AIML Final REPORT
Group-2 CV 1

CAPSTONE PROJECT

TABLE OF CONTENTS

ABOUT THIS DOCUMENT

Title: Final Report

Date: 24/12/2023

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.

```
classInfo.head()
```

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

```
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

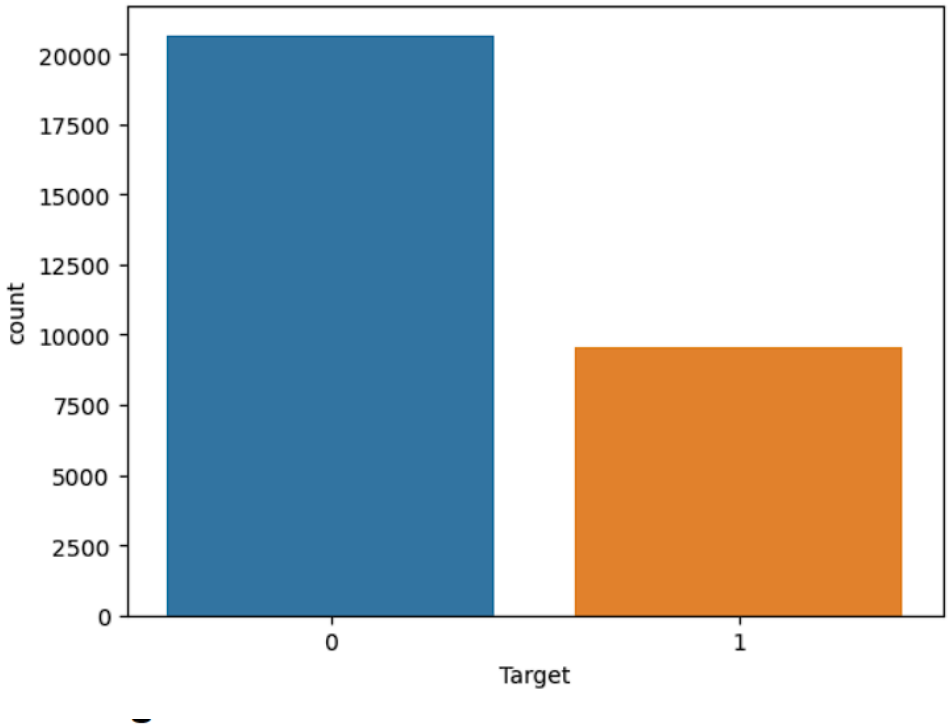
```
trainlabels.info()
```

```
<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
```

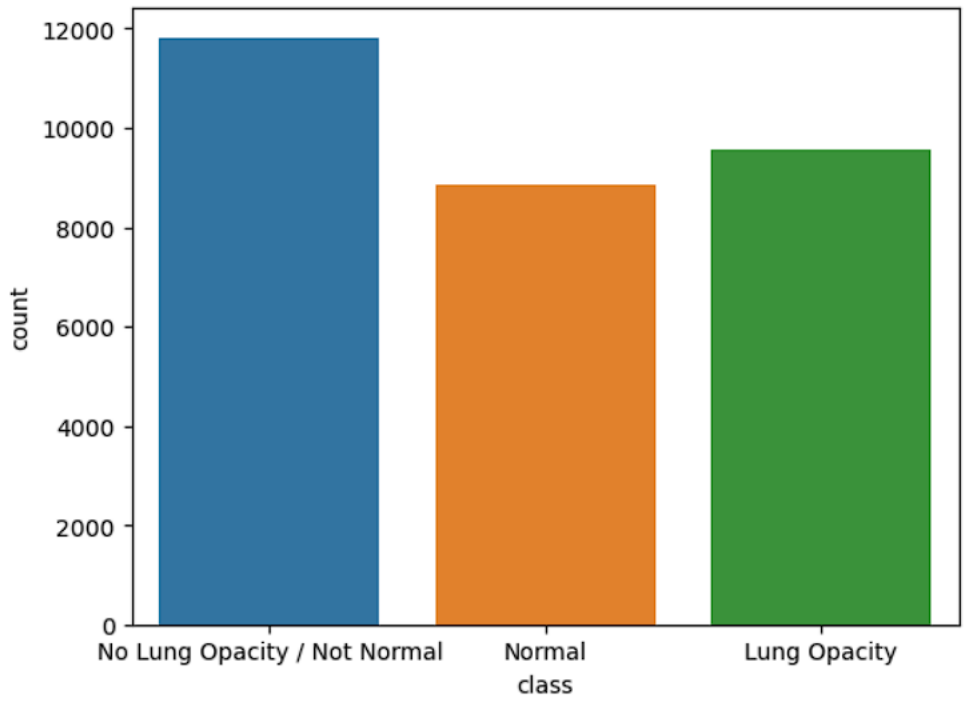
We observed there are X, Y values are missing for few records. This can be due to the fact that for a normal patient these values could be not applicable.

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

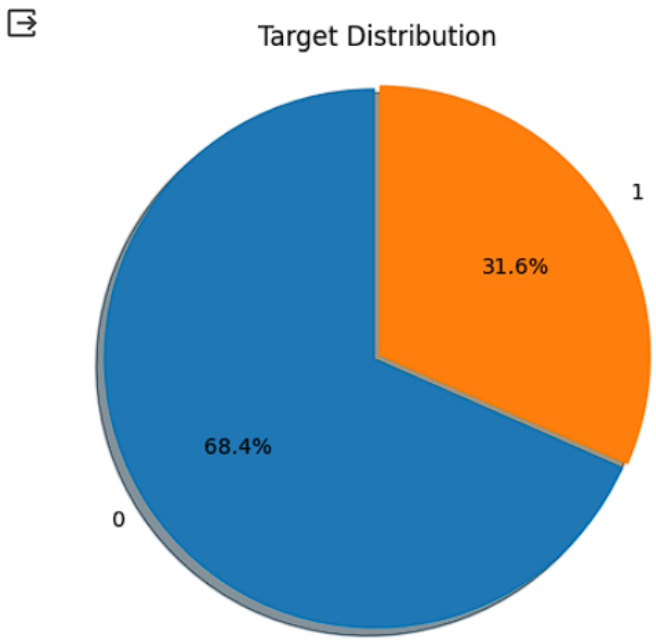
Trainlabels contains 30227 records same as meta data.

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.

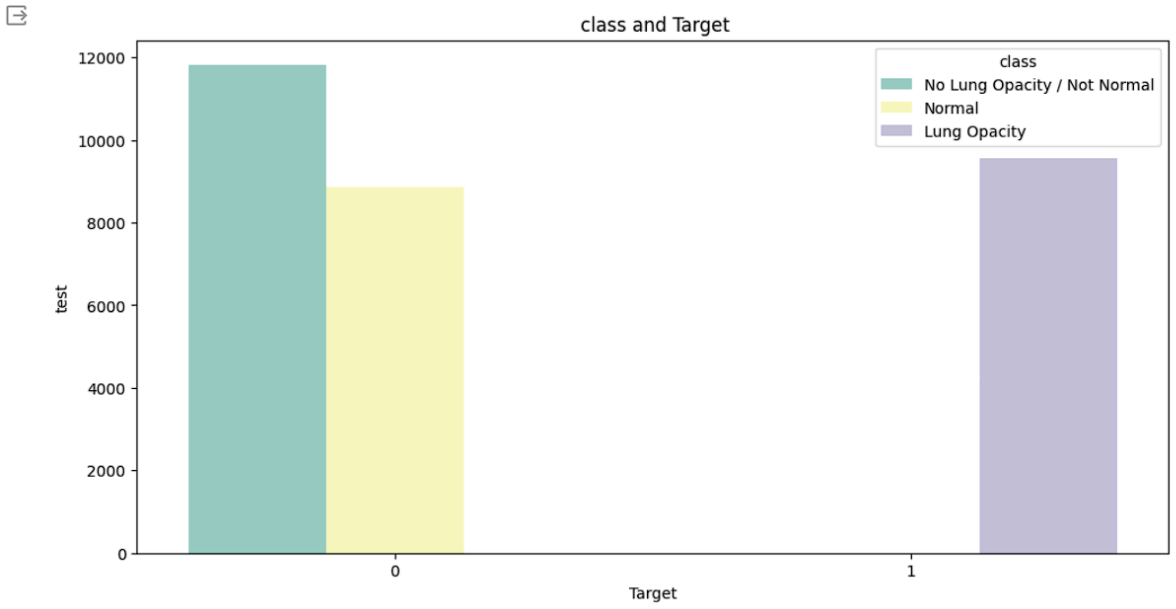
We had done inner merge between two DataFrames, trainlabels and classInfo, based on the common column 'patientId'.

```
# 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

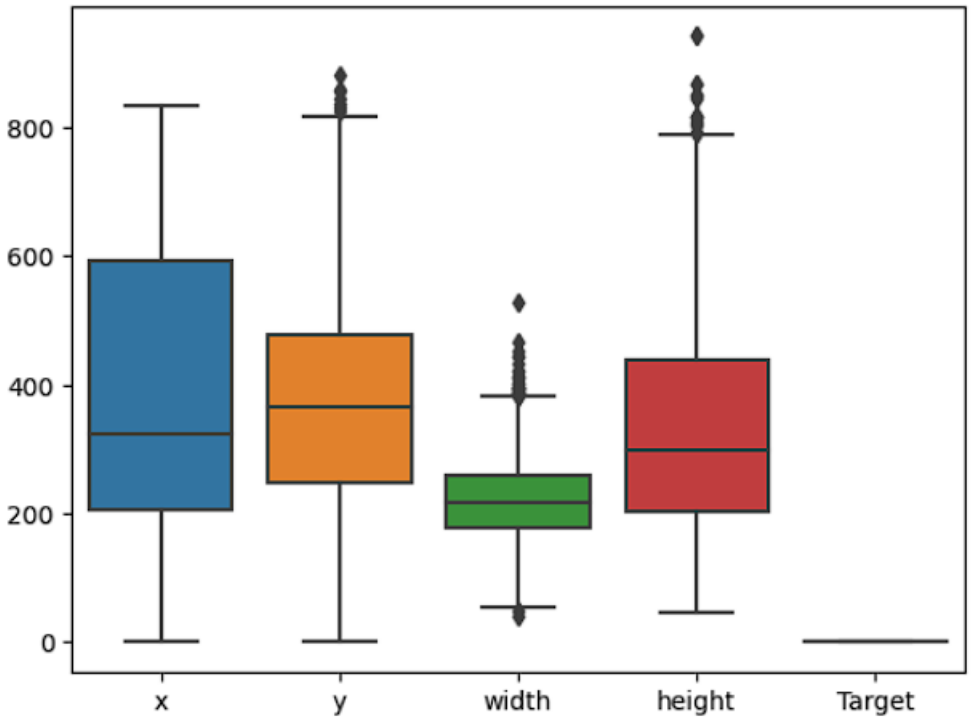
We observed class with Normal and No Lung Opacity / Not Normal has been classified into single target value that is 'O'. So, we can say that the prediction which we have to do is like patient has Lung Opacity or not. Because Normal and not normal patients are combines in same Target.



