AIML - CAPSTONE PROJECT

PNEUMONIA DETECTION CHALLENGE

Mentor : Srihari Nalabolu

Team

NAME	EMAIL	PHONE NUMBER
Guru Siddarth J	siddarth.mine@gmail.com	+91 9743338960
Keyur Singh	krajput@ufl.edu	+1 7866816931
Michael Prasan S	michaelprasansm@gmail.com	+91 9788807095
M V Pavan kumar	pavanmv.147@gmail.com	+91 9743817778
Swastika Samanta	sstika1920@gmail.com	+91 9049630396
Vignesh Jairam	vigirasi@gmail.com	+91 6383442844

Repository

The below link has all the artifacts of the Projects. Following are the documents that are in the repository.

Repo Link: [Capstonse-Project-CV1]

Artifacts:

1. Code of the project in both HTML and IPYNB Formats.
2. Submission file in CSV file on the Object Detection Outputs.
3. Presentation of the Project Outline.
4. Detailed Project Report.

Abstract

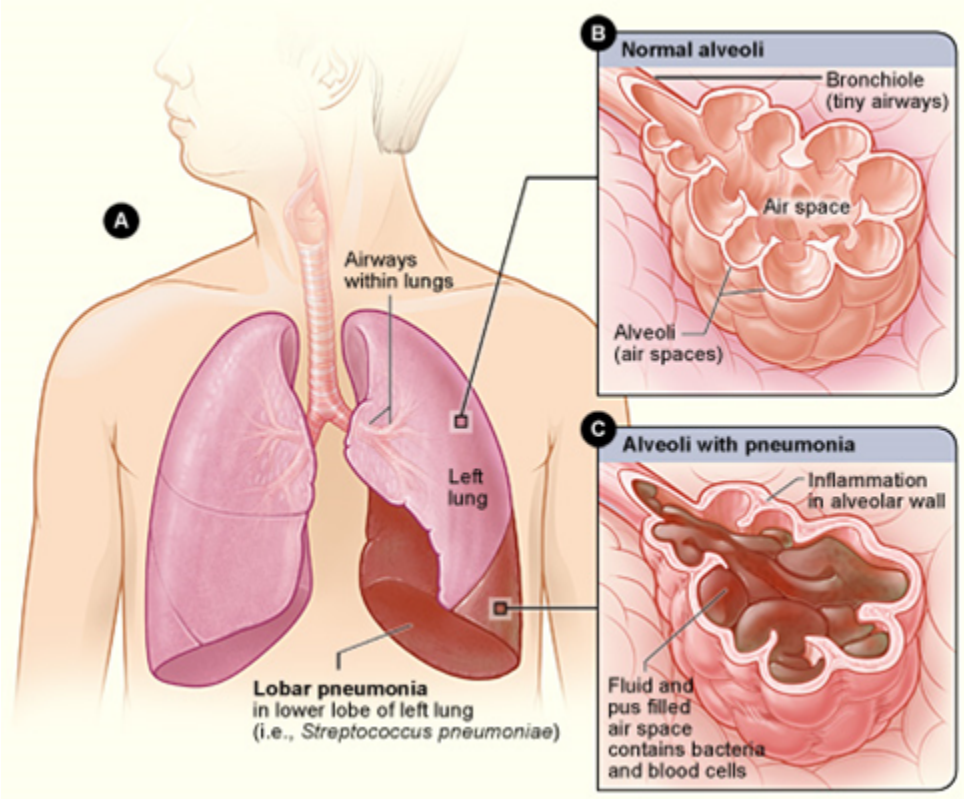
- Pneumonia is an infection of one or both of the lungs caused by bacteria, viruses, or fungi.
- There are more than 30 different causes of pneumonia, and they're grouped by the cause. The main types of pneumonia are bacterial, viral, and mycoplasma pneumonia.
- A cough that produces green, yellow, or bloody mucus is the most common symptom of pneumonia. Other symptoms include fever, shaking chills, shortness of breath, low energy, and extreme tiredness.
- Pneumonia can often be diagnosed with a thorough history and physical exam. Tests used to look at the lungs, blood tests, and tests done on the sputum you cough up may also be used.
- Treatment depends on the type of pneumonia you have. Antibiotics are used for bacterial pneumonia. It may also speed recovery from mycoplasma pneumonia and some special cases. Most viral pneumonias don't have a specific treatment and just get better on their own. Other treatments may include a healthy diet, more fluids, rest, oxygen therapy, and medicine for pain, cough, and fever control.
- Most people with pneumonia respond well to treatment, but pneumonia can cause serious lung and infection problems. It can even be deadly.
- The aim of this project is to develop a software system to detect the disease Pneumonia using Chest x-ray images.
- This is achieved by using multiple convolutional neural network layers where the chest x-ray images are tested, trained and validated.

Problem Statement

Pneumonia is an infection in one or both lungs. Bacteria, viruses, and fungi cause it. The infection causes inflammation in the air sacs in your lungs, which are called alveoli. Pneumonia accounts for over 15% of all deaths of children under 5 years old internationally. In 2017, 920,000 children under the age of 5 died from the disease. It requires review of a chest radiograph (CXR) by highly trained specialists and confirmation through clinical history, vital

signs and laboratory exams. Pneumonia usually manifests as an area or areas of increased opacity on CXR. However, the diagnosis of pneumonia on CXR is complicated because of a number of other conditions in the lungs such as fluid overload (pulmonary edema), bleeding, volume loss (atelectasis or collapse), lung cancer, or postradiation or surgical changes. Outside of the lungs, fluid in the pleural space (pleural effusion) also appears as increased opacity on CXR. When available, comparison of CXRs of the patient taken at different time points and correlation with clinical symptoms and history are helpful in making the diagnosis. CXRs are the most commonly performed diagnostic imaging study. A number of factors such as positioning of the patient and depth of inspiration can alter the appearance of the CXR, complicating interpretation further. In addition, clinicians are faced with reading high volumes of images every shift.

We aim to achieve this by proposing a model using the Deep Learning method. The dataset for the project is collected from kaggle which contains images of infected and normal lungs. Dataset is divided into 3 categories such as train, test and validate.



Objectives

- To classify patients' chest X-ray images as a pneumonia case or a normal case.
- To build a pneumonia detection system, to locate the position of inflammation in an image
- Assist physicians to make better clinical decisions or even replace human judgment in certain functional areas of healthcare (eg, radiology).
- Guided by relevant clinical questions, powerful AI techniques can unlock clinically relevant information hidden in the massive amount of data, which in turn can assist clinical decision making.

Use cases

- Pneumonia is the single largest infectious cause of death in children worldwide. Pneumonia killed 740,180 children under the age of 5 in 2019, accounting for 14% of all deaths of children under five years old but 22% of all deaths in children aged 1 to 5.
- The infection is caused by viruses, bacteria, or fungi and can be prevented by immunization, adequate nutrition, and by addressing environmental factors.
- ·Pneumonia caused by bacteria can be treated with antibiotics, but only one third of children with pneumonia receive the antibiotics they need.
- · Detecting pneumonia is a difficult challenge.
- · Currently, chest X-rays are one of the best methods for the detection of pneumonia. X-rays are the most common and widely available diagnostic imaging technique, playing a crucial role in clinical care and epidemiological studies.
- ·It requires review of a chest radiograph (CXR) by highly trained specialists and confirmation through clinical history, vital signs and laboratory exams.
- ·Pneumonia usually manifests as an area or areas of increased opacity on CXR. However, the diagnosis of pneumonia on CXR is complicated because of a number of other conditions in the lungs such as fluid overload (pulmonary edema), bleeding, volume loss (atelectasis or collapse), lung cancer, or post-radiation or surgical changes.
- Outside of the lungs, fluid in the pleural space (pleural effusion) also appears as increased opacity on CXR.

- When available, comparison of CXRs of the patient taken at different time points and correlation with clinical symptoms and history are helpful in making the diagnosis.
- CXRs are the most commonly performed diagnostic imaging study. A number of factors such as positioning of the patient and depth of inspiration can alter the appearance of the CXR , complicating interpretation further.
- ·In addition to this , experts are faced with the prospect of reading large volumes of images which may prove to be impossible and human error may creep in.
- ·In 2017,researchers at Stanford came up with a deep learning algorithm capable of diagnosing pneumonia from chest X-ray images.In just over a month of development,the algorithm outperformed expert radiologists in detection.
- Artificial Intelligence can help radiologists and physicians in decision making and can replace human intervention in some cases.
- Algorithms can recognise complex patterns and provide insights and interpretation to big data.
- While machines are better at handling large data and computation,humans can be the perfect complement to that by adding the conscious element and use the inputs given by the machine for better clinical decisions and treatment.

Dataset

- We will be using the RSNA Pneumonia Detection Challenge for this project.

- This is a two-stage challenge. You will need the images for the current stage - provided as stage_2_train_images.zip and stage_2_test_images.zip. You will also need the training data - stage_2_train_labels.csv - and the sample submission stage_2_sample_submission.csv, which provides the IDs for the test set, as well as a sample of what your submission should look like. The file stage_2_detailed_class_info.csv contains detailed information about the positive and negative classes in the training set, and may be used to build more nuanced models.

- File descriptions

    - **stage_2_train.csv** - the training set. Contains patientIds and bounding box / target information.

    - **stage_2_sample_submission.csv** - a sample submission file in the correct format. Contains patientIds 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** - provides detailed information about the type of positive or negative class for each image.
- **train_images** - Contains DICOM images used to train the model.
- **test_images** - Contains DICOM images used to test the model.

- Data fields
  - **patientId** - A patientId. Each patientId corresponds to a unique image.
  - **x** - the upper-left x coordinate of the bounding box.
  - **y** - the upper-left y coordinate of the bounding box.
  - **width** - the width of the bounding box.
  - **height** - the height of the bounding box.
  - **Target** - the binary Target, indicating whether this sample has evidence of pneumonia.

Implementation and EDA:

Importing Packages necessary for the project

```
import pandas as pd
import numpy as np
import os
import cv2
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
import PIL
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.layers import Layer, Convolution2D, Flatten, Dense
from tensorflow.keras.layers import Concatenate, UpSampling2D, Conv2D, Reshape,
GlobalAveragePooling2D
from tensorflow.keras import losses,optimizers
from tensorflow.keras.layers import Dense,  Activation,
Flatten,Dropout,MaxPooling2D,BatchNormalization
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras import backend as K
import tensorflow.keras.utils as pltUtil
from tensorflow.keras.utils import Sequence
import math
import scipy.stats as stats
from sklearn.metrics import confusion_matrix
from matplotlib.patches import Rectangle
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense ,LeakyReLU,GlobalMaxPooling2D
from tensorflow.keras import regularizers, optimizers
from sklearn.metrics import r2_score
from tensorflow.keras.models import load_model
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from keras import applications
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications import ResNet50V2
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras import datasets
from tensorflow.keras.layers import BatchNormalization
from tabulate import tabulate
from texttable import Texttable
import warnings
warnings.filterwarnings('ignore')
```

EDA on stage_2_train_labels File

```
train_labels= pd.read_csv('Dataset - New/stage_2_train_labels.csv')
print('First five rows of Training set:\n', train_labels.head())
```

```
First five rows of Training set:
                                  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
```

```
print(train_labels.iloc[0])
```

```
patientId    0004cfab-14fd-4e49-80ba-63a80b6bddd6
x                                    NaN
y                                    NaN
width                                NaN
height                               NaN
Target                                 0
Name: 0, dtype: object
```

train_labels.shape

```
(30227, 6)
The data frame has 30227 rows and 6 Columns
```

train_labels.size

```
181362
```

train_labels.describe

```
<bound method NDFrame.describe of                              patientId      x      y  width  height  \
0      0004cfab-14fd-4e49-80ba-63a80b6bddd6    NaN    NaN    NaN     NaN
1      00313ee0-9eaa-42f4-b0ab-c148ed3241cd    NaN    NaN    NaN     NaN
2      00322d4d-1c29-4943-afc9-b6754be640eb    NaN    NaN    NaN     NaN
3      003d8fa0-6bf1-40ed-b54c-ac657f8495c5    NaN    NaN    NaN     NaN
4      00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0   379.0
...                                     ...    ...    ...    ...     ...
30222  c1ec14ff-f6d7-4b38-b0cb-fe07041cbdc8  185.0  298.0  228.0   379.0
30223  c1edf42b-5958-47ff-a1e7-4f23d99583ba    NaN    NaN    NaN     NaN
30224  c1f6b555-2eb1-4231-98f6-50a963976431    NaN    NaN    NaN     NaN
30225  c1f7889a-9ea9-4acb-b64c-b737c929599a  570.0  393.0  261.0   345.0
30226  c1f7889a-9ea9-4acb-b64c-b737c929599a  233.0  424.0  201.0   356.0

       Target
0           0
```

```
1        0
2        0
3        0
4        1
...    ...
30222    1
30223    0
30224    0
30225    1
30226    1

[30227 rows x 6 columns]>
```

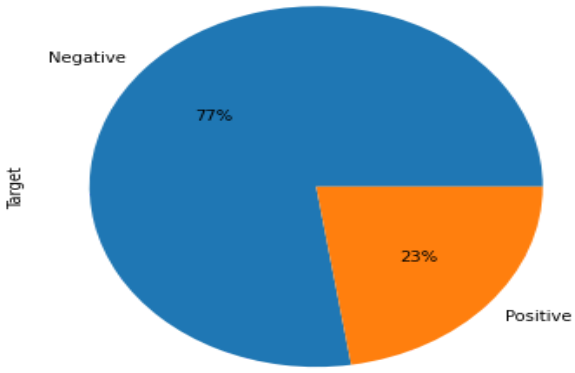
```
train_labels['patientId'].nunique()
```

```
26684
```

There are 26684 Unique patientIds in the data frame, the above shows that each patientId has multiple rows in the dataframe

```
# Let us find how many patients out of 26684 has Pneumonia or not in our dataset
# Creating Variables with and without Pneumonia to final the percentage distribution of data
one = train_labels[train_labels.Target == 1].drop_duplicates('patientId').shape[0]
zero = train_labels[train_labels.Target == 0].drop_duplicates('patientId').shape[0]
total = train_labels.drop_duplicates('patientId').shape[0]
print(f'No of Patients with Pneumonia: {one} - {round(one/total*100, 0)}% of the entire dataset')
print(f'No of Patients without Pneumonia: {zero} - {round(zero/total*100, 0)}% of the entire dataset')
_ = train_labels.drop_duplicates('patientId').drop_duplicates('patientId')['Target'].value_counts().plot(kind = 'pie', autopct = '%.0f%%', labels = ['Negative', 'Positive'], figsize = (10, 6))
```

```
No of Patients with Pneumonia: 6012 - 23.0% of the entire dataset
No of Patients without Pneumonia: 20672 - 77.0% of the entire dataset
```



Checking nulls in bounding box columns:

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

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

Thus, we can see that the number of nulls in bounding box columns are equal to the number of 0's we have in the Target column.

