

|-----SIIM-ACR Pneumothorax Segmentation-----|

Can Artificial Intelligence recognize of pneumothoraces(Collapsed Lung) from Chest X-ray and save lives?



Image Credit: <https://www.medimaging.net/industry-news/articles/294778410/acr-releases-second-research-road-map-on-medical-imaging-ai.html> (<https://www.medimaging.net/industry-news/articles/294778410/acr-releases-second-research-road-map-on-medical-imaging-ai.html>)

1. Business Problem:

1.1 Description:

- **Pneumothorax** is basically a combination of two words Pneumo(air) and Thorax(chest). Pneumothorax is also known as lung collapse. Pneumothorax is caused by abnormal collection of air between the parietal and visceral pleura i. e. pleural space between lungs and chest wall. Pneumothorax is a relatively common respiratory disease that can occur in a wide range of patients and in various clinical settings.

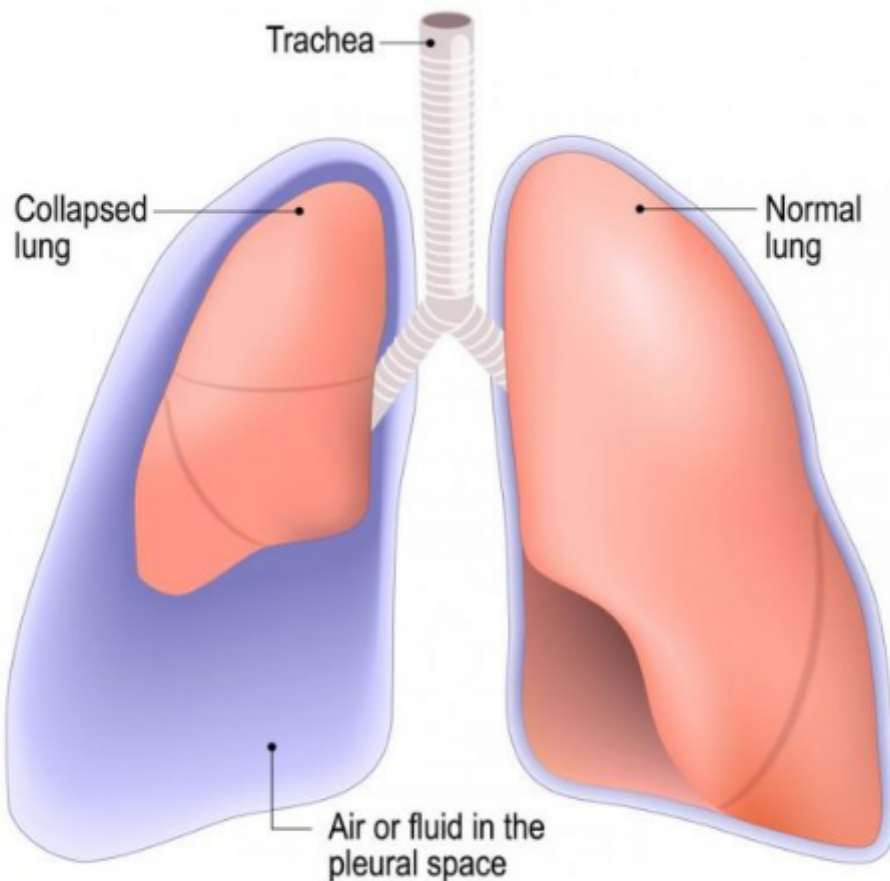


Image Credit: <https://www.firstaidforfree.com/what-is-a-spontaneous-pneumothorax/>

- **Symptoms** of pneumothorax include sudden onset of sharp, one-sided chest pain and shortness of breath. Pneumothorax can be caused by a blunt chest injury, damage from underlying lung disease, or most horrifying—it may occur for no obvious reason at all. On some occasions, a collapsed lung can be a life-threatening event. Diagnosis of pneumothorax by physical examination alone can be difficult, particularly in smaller pneumothoraces. Usually a chest X-ray, CT(Computed Tomography)-Scan or ultrasound is used for detecting or confirming the presence of pneumothorax. Small pneumothorax can be typically resolved without treatment and requires only monitoring. This approach may be appropriate in people who have no underlying disease. In larger pneumothorax or if there is shortness of breath air may be removed by a syringe or chest tube connected to a one way valve system. Occasionally, surgery may be required if tube drainage is unsuccessful. About 17-23 cases of pneumothorax occur per 100,000 people per year. They are more common in men than women.
- **Diagnosing** a pneumothorax in a chest radiography image is not difficult for an experienced physician or radiologist, but in some cases, it can easily be missed. Usually it is diagnosed by a radiologist on a chest x-ray, and can sometimes be very difficult to confirm as discussed above. An accurate AI algorithm to detect pneumothorax would be useful in a lot of clinical scenarios. AI could be used to triage chest radiographs for priority interpretation, or to provide a more confident diagnosis for non-radiologists. In other words, a machine learning-based pneumothorax diagnosis technique from the chest X-ray image is required to assist a physician to diagnose a pneumothorax.

Source: This problem belong to one of the competitions held on kaggle, which can be found on following link:

<https://www.kaggle.com/c/siim-acr-pneumothorax-segmentation/> (<https://www.kaggle.com/c/siim-acr-pneumothorax-segmentation/>)

1.2. Business Objectives

- We have to predict the pneumothorax and segmentize it based on the X-ray images
- Time taken for prediction should be from few seconds to few minutes.

2. Deep Learning problem

2.1 Data

The data is comprised of images in DICOM format and annotations in the form of image IDs and run-length-encoded (RLE) masks. Some of the images contain instances of pneumothorax (collapsed lung), which are indicated by encoded binary masks in the annotations. Some training images have multiple annotations.

```
1.2.276.0.7230010.3.1.4.8323329.14508.1517875252.443873,387620 23 996 33 986 43 977 51 968 58 962
65 956 70 952 74 949 76 946 79
```

Images without pneumothorax have a mask value of -1.

```
1.2.276.0.7230010.3.1.4.8323329.1034.1517875166.8504,-1
```

Link to the data set:

- <https://www.kaggle.com/seesees/siim-train-test> (<https://www.kaggle.com/seesees/siim-train-test>)

2.2 Mapping the real-world problem to a Deep Learning Problem

2.2.1 Type of Deep Learning Problem

As we have seen above in the dataset section we have dataset in the form of images, and our task is to predict the mask of pneumothorax in the X-ray image. This problem is of **Semantic Image Segmentation** problem. This model will assist a physician to diagnose a **Pneumothorax**.