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

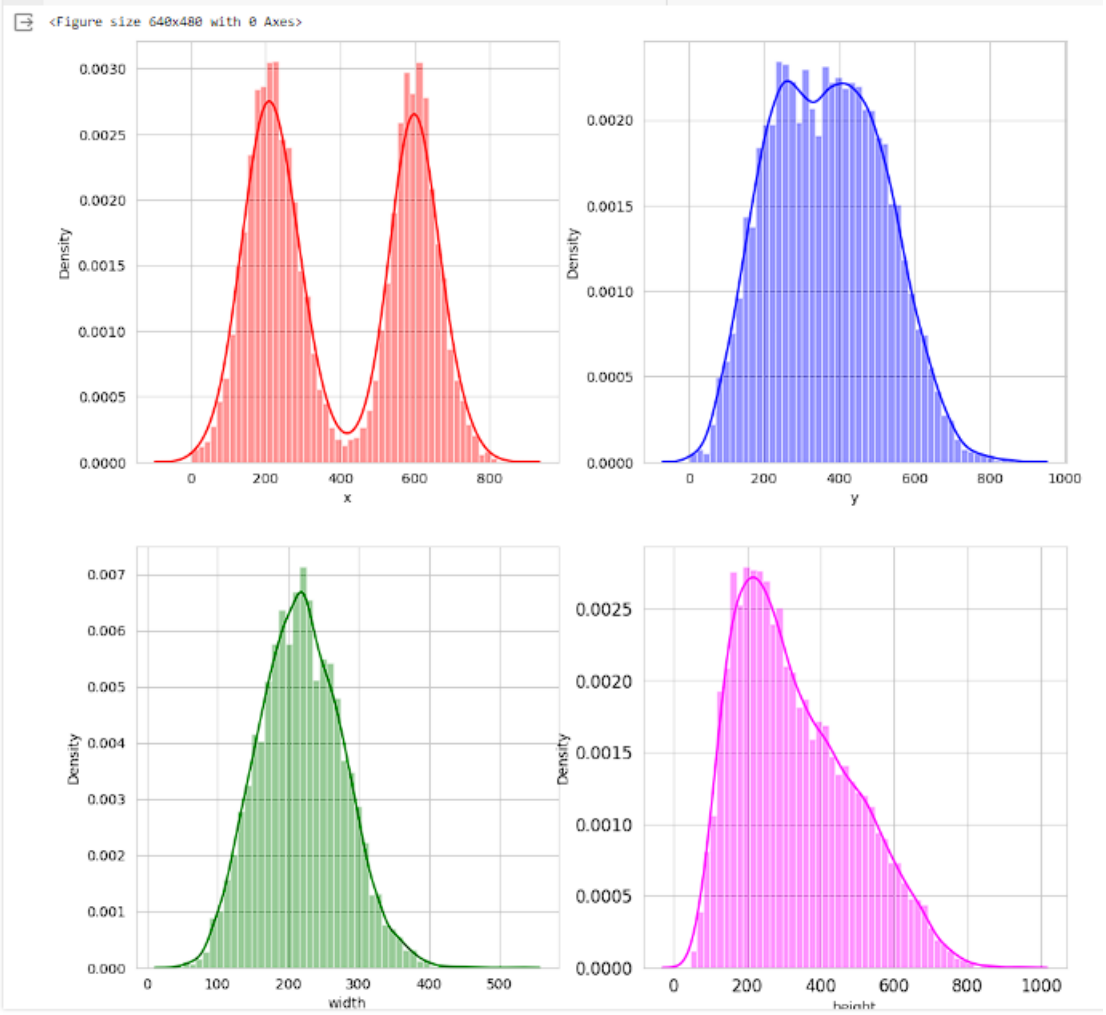
```
[54] sns.boxplot(data=traindf)
```

```
<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

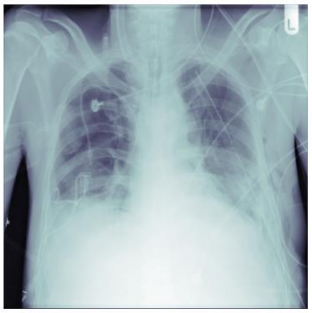
```
def show_dicom_images(data):
    img_data = list(data.T.to_dict().values())
    f, ax = plt.subplots(3,3, figsize=(16,18))
    for i,data_row in enumerate(img_data):
        dcm_path = trainImagesPath/data_row['patientId']
        dcm_path = dcm_path.with_suffix(".dcm")
        data_row_img_data = 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))
```

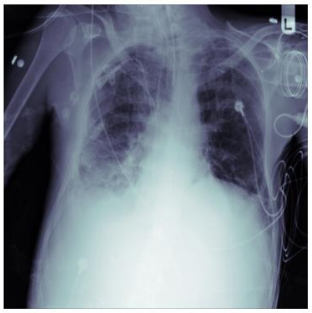
To visualize the images with overlay boxes superimposed, our initial step involves parsing the entire dataset where the target is equal to 1.

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

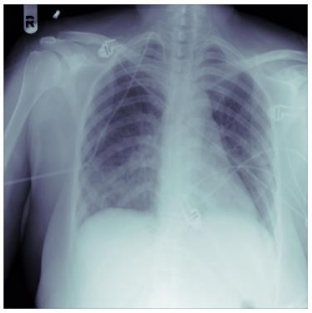
ID: b3a287c5-9fc6-42fb-93c5-310515024948
Modality: CR Age: 52 Sex: M Target: 1
Class: Lung Opacity
Window: 182.0:526.0:210.0:171.0



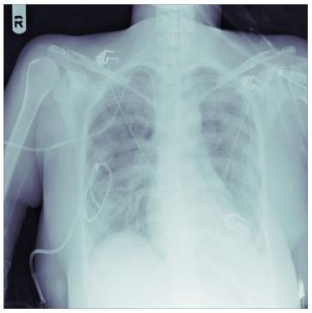
ID: aafda091-8953-40c4-802e-853a206e50d2
Modality: CR Age: 33 Sex: M Target: 1
Class: Lung Opacity
Window: 278.0:431.0:193.0:205.0



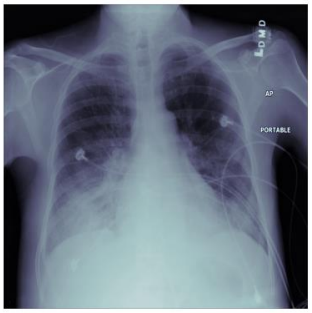
ID: e21bd38d-504c-4256-9168-2f1ccc10ad94
Modality: CR Age: 43 Sex: F Target: 1
Class: Lung Opacity
Window: 279.0:547.0:130.0:146.0



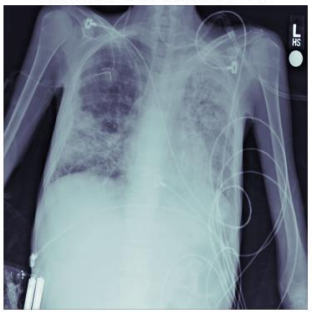
ID: 074f91b5-e915-4b6f-bdf0-af1ee55ebd9b
Modality: CR Age: 40 Sex: F Target: 1
Class: Lung Opacity
Window: 224.0:188.0:329.0:675.0



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

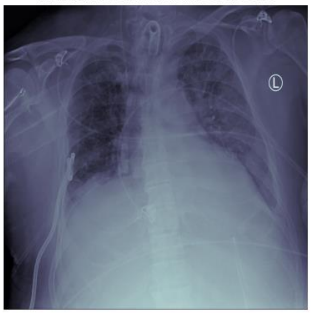


ID: 8fe8e95c-46a1-4ee2-a2d6-2174c6ec50cf
Modality: CR Age: 22 Sex: F Target: 1
Class: Lung Opacity
Window: 590.0:191.0:177.0:306.0



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



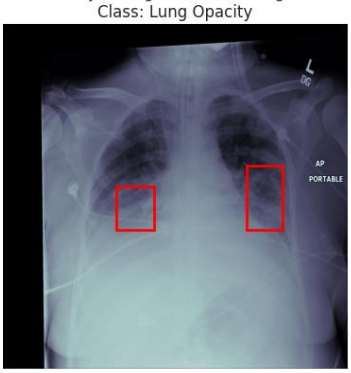
STEP 5: DISPLAY IMAGES WITH BOUNDING BOX.

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

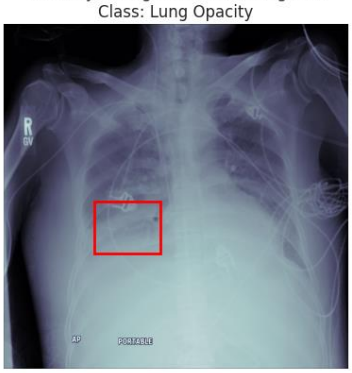
```
def show_dicom_images_with_boxes(data):
    img_data = list(data.T.to_dict().values())
    f, ax = plt.subplots(3,3, figsize=(16,18))
    for i,data_row in enumerate(img_data):
        dcm_path = trainImagesPath/data_row['patientId']
        dcm_path = dcm_path.with_suffix(".dcm")
        data_row_img_data = 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))
```

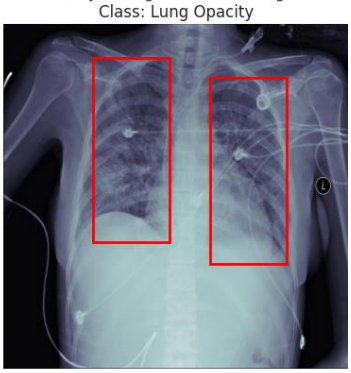
ID: 0ecd37a6-c1bb-40cf-8e48-1020140ae173
Modality: CR Age: 43 Sex: F Target: 1
Class: Lung Opacity



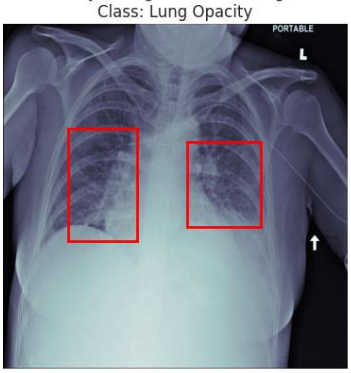
ID: d91b519f-9216-4ff9-9e2a-3356bfe4e391
Modality: CR Age: 59 Sex: M Target: 1
Class: Lung Opacity



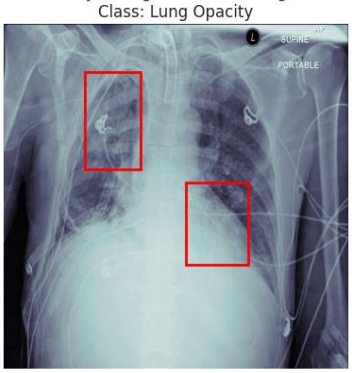
ID: 93351a14-30a7-494c-a02b-7f2d72c724c8
Modality: CR Age: 21 Sex: F Target: 1
Class: Lung Opacity



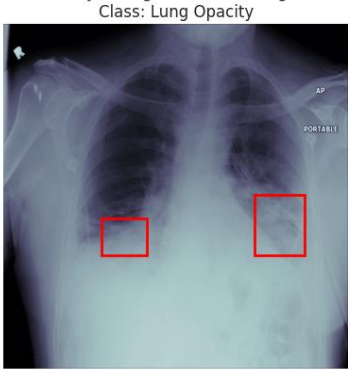
ID: beaee8dc-2b30-41c5-b646-b6f602b965a7
Modality: CR Age: 41 Sex: F Target: 1
Class: Lung Opacity



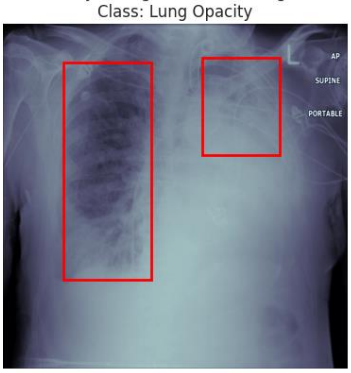
ID: 3bc44999-08aa-4429-ab87-a2958f22b2f8
Modality: CR Age: 51 Sex: M Target: 1
Class: Lung Opacity



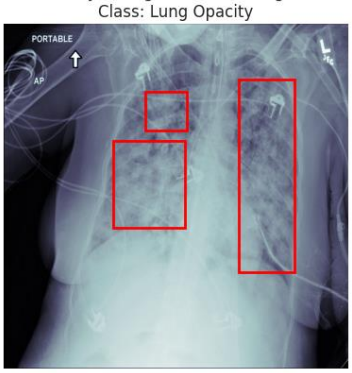
ID: dbc1e726-39b3-4df1-988e-ed7119a4eaa0
Modality: CR Age: 49 Sex: M Target: 1
Class: Lung Opacity



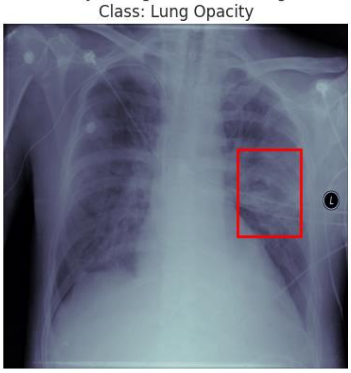
ID: bbd92872-acee-4b27-98b7-07faa4e90919
Modality: CR Age: 67 Sex: M Target: 1
Class: Lung Opacity



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))
```