EDA on stage_2_detailed_class_info File

class_labels = pd.read_csv('Dataset - New/stage_2_detailed_class_info.csv')

print('First five rows of Class label dataset are:\n', class_labels.head())

First five rows of Class label dataset are:

	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

Some information about the data field present in the 'stage_2_detailed_class_info.csv' are:

patientId - A patientId. Each patientId corresponds to a unique image

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

```
# Shape of the dataframe
class_labels.shape
```

```
(30227, 2)
```

The data frame has 30227 rows and 2 Columns

```
# Size of the dataframe
class_labels.size
```

```
60454
```

```
# Basic info about the file
class_labels.describe
```

```
<bound method NDFrame.describe of                                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
...                                     ...                         ...
30222  c1ec14ff-f6d7-4b38-b0cb-fe07041cbdc8                 Lung Opacity
30223  c1edf42b-5958-47ff-a1e7-4f23d99583ba                      Normal
30224  c1f6b555-2eb1-4231-98f6-50a963976431                      Normal
30225  c1f7889a-9ea9-4acb-b64c-b737c929599a                 Lung Opacity
30226  c1f7889a-9ea9-4acb-b64c-b737c929599a                 Lung Opacity
[30227 rows x 2 columns]>
```

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

```
26684
```

```
# Printing the unique values in Class column to do the further analysis
print(sorted(class_labels['class'].unique()))
```

```
['Lung Opacity', 'No Lung Opacity / Not Normal', 'Normal']
```

There are 26684 Unique patientIds in the data frame, the above shows that each patientId has multiple rows in the dataframe

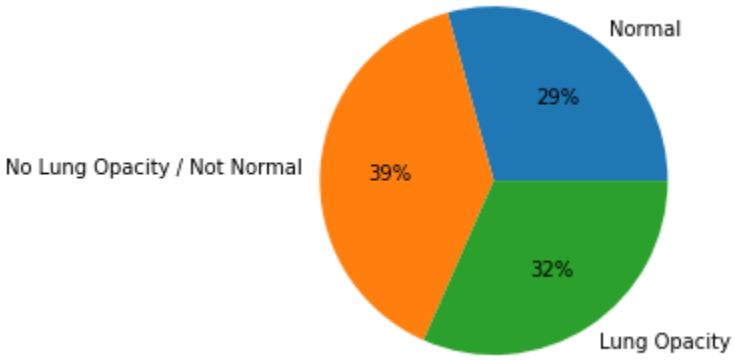
The data is distributed across in the class column as below

(class_labels['class'].value_counts() / class_labels['patientId'].count()) * 100

No Lung Opacity / Not Normal 39.107421
Lung Opacity 31.610812
Normal 29.281768
Name: class, dtype: float64

figsize = (10, 6)

_ = class_labels['class'].value_counts().sort_index(ascending = False).plot(kind = 'pie', autopct = '%.0f%%').set_ylabel('')



Checking nulls in class_labels:

print('Number of nulls in class columns: {}'.format(class_labels['class'].isnull().sum()))

Number of nulls in class columns: 0

Checking whether each patientId has only one type of class or not

class_labels.groupby(['patientId'])['class'].nunique().max()

1
So, each patient is associated with 1 class

Merging of both the dataframes

Both train_lables and class_labels can be merged because of the same number of rows and the same number of unique patient IDs available in both the dataframes.

```
# Merging the two dataset - 'train_labels' and 'class_labels':
training_data = pd.concat([train_labels, class_labels['class']], axis = 1)
print('After merging, the dataset looks like: \n')
training_data.head()
```

After merging, the dataset looks like:

	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

```
# Shape of the dataframe
training_data.shape
```

(30227, 7)

```
# Size of the dataframe
```

```
training_data.size
```

```
211589
```

```
# basic info
training_data.describe
```

```
<bound method NDFrame.describe of                                patientId     x     y  width  height  \
0     0004cfab-14fd-4e49-80ba-63a80b6bddd6    NaN   NaN   NaN    NaN
1     00313ee0-9eaa-42f4-b0ab-c148ed3241cd    NaN   NaN   NaN    NaN
2     00322d4d-1c29-4943-afc9-b6754be640eb    NaN   NaN   NaN    NaN
3     003d8fa0-6bf1-40ed-b54c-ac657f8495c5    NaN   NaN   NaN    NaN
4     00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0  379.0
...                                    ...   ...   ...   ...    ...
30222  c1ec14ff-f6d7-4b38-b0cb-fe07041cbdc8  185.0  298.0  228.0  379.0
30223  c1edf42b-5958-47ff-a1e7-4f23d99583ba    NaN   NaN   NaN    NaN
30224  c1f6b555-2eb1-4231-98f6-50a963976431    NaN   NaN   NaN    NaN
30225  c1f7889a-9ea9-4acb-b64c-b737c929599a  570.0  393.0  261.0  345.0
30226  c1f7889a-9ea9-4acb-b64c-b737c929599a  233.0  424.0  201.0  356.0

      Target                  class
0          0  No Lung Opacity / Not Normal
1          0  No Lung Opacity / Not Normal
2          0  No Lung Opacity / Not Normal
3          0                     Normal
4          1                Lung Opacity
...      ...                        ...
30222      1                Lung Opacity
30223      0                     Normal
30224      0                     Normal
30225      1                Lung Opacity
30226      1                Lung Opacity

[30227 rows x 7 columns]>
```

```
training_data.isna().sum() #checking for null values
```

```
patientId       0
x           20672
y           20672
width       20672
height      20672
Target          0
class           0
dtype: int64
```

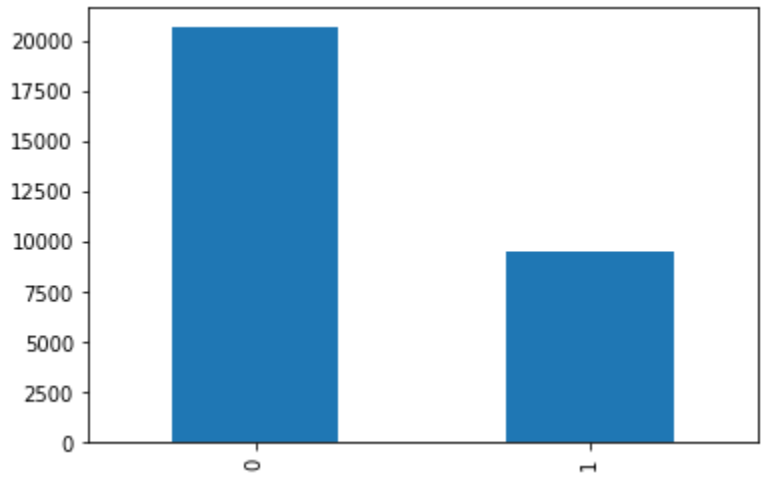
```
training_data['Target'].value_counts() #Value count of the target
```

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

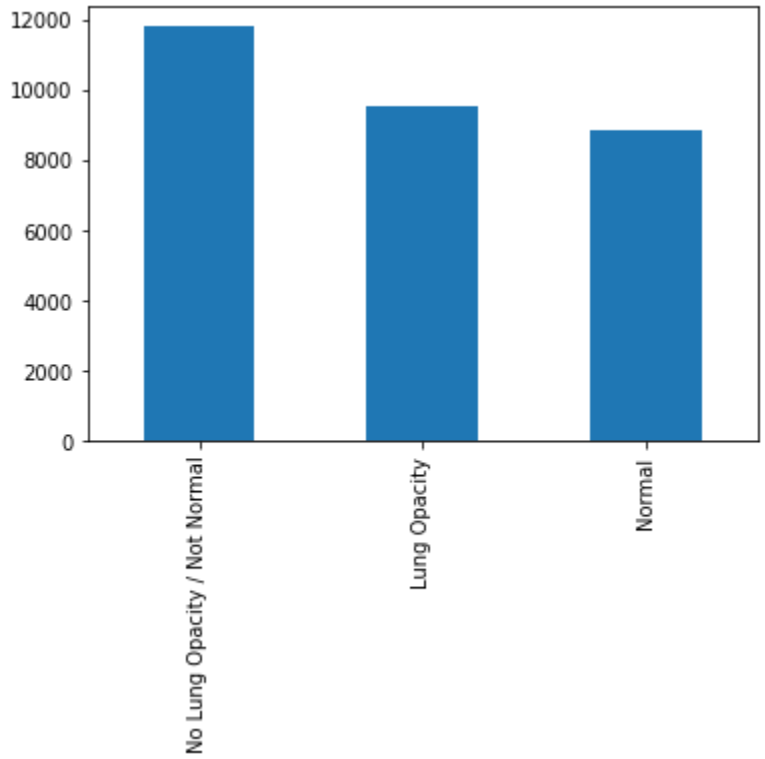
```
training_data['class'].value_counts() #Frequency of the class
```

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

```
pd.value_counts(training_data['Target']).plot(kind="bar") #Bar plot
```



pd.value_counts(training_data['class']).plot(kind="bar") #Bar plot

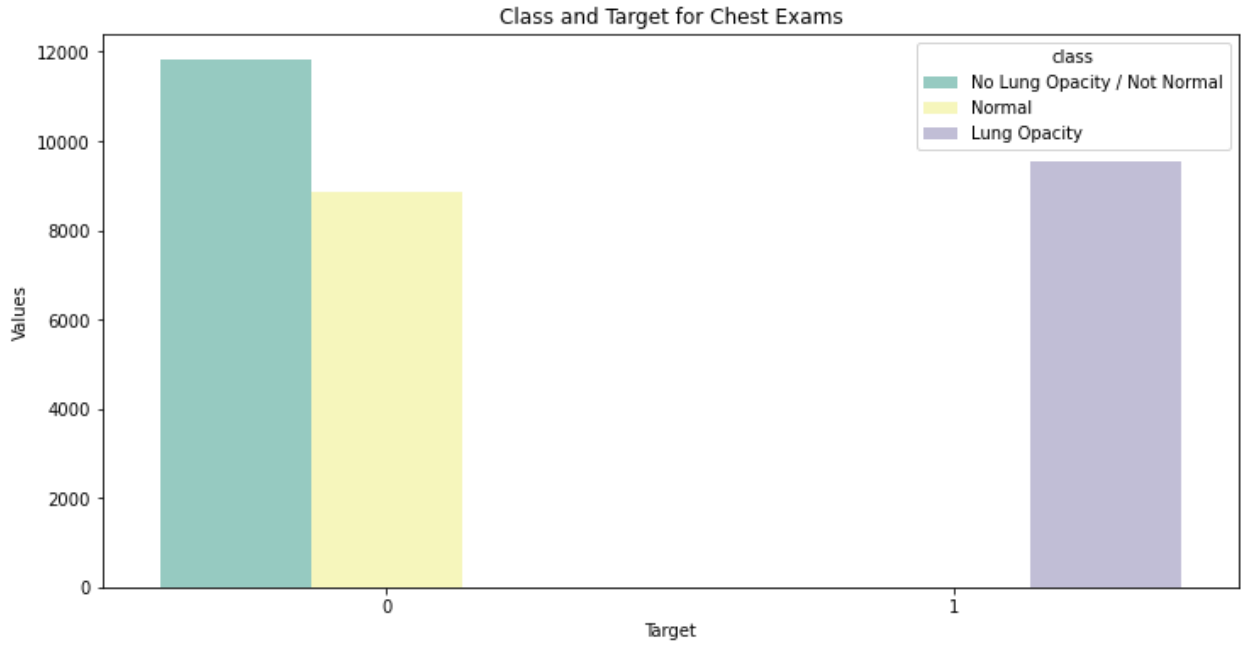


training_data.head(5)

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*

```
## Let us now see how the target in train_labels and class in class_labels are associated
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 for Chest Exams')
```