2.2.2 Evaluation metric

- **Dice coefficient:**

Now we know this problem is a semantic segmentation problem. We have to define a metric for the evaluation of our Deep Learning model. There is a more commonly used and better metric for evaluation of segmentation models called the Dice Coefficient.

Dice coefficient originates from Sørensen–Dice coefficient, which is a statistic developed in the 1940s to gauge the similarity between two samples [Wikipedia]. It was brought to the computer vision community by Milletari et al. in 2016 for 3D medical image segmentation. Dice Loss is also known as the F1 score metric. In simple manner, the Dice coefficient is $2 \times$ the Area of Overlap divided by the total number of pixels in both images. Dice loss ranges from 0 to 1, with 1 signifying the greatest similarity between predicted and truth.

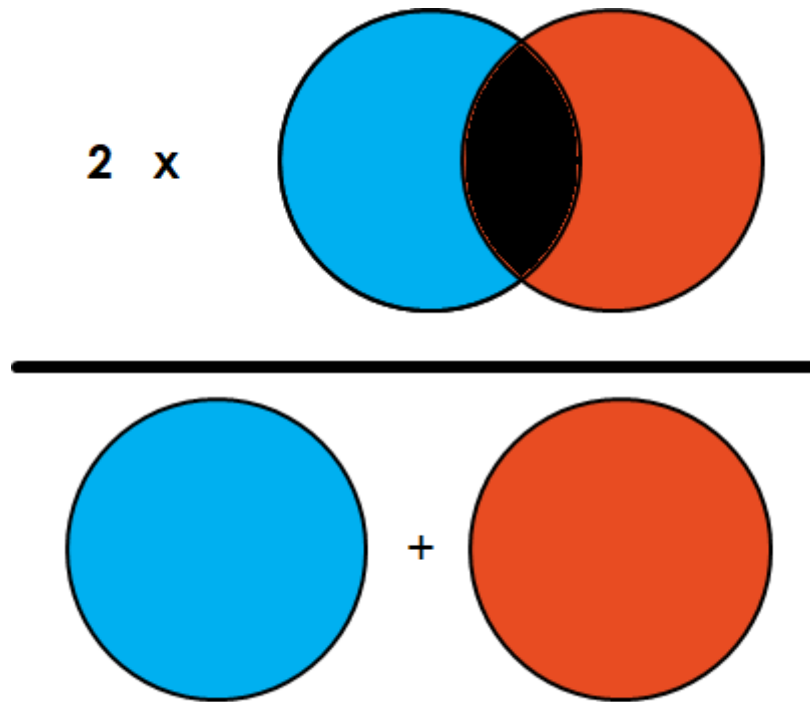


Fig: Dice Coefficient = $(2 \times \text{Area of Overlap}) / (\text{total pixels combined})$

Reference:

<https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2>
<https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2>

- **Loss Metric:**

Whenever we solve a machine learning problem or a deep learning problem, we need a nice trustworthy loss function to check whether our model is performing better or not. Loss functions are basically chosen on the basis of type of dataset and the problem we solve. Our problem is Image Segmentation, for evaluation of Segmentation Models researchers have found that Binary Cross Entropy and the Dice Loss is the best combinational loss function we can use. This combo loss is very helpful for problems with imbalanced datasets. So I will be using this combo loss as loss metric in the whole case study. Combined the two methods allows for some diversity in the loss, while benefitting from the stability of BCE.

$$DiceLoss = 1 - \frac{1}{n} \left(\frac{2 \cdot \sum y_t \cdot y_p}{\sum y_t + \sum y_p} \right)$$

$$BinaryCrossEntropy = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log p(y_i) + (1 - y_i) \cdot (\log(1 - p(y_i)))$$

$$ComboLoss = DiceLoss + BinaryCrossEntropy$$

Reference:

<https://www.kaggle.com/bigironsphere/loss-function-library-keras-pytorch#BCE-Dice-Loss>
(<https://www.kaggle.com/bigironsphere/loss-function-library-keras-pytorch#BCE-Dice-Loss>)

3. Library Imports

In [1]:

```
!pip install pydicom
```

Collecting pydicom

Downloading <https://files.pythonhosted.org/packages/72/7b/6ed88f82dd33a32cdb43432dab7f84fcd40c49d63251442b3cfe0be983d4/pydicom-2.1.1-py3-none-any.whl>
(<https://files.pythonhosted.org/packages/72/7b/6ed88f82dd33a32cdb43432dab7f84fcd40c49d63251442b3cfe0be983d4/pydicom-2.1.1-py3-none-any.whl>) (1.9MB)

|██| 1.9MB 8.3MB/s

Installing collected packages: pydicom

Successfully installed pydicom-2.1.1

In [2]:

```
import pandas as pd
import numpy as np
import pydicom
from glob import glob
import glob2
from tqdm import tqdm
import shutil
import os
import pydicom
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
from PIL import Image
import warnings
from joblib import Parallel, delayed
warnings.filterwarnings("ignore")
```