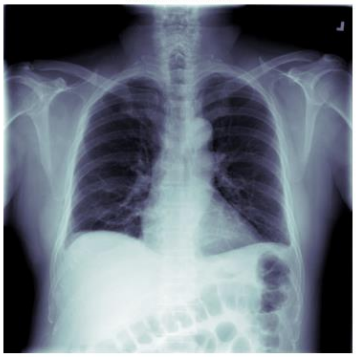
ID: 88a4cd49-f63d-4894-982a-48e7b60bd353
Modality: CR Age: 59 Sex: F Target: 0
Class: Normal

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

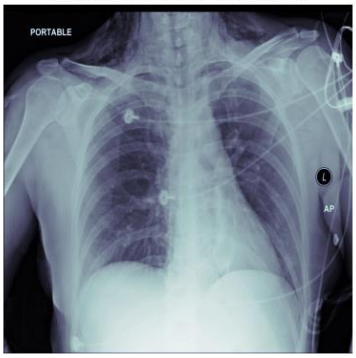
ID: ded4e621-a3ea-4210-8349-06d991c8c056
Modality: CR Age: 57 Sex: M Target: 0
Class: Normal



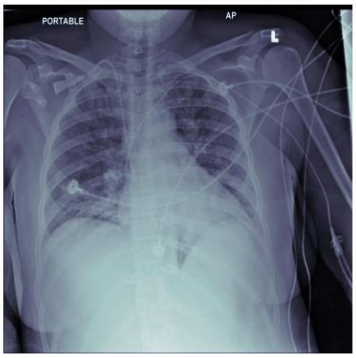
ID: 79deaced-e8d3-4db6-998c-f05798818416
Modality: CR Age: 56 Sex: F Target: 0
Class: No Lung Opacity / Not Normal

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

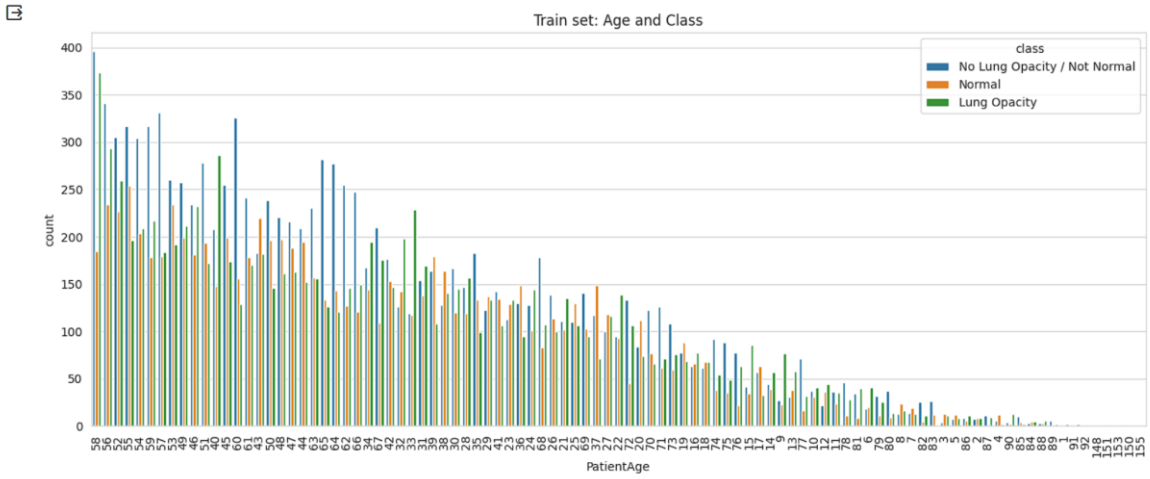
We can see the head of testdf data below.

```
testdf.head()
```

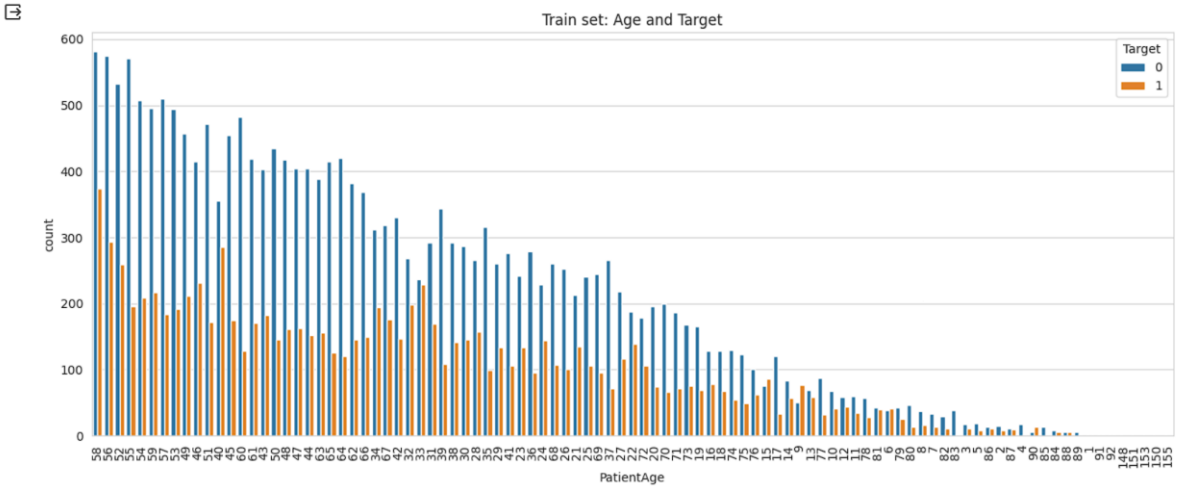
	patientId	Modality	PatientAge	PatientSex	BodyPartExamined	ViewPosition	ConversionType	Rows	Columns	PixelSpacing
0	0000a175-0e68-4ca4-b1af-167204a7e0bc	CR	46	F	CHEST	PA	WSD	1024	1024	0.194
1	0005d3cc-3c3f-40b9-93c3-46231c3eb813	CR	22	F	CHEST	PA	WSD	1024	1024	0.143
2	000686d7-f4fc-448d-97a0-44fa9c5d3aa6	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()
```



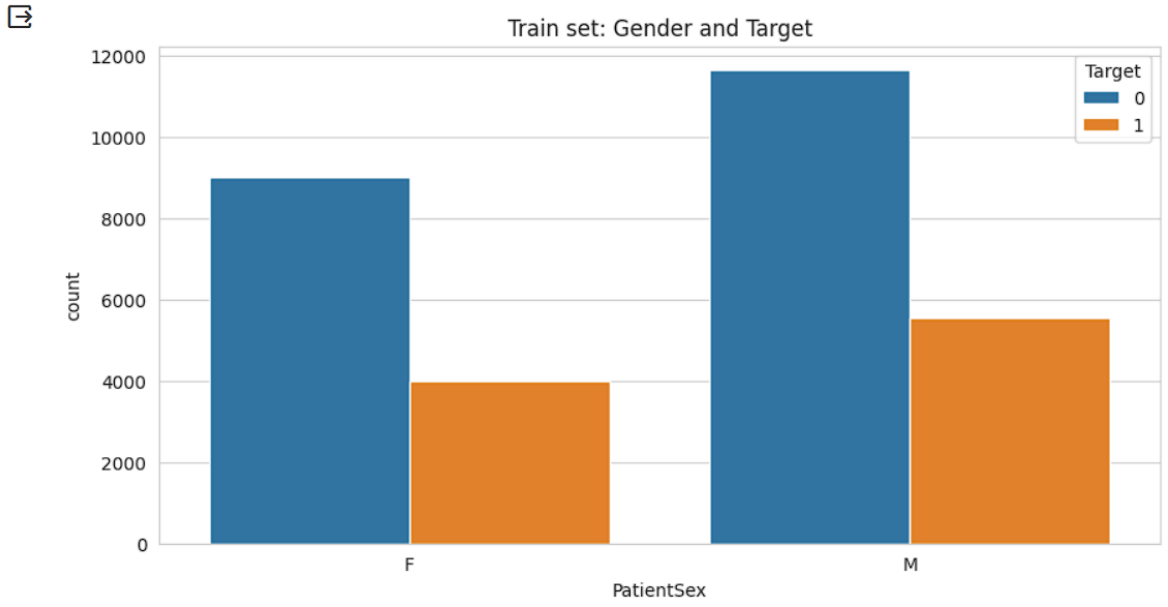

```
fig, (ax) = plt.subplots(nrows=1,figsize=(16,6))
sns.countplot(ax=ax, x = 'PatientAge',hue='Target',data=traindf, order = traindf['PatientAge'].value_counts().index)
plt.title("Train set: Age and Target")
plt.xticks(rotation=90)
plt.show()
```



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



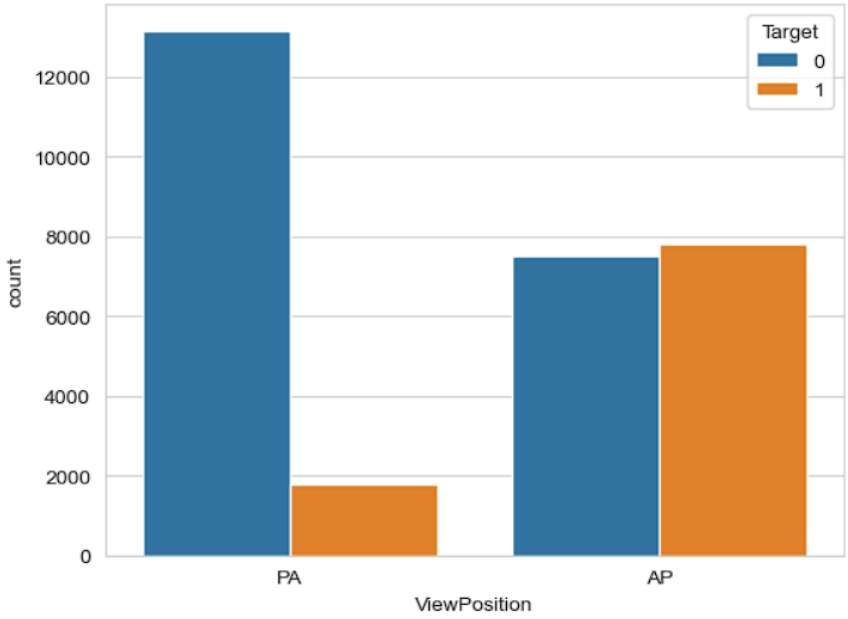
We can see view position below.

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

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

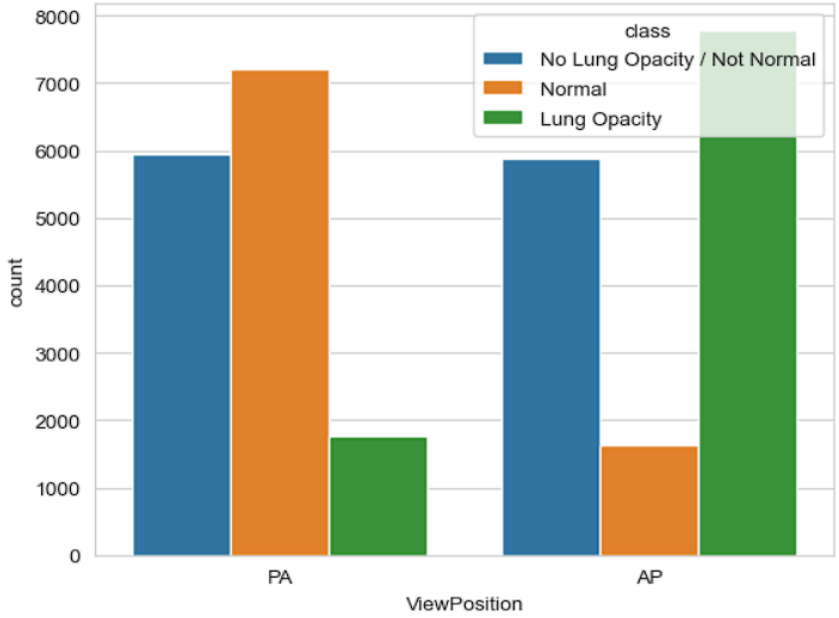
```
sns.countplot(x='ViewPosition',hue='Target',data=traindf)
```