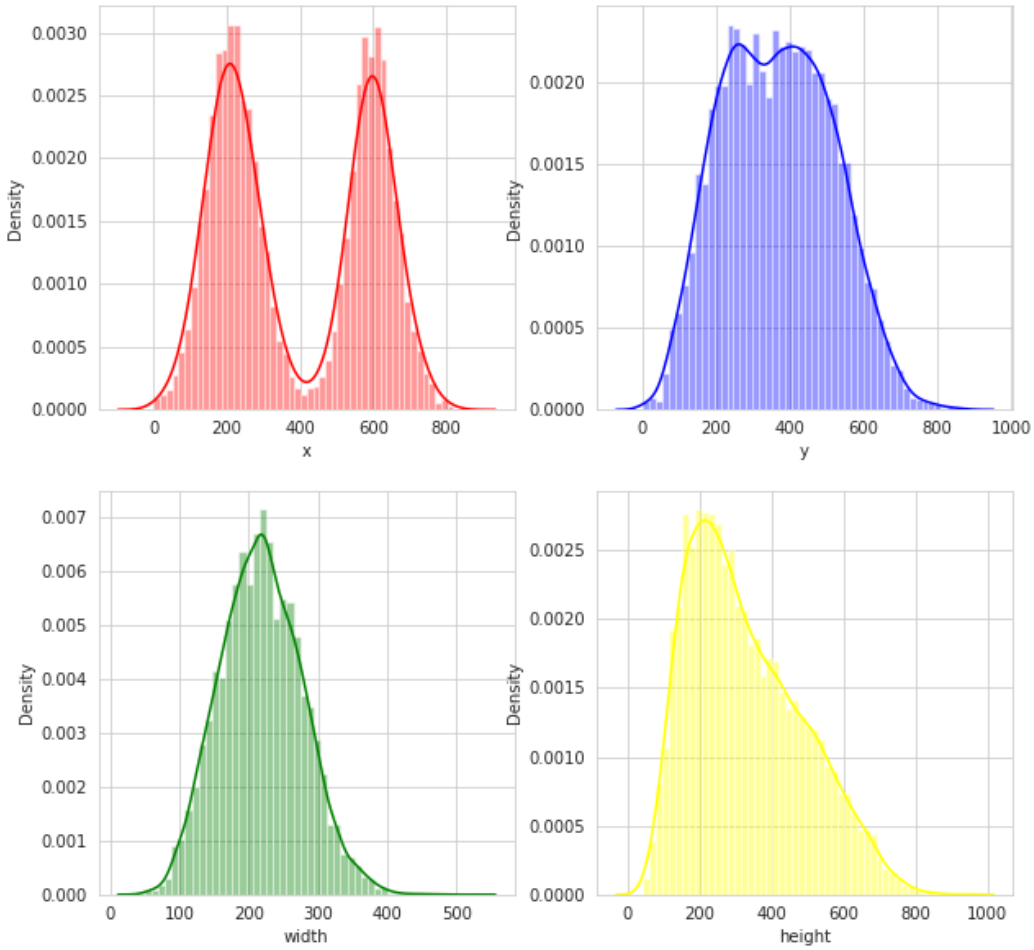
So, from the above we can conclude the below Grouping:

If the Target is 0, the class would be either - No Lung Opacity / Not Normal or Normal

If the Target is 1, the class would be Lung Opacity

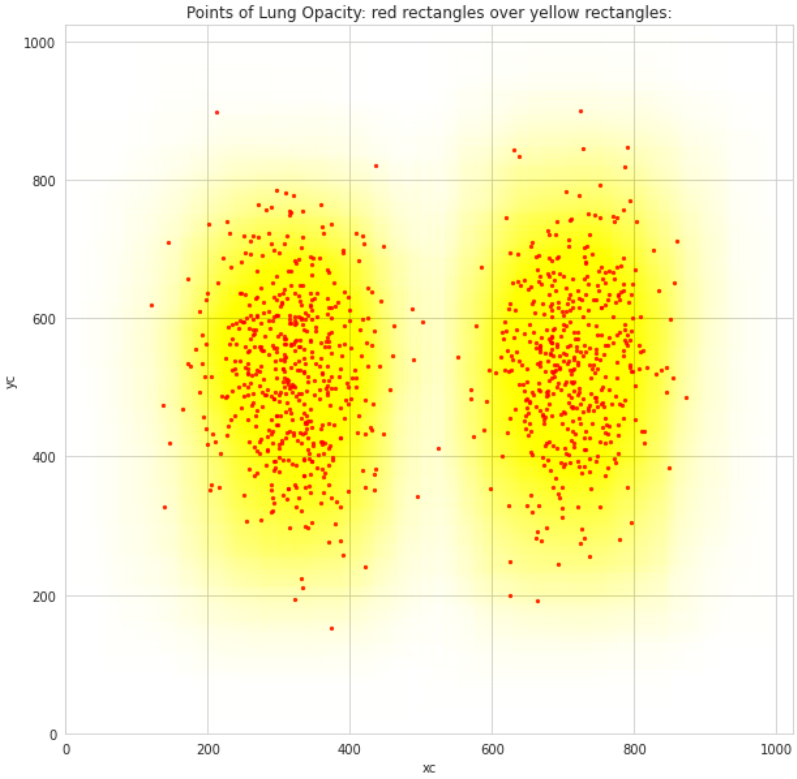
```
#Distribution Plot
target1 = training_data[training_data['Target']==1]
sns.set_style('whitegrid')
plt.figure()
fig, ax = plt.subplots(2,2,figsize=(10,10))
```

```
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="yellow", ax=ax[1,1])
locs, labels = plt.xticks()
plt.tick_params(axis='both', which='major', labelsize=10)
plt.show()
```




```
#Heatmap
fig, ax = plt.subplots(1,1,figsize=(10,10))
target2 = target1.sample(1000)
target2['xc'] = target2['x'] + target2['width'] / 2
target2['yc'] = target2['y'] + target2['height'] / 2
plt.title("Points of Lung Opacity: red rectangles over yellow rectangles:")
target2.plot.scatter(x='xc', y='yc', xlim=(0,1024), ylim=(0,1024), ax=ax, alpha=0.8, marker=".",
color="red")
for i, crt_sample in target2.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="yellow"))
plt.show()
```



Train Folder Images

```
print('Number of images in training images folders are: {}.'.format(len(os.listdir('Dataset -
New/stage_2_train_images (1)-001'))))
```

Number of images in training images folders are: 26684.

We can see that in the training images folder we have just 26684 images which is the same as that of the unique patientId's present in either of the csv files. Thus, we can say that each of the unique patientId's present in either of the csv files corresponds to an image present in the folder.

```
# Creating images dataframe with 2 columns - path of the dicom file and associated patientId
# Every DCM filename is the patientId
from glob import glob
training_image_path = 'Dataset - New/stage_2_train_images (1)-001'
images = pd.DataFrame({'path': glob(os.path.join(training_image_path, '*.dcm'))})
images['patientId'] = images['path'].map(lambda x:os.path.splitext(os.path.basename(x))[0])
```

```
print('Columns in the training images dataframe: {}'.format(list(images.columns)))
```

Columns in the training images dataframe: ['path', 'patientId']

```
# Merging the images data frame with training_data dataframe
training_data = training_data.merge(images, on = 'patientId', how = 'left')
print('After merging the two dataframe, the training_data has {} rows and {}
columns.'.format(training_data.shape[0], training_data.shape[1]))
```

After merging the two dataframe, the training_data has 30227 rows and 8 columns.

```
training_data.head()
```

	patientId	x	y	width	height	Target	class	path
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	Dataset - New/stage_2_train_images (1)-001\000...
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	Dataset - New/stage_2_train_images (1)-001\003...
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	Dataset - New/stage_2_train_images (1)-001\003...
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0	Normal	Dataset - New/stage_2_train_images (1)-001\003...
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1	Lung Opacity	Dataset - New/stage_2_train_images (1)-001\004...

```
# Shape of the dataframe
training_data.shape
```

```
(30227, 8)
```

```
# Size of the dataframe
training_data.size
```

```
241816
```

```
# basic info of the dataframe
training_data.describe
```

```
<bound method NDFrame.describe of                              patientId     x     y  width  height  \
0      0004cfab-14fd-4e49-80ba-63a80b6bddd6   NaN   NaN   NaN    NaN
1      00313ee0-9eaa-42f4-b0ab-c148ed3241cd   NaN   NaN   NaN    NaN
2      00322d4d-1c29-4943-afc9-b6754be640eb   NaN   NaN   NaN    NaN
3      003d8fa0-6bf1-40ed-b54c-ac657f8495c5   NaN   NaN   NaN    NaN
4      00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0   379.0
...                                     ...   ...   ...   ...     ...
30222  c1ec14ff-f6d7-4b38-b0cb-fe07041cbdc8  185.0  298.0  228.0   379.0
30223  c1edf42b-5958-47ff-a1e7-4f23d99583ba   NaN   NaN   NaN    NaN
30224  c1f6b555-2eb1-4231-98f6-50a963976431   NaN   NaN   NaN    NaN
30225  c1f7889a-9ea9-4acb-b64c-b737c929599a  570.0  393.0  261.0   345.0
30226  c1f7889a-9ea9-4acb-b64c-b737c929599a  233.0  424.0  201.0   356.0

      Target                  class  \
0          0  No Lung Opacity / Not Normal
1          0  No Lung Opacity / Not Normal
2          0  No Lung Opacity / Not Normal
3          0                 Normal
4          1           Lung Opacity
...      ...                    ...
30222      1           Lung Opacity
30223      0                 Normal
30224      0                 Normal
30225      1           Lung Opacity
30226      1           Lung Opacity
```

```
                                        path
0      Dataset - New/stage_2_train_images (1)-001\000...
1      Dataset - New/stage_2_train_images (1)-001\003...
2      Dataset - New/stage_2_train_images (1)-001\003...
3      Dataset - New/stage_2_train_images (1)-001\003...
4      Dataset - New/stage_2_train_images (1)-001\004...
...                                          ...
30222  Dataset - New/stage_2_train_images (1)-001\c1e...
30223  Dataset - New/stage_2_train_images (1)-001\c1e...
30224  Dataset - New/stage_2_train_images (1)-001\c1f...
30225  Dataset - New/stage_2_train_images (1)-001\c1f...
30226  Dataset - New/stage_2_train_images (1)-001\c1f...

[30227 rows x 8 columns]>
```

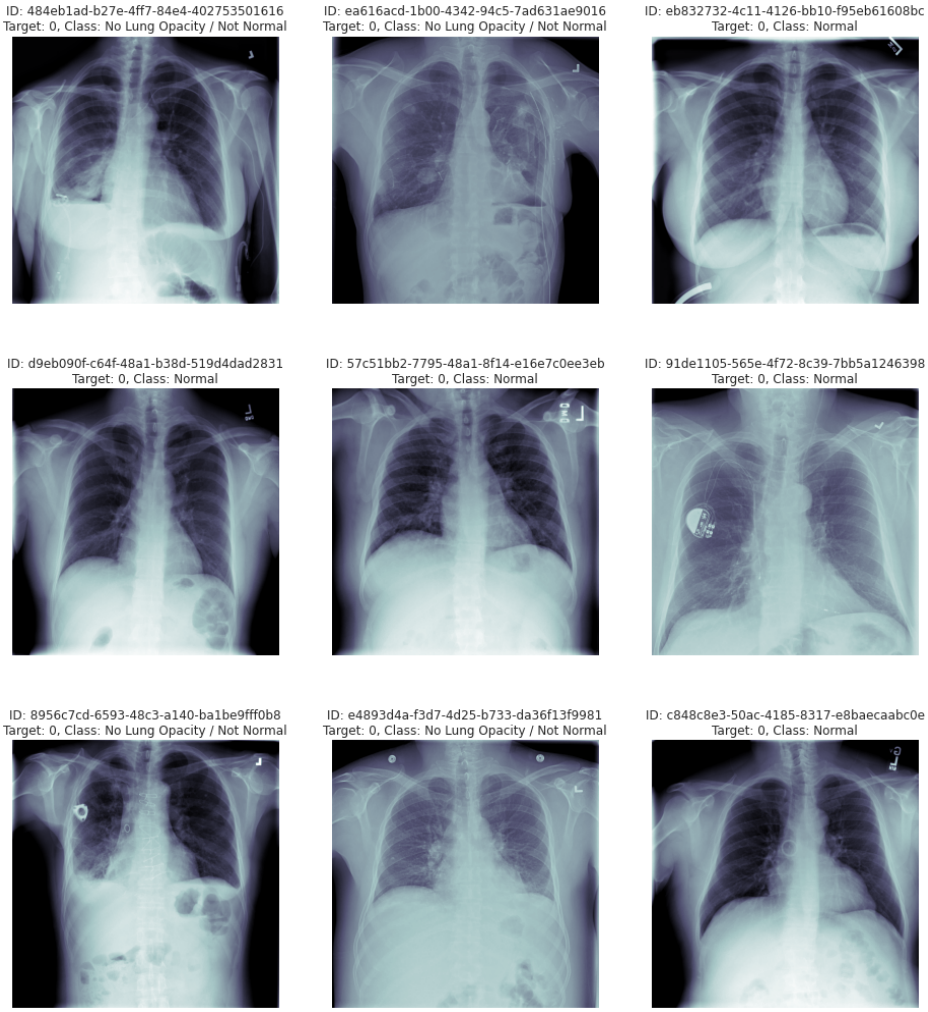
Function to show images from train folder with bounding boxes

```
from matplotlib.patches import Rectangle
import pydicom as dcm

def show_dicom_images(data, df, img_path):
  img_data = list(data.T.to_dict().values())
  f, ax = plt.subplots(3, 3, figsize = (16, 18))

  for i, row in enumerate(img_data):
    image = row['patientId'] + '.dcm'
    path = os.path.join(img_path, image)
    data = dcm.read_file(path)
    rows = df[df['patientId'] == row['patientId']]
    data_img = dcm.dcmread(path)
    ax[i//3, i%3].imshow(data_img.pixel_array, cmap = plt.cm.bone)
    ax[i//3, i%3].axis('off')
    ax[i//3, i%3].set_title('ID: {}\nTarget: {}, Class: {}'\
                        .format(row['patientId'], row['Target'],
                              row['class'], row['x'],
                              row['y'], row['width'],
                              row['height']))
    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'],
                                      color = 'blue', alpha = 0.15))
plt.show()
```