In [3]:

```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT
```

```
--2020-11-16 09:13:28-- https://storage.googleapis.com/kaggle-data-sets/245622/651264/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20201116%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20201116T051841Z&X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=a8af9184db1c33f546b3fcd3f2b3ecdbb0449a472f9cf54e0dc267851cb7aebe8971bfff88c6fc37a343446dceb8cbb2433f7f1400a47a4b1f9d11455353868f11ef3af0d798e2f012b5019ea040972221f49b659bfb7f482676d8ebdb38e1d08ab39feb36f6b38209fcb7027c05e6fde58dfb0bf1a0a415b7e53ca0dd9a6e5b8bcc6925d16c0a0726b4f65c292e0ca10964ecbc13341cae8869d94d00e522e5aad7fbad09e1ba9c756c4b1ac05ed53f31ecadc2d589da2ef3c403ca709d7ba5fdd7705dc626f8434610aa1473529fa59c6efd182a1a1bc1300cd923577c964dfe4623627608479b2d87038e94c34b8c80dde3482fc581c011d10e849a35af (https://storage.googleapis.com/kaggle-data-sets/245622/651264/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20201116%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20201116T051841Z&X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=a8af9184db1c33f546b3fcd3f2b3ecdbb0449a472f9cf54e0dc267851cb7aebe8971bfff88c6fc37a343446dceb8cbb2433f7f1400a47a4b1f9d11455353868f11ef3af0d798e2f012b5019ea040972221f49b659bfb7f482676d8ebdb38e1d08ab39feb36f6b38209fcb7027c05e6fde58dfb0bf1a0a415b7e53ca0dd9a6e5b8bcc6925d16c0a0726b4f65c292e0ca10964ecbc13341cae8869d94d00e522e5aad7fbad09e1ba9c756c4b1ac05ed53f31ecadc2d589da2ef3c403ca709d7ba5fdd7705dc626f8434610aa1473529fa59c6efd182a1a1bc1300cd923577c964dfe4623627608479b2d87038e94c34b8c80dde3482fc581c011d10e849a35af)
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.142.128, 74.125.20.128, 74.125.195.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.142.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2059765561 (1.9G) [application/zip]
Saving to: 'archive.zip'
```

```
archive.zip          100%[=====>]    1.92G   84.3MB/s   in 26s
```

```
2020-11-16 09:13:54 (76.6 MB/s) - 'archive.zip' saved [2059765561/2059765561]
```

In [4]:

```
!unzip -qq 'archive.zip'
print("Done Extracting 'archive.zip'")
```

4. Exploratory Data Analysis

4.1 Data Loading and EDA

Let's analyze training data provided to us,

In [5]:

```
train_data = pd.read_csv('siim/train-rle.csv', delimiter=',')
train_data.head()
```

Out[5]:

	ImageId	EncodedPixels
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	557374 2 1015 8 1009 14 1002 20 997 26 990 32 ...
2	1.2.276.0.7230010.3.1.4.8323329.11028.15178752...	-1
3	1.2.276.0.7230010.3.1.4.8323329.10366.15178752...	514175 10 1008 29 994 30 993 32 991 33 990 34 ...
4	1.2.276.0.7230010.3.1.4.8323329.10016.15178752...	592184 33 976 58 956 73 941 88 926 102 917 109...

In [6]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12954 entries, 0 to 12953
Data columns (total 2 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   ImageId         12954 non-null  object
1   EncodedPixels   12954 non-null  object
dtypes: object(2)
memory usage: 202.5+ KB
```

- In above information I can see we have total 12954 X-ray files as training data
- We have two columns in the dataset:
 - ImageId = ID of X-rays for each patient checked
 - EncodedPixels = Run Length Encoded Pixel data for each X-ray image

Let's check if there are any duplicate ImageId in the dataset.

In [7]:

```
# add column if the file is duplicate or not
train_data['isDuplicate'] = train_data['ImageId'].duplicated()
train_data.head()
```

Out[7]:

	ImageId	EncodedPixels	isDuplicate
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1	False
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	557374 2 1015 8 1009 14 1002 20 997 26 990 32 ...	False
2	1.2.276.0.7230010.3.1.4.8323329.11028.15178752...	-1	False
3	1.2.276.0.7230010.3.1.4.8323329.10366.15178752...	514175 10 1008 29 994 30 993 32 991 33 990 34 ...	False
4	1.2.276.0.7230010.3.1.4.8323329.10016.15178752...	592184 33 976 58 956 73 941 88 926 102 917 109...	False

In [8]:

```
# check where the files are duplicate
dupImages = train_data.index[train_data['isDuplicate']==True]
print(f"We have total {len(dupImages)} duplicate image ids")
```

We have total 907 duplicate image ids

We should always drop duplicate files,

In [9]:

```
print(f"With duplicates we have total {len(train_data)} files.")
train_data = train_data.drop(list(dupImages))
print(f"Without duplicates we have total {len(train_data)} files.")
```

With duplicates we have total 12954 files.
Without duplicates we have total 12047 files.

Now I have dropped the duplicate ImageIds now I have to add path for each Image Id for further processing of X-ray images

In [10]:

```
train_data = train_data.drop('isDuplicate', axis=1)
train_data['ImagePath'] = 'siim/train_dicom/' + train_data['ImageId'] + '.dcm'
# save the .csv file for further use
train_data.to_csv('train_images_dicom.csv', index=False)
train_data.head()
```

Out[10]:

	ImageId	EncodedPixels
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1 siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.6904.151787520.dcm
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	557374 2 1015 8 1009 14 1002 20 997 26 990 32 ... siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.13666.151787520.dcm
2	1.2.276.0.7230010.3.1.4.8323329.11028.15178752...	-1 siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.11028.151787520.dcm
3	1.2.276.0.7230010.3.1.4.8323329.10366.15178752...	514175 10 1008 29 994 30 993 32 991 33 990 34 ... siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.10366.151787520.dcm
4	1.2.276.0.7230010.3.1.4.8323329.10016.15178752...	592184 33 976 58 956 73 941 88 926 102 917 109... siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.10016.151787520.dcm

In [11]:

```
print(f"Now we have total {train_data.shape[0]} unique X-ray images in the Train Dataset.")
```

Now we have total 12047 unique X-ray images in the Train Dataset.

Let's analyze testing data provided to us,

In [12]:

```
test_data = pd.read_csv('siim/stage_2_sample_submission.csv', delimiter=',')
test_data = test_data.drop('EncodedPixels', axis=1)
test_data['ImagePath'] = 'siim/test_dicom/' + test_data['ImageId'] + '.dcm'
# save the .csv file for further use
test_data.to_csv('test_images_dicom.csv', index=False)
test_data.head()
```

Out[12]:

	ImageId	ImagePath
0	ID_c68e114ba	siim/test_dicom/ID_c68e114ba.dcm
1	ID_b5a797789	siim/test_dicom/ID_b5a797789.dcm
2	ID_490a04f54	siim/test_dicom/ID_490a04f54.dcm
3	ID_823ca20e1	siim/test_dicom/ID_823ca20e1.dcm
4	ID_5face2763	siim/test_dicom/ID_5face2763.dcm

In [13]:

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3205 entries, 0 to 3204
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  -
0   ImageId     3205 non-null   object
1   ImagePath   3205 non-null   object
dtypes: object(2)
memory usage: 50.2+ KB
```

- In above information I can see we have total 3205 X-ray files as testing data
- We have one column in the dataset:
 - ImageId = ID of X-rays for each patient checked

Now I have the final CSV file as obtained above, so the next step is to analyze the actual '.dcm' files

First I will move all the files to a single directory for both train and test data, because we are given separate folder for each file,

Note: Run the below cell only once.

In [14]:

```
# Create Directories to move files
destination_1 = 'siim/train_dicom'
destination_2 = 'siim/test_dicom'

if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)

def move_files(source, destination):
    """
    This function takes source and destination paths as input
    and moves files from source to the destination folder.
    """
    print(source)
    for filename in tqdm(glob2.glob(source)):
        shutil.move(str(filename), destination)

train_path = 'siim/dicom-images-train/**/*.dcm'
test_path = 'siim/dicom-images-test/**/*.dcm'

move_files(train_path, destination_1)
move_files(test_path, destination_2)
```

siim/dicom-images-train/**/*.dcm

100%|██████████| 12089/12089 [00:00<00:00, 22806.62it/s]

100%|██████████| 3205/3205 [00:00<00:00, 28081.95it/s]

siim/dicom-images-test/**/*.dcm

In [15]:

```
print('Sample train:\n', os.listdir('siim/train_dicom')[:2])
print('\nSample test:\n', os.listdir('siim/test_dicom')[:2])
```

Sample train:

```
['1.2.276.0.7230010.3.1.4.8323329.5022.1517875185.999085.dcm', '1.2.276.0.7
230010.3.1.4.8323329.10658.1517875224.950294.dcm']
```

Sample test:

```
['ID_69c158b1f.dcm', 'ID_1086d1e68.dcm']
```

Above I have printed few file names from each train and test set, Here we have files which are having **.dcm** format. This format is commonly used in medical imaging field. This format is known as **DICOM(Digital Imaging and Communications in Medicine)**. Now, almost all forms of medical imaging have become digitized and the spectrum of radiology includes not just digital radiographs but also CT scans, MRIs, ultrasound, and nuclear imaging. DICOM is the file format used for storing the images which can be X-ray scan, an CT-scan, etc. along with the metadata.

Let's start analyzing DICOM files, we have a great library in python to work around DICOM files i.e. **'pydicom'**

In [16]:

```
train_data = pd.read_csv('train_images_dicom.csv')
train_data.head(2)
```

Out[16]:

	ImageId	EncodedPixels
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1 siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.6904.151787520... 557374 2 1015
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	8 1009 14 1002 siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.13666.15178752... 20 997 26 990 32 ...

In [17]:

```
test_data = pd.read_csv('test_images_dicom.csv')
test_data.head(2)
```

Out[17]:

	ImageId	ImagePath
0	ID_c68e114ba	siim/test_dicom/ID_c68e114ba.dcm
1	ID_b5a797789	siim/test_dicom/ID_b5a797789.dcm

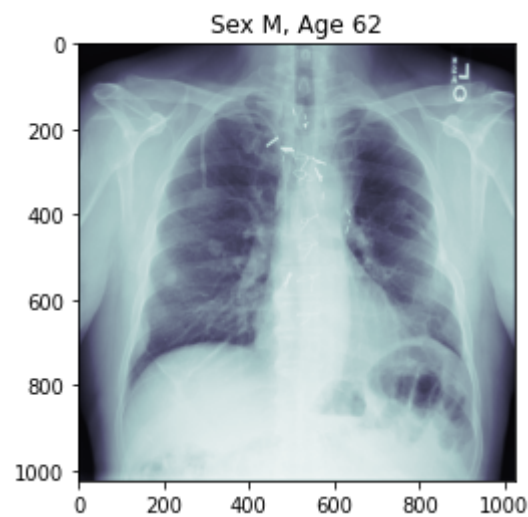
Below is the sample X-ray image:

In [18]:

```
#displaying the X-ray image
sample_img = pydicom.read_file(train_data['ImagePath'][0])
fig, ax = plt.subplots(1, 1)
ax.set_title("Sex {}, Age {}".format(sample_img.PatientSex, sample_img.PatientAge))
ax.imshow(sample_img.pixel_array, cmap='bone')
```

Out[18]:

<matplotlib.image.AxesImage at 0x7f618c5312b0>



In [19]:

```
#displaying metadata
```

```
sample_data = pydicom.dcmread(train_data['ImagePath'])[0]
```

```
print(sample_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: 1.2.276.0.7230010.3.1.4.8323329.6904.1517875201.850819
(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: 1.2.276.0.7230010.3.1.4.8323329.6904.1517875201.850819
(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: '16d7f894-55d7-4d95-8957-d18987f0e981'
(0010, 0020) Patient ID	LO: '16d7f894-55d7-4d95-8957-d18987f0e981'
(0010, 0030) Patient's Birth Date	DA: ''
(0010, 0040) Patient's Sex	CS: 'M'
(0010, 1010) Patient's Age	AS: '62'
(0018, 0015) Body Part Examined	CS: 'CHEST'
(0018, 5101) View Position	CS: 'PA'
(0020, 000d) Study Instance UID	UI: 1.2.276.0.7230010.3.1.2.8323329.6904.1517875201.850818
(0020, 000e) Series Instance UID	UI: 1.2.276.0.7230010.3.1.3.8323329.6904.1517875201.850817
(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: [0.168, 0.168]
(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 126284 element

s

Above I have printed a sample X-ray DICOM file and it's corresponding Metadata. We can see the meta data is too large, it is having a lot of information. Not all information is useful for us so we will analyse some of the important data from the file like patient age, sex, etc.

One thing to notice in the above data is: 'Media Storage SOP Instance UID' is our 'ImageId' provided to us. Now let's create a DataFrame with some of the metadata for analysis purpose.

In [20]:

```
gender = []
age     = []

for fname in tqdm(train_data['ImagePath']):
    dcm = pydicom.dcmread(fname)
    gender.append(dcm.PatientSex)
    age.append(dcm.PatientAge)

meta_data = pd.DataFrame()
meta_data['ImageId'] = train_data['ImageId'].copy()
meta_data['Gender'] = gender
meta_data['Age'] = age
meta_data.head()
```

100%|██████████| 12047/12047 [00:10<00:00, 1141.86it/s]

Out[20]:

	ImageId	Gender	Age
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	M	62
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	M	34
2	1.2.276.0.7230010.3.1.4.8323329.11028.15178752...	F	52
3	1.2.276.0.7230010.3.1.4.8323329.10366.15178752...	F	30
4	1.2.276.0.7230010.3.1.4.8323329.10016.15178752...	F	57

In [21]:

```
# Lets add column for Pneumothorax present or not column
meta_data['Affection'] = train_data['EncodedPixels'].apply(lambda x: 'No' if x=='-1' else 'Yes')
# save CSV for future use
meta_data.to_csv('train_meta_data.csv', index=False)
print("Saved 'train_meta_data.csv'")
meta_data.head()
```

Saved 'train_meta_data.csv'

Out[21]:

	ImageId	Gender	Age	Affection
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	M	62	No
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	M	34	Yes
2	1.2.276.0.7230010.3.1.4.8323329.11028.15178752...	F	52	No
3	1.2.276.0.7230010.3.1.4.8323329.10366.15178752...	F	30	Yes
4	1.2.276.0.7230010.3.1.4.8323329.10016.15178752...	F	57	Yes

In [22]:

```
# Load the saved 'train_meta_data.csv' file
meta_data = pd.read_csv('train_meta_data.csv')
```

I have the required details for analysis as obtained above, now I will start with the simple analysis

4.2 Analysis of Metadata

In [23]:

```
# Check the total no. of males and females in the dataset
mens = len(meta_data[meta_data["Gender"] == "M"])
women = meta_data.shape[0] - mens
print(f"We have total {mens} Males, and total {women} Females in the DataSet.")
```

We have total 6626 Males, and total 5421 Females in the DataSet.

In [24]:

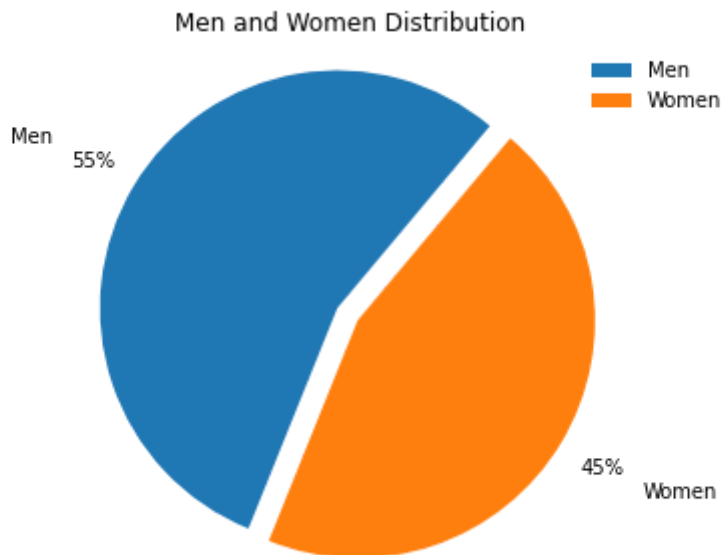
```
#Check the number of pneumothorax affected people and healthy
healthy = len(meta_data[meta_data["Affection"] == "No"])
ill = len(meta_data) - healthy
print(f"We have total {healthy} healthy patients, and {ill} pneumothorax affected patients")
```

We have total 9378 healthy patients, and 2669 pneumothorax affected patients

Let's visualize the above found information in the form of Pie charts,

In [25]:

```
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
labels = ['Men', 'Women']
percentages = [mens, women]
explode=(0.1,0)
ax.pie(percentages, explode=explode, labels=labels, autopct='%1.0f%%',
        shadow=False, startangle=50, pctdistance=1.2, labeldistance=1.4)
ax.axis('equal')
ax.set_title("Men and Women Distribution")
ax.legend(frameon=False, bbox_to_anchor=(0.85,1))
plt.show()
```

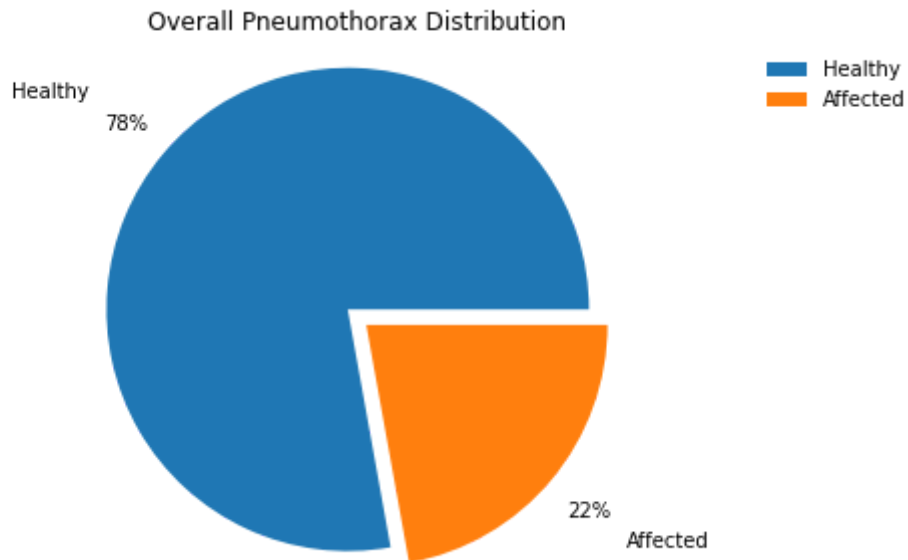


- From above Pie chart I can see 55% of the total patients are Males and 45% are Females.

In [26]:

```
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
labels = ['Healthy', 'Affected']
percentages = [healthy, ill]
explode=(0.1,0)
ax.pie(percentages, explode=explode, labels=labels, autopct='%1.0f%%',
        shadow=False, startangle=0, pctdistance=1.2, labeldistance=1.4)
ax.axis('equal')
ax.set_title("Overall Pneumothorax Distribution")
ax.legend(frameon=False, bbox_to_anchor=(0.85,1))

plt.show()
```



- This Pie charts shows that the 78% patients are Safe, they do not have pneumothorax and only 22% of the people are affected by Pneumothorax.
- The data is imbalanced.

In [27]:

```
#gender
mens = len(meta_data[meta_data["Gender"] == "M"])
women = meta_data.shape[0] - mens
print(f"Total Men      : {mens}")
print("--"*10)
print(f"Total Women    : {women}")
print("--"*10)

#gender + illness
healthy_men = len(meta_data[(meta_data["Gender"] == "M") & (meta_data["Affection"] == "No")])
ill_men = mens - healthy_men
healthy_women = len(meta_data[(meta_data["Gender"] == "F") & (meta_data["Affection"] == "No")])
ill_women = women - healthy_women
print(f"Healthy Men     : {healthy_men}")
print("--"*10)
print(f"Affected Men     : {ill_men}")
print("--"*10)
print(f"Healthy Women    : {healthy_women}")
print("--"*10)
print(f"Affected Women: {ill_women}")
print("--"*10)
```

Total Men : 6626

Total Women : 5421

Healthy Men : 5137

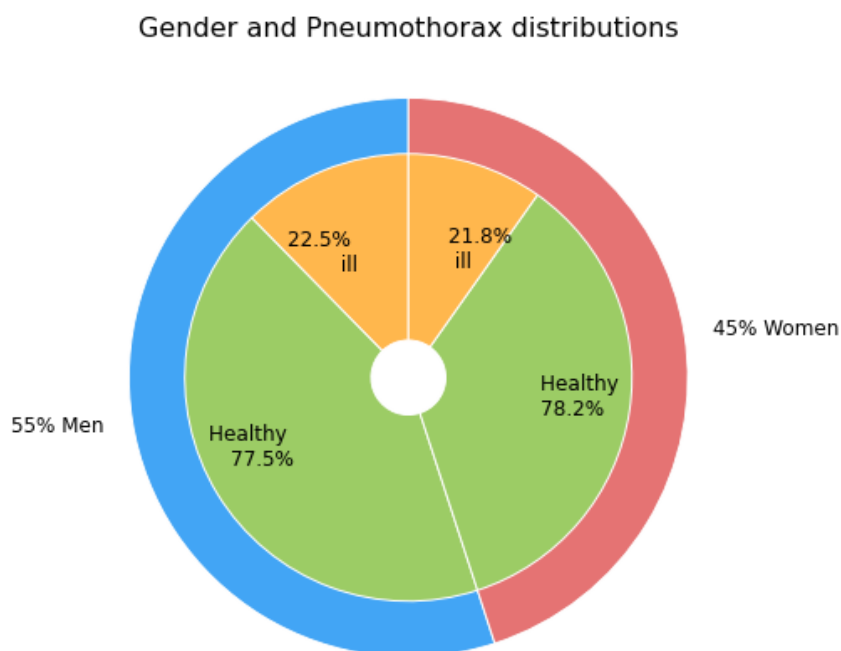
Affected Men : 1489

Healthy Women : 4241

Affected Women: 1180

In [28]:

```
perc = [str(round(ill_men/mens, 3)*100) + "% \n ill", "Healthy \n" +  
        str(round(healthy_men/mens, 3)*100) + "%", "Healthy \n" +  
        str(round(healthy_women/women, 3)*100) + "%",  
        str(round(ill_women/women, 3)*100) + "% \n ill"]  
  
fig, ax = plt.subplots(1, 1, figsize=(10, 5))  
  
fig.suptitle("Gender and Pneumothorax distributions", fontsize=16, y=1.1)  
  
plt.rcParams['font.size'] = 12.0  
  
ax.axis('equal')  
  
#gender healthy  
mypie, _ = ax.pie([mens, women], radius=1.5, labels=["55% Men", "45% Women"],  
                  colors=["#42A5F5", "#E57373"], startangle=90)  
plt.setp( mypie, width=0.5, edgecolor='white')  
  
mypie2, _ = ax.pie([ ill_men, healthy_men, healthy_women, ill_women], radius = 1.5 - 0.3,  
                  labels=perc, labeldistance=0.6,  
                  colors = ["#FFB74D", "#9CCC65", "#9CCC65", "#FFB74D"], startangle=90)  
plt.setp( mypie2, width=1, edgecolor='white')  
plt.margins(0,0)  
  
plt.tight_layout()  
plt.show()
```



- So above is the Pie chart for distribution of pneumothorax with respect to patients Gender, We can see the distribution for Males and Females is nearly same.
- 77.5% of males are healthy and 22.5% males are affected by Pneumothorax.
- 78.2% females are healthy and 21.8% females are affected by Pneumothorax.

It is evident from above chart that the percentage of affected Males and Females is nearly same, So now just plot age histogram for checking age wise distribution of patients

In [29]:

```
#calculating all and ill men and women histograms
bins = [i for i in range(100)]
plt.style.use('seaborn-whitegrid')