```
<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)
```

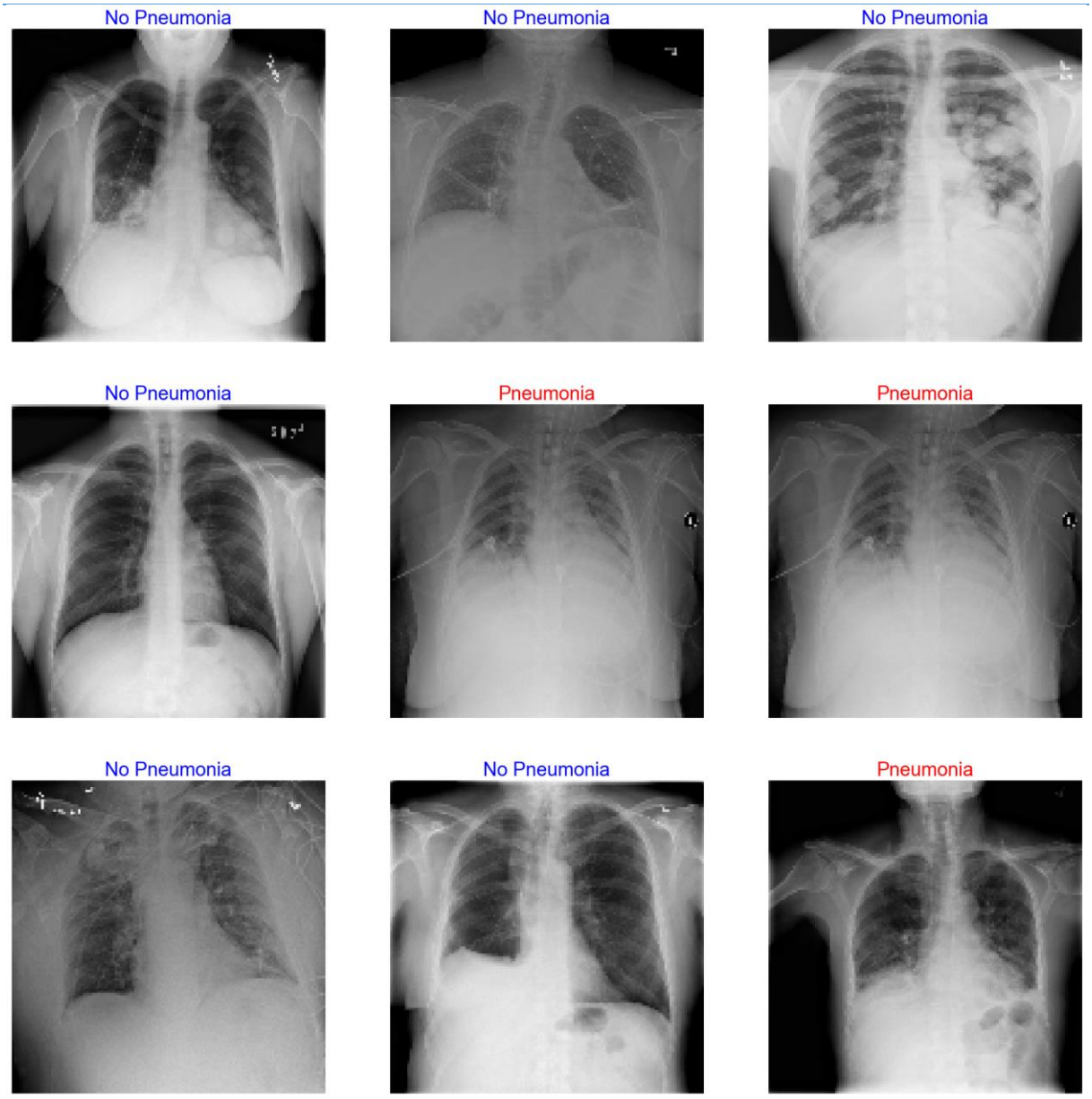
Reading the test data.

```
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"]))
```

```
1512/1512 [==============================] - 76s 49ms/step - loss: 0.55
25 - accuracy: 0.7063 - val_loss: 0.5068 - val_accuracy: 0.7536 - lr: 5
.0000e-04
Epoch 2/10
1512/1512 [==============================] - 74s 49ms/step - loss: 0.51
28 - accuracy: 0.7537 - val_loss: 0.5169 - val_accuracy: 0.7545 - lr: 5
.0000e-04
Epoch 3/10
```

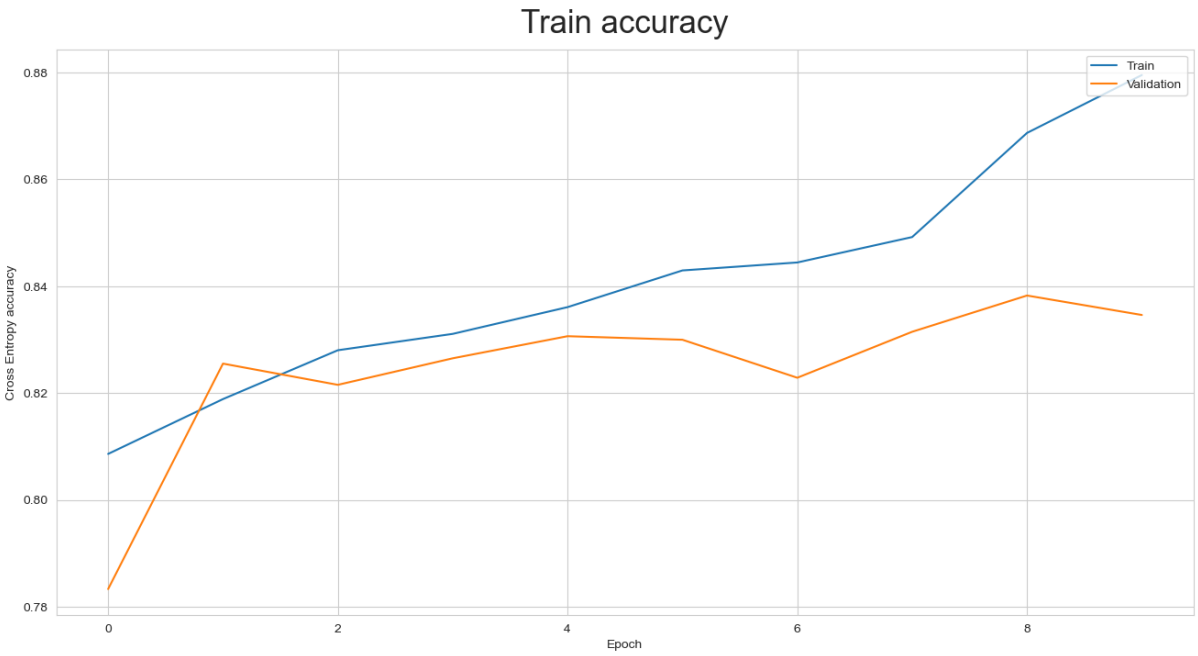
```
1512/1512 [==============================] - 74s 49ms/step - loss: 0.49
91 - accuracy: 0.7684 - val_loss: 0.4839 - val_accuracy: 0.7645 - lr: 5
.0000e-04
Epoch 4/10
1512/1512 [==============================] - 74s 49ms/step - loss: 0.49
03 - accuracy: 0.7733 - val_loss: 0.4840 - val_accuracy: 0.7721 - lr: 5
.0000e-04
Epoch 5/10
1512/1512 [==============================] - 74s 49ms/step - loss: 0.48
29 - accuracy: 0.7775 - val_loss: 0.4775 - val_accuracy: 0.7713 - lr: 5
.0000e-04
Epoch 6/10
1512/1512 [==============================] - 74s 49ms/step - loss: 0.48
21 - accuracy: 0.7784 - val_loss: 0.4684 - val_accuracy: 0.7749 - lr: 5
.0000e-04
Epoch 7/10
1512/1512 [==============================] - 74s 49ms/step - loss: 0.47
54 - accuracy: 0.7790 - val_loss: 0.4724 - val_accuracy: 0.7717 - lr: 5
.0000e-04
Epoch 8/10
1512/1512 [==============================] - 74s 49ms/step - loss: 0.47
28 - accuracy: 0.7792 - val_loss: 0.4920 - val_accuracy: 0.7653 - lr: 5
.0000e-04
Epoch 9/10
1512/1512 [==============================] - 74s 49ms/step - loss: 0.46
42 - accuracy: 0.7850 - val_loss: 0.4614 - val_accuracy: 0.7807 - lr: 1
.0000e-04
Epoch 10/10
1512/1512 [==============================] - 73s 49ms/step - loss: 0.46
47 - accuracy: 0.7857 - val_loss: 0.4574 - val_accuracy: 0.7792 - lr: 1
.0000e-04
189/189 [==============================] - 4s 20ms/step - loss: 0.4574
- accuracy: 0.7792
Test loss: 0.4573941230773926
Test accuracy: 0.7791928648948669
189/189 [==============================] - 4s 20ms/step
              precision    recall  f1-score   support

           0       0.82      0.87      0.84      4135
           1       0.67      0.58      0.63      1911

    accuracy                           0.78      6046
   macro avg       0.75      0.73      0.73      6046
weighted avg       0.77      0.78      0.77      6046
```

WE CAN SEE THE CNN MODEL ACCURACY IS 78%.

```
189/189 [==============================] - 4s 20ms/step
              precision    recall  f1-score   support

           0       0.82      0.87      0.84      4135
           1       0.67      0.58      0.63      1911

    accuracy                           0.78      6046
   macro avg       0.75      0.73      0.73      6046
weighted avg       0.77      0.78      0.77      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.

MILESTONE 2

STEP 1: FINE TUNE THE TRAINED BASIC CNN MODELS FOR CLASSIFICATION.

Precision, recall, F1-score, support, and accuracy are metrics commonly used to evaluate the performance of classification models. These metrics are calculated based on the confusion matrix, which is a table that summarizes the performance of a classification algorithm. Here's a brief description of each:

Precision:

- Precision is the ratio of correctly predicted positive observations to the total predicted positives.
- It is a measure of how many of the predicted positive instances are actually positive.
- Precision is calculated as TP/(TP+FP), where TP is the number of true positives and FP is the number of false positives.

Recall (Sensitivity or True Positive Rate):

- Recall is the ratio of correctly predicted positive observations to the all observations in the actual class.
- It is a measure of how many of the actual positive instances were correctly predicted.
- Recall is calculated as TP/(TP+FN), where TP is the number of true positives and FN is the number of false negatives.

F1-score:

- The F1-score is the harmonic mean of precision and recall.
- It provides a balance between precision and recall.
- F1-score is calculated as (2×Precision×Recall)/(Precision+Recall)

Support:

- Support is the number of actual occurrences of the class in the specified dataset.
- It is the number of true instances in each class.

Accuracy:

- Accuracy is the ratio of correctly predicted observations to the total observations.
- It is a general measure of the model's correctness.
- Accuracy is calculated as (TP+TN)/(TP+TN+FP+FN), where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

In summary, precision focuses on the accuracy of positive predictions, recall emphasizes the coverage of actual positive instances, the F1-score balances precision and recall, support indicates the number of instances in each class, and accuracy measures the overall correctness of predictions across all classes. It's important to consider the specific goals and requirements of your application when choosing which metric to prioritize.