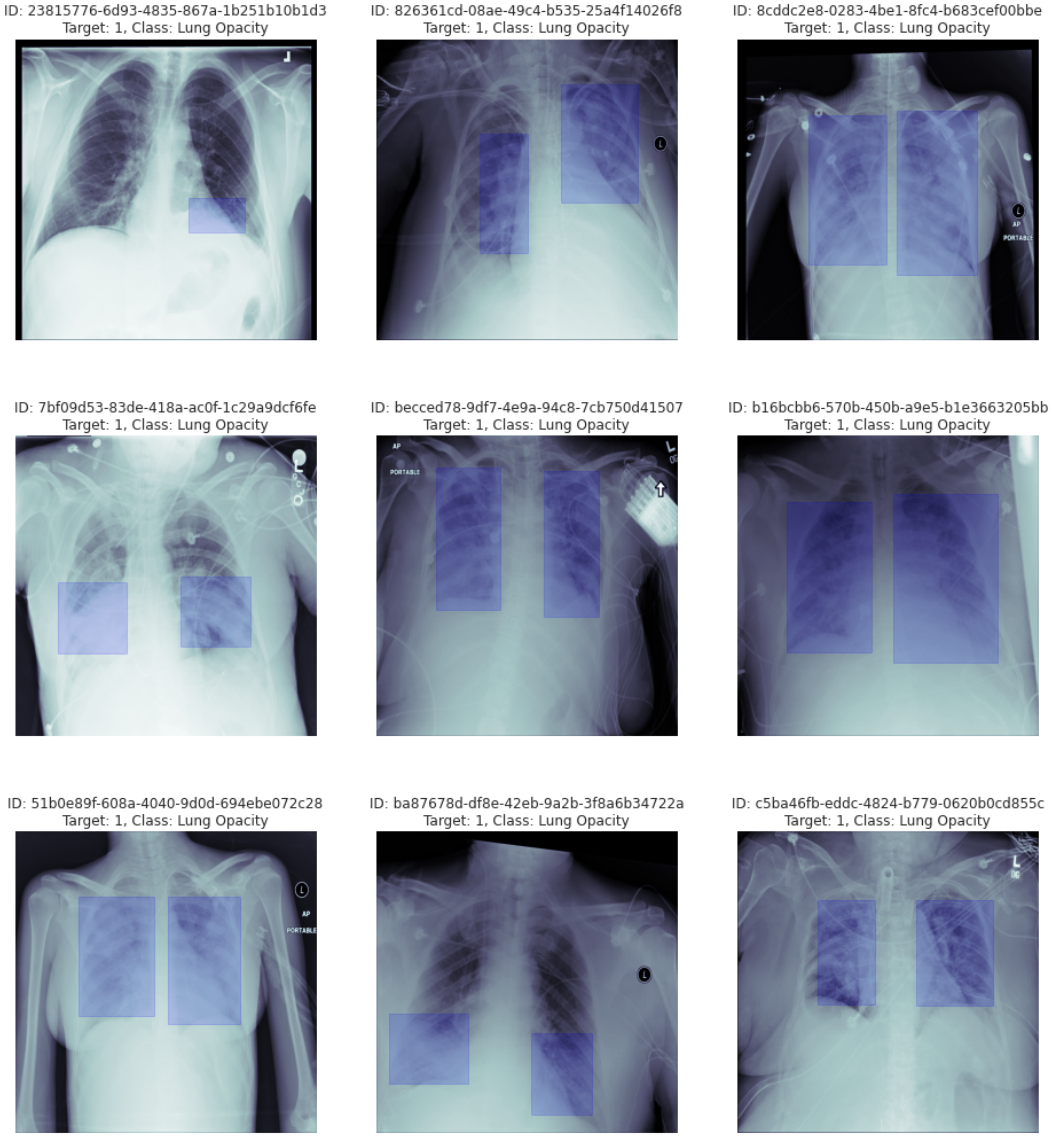
Showing images without pneumonia

```
show_dicom_images(data = training_data.loc[(training_data['Target'] == 0)].sample(9),
                  df = training_data, img_path = 'Dataset - New/stage_2_train_images (1)-001')
```



Showing images with pneumonia

```
show_dicom_images(data = training_data.loc[(training_data['Target'] == 1)].sample(9),
                  df = training_data, img_path = 'Dataset - New/stage_2_train_images (1)-001')
```



EDA on Metadata of DICOM Images

Medical images are stored in a special format known as DICOM files (*.dcm). They contain a combination of header metadata as well as underlying raw image arrays for pixel data. In Python, one popular library to access and manipulate DICOM files is the pydicom module. To use the pydicom library, first find the DICOM file for a given patientId by simply looking for the matching file in the stage_2_train_images/ folder, and the use the pydicom.read_file() method to load the data

```
sample_patientId = train_labels['patientId'][0]
dcm_file = training_image_path+'/{}.dcm'.format(sample_patientId)
dcm_data = dcm.read_file(dcm_file)
print('Metadata of the image consists of \n', dcm_data)
```

```
Metadata of the image consists of
 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

Adding necessary Columns for EDA from Metadata of the DICOM Images

```
columns_to_add = ['Modality', 'PatientAge', 'PatientSex', 'BodyPartExamined', 'ViewPosition',
'ConversionType', 'Rows', 'Columns', 'PixelSpacing']
from tqdm import tqdm, tqdm_notebook

def parse_dicom_data(data_df, data_path):
  for col in columns_to_add:
    data_df[col] = None
  image_names = os.listdir('Dataset - New/stage_2_train_images (1)-001/')
  for i, img_name in tqdm_notebook(enumerate(image_names)):
    imagepath = os.path.join('Dataset - New/stage_2_train_images (1)-001/', img_name)
    data_img = dcm.read_file(imagepath)
    idx = (data_df['patientId'] == data_img.PatientID)
    data_df.loc[idx, 'Modality'] = data_img.Modality
    data_df.loc[idx, 'PatientAge'] = pd.to_numeric(data_img.PatientAge)
```

```
    data_df.loc[idx, 'PatientSex'] = data_img.PatientSex
    data_df.loc[idx, 'BodyPartExamined'] = data_img.BodyPartExamined
    data_df.loc[idx, 'ViewPosition'] = data_img.ViewPosition
    data_df.loc[idx, 'ConversionType'] = data_img.ConversionType
    data_df.loc[idx, 'Rows'] = data_img.Rows
    data_df.loc[idx, 'Columns'] = data_img.Columns
    data_df.loc[idx, 'PixelSpacing'] = str.format("{:4.3f}", data_img.PixelSpacing[0])

# Parse DICOM Images to pull the metadata from them
parse_dicom_data(training_data,'Dataset - New/stage_2_train_images (1)-001/')

print('So after parsing the information from the dicom images, our training_data data frame has {} rows and {} columns and it looks like:\n'.format(training_data.shape[0], training_data.shape[1]))
training_data.head()
```

patientId	x	y	width	height	Target	class	path	Modality	PatientAge	PatientSex	BodyPartExamined	ViewPosition	ConversionType	Rows	Columns	PixelSpacing
0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	Dataset - New/stage_2_train_images (1)-001\000...	CR	51	F	CHEST	PA	WSD	1024	1024	0.143
00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	Dataset - New/stage_2_train_images (1)-001\003...	CR	48	F	CHEST	PA	WSD	1024	1024	0.194
00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	Dataset - New/stage_2_train_images (1)-001\003...	CR	19	M	CHEST	AP	WSD	1024	1024	0.168
003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0	Normal	Dataset - New/stage_2_train_images (1)-001\003...	CR	28	M	CHEST	PA	WSD	1024	1024	0.143
00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1	Lung Opacity	Dataset - New/stage_2_train_images (1)-001\004...	CR	32	F	CHEST	AP	WSD	1024	1024	0.139

EDA on Modality

```
print('Modality for the images obtained is: {} \n'.format(training_data['Modality'].unique()[0]))
```

Modality for the images obtained is: CR

EDA on Body Part Examined

```
print('The images obtained are of {} areas.'.format(training_data['BodyPartExamined'].unique()[0]))
```

The images obtained are of CHEST areas.

EDA on Understanding Different Positions

```
(training_data['ViewPosition'].value_counts() / training_data['patientId'].count()) * 100
```

AP 50.607073

PA 49.392927

Name: ViewPosition, dtype: float64

AP : Anterior/Posterior (AP)

PA : Posterior/Anterior (PA)

EDA on Conversion Type

```
print('Conversion Type for the data in Training Data: ',
training_data['ConversionType'].unique()[0])
```

Conversion Type for the data in Training Data: WSD

EDA on Gender

```
(training_data['PatientSex'].value_counts() / training_data['patientId'].count()) * 100
```

M   56.955702

F   43.044298

Name: PatientSex, dtype: float64

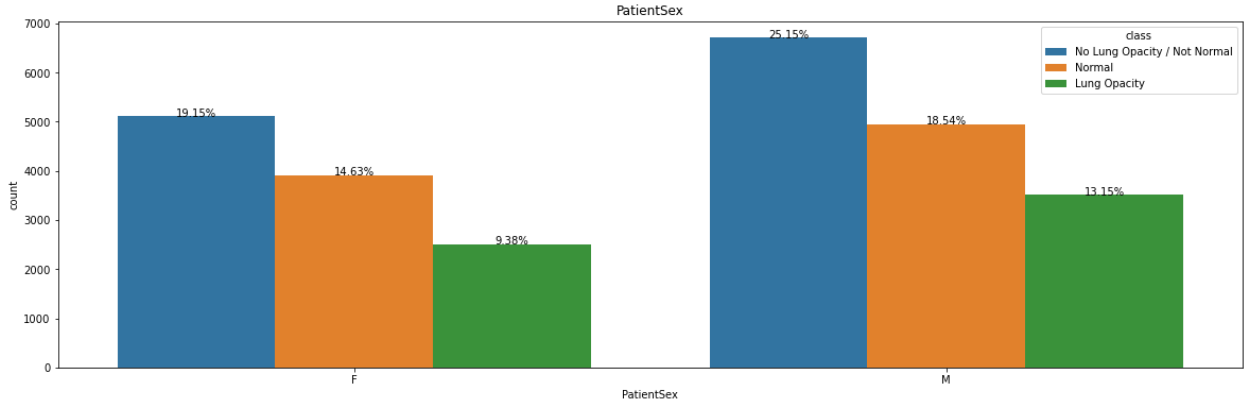
Function to plot graphs on different attributes of the dataframe

```
def drawgraphs(data_file, columns, hue = False, width = 15, showdistribution = True):
  if (hue):
    print('Creating graph for: {} and {}'.format(columns, hue))
  else:
    print('Creating graph for : {}'.format(columns))
  length = len(columns) * 6
  total = float(len(data_file))
  fig, axes = plt.subplots(nrows = len(columns) if len(columns) > 1 else 1, ncols = 1, figsize =
(width, length))
  for index, content in enumerate(columns):
    plt.title(content)
    currentaxes = 0
    if (len(columns) > 1):
      currentaxes = axes[index]
    else:
      currentaxes = axes
    if (hue):
      sns.countplot(x = columns[index], data = data_file, ax = currentaxes, hue = hue)
    else:
      sns.countplot(x = columns[index], data = data_file, ax = currentaxes)
    if(showdistribution):
      for p in (currentaxes.patches):
        height = p.get_height()
```

```
    if (height > 0 and total > 0):
      currentaxes.text(p.get_x() + p.get_width()/2., height + 3, '{:1.2f}%'.format(100*height/total), ha = "center")
```

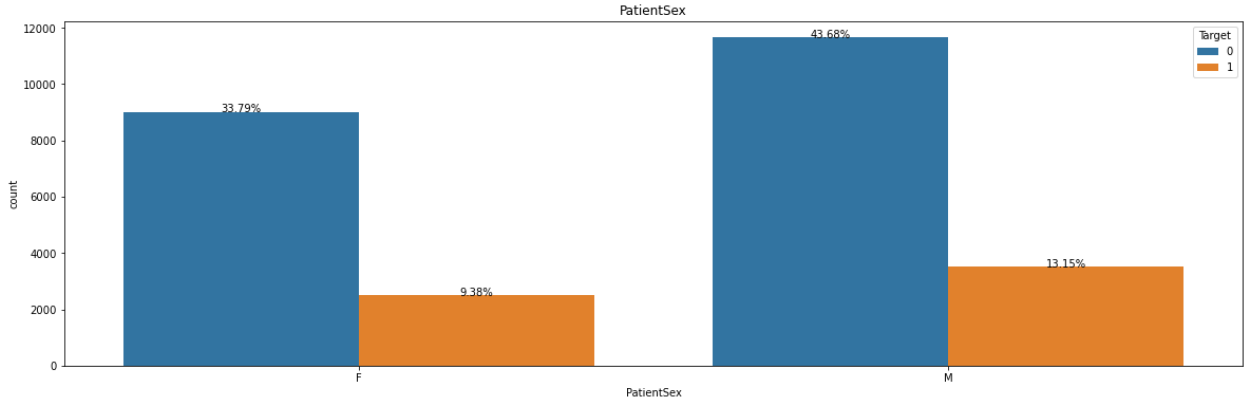
EDA on PatientSex and Count of patientId - Hue by Class

drawgraphs(data_file = training_data.drop_duplicates('patientId'), columns = ['PatientSex'], hue = 'class', width = 20, showdistribution = True)



EDA on PatientSex and Count of patientId. Split by Target

drawgraphs(data_file = training_data.drop_duplicates('patientId'), columns = ['PatientSex'], hue = 'Target', width = 20, showdistribution = True)



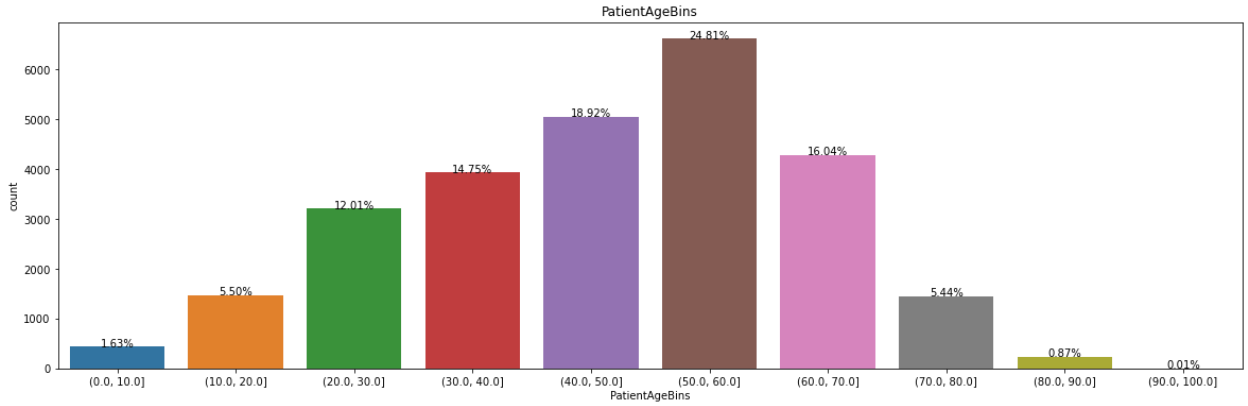
EDA on PatientAge

# Split PatientAge based on into PatientAgeBins
custom_array = np.linspace(0, 100, 11)
training_data['PatientAgeBins'] = pd.cut(training_data['PatientAge'], custom_array)
training_data.drop_duplicates('patientId')['PatientAgeBins'].value_counts()

(50.0, 60.0] 6619
(40.0, 50.0] 5048
(60.0, 70.0] 4279
(30.0, 40.0] 3936
(20.0, 30.0] 3205
(10.0, 20.0] 1468
(70.0, 80.0] 1452
(0.0, 10.0] 435
(80.0, 90.0] 233
(90.0, 100.0] 4
Name: PatientAgeBins, dtype: int64

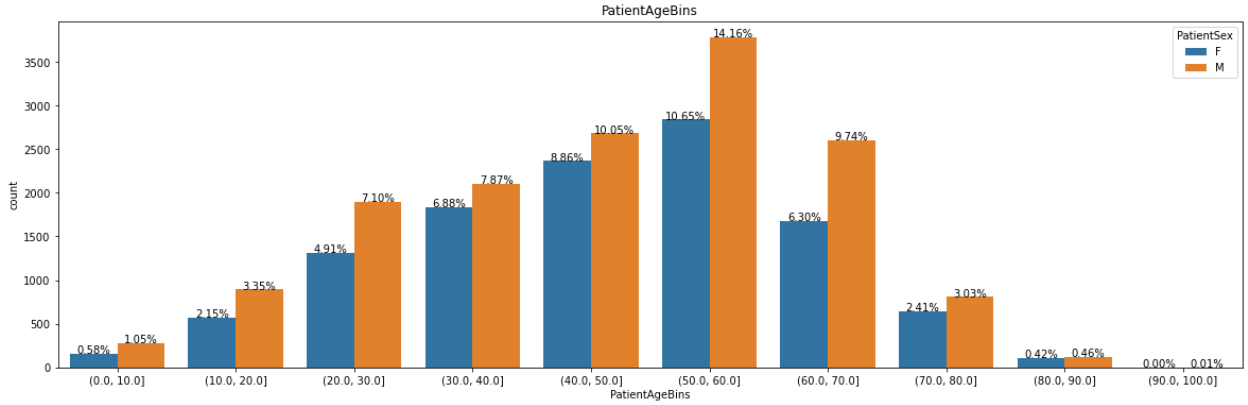
Distribution on Count of Patients based on PatientAgeBins

drawgraphs(data_file = training_data.drop_duplicates('patientId'), columns = ['PatientAgeBins'], width = 20, showdistribution = True)



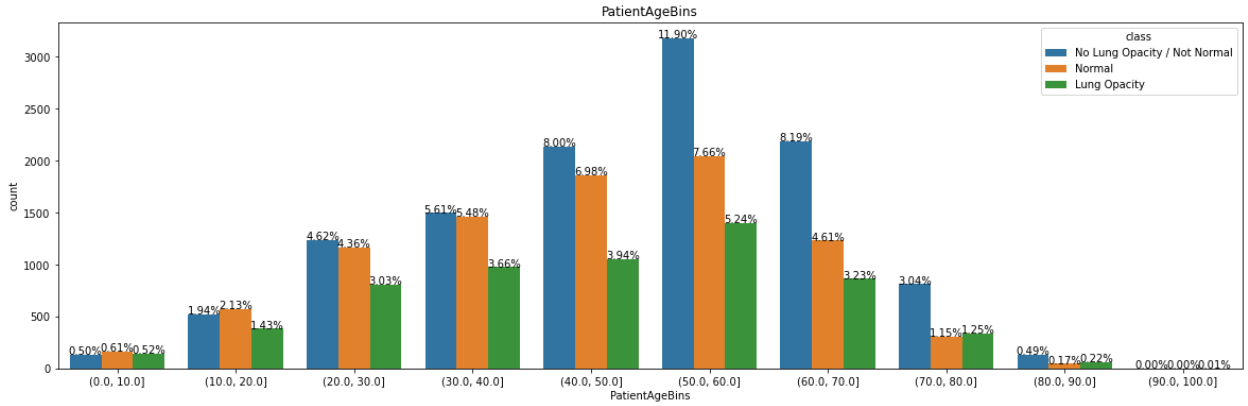
As per our dataset the major images belong to age group between 50 and 60

drawgraphs(data_file = training_data.drop_duplicates('patientId'), columns = ['PatientAgeBins'], hue = 'PatientSex', width = 20, showdistribution = True)

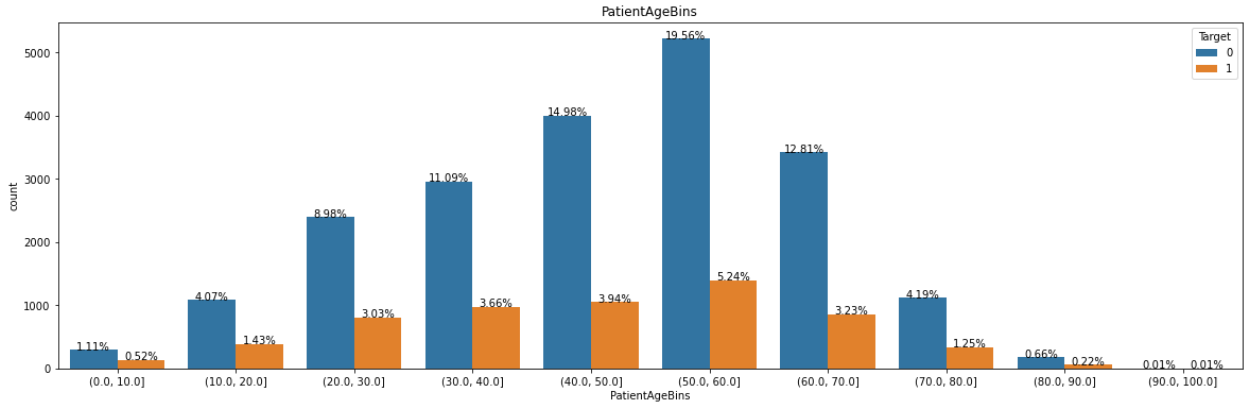


The Images for Male are more than of Female

drawgraphs(data_file = training_data.drop_duplicates('patientId'), columns = ['PatientAgeBins'], hue = 'class', width = 20, showdistribution = True)



drawgraphs(data_file = training_data.drop_duplicates('patientId'), columns = ['PatientAgeBins'], hue = 'Target', width = 20, showdistribution = True)



Test Folder Images

print('Number of images in testing images folders are: {}.'.format(len(os.listdir('Dataset - New/stage_2_test_images'))))

Number of images in testing images folders are: 3000.

Image Classification

Image classification is the process of predicting a specific class, or label, for something that is defined by a set of data points. In this case, we are building a deep learning architecture to classify the given patient's chest X-ray image as a case of pneumonia or not. The dataset primarily was not designed for image classification and this is where we were creative. Usually, the data is organized as image files in folders for each class. The generator will take each

folder as the classes to be trained. Keras generator function flow_from_dataframe() allows us to organize the data as a dataframe where we have the image filename and their corresponding target. Here we only use the train images as it is labeled. The DICOM images are converted into JPEG images as it is easier for the model to process and gives better metrics. Then, pre-processing of the converted images is done after splitting the data into train, validation and test sets. The Keras ImageDataGenerator takes in these images, transforms them with the arguments we give and feeds them into the model, which results in a better model performance, although it takes more time.

Transfer learning allows us to use a pre-existing model, trained on a huge dataset, for our own tasks. Consequently reducing the cost of training new deep learning models and since the datasets have been vetted, we can be assured of the quality. We have built three image classification models with transfer learning:

1.VGG16 Model

2.ResNet50 Model

3.DenseNet121 Model

The three models are fairly consistent and have a similar framework. In all three models, pre-trained imagenet weights are used as it provides a standard measure for model comparison and helps in model coverage with less number of epochs. The input shape is also the same and include_top =False as we are using pre-trained material and we do not want the final dense layer that gives classification prediction. After the architecture, we compile the model with three metrics: accuracy, precision and recall with loss being set to categorical cross entropy and binary cross entropy as it is a classification problem and the optimizer is Adam. We then display the summary. Then, it is fit with the training and validation data along with other parameters. The average accuracy of the training data is shown and plots displaying the accuracy and loss metrics against the number epochs are visualized. Finally, the test metrics are displayed individually and in a summary table.

VGG-16 Model:

We set some layers frozen. To "freeze" a layer means to exclude it from training. Flattening of the last layer is done. It converts a multi-dimensional feature map to a single dimension. The prediction Dense layer has 2 nodes and softmax activation.

ResNet50 Model and Dense121 Model:

Even in the ResNet50,we freeze some layers. The ResNet50 model and DenseNet121 model architectures are almost identical. Dropout threshold is set to 0.5 to prevent overfitting and to regularize the model. There are 2 neurons in the output layer with sigmoid activation.

In this context, accuracy is the most important metric for image classification, not taking anything away from the other metrics. The train accuracy is most important as we are interested in the performance of the model in the real world setting. All three models have accuracies in the range of 70% - 80% with VGG16 having the best test accuracy. The ResNet50 model closely follows and DenseNet121 has the least test accuracy. We can add more trainable layers for performance improvements.

Converting the train images from DICOM to JPEG

```
import pydicom as dicom
PNG = False
jpg_folder_path = "Dataset - New/JPG_test" #Output folder
images_path = os.listdir(training_image_path)
for n, image in enumerate(images_path):
    ds = dicom.dcmread(os.path.join(training_image_path, image))
    pixel_array_numpy = ds.pixel_array
    if PNG == False:
        image = image.replace('.dcm', '.jpg')
    else:
        image = image.replace('.dcm', '.png')
    cv2.imwrite(os.path.join(jpg_folder_path, image), pixel_array_numpy)
    if n % 50 == 0:
        print('{} image converted'.format(n))
```

```
0 image converted
50 image converted
100 image converted
150 image converted
200 image converted
.
.
.
.
26500 image converted
26550 image converted
26600 image converted
26650 image converted
```

```
df = pd.merge(train_labels, class_labels, on="patientId")
#Merging the images with their labels in a dataframe
images = pd.DataFrame({'path': glob(os.path.join(jpg_folder_path, '*.jpg'))})
images['patientId'] = images['path'].map(lambda x:os.path.splitext(os.path.basename(x))[0])
print('Columns in the training images dataframe: {}'.format(list(images.columns)))
merge = df.merge(images, on = 'patientId', how = 'left')
print('After merging the two dataframe, the training_data has {} rows and {}
columns.'.format(df.shape[0], df.shape[1]))
```

Columns in the training images dataframe: ['path', 'patientId']
After merging the two dataframe, the training_data has 37629 rows and 7 columns.