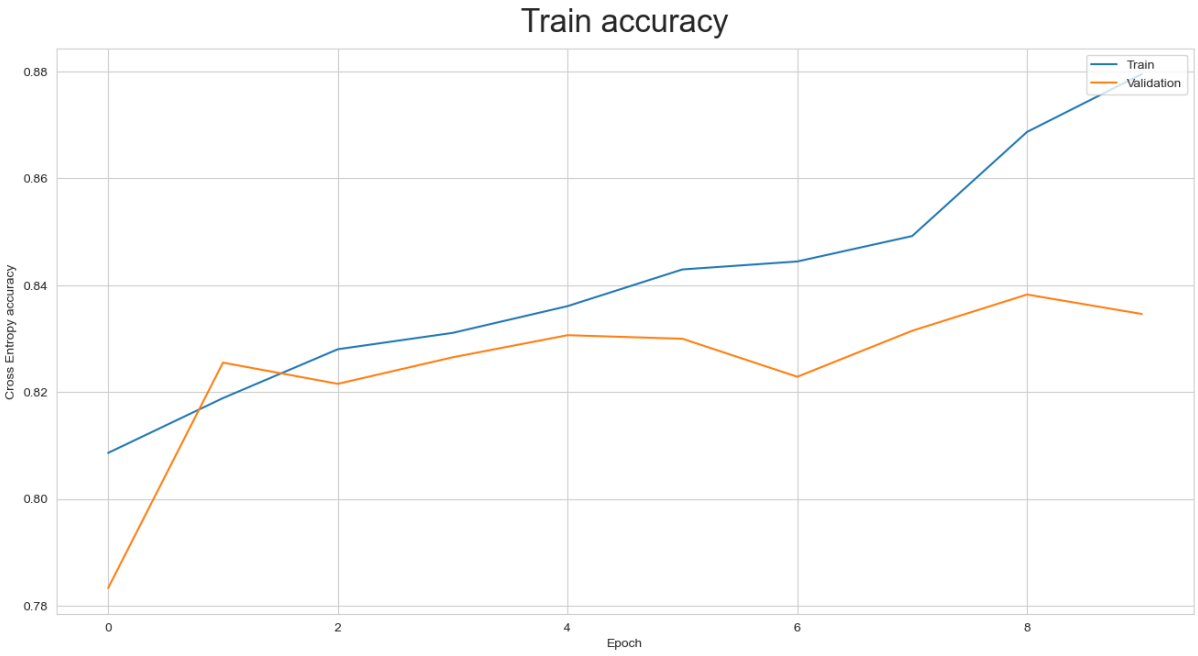
```
base_model.trainable = True
for layer in base_model.layers:
    if isinstance(layer, BatchNormalization): # set BatchNorm layers as not trainable
        layer.trainable = False

model.compile(optimizer=AdamW(learning_rate=0.00001), #
                  loss=losses.categorical_crossentropy,
                  metrics=['accuracy']) # ,f1_m

history = model.fit(X_train,
                        y_train,
                        batch_size=16,
                        epochs=10    ,
                        validation_data=(X_test, y_test),
                        callbacks=[
                            EarlyStopping(monitor = "val_loss", patience = 15,
                                          restore_best_weights = True, mode='min'),
                            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 1/10
1512/1512 [==============================] - 1714s 1s/step - loss: 0.4116 - accuracy: 0.8086 - val_loss: 0.4389 - val_accuracy:
0.7833 - lr: 1.0000e-05
Epoch 2/10
1512/1512 [==============================] - 1712s 1s/step - loss: 0.3930 - accuracy: 0.8189 - val_loss: 0.4000 - val_accuracy:
0.8255 - lr: 1.0000e-05
Epoch 3/10
1512/1512 [==============================] - 1708s 1s/step - loss: 0.3793 - accuracy: 0.8280 - val_loss: 0.3971 - val_accuracy:
0.8215 - lr: 1.0000e-05
Epoch 4/10
1512/1512 [==============================] - 1710s 1s/step - loss: 0.3714 - accuracy: 0.8311 - val_loss: 0.3992 - val_accuracy:
0.8265 - lr: 1.0000e-05
Epoch 5/10
1512/1512 [==============================] - 1709s 1s/step - loss: 0.3602 - accuracy: 0.8361 - val_loss: 0.3787 - val_accuracy:
0.8306 - lr: 1.0000e-05
Epoch 6/10
1512/1512 [==============================] - 1709s 1s/step - loss: 0.3478 - accuracy: 0.8429 - val_loss: 0.3781 - val_accuracy:
0.8300 - lr: 1.0000e-05
Epoch 7/10
1512/1512 [==============================] - 1709s 1s/step - loss: 0.3381 - accuracy: 0.8444 - val_loss: 0.3970 - val_accuracy:
0.8229 - lr: 1.0000e-05
Epoch 8/10
1512/1512 [==============================] - 1716s 1s/step - loss: 0.3277 - accuracy: 0.8492 - val_loss: 0.4195 - val_accuracy:
0.8315 - lr: 1.0000e-05
Epoch 9/10
1512/1512 [==============================] - 1710s 1s/step - loss: 0.2914 - accuracy: 0.8687 - val_loss: 0.3802 - val_accuracy:
0.8382 - lr: 2.0000e-06
Epoch 10/10
1512/1512 [==============================] - 1711s 1s/step - loss: 0.2740 - accuracy: 0.8795 - val_loss: 0.3959 - val_accuracy:
0.8346 - lr: 2.0000e-06
189/189 [==============================] - 97s 515ms/step - loss: 0.3959 - accuracy: 0.8346
Test loss: 0.3958609700202942
Test accuracy: 0.8346014022827148
189/189 [==============================] - 98s 520ms/step
```

	precision	recall	f1-score	support
0	0.89	0.87	0.88	4135
1	0.73	0.76	0.74	1911
accuracy			0.83	6046
macro avg	0.81	0.81	0.81	6046
weighted avg	0.84	0.83	0.84	6046

We observed accuracy is improved to 83% after fine tuning the model.

```
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()
```




Train Loss
Train
Validation
Cross Entropy Loss
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0
2
4
6
8
Epoch

STEP 2: APPLY TRANSFER LEARNING MODEL FOR CLASSIFICATION

Transfer learning is a machine learning technique that involves taking knowledge learned from one task and applying it to a different but related task. In the context of classification, transfer learning is often used in image, text, or speech recognition tasks. Here's a step-by-step explanation of how transfer learning works for classification:

Select a Pre-trained Model:

Choose a pre-trained model that has been trained on a large dataset for a specific task. This model is typically trained on a more general problem, such as image classification, and has learned to extract useful features from the input data. Common pre-trained models include VGG, ResNet, Inception, and BERT for various tasks.

Remove or Modify Top Layers (Optional):

Depending on your specific classification task, you may need to modify the architecture of the pre-trained model. If the original model was trained for a different number of classes, you might need to remove or adapt the final classification layers. This step ensures that the model is compatible with your target task.

Freeze Pre-trained Layers:

Freeze the weights of the pre-trained layers. This means that during the training of your new classification model, the weights of the pre-trained layers remain fixed and do not get updated. This helps to preserve the knowledge learned by the model on the original task.

Add New Classification Layers:

Add new layers to the model that are specific to your classification task. These layers are often referred to as the "head" of the model and are responsible for making predictions on your specific classes. These layers are initialized with random weights and will be trained on your dataset.

Compile the Model:

Compile the model with an appropriate optimizer, loss function, and metrics for your classification task. The choice of these components depends on whether you are dealing with binary or multiclass classification and the characteristics of your specific problem.

Train the Model:

Train the model on your target dataset using the frozen pre-trained layers and the newly added classification layers. The pre-trained layers help the model quickly adapt to the features present in your dataset, while the new layers learn to make task-specific predictions.

Fine-tuning (Optional):

Optionally, you can perform fine-tuning by unfreezing some of the pre-trained layers and allowing them to be updated during training. This is especially useful when you have a larger dataset for your specific task and want to fine-tune the model's learned features.

Evaluate and Adjust:

Evaluate the performance of the model on a separate validation or test dataset. Adjust the model architecture, hyperparameters, or perform additional fine-tuning as needed to achieve satisfactory results.

Transfer learning is beneficial when you have limited data for your target task, as it allows you to leverage the knowledge acquired by a model on a more general task. This can lead to improved performance and faster convergence during training.

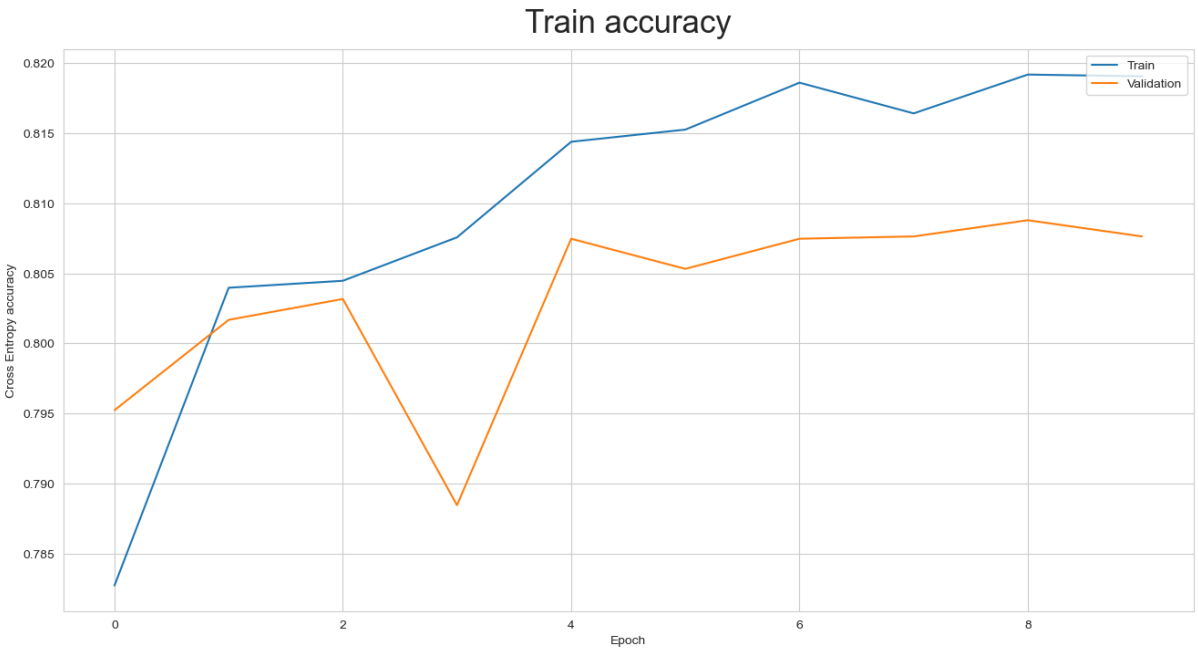
```
base_model = tf.keras.applications.VGG16(weights = 'imagenet', include_top = False)
for layer in base_model.layers:
      layer.trainable = False

# Data Augmentation Step
augment = Sequential([
#   RandomFlip("horizontal"),
#     Rescaling(1./255), # pretrained model already include rescaling
    RandomRotation(0.15)
], name='AugmentationLayer')

inputs = Input(shape = input_shape, name='inputLayer')
x = augment(inputs)
pretrain_out = base_model(x, training = False)
x = Flatten()(pretrain_out)
x = Dense(64, activation='relu')(x)
x = Dense(y_train.shape[1], name='outputLayer')(x)
outputs = Activation(activation="softmax", dtype=tf.float32, name='activationLayer')(x)
model = Model(inputs=inputs, outputs=outputs)

model.compile(optimizer=AdamW(learning_rate=0.0005), #
                  loss=losses.categorical_crossentropy,
                  metrics=['accuracy']) # ,f1_m


history = model.fit(X_train,
                        y_train,
                        batch_size=16,
                        epochs=10,
                        validation_data=(X_test, y_test),
                        callbacks=[
                            EarlyStopping(monitor = "val_loss", patience = 15,
                                          restore_best_weights = True, mode='min'),
                            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 1/10
1512/1512 [==============================] - 536s 354ms/step - loss: 0.5102 - accuracy: 0.7827 - val_loss: 0.4546 - val_accurac
y: 0.7952 - lr: 5.0000e-04
Epoch 2/10
1512/1512 [==============================] - 542s 358ms/step - loss: 0.4296 - accuracy: 0.8040 - val_loss: 0.4302 - val_accurac
y: 0.8017 - lr: 5.0000e-04
Epoch 3/10
1512/1512 [==============================] - 515s 341ms/step - loss: 0.4187 - accuracy: 0.8045 - val_loss: 0.4569 - val_accurac
y: 0.8032 - lr: 5.0000e-04
Epoch 4/10
1512/1512 [==============================] - 509s 337ms/step - loss: 0.4133 - accuracy: 0.8076 - val_loss: 0.4659 - val_accurac
y: 0.7885 - lr: 5.0000e-04
Epoch 5/10
1512/1512 [==============================] - 507s 335ms/step - loss: 0.3989 - accuracy: 0.8144 - val_loss: 0.4308 - val_accurac
y: 0.8075 - lr: 1.0000e-04
Epoch 6/10
1512/1512 [==============================] - 508s 336ms/step - loss: 0.3953 - accuracy: 0.8153 - val_loss: 0.4447 - val_accurac
y: 0.8053 - lr: 1.0000e-04
Epoch 7/10
1512/1512 [==============================] - 511s 338ms/step - loss: 0.3893 - accuracy: 0.8186 - val_loss: 0.4318 - val_accurac
y: 0.8075 - lr: 2.0000e-05
Epoch 8/10
1512/1512 [==============================] - 508s 336ms/step - loss: 0.3887 - accuracy: 0.8164 - val_loss: 0.4288 - val_accurac
y: 0.8076 - lr: 2.0000e-05
Epoch 9/10
1512/1512 [==============================] - 508s 336ms/step - loss: 0.3875 - accuracy: 0.8192 - val_loss: 0.4295 - val_accurac
y: 0.8088 - lr: 2.0000e-05
Epoch 10/10
1512/1512 [==============================] - 507s 335ms/step - loss: 0.3882 - accuracy: 0.8191 - val_loss: 0.4286 - val_accurac
y: 0.8076 - lr: 2.0000e-05
189/189 [==============================] - 98s 516ms/step - loss: 0.4286 - accuracy: 0.8076
Test loss: 0.4285995364189148
Test accuracy: 0.8076413869857788
189/189 [==============================] - 98s 517ms/step
```