```
#Changing the target variable to string as it a classification problem
new['Target'] = new['Target'].astype(str)

#Splitting the dataset into train,validation and test with the below size
t1=new.iloc[0:31625] #train
t2=new.iloc[31625:32625] #validation
t3=new.iloc[32625:37625] #test
print("t1 : ", t1.shape )
print("t2 : ", t2.shape )
print("t3 : ", t3.shape )
```

t1 : (31625, 2)
t2 : (1000, 2)
t3 : (5000, 2)

Image Pre-processing

```
from keras.preprocessing.image import ImageDataGenerator
train_datagenerator = ImageDataGenerator(rescale=1. / 255,shear_range=0.2,zoom_range=0.2,rotation_range=20,brightness_range=(1.2, 1.5),horizontal_flip=True)
test_datagenerator = ImageDataGenerator(rescale=1. / 255)
val_datagenerator = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagenerator.flow_from_dataframe(dataframe=t1,x_col='path',y_col="Target",target_size=(224, 224),batch_size=16,class_mode='categorical')
val_generator = val_datagenerator.flow_from_dataframe(dataframe=t2,x_col='path',y_col="Target",target_size=(224, 224),batch_size=16,class_mode='categorical')
test_generator = test_datagenerator.flow_from_dataframe(dataframe=t3,x_col='path',y_col="Target",target_size=(224, 224),batch_size=16,class_mode='categorical')
```

Found 31625 validated image filenames belonging to 2 classes.

Found 1000 validated image filenames belonging to 2 classes.

Found 5000 validated image filenames belonging to 2 classes.

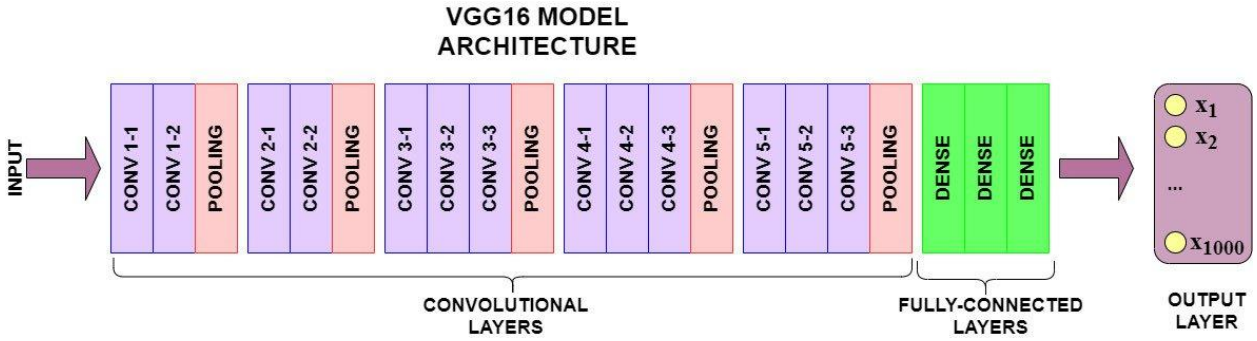
Functions for Re-usability:

```
headAccTable = ["Model","Train Accuracy", "Test Accuracy"]
headDataMetrics = ["Loss","Accuracy", "Precision", "Recall"]

def printTable(data, head):
    # display table
    print(tabulate(data, headers=head, tablefmt="fancy_grid"))

def getScore(modelHistory):
    score = np.mean(modelHistory.history['accuracy'])
    print('The Average Training Accuracy is', score)
    return score
```

```
def plotAccAndLoss(modelHistory):
    #Plots-accuracy and loss in train and validation data
    fig, ax = plt.subplots(1, 2, figsize=(10, 3))
    for i, met in enumerate(['accuracy', 'loss']):
        ax[i].plot(modelHistory.history[met])
        ax[i].plot(modelHistory.history['val_' + met])
        ax[i].set_title('Model {}'.format(met))
        ax[i].set_xlabel('epochs')
        ax[i].set_ylabel(met)
        ax[i].legend(['train', 'val'])
```

VGG-16 Model

A convolutional neural network is also known as a ConvNet, which is a kind of artificial neural network. A convolutional neural network has an input layer, an output layer, and various hidden layers. VGG16 is a type of CNN (Convolutional Neural Network) that is considered to be one of the best computer vision models to date. The creators of this model evaluated the networks and increased the depth using an architecture with very small (3 × 3) convolution filters, which showed a significant improvement on the prior-art configurations. They pushed the depth to 16–19 weight layers making it approx — 138 trainable parameters. VGG16 is an object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with 92.7% accuracy.



Vgg model building

```
vgg = VGG16(weights='imagenet',include_top=False,input_shape=(224,224,3))
for layer in vgg.layers:
    layer.trainable = False #layers as non-trainable
#flattening last layer
x = Flatten()(vgg.output)
#Dense layer
```

```
predictions = Dense(2,activation='softmax')(x)
model1 = Model(inputs=vgg.input, outputs=predictions)
model1.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy','Precision','Recall'])
model1.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
flatten (Flatten)            (None, 25088)             0
dense (Dense)                (None, 2)                 50178
=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
history1 =
model1.fit(train_generator,validation_data=val_generator,steps_per_epoch=25,epochs=10)
```

Epoch 1/10

25/25 [==============================] - 199s 8s/step - loss: 0.9998 - accuracy: 0.6125 - precision: 0.6125 - recall: 0.6125 - val_loss: 0.6864 - val_accuracy: 0.6940 - val_precision: 0.6940 - val_recall: 0.6940

Epoch 2/10

25/25 [==============================] - 226s 9s/step - loss: 0.8257 - accuracy: 0.6675 - precision: 0.6675 - recall: 0.6675 - val_loss: 0.6705 - val_accuracy: 0.6890 - val_precision: 0.6890 - val_recall: 0.6890

Epoch 3/10

25/25 [==============================] - 237s 10s/step - loss: 0.6190 - accuracy: 0.7455 - precision: 0.7455 - recall: 0.7455 - val_loss: 0.4521 - val_accuracy: 0.7880 - val_precision: 0.7880 - val_recall: 0.7880

Epoch 4/10

25/25 [==============================] - 242s 10s/step - loss: 0.6327 - accuracy: 0.7150 - precision: 0.7150 - recall: 0.7150 - val_loss: 0.9789 - val_accuracy: 0.5810 - val_precision: 0.5810 - val_recall: 0.5810

Epoch 5/10

25/25 [==============================] - 253s 10s/step - loss: 0.8115 - accuracy: 0.6825 - precision: 0.6825 - recall: 0.6825 - val_loss: 1.3225 - val_accuracy: 0.5300 - val_precision: 0.5300 - val_recall: 0.5300

Epoch 6/10

25/25 [==============================] - 234s 10s/step - loss: 0.6016 - accuracy: 0.7300 - precision: 0.7300 - recall: 0.7300 - val_loss: 0.6002 - val_accuracy: 0.7310 - val_precision: 0.7310 - val_recall: 0.7310

Epoch 7/10

25/25 [==============================] - 234s 10s/step - loss: 0.4591 - accuracy: 0.8150 - precision: 0.8150 - recall: 0.8150 - val_loss: 0.4389 - val_accuracy: 0.8120 - val_precision: 0.8120 - val_recall: 0.8120

Epoch 8/10

25/25 [==============================] - 241s 10s/step - loss: 0.5959 - accuracy: 0.7650 - precision: 0.7650 - recall: 0.7650 - val_loss: 0.7058 - val_accuracy: 0.7040 - val_precision: 0.7040 - val_recall: 0.7040

Epoch 9/10

25/25 [==============================] - 250s 10s/step - loss: 0.6067 - accuracy: 0.7475 - precision: 0.7475 - recall: 0.7475 - val_loss: 0.9398 - val_accuracy: 0.6270 - val_precision: 0.6270 - val_recall: 0.6270

Epoch 10/10

25/25 [==============================] - 241s 10s/step - loss: 0.7104 - accuracy: 0.7075 - precision: 0.7075 - recall: 0.7075 - val_loss: 0.8774 - val_accuracy: 0.6390 - val_precision: 0.6390 - val_recall: 0.6390

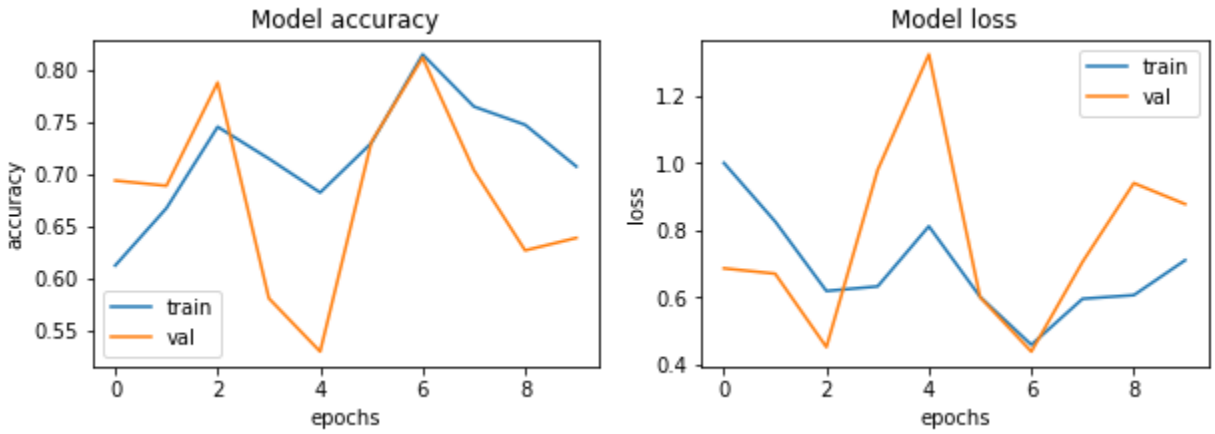
#Training Accuracy of the VGG-16 Model

score1 = getScore(history1)

The Average Training Accuracy is 0.7188047051429749

#Plots-accuracy and loss in train vs validation data

plotAccAndLoss(history1)



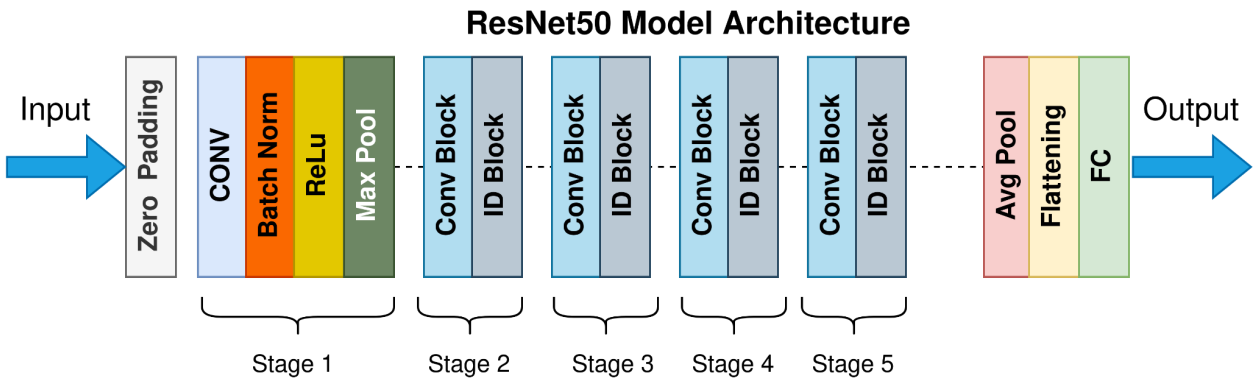
result1 = model1.evaluate_generator(test_generator, 5000)

printTable([result1], headDataMetrics)

Loss	Accuracy	Precision	Recall
0.587347	0.754	0.754	0.754

ResNet50 Model

ResNet stands for Residual Network. This model was immensely successful, as can be ascertained from the fact that its ensemble won the top position at the ILSVRC 2015 classification competition with an error of only 3.57%.ResNet has many variants that run on the same concept but have different numbers of layers. Resnet50 is used to denote the variant that can work with 50 neural network layers.



Resnet50 Model building

```
resnet50 = ResNet50V2(weights = "imagenet", input_shape = (224,224,3), include_top = False)
for layer in resnet50.layers:
    layer.trainable = False
model2 = Sequential()
model2.add(resnet50)
model2.add(Flatten())
model2.add(Dense(units = 128, activation = "relu"))
model2.add(Dropout(0.5))
model2.add(Dense(units = 2, activation = "sigmoid"))
model2.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy','Precision','Recall'])
model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800
 flatten_1 (Flatten)         (None, 100352)            0
 dense_1 (Dense)             (None, 128)               12845184
 dropout (Dropout)           (None, 128)               0
 dense_2 (Dense)             (None, 2)                 258
=================================================================
Total params: 36,410,242
Trainable params: 12,845,442
```

Non-trainable params: 23,564,800

_____

history2 = model2.fit(train_generator,validation_data=val_generator,steps_per_epoch=25,epochs=10)

Epoch 1/10

25/25 [==============================] - 89s 4s/step - loss: 4.5310 - accuracy: 0.7225 - precision: 0.6856 - recall: 0.7250 - val_loss: 1.7911 - val_accuracy: 0.7560 - val_precision: 0.7885 - val_recall: 0.6450

Epoch 2/10

25/25 [==============================] - 84s 3s/step - loss: 1.0623 - accuracy: 0.7125 - precision: 0.7200 - recall: 0.6300 - val_loss: 0.5720 - val_accuracy: 0.7730 - val_precision: 0.7588 - val_recall: 0.7740

Epoch 3/10

25/25 [==============================] - 82s 3s/step - loss: 0.6034 - accuracy: 0.6575 - precision: 0.6035 - recall: 0.8525 - val_loss: 0.5324 - val_accuracy: 0.7610 - val_precision: 0.6350 - val_recall: 0.9150

Epoch 4/10

25/25 [==============================] - 86s 4s/step - loss: 0.6393 - accuracy: 0.6075 - precision: 0.5879 - recall: 0.9025 - val_loss: 0.6319 - val_accuracy: 0.5010 - val_precision: 0.5891 - val_recall: 0.9390

Epoch 5/10

25/25 [==============================] - 85s 3s/step - loss: 0.6649 - accuracy: 0.6600 - precision: 0.5994 - recall: 0.9275 - val_loss: 0.6127 - val_accuracy: 0.7660 - val_precision: 0.6063 - val_recall: 0.8240

Epoch 6/10

25/25 [==============================] - 88s 4s/step - loss: 0.5638 - accuracy: 0.6875 - precision: 0.6271 - recall: 0.9125 - val_loss: 0.6146 - val_accuracy: 0.5750 - val_precision: 0.6240 - val_recall: 0.7500

Epoch 7/10

25/25 [==============================] - 87s 4s/step - loss: 0.6241 - accuracy: 0.6350 - precision: 0.5919 - recall: 0.9100 - val_loss: 0.6726 - val_accuracy: 0.4450 - val_precision: 0.5820 - val_recall: 0.8130

Epoch 8/10

25/25 [==============================] - 87s 4s/step - loss: 0.5461 - accuracy: 0.6925 - precision: 0.6339 - recall: 0.9175 - val_loss: 0.5942 - val_accuracy: 0.7530 - val_precision: 0.6312 - val_recall: 0.7960

Epoch 9/10> =====] - 84s 3s/step - loss: 0.6497 - accuracy: 0.6725 - precision: 0.5890 - recall: 0.9100 - val_loss: 0.5576 - val_accuracy: 0.7630 - val_precision: 0.6850 - val_recall: 0.8090

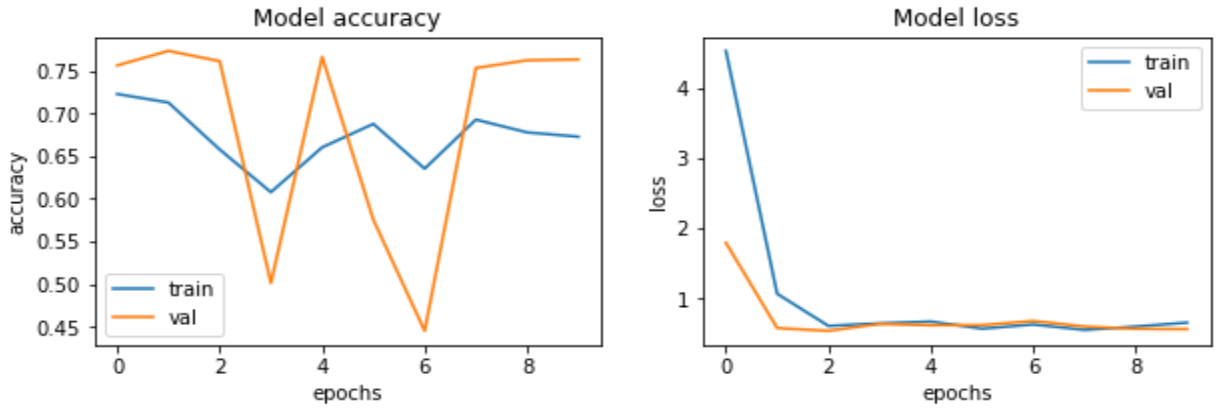
#Training Accuracy of the ResNet50 Model

score2 = getScore(history2)

The Average Training Accuracy is 0.6725000083446503

#Plots-accuracy and loss in train vs validation data

plotAccAndLoss(history2)



#Test Data Metrics-Accuracy,Precision and Recall

result2 = model2.evaluate_generator(test_generator, 5000)

printTable([result2], headDataMetrics)

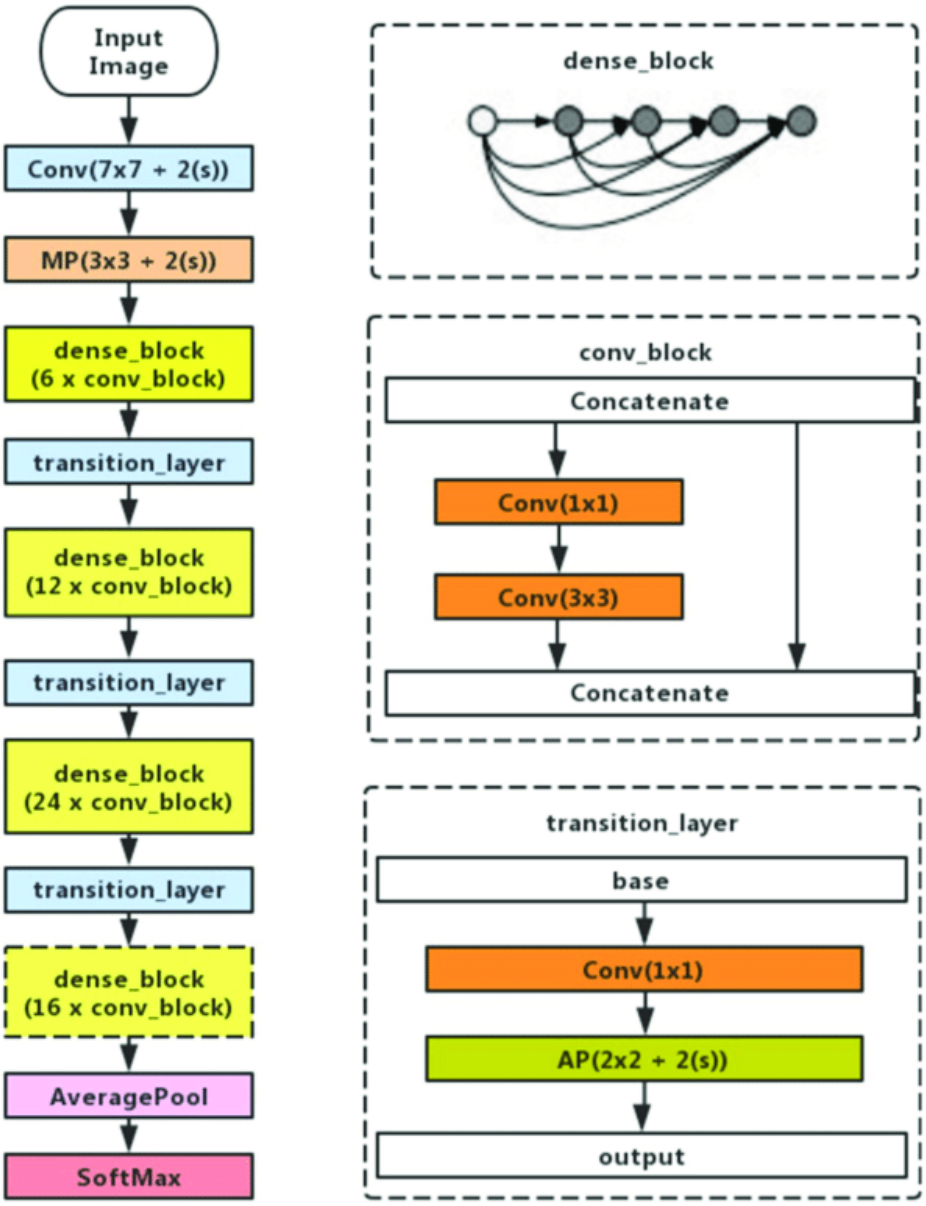
Loss	Accuracy	Precision	Recall
0.50237	0.8024	0.74935	0.8652

DenseNet121 Model

DenseNet (Dense Convolutional Network) is an architecture that focuses on making the deep learning networks go even deeper, but at the same time making them more efficient to train, by using shorter connections between the layers. DenseNet is a convolutional neural network where each layer is connected to all other layers that are deeper in the network, that is, the first layer is connected to the 2nd, 3rd, 4th and so on, the second layer is connected to the 3rd, 4th, 5th and so on. This is done to enable maximum information flow between the layers of the network. To preserve the feed-forward nature, each layer obtains inputs from all the previous layers and passes on its own feature maps to all the layers which will come after it. Unlike Resnets it does not combine features through summation but combines the features by concatenating them. So the 'ith' layer has 'i' inputs and consists of feature maps of all its preceding convolutional blocks. Its own feature maps are passed on to all the next 'I-i' layers.

This introduces '(I(I+1))/2' connections in the network, rather than just 'I' connections as in traditional deep learning architectures. It hence requires fewer parameters than traditional convolutional neural networks, as there is no need to learn unimportant feature maps.

DenseNet consists of two important blocks other than the basic convolutional and pooling layers. They are the Dense Blocks and the Transition layers.



DenseNet121 Model Building:

```
base_model = DenseNet121(include_top=False, weights="imagenet", input_shape=(224,224,3), pooling="avg")

model3 = Sequential()
model3.add(base_model)
model3.add(layers.Flatten())
model3.add(layers.Dense(2048 ,activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))
model3.add(layers.Dense(2, activation ='sigmoid'))

model3.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy','Precision','Recall'])

model3.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Functional)     (None, 1024)              7037504

flatten_2 (Flatten)          (None, 1024)              0

dense_3 (Dense)              (None, 2048)              2099200

batch_normalization (BatchN  (None, 2048)              8192
ormalization)

dropout_1 (Dropout)          (None, 2048)              0

dense_4 (Dense)              (None, 2)                 4098

=================================================================
Total params: 9,148,994
Trainable params: 9,061,250
Non-trainable params: 87,744
```

```
history3 =
model3.fit(train_generator,steps_per_epoch=25,epochs=10,validation_data=val_generator)
```

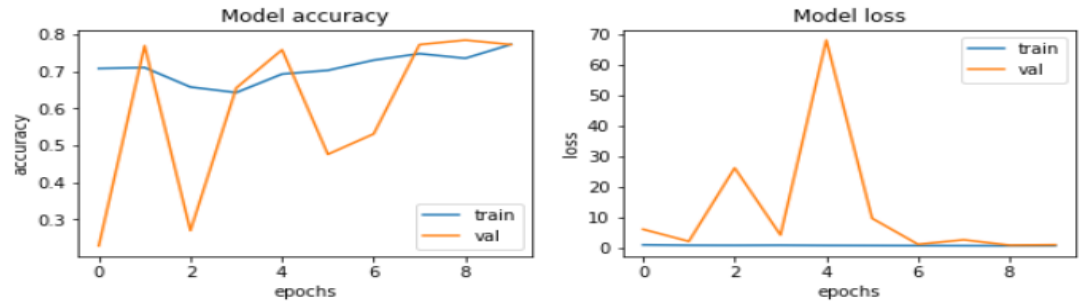
```
Epoch 1/10
25/25 [==============================] - 227s 9s/step - loss: 0.8712 - accuracy: 0.7075 -
precision: 0.6990 - recall: 0.7025 - val_loss: 5.9835 - val_accuracy: 0.2290 - val_precision: 0.2290 -
val_recall: 0.2290
Epoch 2/10
25/25 [==============================] - 222s 9s/step - loss: 0.7363 - accuracy: 0.7100 -
precision: 0.6897 - recall: 0.6725 - val_loss: 2.0226 - val_accuracy: 0.7690 - val_precision: 0.7353 -
val_recall: 0.0250
Epoch 3/10
25/25 [==============================] - 225s 9s/step - loss: 0.7071 - accuracy: 0.6575 -
precision: 0.6430 - recall: 0.6800 - val_loss: 26.1229 - val_accuracy: 0.2700 - val_precision: 0.2510 -
val_recall: 0.2440
Epoch 4/10
25/25 [==============================] - 243s 10s/step - loss: 0.7523 - accuracy: 0.6425 -
precision: 0.6574 - recall: 0.6525 - val_loss: 4.0967 - val_accuracy: 0.6550 - val_precision: 0.6574 -
val_recall: 0.6620
Epoch 5/10
25/25 [==============================] - 233s 9s/step - loss: 0.6738 - accuracy: 0.6925 -
precision: 0.6863 - recall: 0.7000 - val_loss: 68.0669 - val_accuracy: 0.7580 - val_precision: 0.7608 -
val_recall: 0.7410
Epoch 6/10
25/25 [==============================] - 227s 9s/step - loss: 0.6446 - accuracy: 0.7025 -
precision: 0.7080 - recall: 0.6850 - val_loss: 9.5813 - val_accuracy: 0.4760 - val_precision: 0.4779 -
val_recall: 0.4650
Epoch 7/10
25/25 [==============================] - 236s 9s/step - loss: 0.6081 - accuracy: 0.7300 -
precision: 0.7146 - recall: 0.7450 - val_loss: 1.0643 - val_accuracy: 0.5310 - val_precision: 0.5313 -
val_recall: 0.5350
Epoch 8/10
25/25 [==============================] - 227s 9s/step - loss: 0.5914 - accuracy: 0.7475 -
precision: 0.7410 - recall: 0.7225 - val_loss: 2.5277 - val_accuracy: 0.7720 - val_precision: 0.7718 -
val_recall: 0.7710
Epoch 9/10
25/25 [==============================] - 228s 9s/step - loss: 0.5502 - accuracy: 0.7350 -
precision: 0.7291 - recall: 0.7400 - val_loss: 0.7689 - val_accuracy: 0.7840 - val_precision: 0.7850 -
val_recall: 0.7850
Epoch 10/10
```

25/25 [==============================] - 211s 9s/step - loss: 0.5787 - accuracy: 0.7725 - precision: 0.7625 - recall: 0.7625 - val_loss: 0.8964 - val_accuracy: 0.7720 - val_precision: 0.7720 - val_recall: 0.7720

#Training Accuracy of the DenseNet121 Model
score3 = getScore(history3)

The Average Training Accuracy is 0.7097499966621399

#Plots-accuracy and loss in train vs validation data
plotAccAndLoss(history3)



#Test Data Metrics-Accuracy,Precision and Recall
result3 = model3.evaluate_generator(test_generator, 5000)
printTable([result3], headDataMetrics)

Loss	Accuracy	Precision	Recall
1.68854	0.573	0.573	0.573

# assign data
accData= [
    ["Model-1 VGG-16   ", score1, result1[1]],
    ["Model-2 ResNet50   ", score2, result2[1]],
    ["Model-3 DenseNet121   ", score3, result3[1]]
]

# display table
printTable(accData, headAccTable)

Model	Train Accuracy	Test Accuracy

Model-1 VGG-16	0.718805	0.754
Model-2 ResNet50	0.6725	0.8024
Model-3 DenseNet121	0.70975	0.573

Object Detection:

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. Well-researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance.

For object detection, we have implemented a specific model whose steps are very clearly explained in the comments after each piece of code. First we load the needed packages and then import the dataset. Then, we split the data into train and validation samples. Next we define some functions and classes to be implemented in the model. This is tailored for the dataset on hand. Intersection Over Union metric and loss function are defined. The IOU score is the most important metric in object detection. It is the area of overlap divided by the area of the union. Generally, an IOU score of 0.5 and more is considered good for an object detection model, a threshold which is achieved by our model in the validation.After the model is built, we create a network and compile it with Adam optimizer, loss as iou_bce_loss as defined above and two metrics: accuracy and IOU score. Cosine function is used as the learning rate annealing function and ‘generator’ is used to create train and validation generators and later test generators.Finally, the model is fit with the parameters. Training vs validation metrics plots are visualized. At the end, the test data is used and the submission CSV file with predictions is given as an output. For further improvement, we can try different models like YOLO, Mask R-CNN, etc…

Object Detection Model Building:

The table contains [filename : pneumonia location] pairs per row.

A filename that has no pneumonia does not have a pneumonia location.

Likewise,a filename that contains multiple pneumonia contains multiple rows with the same filename but different pneumonia locations.
The table is loaded and transformed it into a dictionary.
The dictionary uses the filename as key and a list of pneumonia locations in that filename as value.