	precision	recall	f1-score	support
0	0.84	0.89	0.86	4135
1	0.72	0.63	0.68	1911
accuracy			0.81	6046
macro avg	0.78	0.76	0.77	6046
weighted avg	0.80	0.81	0.80	6046

Applying transfer learning model accuracy improves to 81%.

```
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()
```



STEP 3: DESIGN, TRAIN AND TEST RCNN & ITS HYBRIDS BASED OBJECT DETECTION MODELS TO IMPOSE THE BOUNDING BOX OR MASK OVER THE AREA OF INTEREST.

RCNN Model:

The term **"RCNN" (Region-based Convolutional Neural Network)** actually refers to a family of convolutional neural network architectures used primarily for object detection in computer vision. The RCNN models, including their variations like Fast R-CNN, Faster R-CNN, and Mask R-CNN, have significantly improved object detection accuracy in various tasks.

Here's an overview of the general architecture and evolution:

1. **RCNN (Region-based Convolutional Neural Network):**
   - Introduced by Ross Girshick, et al. in 2014.
   - A two-stage method where regions of interest (RoIs) are proposed using selective search.
   - Each region is independently processed through a CNN to extract features, then classified using SVMs.

2. **Fast R-CNN:**
   - Proposed as an improvement to RCNN in 2015 by the same authors.
   - Introduces a region of interest pooling layer, allowing feature extraction for all regions simultaneously.
   - Speeds up the process by sharing convolutional layers for feature extraction.

3. **Faster R-CNN:**
   - Proposed by Shaoqing Ren, et al. in 2015.
   - Integrates the Region Proposal Network (RPN) into the network architecture.
   - The RPN generates region proposals directly from the CNN's feature maps, making the process end-to-end trainable.
   - Significantly faster than the previous methods.

4. **Mask R-CNN:**
   - Proposed by Kaiming He, et al. in 2017.
   - Extends Faster R-CNN by adding a branch for predicting segmentation masks on each RoI.
   - Enables instance segmentation, allowing precise pixel-level delineation of objects.

The key features of these RCNN variants include the ability to generate region proposals, extract features, and perform object classification and bounding box regression in a single unified framework. The introduction of RPNs and the integration of these modules into a single network architecture have significantly improved accuracy and efficiency in object detection tasks.

These models have been instrumental in advancing state-of-the-art performance in object detection, instance segmentation, and related computer vision tasks, making them widely used in research and applications across various domains.

The purpose of these variables is to handle the resizing or adjustment of image sizes in a consistent manner, especially when dealing with deep learning models that might have specific input size requirements. The FACTOR is used to scale down the original images to the desired size during preprocessing.

```
IMAGE_SIZE = 1024 #sample_pixel_array.shape[0]
ADJUSTED_IMAGE_SIZE=224
FACTOR = ADJUSTED_IMAGE_SIZE/IMAGE_SIZE
```

```
train_df.fillna(0,inplace=True)
```

```
train_df['x']            = train_df['x'].astype('int');
train_df['y']            = train_df['y'].astype('int');
train_df['width']        = train_df['width'].astype('int');
train_df['height']       = train_df['height'].astype('int');
```

```
print('Original dataframe shape:', train_df.shape)

train_labels_df_pos = pd.DataFrame(columns=['patientId', 'x', 'y', 'width', 'height'])

k = 0
for i in range(len(train_df)):
    if train_df.iloc[i]['Target'] == 1:
        train_labels_df_pos.loc[k] = train_df.loc[i]
        k += 1

print('Positive instances dataframe shape:', train_labels_df_pos.shape)
#train_paths = [os.path.join(IMAGE_PATH, image[0]) for image in train_labels_df_pos.values]
```

```
Original dataframe shape: (30227, 6)
Positive instances dataframe shape: (9555, 5)
```

Below code defines a PyTorch dataset class, Pneumonia Dataset, tailored for pneumonia detection tasks. It encapsulates functionalities for loading and preprocessing DICOM images along with associated bounding box information.

The dataset initialization captures unique patient IDs, and the __getitem__ method retrieves preprocessed images and target dictionaries containing bounding boxes, labels, and additional information.

The code also includes functions for data augmentation transformations using the Augmentations library, a collate function for batch processing, and an Averager utility class for computing average values during training.

This dataset class is designed to facilitate the training and validation of a pneumonia detection model in PyTorch.

```
from torch.utils.data import DataLoader, Dataset
class PneumoniaDataset(Dataset):
    def __init__(self, dataframe, image_dir, transforms=None):
        super().__init__()

        self.image_ids = dataframe['patientId'].unique()
        self.df = dataframe
        self.image_dir = image_dir
        self.transforms = transforms

    def __getitem__(self, index):
        # Load images and bounding boxes
        image_id = self.image_ids[index]
        records = self.df[self.df['patientId'] == image_id]

        dicom_data = pydicom.read_file(os.path.join(self.image_dir + '%s.dcm' % image_id))

        img = dicom_data.pixel_array
        img = cv2.resize(img, (ADJUSTED_IMAGE_SIZE, ADJUSTED_IMAGE_SIZE))

        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB).astype(np.float32)
        img /= 255.0
        boxes = records[['x', 'y', 'width', 'height']].values
        boxes[:, 2] = boxes[:, 0] + boxes[:, 2]
        boxes[:, 3] = boxes[:, 1] + boxes[:, 3]

        #Resizing
        boxes[:, 0]=boxes[:, 0].astype(np.int32)* FACTOR
        boxes[:, 1]=boxes[:, 1].astype(np.int32)* FACTOR
        boxes[:, 2]=boxes[:, 2].astype(np.int32)* FACTOR
        boxes[:, 3]=boxes[:, 3].astype(np.int32)* FACTOR
        boxes=np.vstack(boxes).astype(np.float32)
        #print("Final co:")
        #print(boxes[:,0],boxes[:,2],boxes[:,1],boxes[:,3])
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
        area = np.vstack(area).astype(np.float32)
        area = torch.as_tensor(area, dtype=torch.float32)


        # there is only one class
        labels = torch.ones((records.shape[0],), dtype=torch.int64)
        # suppose all instances are not crowd
        iscrowd = torch.zeros((records.shape[0],), dtype=torch.int64)

        target = {}
        target['boxes'] = boxes
        target['labels'] = labels
        target['patientId'] = torch.tensor([index])
        target['area'] = area
        target['iscrowd'] = iscrowd

        if self.transforms:
            sample = {
                'image': img,
                'bboxes': target['boxes'],
                'labels': labels
            }
            sample = self.transforms(**sample)
            img = sample['image']

            target['boxes'] = torch.stack(tuple(map(torch.FloatTensor, zip(*sample['bboxes'])))).permute(1, 0)

        return img, target


    def __len__(self):
        return self.image_ids.shape[0]

# Albumentations
def get_train_transform():
    return A.Compose([
        A.Resize(width=ADJUSTED_IMAGE_SIZE,height=ADJUSTED_IMAGE_SIZE),
        A.Flip(0.5),
        A.RandomRotate90(0.5),
        A.MotionBlur(p=0.2),
        A.MedianBlur(blur_limit=3, p=0.1),
        A.Blur(blur_limit=3, p=0.1),
        ToTensorV2(p=1.0)
```

```
class Averager:
    def __init__(self):
        self.current_total = 0.0
        self.iterations = 0.0

    def send(self, value):
        self.current_total += value
        self.iterations += 1

    @property
    def value(self):
        if self.iterations == 0:
            return 0
        else:
            return 1.0 * self.current_total / self.iterations

    def reset(self):
        self.current_total = 0.0
        self.iterations = 0.0
```

The below code takes a list of patients with pneumonia, splits them into two groups—one for training the model and the other for validating its performance. The sizes of these groups are printed out. Then, it creates datasets for training and validation using these patient groups and applies some image transformations to make the model more robust.

Finally, it sets up data loaders, which help efficiently feed batches of data to the model during training and validation. The printed messages provide information about how many patients are in each group and how many examples are available for the model to learn from.

```
import albumentations as A
from albumentations.pytorch.transforms import ToTensorV2


image_ids = train_labels_df_pos['patientId'].unique()
valid_ids = image_ids[-300:]
train_ids = image_ids[:-300]
print(f"Training instance: {len(train_ids)}")
print(f"Validation instances: {len(valid_ids)}")

valid_df = train_labels_df_pos[train_labels_df_pos['patientId'].isin(valid_ids)]
train_df = train_labels_df_pos[train_labels_df_pos['patientId'].isin(train_ids)]

print('Train dataframe shape:', train_df.shape)
print('Valid dataframe shape:', valid_df.shape)

train_dataset = PneumoniaDataset(train_df, IMAGE_PATH, get_train_transform())
valid_dataset = PneumoniaDataset(valid_df, IMAGE_PATH, get_valid_transform())
print('train_dataset and valid_dataset are loaded :)')
print("We have: {} training examples and {} validation examples".format(len(train_dataset), len(valid_dataset)))

train_data_loader = DataLoader(train_dataset, batch_size=8, shuffle=False, num_workers=2, collate_fn=collate_fn)
valid_data_loader = DataLoader(valid_dataset, batch_size=8, shuffle=False, num_workers=2, collate_fn=collate_fn)
```

```
Training instance: 5712
Validation instances: 300
Train dataframe shape: (9057, 5)
Valid dataframe shape: (498, 5)
train_dataset and valid_dataset are loaded :)
We have: 5712 training examples and 300 validation examples
```

Below code is part of an evaluation process for an object detection model. The primary focus is on measuring the accuracy and precision of the model's predictions.

The Intersection over Union (IoU) is used to assess how well the predicted bounding boxes match the actual objects. The code calculates precision, which quantifies the accuracy of the model by considering true positives (correctly identified objects) and false positives (incorrectly identified objects).