```
pneumonia_locations = {}
with open(os.path.join('stage_2_train_labels.csv'), mode='r') as infile:
    # open reader
    reader = csv.reader(infile)
    # skip header
    next(reader, None)
    # loop through rows
    for rows in reader:
        # retrieve information
        filename = rows[0]
        location = rows[1:5]
        pneumonia = rows[5]
        if pneumonia == '1':
            # convert string to float to int
            location = [int(float(i)) for i in location]
            # save pneumonia location in dictionary
            if filename in pneumonia_locations:
                pneumonia_locations[filename].append(location)
            else:
                pneumonia_locations[filename] = [location]

#load and shuffle filenames
folder = 'train_images'
filenames = os.listdir(folder)
random.shuffle(filenames)
#split into train and validation filenames
```

```
n_valid_samples = 2500
train_filenames = filenames[n_valid_samples:]
valid_filenames = filenames[:n_valid_samples]
print('train samples =', len(train_filenames))
print('validation samples =', len(valid_filenames))
n_train_samples = len(filenames) - n_valid_samples
```

```
train samples = 24218
validation samples = 2500
```

```
class generator(keras.utils.Sequence):

    def __init__(self, folder, filenames, pneumonia_locations=None, batch_size=32,
image_size=224, shuffle=True, augment=False, predict=False):
        self.folder = folder
        self.filenames = filenames
        self.pneumonia_locations = pneumonia_locations
        self.batch_size = batch_size
        self.image_size = image_size
        self.shuffle = shuffle
        self.augment = augment
        self.predict = predict
        self.on_epoch_end()

    def __load__(self, filename):
        # load dicom file as numpy array
        img = dicom.dcmread(os.path.join(self.folder, filename)).pixel_array
        # create empty mask
        msk = np.zeros(img.shape)
        # get filename without extension
        filename = filename.split('.')[0]
        # if image contains pneumonia
        if filename in pneumonia_locations:
```

```
            # loop through pneumonia
            for location in pneumonia_locations[filename]:
                # add 1's at the location of the pneumonia
                x, y, w, h = location
                msk[y:y+h, x:x+w] = 1
        # if augment then horizontal flip half the time
        if self.augment and random.random() > 0.5:
            img = np.fliplr(img)
            msk = np.fliplr(msk)
        # resize both image and mask
        img = resize(img, (self.image_size, self.image_size), mode='reflect')
        msk = resize(msk, (self.image_size, self.image_size), mode='reflect') > 0.5
        # add trailing channel dimension
        img = np.expand_dims(img, -1)
        msk = np.expand_dims(msk, -1)
        return img, msk

    def __loadpredict__(self, filename):
        # load dicom file as numpy array
        img = dicom.dcmread(os.path.join(self.folder, filename)).pixel_array
        # resize image
        img = resize(img, (self.image_size, self.image_size), mode='reflect')
        # add trailing channel dimension
        img = np.expand_dims(img, -1)
        return img

    def __getitem__(self, index):
        # select batch
        filenames = self.filenames[index*self.batch_size:(index+1)*self.batch_size]
        # predict mode: return images and filenames
        if self.predict:
            # load files
```

```
            imgs = [self.__loadpredict__(filename) for filename in filenames]
            # create numpy batch
            imgs = np.array(imgs)
            return imgs, filenames
        # train mode: return images and masks
        else:
            # load files
            items = [self.__load__(filename) for filename in filenames]
            # unzip images and masks
            imgs, msks = zip(*items)
            # create numpy batch
            imgs = np.array(imgs)
            msks = np.array(msks)
            return imgs, msks

    def on_epoch_end(self):
        if self.shuffle:
            random.shuffle(self.filenames)

    def __len__(self):
        if self.predict:
            # return everything
            return int(np.ceil(len(self.filenames) / self.batch_size))
        else:
            # return full batches only
            return int(len(self.filenames) / self.batch_size)

#Intersection Over Union or jaccard loss function
def iou_loss(y_true, y_pred):
    #print(y_true)
    y_true=tf.cast(y_true, tf.float32)
    y_pred=tf.cast(y_pred, tf.float32)
```

```
    y_true = tf.reshape(y_true, [-1])
    y_pred = tf.reshape(y_pred, [-1])

    intersection = tf.reduce_sum(y_true * y_pred)
    score = (intersection + 1.) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) - intersection + 1.)
    return 1 - score

# combine bce loss and iou loss
def iou_bce_loss(y_true, y_pred):
    return 0.5 * keras.losses.binary_crossentropy(y_true, y_pred) + 0.5 * iou_loss(y_true, y_pred)

# mean iou as a metric
def mean_iou(y_true, y_pred):
    y_pred = tf.round(y_pred)
    intersect = tf.reduce_sum(y_true * y_pred, axis=[1, 2, 3])
    union = tf.reduce_sum(y_true, axis=[1, 2, 3]) + tf.reduce_sum(y_pred, axis=[1, 2, 3])
    smooth = tf.ones(tf.shape(intersect))
    return tf.reduce_mean((intersect + smooth) / (union - intersect + smooth))

def create_downsample(channels, inputs):
    x = keras.layers.BatchNormalization(momentum=0.9)(inputs)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(channels, 1, padding='same', use_bias=False)(x)
    x = keras.layers.MaxPool2D(2)(x)
    return x

def create_resblock(channels, inputs):
    x = keras.layers.BatchNormalization(momentum=0.9)(inputs)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=False)(x)
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
```

```
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=False)(x)
    return keras.layers.add([x, inputs])

def create_network(input_size, channels, n_blocks=2, depth=4):
    # input
    inputs = keras.Input(shape=(input_size, input_size, 1))
    x = keras.layers.Conv2D(channels, 3, padding='same', use_bias=False)(inputs)
    # residual blocks
    for d in range(depth):
        channels = channels * 2
        x = create_downsample(channels, x)
        for b in range(n_blocks):
            x = create_resblock(channels, x)
    # output
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(128, 1, activation=None)(x)
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2DTranspose(64, (8,8), (4,4), padding="same", activation=None)(x)
    x = keras.layers.BatchNormalization(momentum=0.9)(x)
    x = keras.layers.LeakyReLU(0)(x)
    x = keras.layers.Conv2D(1, 1, activation='sigmoid')(x)
    outputs = keras.layers.UpSampling2D(2**(depth-2))(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
    return model


# create network and compiler
model = create_network(input_size=224, channels=32, n_blocks=2, depth=4)
model.compile(optimizer='adam', loss=iou_bce_loss, metrics=['accuracy', mean_iou])
```

```
# cosine learning rate annealing
def cosine_annealing(x):
    lr = 0.001
    epochs = 7
    return lr*(np.cos(np.pi*x/epochs)+1.)/2
learning_rate = tf.keras.callbacks.LearningRateScheduler(cosine_annealing)

# create train and validation generators
folder = 'train_images'
train_gen = generator(folder, train_filenames, pneumonia_locations, batch_size=32,
image_size=224, shuffle=True, augment=False, predict=False)
valid_gen = generator(folder, valid_filenames, pneumonia_locations, batch_size=32,
image_size=224, shuffle=False, predict=False)

MULTI_PROCESSING = True
history = model.fit_generator(train_gen, validation_data=valid_gen, callbacks=[learning_rate],
use_multiprocessing=True, epochs=5,steps_per_epoch=100, shuffle=True, verbose=1)
```

Epoch 1/5

100/100 [==============================] - 5556s 55s/step - loss: 0.5959 - accuracy: 0.9508 - mean_iou: 0.6321 - val_loss: 0.5386 - val_accuracy: 0.9634 - val_mean_iou: 0.5955 - lr: 0.0010

Epoch 2/5

100/100 [==============================] - 5440s 54s/step - loss: 0.5207 - accuracy: 0.9543 - mean_iou: 0.5617 - val_loss: 0.5302 - val_accuracy: 0.9254 - val_mean_iou: 0.5042 - lr: 9.5048e-04

Epoch 3/5

100/100 [==============================] - 5546s 55s/step - loss: 0.4926 - accuracy: 0.9609 - mean_iou: 0.5810 - val_loss: 0.4832 - val_accuracy: 0.9635 - val_mean_iou: 0.6549 - lr: 8.1174e-04

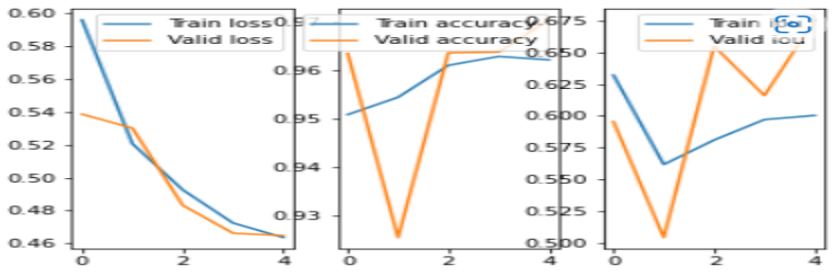
Epoch 4/5

100/100 [==============================] - 5637s 56s/step - loss: 0.4724 - accuracy: 0.9628 - mean_iou: 0.5971 - val_loss: 0.4662 - val_accuracy: 0.9637 - val_mean_iou: 0.6159 - lr: 6.1126e-04

Epoch 5/5

100/100 [==============================] - 5723s 57s/step - loss: 0.4636 - accuracy: 0.9621 - mean_iou: 0.6003 - val_loss: 0.4648 - val_accuracy: 0.9703 - val_mean_iou: 0.6755 - lr: 3.8874e-04

```
#Plots:Training vs validation-Loss,Accuracy and Intersection Over Union Score
plt.subplot(131)
plt.plot(history.epoch, history.history["loss"], label="Train loss")
plt.plot(history.epoch, history.history["val_loss"], label="Valid loss")
plt.legend()

plt.subplot(132)
plt.plot(history.epoch, history.history["accuracy"], label="Train accuracy")
plt.plot(history.epoch, history.history["val_accuracy"], label="Valid accuracy")
plt.legend()

plt.subplot(133)
plt.plot(history.epoch, history.history["mean_iou"], label="Train iou")
plt.plot(history.epoch, history.history["val_mean_iou"], label="Valid iou")
plt.legend()
plt.show()
```




```
# load and shuffle filenames
folder = 'test_images'
test_filenames = os.listdir(folder)
print('test samples =', len(test_filenames))
```

```
# create test generator with predict flag set to True
test_gen = generator(folder, test_filenames, None, batch_size=32, image_size=224,
shuffle=False, predict=True)


# create submission dictionary
submission_dict = {}
# loop through testset
for imgs, filenames in tqdm(test_gen):
    # predict batch of images
    preds = model.predict(imgs)
    # loop through batch
    for pred, filename in zip(preds, filenames):
        # resize predicted mask
        pred = resize(pred, (1024, 1024), mode='reflect')
        # threshold predicted mask
        comp = pred[:, :, 0] > 0.5
        # apply connected components
        comp = measure.label(comp)
        # apply bounding boxes
        predictionString = ''
        for region in measure.regionprops(comp):
            # retrieve x, y, height and width
            y, x, y2, x2 = region.bbox
            height = y2 - y
            width = x2 - x
            # proxy for confidence score
            conf = np.mean(pred[y:y+height, x:x+width])
            # add to predictionString
            predictionString += str(conf) + ' ' + str(x) + ' ' + str(y) + ' ' + str(width) + ' ' + str(height) + ' '
        # add filename and predictionString to dictionary
        filename = filename.split('.')[0]
        submission_dict[filename] = predictionString
```

```
    # stop if we've got them all
    if len(submission_dict) >= len(test_filenames):
        break

print("Prediction complete")

# save dictionary as csv file-submission
sub = pd.DataFrame.from_dict(submission_dict,orient='index')
sub.index.names = ['patientId']
sub.columns = ['PredictionString']
sub.to_csv('submission.csv')
```

```
test samples = 3000
 99%|▉| 93/94 [27:11<00:17, 17.54s/it]
Prediction complete
```