The Image Precision is an average precision score at different IoU thresholds, providing an overall evaluation metric for the entire model's performance in detecting and localizing objects in images. This analysis aids in understanding the effectiveness of the object detection algorithm.

```
def calculate_image_precision(gts, preds, thresholds = (0.5, ), form = 'coco') -> float:
    # https://www.kaggle.com/sadmanoraf/wheat-detection-using-faster-rcnn-train
    """Calculates image precision.

    Args:
        gts: (List[List[Union[int, float]]]) Coordinates of the available ground-truth boxes
        preds: (List[List[Union[int, float]]]) Coordinates of the predicted boxes,
               sorted by confidence value (descending)
        thresholds: (float) Different thresholds
        form: (str) Format of the coordinates

    Return:
        (float) Precision
    """
    n_threshold = len(thresholds)
    image_precision = 0.0

    ious = np.ones((len(gts), len(preds))) * -1
    # ious = None

    for threshold in thresholds:
        precision_at_threshold = calculate_precision(gts.copy(), preds, threshold=threshold,
                                                     form=form, ious=ious)
        image_precision += precision_at_threshold / n_threshold

    return image_precision


def calculate_iou(gt, pr, form='pascal_voc') -> float:
    # https://www.kaggle.com/sadmanoraf/wheat-detection-using-faster-rcnn-train
    """Calculates the Intersection over Union.

    Args:
        gt: (np.ndarray[Union[int, float]]) coordinates of the ground-truth box
        pr: (np.ndarray[Union[int, float]]) coordinates of the prdected box
        form: (str) gt/pred coordinates format
            - pascal_voc: [xmin, ymin, xmax, ymax]
            - coco: [xmin, ymin, w, h]
    Returns:
        (float) Intersection over union (0.0 <= iou <= 1.0)
    """
    if form == 'coco':
        gt = gt.copy()
        pr = pr.copy()

        gt[2] = gt[0] + gt[2]
        gt[3] = gt[1] + gt[3]
        pr[2] = pr[0] + pr[2]
        pr[3] = pr[1] + pr[3]

    # Calculate overlap area
    dx = min(gt[2], pr[2]) - max(gt[0], pr[0]) + 1

    if dx < 0:
        return 0.0
    dy = min(gt[3], pr[3]) - max(gt[1], pr[1]) + 1

    if dy < 0:
        return 0.0

    overlap_area = dx * dy

    # Calculate union area
    union_area = (
            (gt[2] - gt[0] + 1) * (gt[3] - gt[1] + 1) +
            (pr[2] - pr[0] + 1) * (pr[3] - pr[1] + 1) -
            overlap_area
    )

    return overlap_area / union_area
```

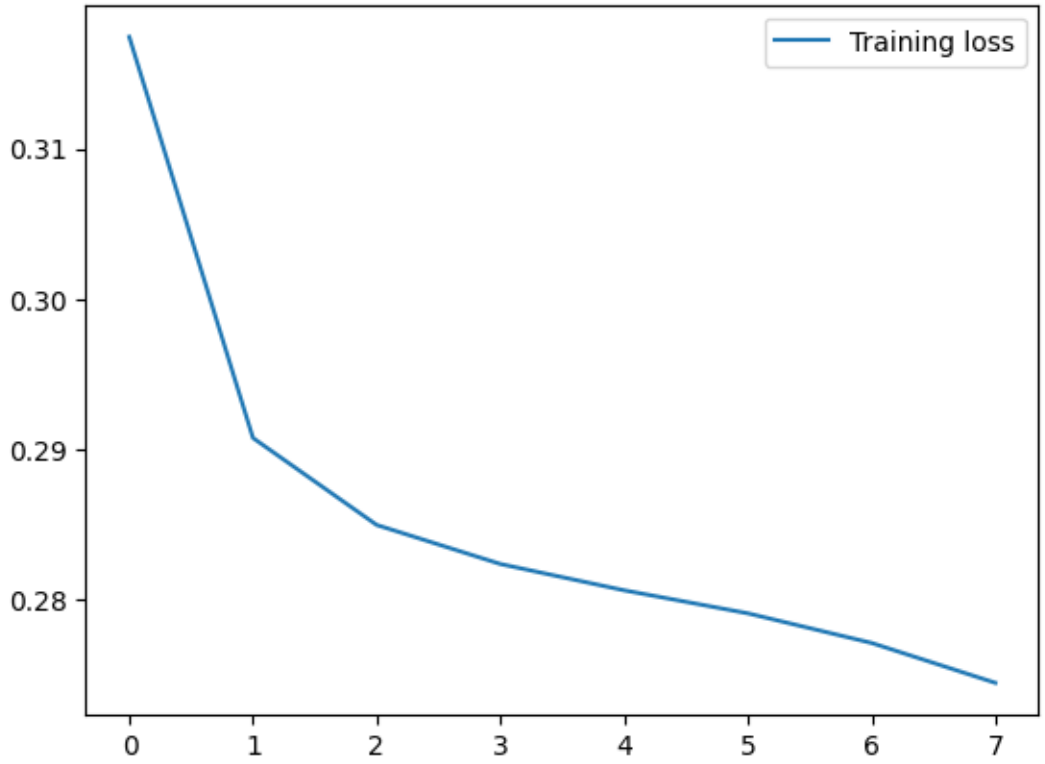
```
    Args:
        gts: (List[List[Union[int, float]]]) Coordinates of the available ground-truth boxes
        pred: (List[Union[int, float]]) Coordinates of the predicted box
        pred_idx: (int) Index of the current predicted box
        threshold: (float) Threshold
        form: (str) Format of the coordinates
        ious: (np.ndarray) len(gts) x len(preds) matrix for storing calculated ious.

    Return:
        (int) Index of the best match GT box (-1 if no match above threshold)
    """
    best_match_iou = -np.inf
    best_match_idx = -1
    for gt_idx in range(len(gts)):

        if gts[gt_idx][0] < 0:
            # Already matched GT-box
            continue

        iou = -1 if ious is None else ious[gt_idx][pred_idx]

        if iou < 0:
            iou = calculate_iou(gts[gt_idx], pred, form=form)

            if ious is not None:
                ious[gt_idx][pred_idx] = iou

        if iou < threshold:
            continue

        if iou > best_match_iou:
            best_match_iou = iou
            best_match_idx = gt_idx

    return best_match_idx

def calculate_precision(gts, preds, threshold = 0.5, form = 'coco', ious=None) -> float:
    # https://www.kaggle.com/sadmansraf/wheat-detection-using-faster-rcnn-train
    """Calculates precision for GT - prediction pairs at one threshold.

    Args:
        gts: (List[List[Union[int, float]]]) Coordinates of the available ground-truth boxes
        preds: (List[List[Union[int, float]]]) Coordinates of the predicted boxes,
               sorted by confidence value (descending)
        threshold: (float) Threshold
        form: (str) Format of the coordinates
        ious: (np.ndarray) len(gts) x len(preds) matrix for storing calculated ious.

    Return:
        (float) Precision
    """
    n = len(preds)
    tp = 0
    fp = 0

    for pred_idx in range(n):

        best_match_gt_idx = find_best_match(gts, preds[pred_idx], pred_idx,
                                            threshold=threshold, form=form, ious=ious)

        if best_match_gt_idx >= 0:
            # True positive: The predicted box matches a gt box with an IoU above the threshold.
            tp += 1
            # Remove the matched GT box
            gts[best_match_gt_idx] = -1
        else:
            # No match
            # False positive: indicates a predicted box had no associated gt box.
            fp += 1

    # False negative: indicates a gt box had no associated predicted box.
    fn = (gts.sum(axis=1) > 0).sum()

    return tp / (tp + fp + fn)
```

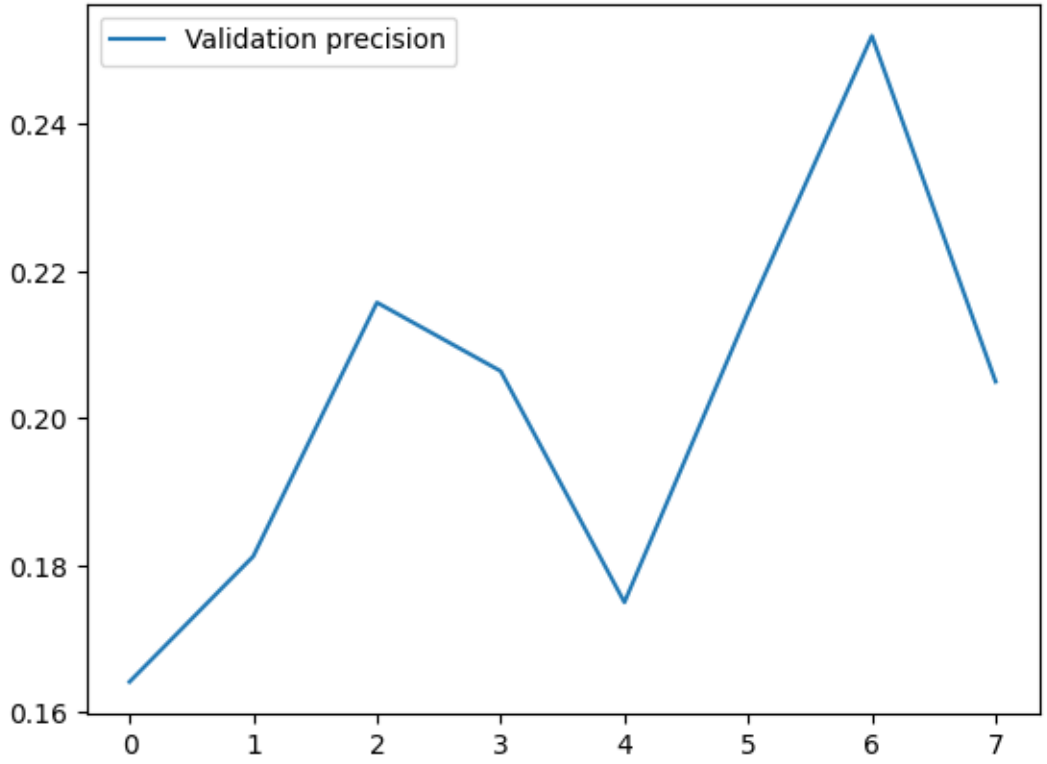
```
def train(dataloader, lr_scheduler, model, optimizer,
          device, epoch, loss_hist, itr):
    model.train()
    start = time.time()
    loss_hist.reset()
    for images, targets in dataloader:
        images = list(image.to(device) for image in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())
        loss_value = losses.item()
        loss_hist.send(loss_value)
        optimizer.zero_grad()
        losses.backward()
        optimizer.step()
        if itr % 100 == 0:
            print(f"Epoch #{epoch} iteration #{itr} loss: {loss_value}")
        itr += 1
    end = time.time()
    return loss_hist, end, start

def validate(dataloader, model, device, iou_thresholds):
    valid_image_precision = []
    model.eval()
    with torch.no_grad():
        for images, targets in dataloader:

            images = list(image.to(device) for image in images)
            targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

            outputs = model(images)
    for i, image in enumerate(images):
        boxes = outputs[i]['boxes'].data.cpu().numpy()
        scores = outputs[i]['scores'].data.cpu().numpy()
        gt_boxes = targets[i]['boxes'].cpu().numpy()
        preds_sorted_idx = np.argsort(scores)[::-1]
        preds_sorted = boxes[preds_sorted_idx]
        image_precision = calculate_image_precision(preds_sorted,
                                                    gt_boxes,
                                                    thresholds=iou_thresholds,
                                                    form='coco')
        valid_image_precision.append(image_precision)

    valid_prec = np.mean(valid_image_precision)
    return valid_prec
```

```
torch.cuda.is_available()
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
def get_model():
    # Load the COCO pre-trained model
    model = torchvision.models.detection.fasterrcnn_resnet50_fpn_v2(weights='DEFAULT')
    # one class is pneumonia, and the other is background
    num_classes = 2
    # get the input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace pre-trained head with our features head
    # the head layer will classify the images based on our data input features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
    return model
```

```
# Learning parameters
num_epochs = 8
lr = 0.001
batch_size = 6
train_loss = []
precision = []

# initialize the Averager
loss_hist = Averager()
iou_thresholds = [x for x in np.arange(0.5, 0.76, 0.05)]

model = get_model().to(device)
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=lr, momentum=0.9, weight_decay=0.0005)
lr_scheduler = None

for epoch in range(num_epochs):
    itr = 1
    train_loss_hist, end, start = train(train_data_loader, lr_scheduler,
                                        model, optimizer, device,
                                        epoch, loss_hist, itr)
    valid_prec = validate(valid_data_loader, model, device, iou_thresholds)
    print(f"Took {(end-start)/60:.3f} minutes for epoch# {epoch} to train")
    print(f"Epoch #{epoch} Train loss: {train_loss_hist.value}")
    print(f"Epoch #{epoch} Validation Precision: {valid_prec}")
    train_loss.append(train_loss_hist.value)
    precision.append(valid_prec)

    # update the learning rate
    if lr_scheduler is not None:
        lr_scheduler.step()

torch.save(model.state_dict(), 'fasterrcnn_resnet50_fpn_pneumonia_detection.pth')
```

```
# plot the training loss
plt.figure()
plt.plot(train_loss, label='Training loss')
plt.legend()
plt.show()

# plot the validation precision
plt.figure()
plt.plot(precision, label='Validation precision')
plt.legend()
plt.show()
```


Training loss
0.31
0.30
0.29
0.28
0
1
2
3
4
5
6
7


Validation precision
0.24
0.22
0.20
0.18
0.16
0
1
2
3
4
5
6
7

```
images_test_path = 'C:/Users/manve/Downloads/rsna-pneumonia-detection-challenge/stage_2_test_images/'
test_images = os.listdir(images_test_path)
print(f"Validation instances: {len(test_images)}")

# # load a model; pre-trained on COCO
model = get_model()

# # os.makedirs('../validation_predictions', exist_ok=True)
model.load_state_dict(torch.load('fasterrcnn_resnet50_fpn_pneumonia_detection.pth'))
model.to(device)
```

```
def format_prediction_string(boxes, scores):
    pred_strings = []
    for j in zip(scores, boxes):
        pred_strings.append("{0:.4f} {1} {2} {3} {4}".format(j[0],
                                                             int(j[1][0]), int(j[1][1]),
                                                             int(j[1][2]), int(j[1][3])))

    return " ".join(pred_strings)
```

```
detection_threshold = 0.8
results = []
model.eval()
f, axarr = plt.subplots(10, 3, figsize=(24,36))
axarr = axarr.ravel()
axidx = 0
with torch.no_grad():
   for i, image in tqdm(enumerate(test_images), total=len(test_images)):

        # plot image
        ID = "ID: ",format(image)
        #orig_image = cv2.imread(f"{images_test_path}/{test_images[i]}", cv2.IMREAD_COLOR)
        orig_image = pydicom.read_file(os.path.join(images_test_path + '%s' % image))
        image = orig_image.pixel_array
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB).astype(np.float32)
        image /= 255.0
        image = np.transpose(image, (2, 0, 1)).astype(np.float32)
        image = torch.tensor(image, dtype=torch.float).cuda()
        image = torch.unsqueeze(image, 0)

        model.eval()
        cpu_device = torch.device("cpu")

        outputs = model(image)

        outputs = [{k: v.to(cpu_device) for k, v in t.items()} for t in outputs]
        if len(outputs[0]['boxes']) != 0:
            for counter in range(len(outputs[0]['boxes'])):
                boxes = outputs[0]['boxes'].data.cpu().numpy()
                scores = outputs[0]['scores'].data.cpu().numpy()
                boxes = boxes[scores >= detection_threshold].astype(np.int32)
                draw_boxes = boxes.copy()
                boxes[:, 2] = boxes[:, 2] - boxes[:, 0]
                boxes[:, 3] = boxes[:, 3] - boxes[:, 1]
                #print(image.shape)
                #print(boxes)
                draw_boxes = boxes.copy()
                axarr[axidx].imshow(cv2.cvtColor(orig_image.pixel_array, cv2.COLOR_BGR2RGB))
                axarr[axidx].set_title(ID)

                for box in draw_boxes:
                   rectangle = Rectangle(xy= ((int(box[0]), int(box[1]))),width=int(box[2]),
                                         height=int(box[3]), color="red",alpha = 0.1)
                   axarr[axidx].add_patch(rectangle)

            axidx+=1
            result = {
               'patientId': test_images[i].split('.')[0],
               'PredictionString': format_prediction_string(boxes, scores)
            }
            results.append(result)
        else:
            result = {
               'patientId': test_images[i].split('.')[0],
               'PredictionString': None
            }
            results.append(result)
        i=i+1
        #exit after 32 images
        if i>29:
            break
```

```
detection_threshold = 0.8
 results = []
 model.eval()
 with torch.no_grad():
    for i, image in tqdm(enumerate(test_images), total=len(test_images)):

         # plot image
         orig_image = pydicom.read_file(os.path.join(images_test_path + '%s' % image))
         image = orig_image.pixel_array

         image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB).astype(np.float32)
         image /= 255.0
         image = np.transpose(image, (2, 0, 1)).astype(np.float32)
         image = torch.tensor(image, dtype=torch.float).cuda()
         image = torch.unsqueeze(image, 0)

         model.eval()
         cpu_device = torch.device("cpu")

         outputs = model(image)

         outputs = [{k: v.to(cpu_device) for k, v in t.items()} for t in outputs]
         if len(outputs[0]['boxes']) != 0:
             for counter in range(len(outputs[0]['boxes'])):
                 boxes = outputs[0]['boxes'].data.cpu().numpy()
                 scores = outputs[0]['scores'].data.cpu().numpy()
                 boxes = boxes[scores >= detection_threshold].astype(np.int32)
                 draw_boxes = boxes.copy()
                 boxes[:, 2] = boxes[:, 2] - boxes[:, 0]
                 boxes[:, 3] = boxes[:, 3] - boxes[:, 1]

                 draw_boxes = boxes.copy()

             result = {
                'patientId': test_images[i].split('.')[0],
                'PredictionString': format_prediction_string(boxes, scores)
            }
             results.append(result)
         else:
             result = {
                'patientId': test_images[i].split('.')[0],
                'PredictionString': ''
            }
             results.append(result)

 sub_df = pd.DataFrame(results, columns=['patientId', 'PredictionString'])
 print(sub_df.head())
 sub_df.to_csv('submission_frcnn.csv', index=False)
```

```
sub_df
```

```
sub_df[sub_df['PredictionString']!='']
```

```
validation_threshold=0.8
def show_pred_image_with_bboxes(patientId,axarr,axidx):
    with torch.no_grad():
        ID = "Predicted: ",format(patientId)

        orig_image = pydicom.read_file(os.path.join(IMAGE_PATH + '%s.dcm' % patientId))
        image = orig_image.pixel_array

        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB).astype(np.float32)
        image /= 255.0
        image = np.transpose(image, (2, 0, 1)).astype(np.float32)
        image = torch.tensor(image, dtype=torch.float).cuda()
        image = torch.unsqueeze(image, 0)

        model.eval()
        cpu_device = torch.device("cpu")

        outputs = model(image)

        outputs = [{k: v.to(cpu_device) for k, v in t.items()} for t in outputs]
        if len(outputs[0]['boxes']) != 0:
            for counter in range(len(outputs[0]['boxes'])):
                boxes = outputs[0]['boxes'].data.cpu().numpy()
                scores = outputs[0]['scores'].data.cpu().numpy()
                boxes = boxes[scores >= validation_threshold].astype(np.int32)
                draw_boxes = boxes.copy()
                boxes[:, 2] = boxes[:, 2] - boxes[:, 0]
                boxes[:, 3] = boxes[:, 3] - boxes[:, 1]
                draw_boxes = boxes.copy()
                axarr[axidx].imshow(cv2.cvtColor(orig_image.pixel_array, cv2.COLOR_BGR2RGB))
                axarr[axidx].set_title(ID)

                for box in draw_boxes:
                    rectangle = Rectangle(xy= ((int(box[0]), int(box[1]))),width=int(box[2]),
                                          height=int(box[3]), color="red",alpha = 0.1)
                    axarr[axidx].add_patch(rectangle)
```

```
def show_image_with_bboxes(patientId,axarr,axidx):
        ID = "Actual: ",format(patientId)
        id_= np.random.choice(train_labels_df_pos['patientId'].values)
        orig_image = pydicom.read_file(os.path.join(IMAGE_PATH + '%s.dcm' % patientId))
        image = orig_image.pixel_array
        axarr[axidx].imshow(cv2.cvtColor(orig_image.pixel_array, cv2.COLOR_BGR2RGB))
        axarr[axidx].set_title(ID)
        boxes=train_labels_df_pos[['x','y','width','height']][train_labels_df_pos['patientId']==id_].values
        for box in boxes:
            x=box[0]
            y=box[1]
            w=box[2]
            h=box[3]
            axarr[axidx].add_patch(plt.Rectangle((x, y), w, h, color='red', fill=False, linewidth=3))
```

```
dataset = valid_df
f, axarr = plt.subplots(6, 2, figsize=(16,36))
axarr = axarr.ravel()
axidx = 0

image_id = valid_df.sample(6)['patientId'].reset_index(drop=True)
for i in range(6):

    show_image_with_bboxes(image_id[i],axarr,axidx)
    axidx+=1
    show_pred_image_with_bboxes(image_id[i],axarr,axidx)
    axidx+=1
```

STEP 4: PICKLE THE MODEL FOR FUTURE PREDICTION

```
import pickle
```

```
pickle.dump(model, open('model.pkl', 'wb'))
```

```
# Loading model to compare the results
pickled_model = pickle.load(open('model.pkl', 'rb'))
pickled_model.predict(X_test)
```

CONCLUSION

We have used transfer learning model and found the accuracy 81% and have fine tunned the model which results the accuracy to 83%. Then we have used the Faster RCNN model to find the prediction of our Pneumonia detection dataset.

Project Conclusion and future work

We tried various models to achieve our goal of building a model with minimal computational resources so that faster and accurate results can be brought out in the healthcare industry, especially the imaging side as it is very computationally heavy. Using the existing data, it was possible to train several models with different configurations and parameters whose accuracies vary between ~74% and ~97%; we can conclude that the model's capabilities can be expanded if sufficient training data and advanced computational capability to run further iterations is available as the loss was still decreasing.

In this work, we have presented our approach for identifying pneumonia and understanding how the lung image size plays an important role for the model performance. We found that the distinction is quite subtle for images among presence or absence of pneumonia, large image can be more beneficial for deeper information. However, the computation cost also burden exponentially when dealing with large image.

Our proposed architecture with regional context RCNN with selective search supplied extra context for generating accurate results. Also, using thresholds in background while training tuned our network to perform well in this task. With the usage of image augmentation, dropout and L2 regularization prevented the overfitting, but are obtained something weaker results on the training set with respect to the test. Our model can be improved by adding new layers, but this would introduce even more hyperparameters that should be adjusted. We intend to extend our model architecture in other areas of medical imaging with the usage of deep learning and computer vision techniques.

RCNN with selective search has very limited application for Dicom image analysis due to its architecture. The performance of other models has been better as compared to RCNN. Yolov3 was able to predict the test images with good performance. But Faster-RCNN predicted 1098 test images out of 3000 samples with high confidence score. Hence, both YOLO and Faster RCNN can be taken up for further tuning in next phase of project.

Improvements Area:

Due to memory and GPU requirements, not all training data could be loaded. This can be fully used for better model training.

Running the training for more epocs with higher patience levels

Increasing the image dimension being used from 224 to 512px or above.

Stratified sampling of data for training gives the model less opportunity of learning the needed feature in turn leaving the model with fewer accuracies as here only 1/3rd data is of the positive class.

Different variants of image augmentation can be tried.

While the results are promising, there are several avenues for future research:

- **Clinical Validation:** Our model needs to be tested in a real-world clinical setting to assess its efficacy and reliability further.
- **Model Explainability:** Future work can also focus on making the model more interpretable for clinicians, possibly through the use of attention mechanisms or other explainability techniques.
- **Multi-Modal Approaches:** Incorporating additional data types, like patient history or other diagnostic tests, could potentially improve the model's predictive power.
- **Low-Resource Settings:** Optimizing the model for deployment in low-resource settings could have a significant impact, making high-quality diagnosis accessible to underserved populations.

In future, it would be interesting to see or use medical dataset repository as it would be instrumental in advancing computer vision and deep learning research for medical field. As more and more advanced architectures are emerging, we can look at more accurate classification and localization models to diagnose pneumonia. We could also look at a historical progressive monitoring of the patient and affected region, assisting the pulmonologist in a visual analysis of the patient’s on-going condition.

Thank You