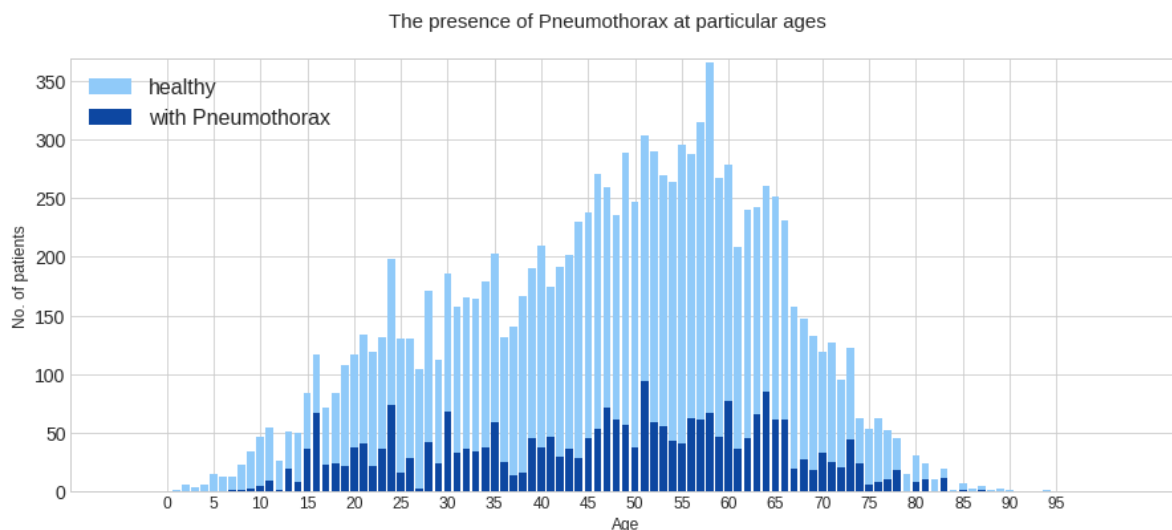
all_patients = np.histogram(meta_data["Age"].values, bins=bins)[0]

ill_patients = np.histogram(meta_data[(meta_data["Affection"] == 'Yes')]["Age"].values, bins=bins)[0]

fig, axes = plt.subplots(figsize=(15, 6))

fig.suptitle("The presence of Pneumothorax at particular ages", fontsize=15, y=0.96)

axes.margins(x=0.1, y=0.01)
m1 = axes.bar(bins[:-1], all_patients, color='#90CAF9')
m2 = axes.bar(bins[:-1], ill_patients, color='#0D47A1')
axes.set(xticks=[i*5 for i in range(20)])
axes.tick_params(axis="y", labelsize=14)
axes.legend((m1[0], m2[0]), ('healthy', 'with Pneumothorax'), loc=2, prop={'size': 16})
plt.xlabel('Age')
plt.ylabel('No. of patients')
plt.show()
```



Observations:

- First of all the overall distribution of age looks almost normally distributed but not exactly.
- 0-6 years babies are not affected by Pnuemothorax in this dataset.
- From the age 7-85 years there is at least one patient which is affected.
- Most of the affected patients are of age 51 years.
- But we cannot say particular age group is Affected because we have much variance i.e. patients of almost all ages above 6 years are affected.

5. Preprocessing

5.1 PNG Conversion

We have the files in the form of **.dcm** files, we cannot use them directly for training the model. So we have to convert them into **.png** format. Also I have to create masks for respective images which also be in **.png** format.

So let's start,

In [30]:

```
train_data = pd.read_csv('train_images_dicom.csv')
train_data.head(2)
```

Out[30]:

	ImageId	EncodedPixels
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1 siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.6904.151787520... 557374 2 1015
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	8 1009 14 1002 siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.13666.15178752... 20 997 26 990 32 ...

In [31]:

```
test_data = pd.read_csv('test_images_dicom.csv')
test_data.head(2)
```

Out[31]:

	ImageId	ImagePath
0	ID_c68e114ba	siim/test_dicom/ID_c68e114ba.dcm
1	ID_b5a797789	siim/test_dicom/ID_b5a797789.dcm

Note: Below 4 cells to be executed only when needed .png conversion

In [32]:

```
from skimage import exposure
def convert_to_png(filename):
    """
    filename : filename with extension '.dcm' with it's full path
    'This function creates png images from the dicom files'
    """
    # read dicom file
    ds = pydicom.read_file(str(filename))
    # convert dicom image to array
    img = ds.pixel_array
    # resize the image for fast computation
    img = cv2.resize(img, (256, 256))
    # create new file name
    fname = filename.replace(".dcm", ".png")
    fname = fname.replace("_dicom", "_png")
    # save the png image to disk
    cv2.imwrite(fname, img)
```

In [33]:

```
# Create Directories for png files
destination_1 = 'siim/train_png'
destination_2 = 'siim/test_png'

if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)
```

In [34]:

```
train_conversion = Parallel(n_jobs=-1, backend='threading')(delayed(
    convert_to_png)(file) for file in tqdm(train_data['ImagePath'],
    total=len(train_data['ImagePath'])))
```

100%|██████████| 12047/12047 [01:32<00:00, 130.49it/s]

In [35]:

```
test_conversion = Parallel(n_jobs=-1, backend='threading')(delayed(
    convert_to_png)(file) for file in tqdm(test_data['ImagePath'],
    total=len(test_data['ImagePath'])))
```

100%|██████████| 3205/3205 [00:24<00:00, 129.83it/s]

In [36]:

```
print(os.listdir('siim/train_png/')[0])
print(os.listdir('siim/test_png/')[0])
```

1.2.276.0.7230010.3.1.4.8323329.10185.1517875222.523417.png
ID_1c982f819.png

5.2 Mask Creation

We have data with masks are Run Length encoded, We have to understand what is this actually so the above video gives a clear idea about RLE. Run-length encoding (RLE) is a form of lossless data compression in which runs of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs. Consider, for example, simple graphic images such as icons, line drawings, Conway's Game of Life, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size.



Image Credit: <https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/31123/versions/3/screenshot.jpg>
(<https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/31123/versions/3/screenshot.jpg>)

E.g.

Input : aaaabbbccc

RLE : a4b3c3

Now I have all the files in **.png** format, Next step is to create ground truth masks for each image in the train dataset. We have mask data in the form of Run Length Encoded Pixels so we have to convert them into **.png** image. Organizers have provided functions for creating masks using RLE pixels and vice-versa.

In [37]:

```
train_data = pd.read_csv('train_images_dicom.csv')
train_data.head(2)
```

Out[37]:

	ImageId	EncodedPixels
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1 siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.6904.151787520... 557374 2 1015
1	1.2.276.0.7230010.3.1.4.8323329.13666.15178752...	8 1009 14 1002 siim/train_dicom/1.2.276.0.7230010.3.1.4.8323329.13666.15178752... 20 997 26 990 32 ...

In [38]:

```
def rle2mask(rle, width, height):
    """
    RLE to mask conversion provided by competition organizers with the dataset.
    """
    mask= np.zeros(width* height)
    array = np.asarray([int(x) for x in rle.split()])
    starts = array[0::2]
    lengths = array[1::2]

    current_position = 0
    for index, start in enumerate(starts):
        current_position += start
        mask[current_position:current_position+lengths[index]] = 1
        current_position += lengths[index]

    return mask.reshape(width, height)
```

Below I am creating and visualizing sample image with it's mask

In [39]:

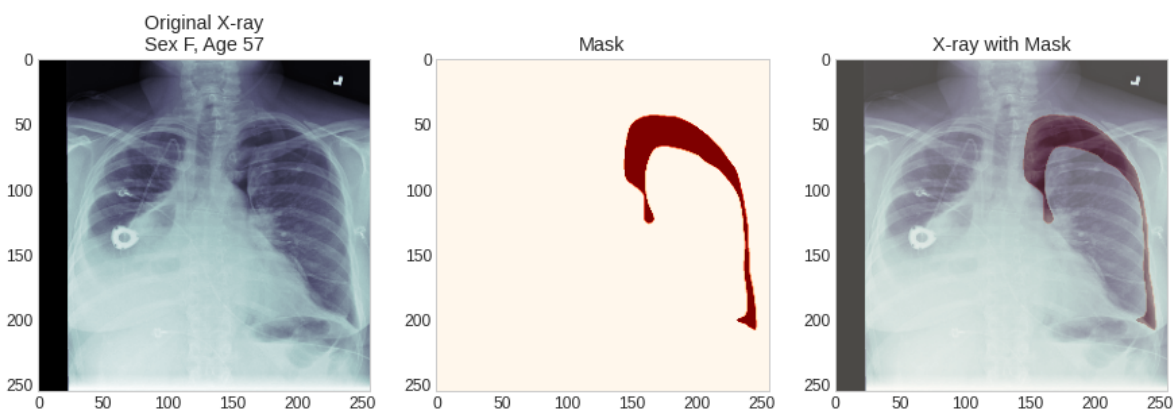
```
# read the original file
ds = pydicom.read_file('siim/train_dicom/' + train_data['ImageId'].iloc[4] + '.dcm')
img = ds.pixel_array
# resize the image for fast computation
img = cv2.resize(img, (256, 256))

# create mask
sam_mask = rle2mask(train_data['EncodedPixels'][4], 1024, 1024).T
sam_mask = cv2.resize(sam_mask, (256, 256))

# plot Original X-ray
fig = plt.figure(figsize=(15, 10))
a = fig.add_subplot(1, 3, 1)
plt.imshow(img, cmap='bone')
a.set_title(f"Original X-ray\nSex {ds.PatientSex}, Age {ds.PatientAge}")
plt.grid(False)
# plt.axis("off")

# plotting mask only
a = fig.add_subplot(1, 3, 2)
plt.imshow(sam_mask, cmap='OrRd', alpha=1)
a.set_title("Mask")
plt.grid(False)
# plt.axis("off")

# plotting X-ray with it's mask
a = fig.add_subplot(1, 3, 3)
plt.imshow(img, cmap='bone')
plt.imshow(sam_mask, cmap='OrRd', alpha=0.3)
a.set_title("X-ray with Mask")
plt.grid(False)
# plt.axis("off")
```



- Above is a sample X-ray and it's corresponding mask
- In the original X-ray it is difficult to recognize if there is pneumothorax or not.
- But the ground truths are provided to us and I have printed mask image for the X-ray which I have shown in Red colored patch, this patch is the presence of Pneumothorax.
- The third image is X-ray with it's mask, I can see the exact location of the mask on the X-ray.

Just like above I will now create masks for each image file in the dataset

In [40]:

```
# Create Directories for mask png files
destination_1 = 'siim/train_mask_png'
destination_2 = 'siim/test_mask_png'

if not os.path.isdir(destination_1):
    os.makedirs(destination_1)
if not os.path.isdir(destination_2):
    os.makedirs(destination_2)
```

In [41]:

```
def get_masks(data, destination):
    """
    data : DataFrame with Columns 'ImageId' and 'EncodedPixels'
    destination: Path for saving masks
    """
    print("Creating masks...")
    # for each image in the data
    for Id,pix in tqdm(data.values):
        # create filename for mask image
        fname = f"{destination}{Id}_mask.png"
        # check if the encoding present
        # if present then decode the mask using rle2mask(rle, width, height) function
        if pix != "-1":
            mask = rle2mask(pix, 1024, 1024).T
            # resize for fast computation
            mask = cv2.resize(mask, (256, 256))
            cv2.imwrite(fname, mask)
        else:
            mask = np.zeros((256, 256), dtype=np.uint8)
            cv2.imwrite(fname, mask)
    print("\nDone!")

# path for saving mask images
train_mask_path = 'siim/train_mask_png/'
# call the function for creating masks
get_masks(train_data[['ImageId', 'EncodedPixels']], train_mask_path)
```

0%| | 44/12047 [00:00<00:28, 419.72it/s]

Creating masks...

100%|██████████| 12047/12047 [00:19<00:00, 624.74it/s]

Done!

Now I am creating a DataFrame with Actual Images and their corresponding Masks

In [42]:

```
final_data = pd.read_csv('train_images_dicom.csv')
final_data = final_data.drop(['ImageId', 'EncodedPixels'], axis=1)
final_data['ImagePath'] = 'siim/train_png/' + train_data['ImageId'] + '.png'
final_data['MaskPath'] = 'siim/train_mask_png/' + train_data['ImageId'] + '_mask.png'
# save file for future use
final_data.to_csv("final_data.csv", index=False)
print("Saved 'final_data.csv'")
final_data.head()
```

Saved 'final_data.csv'

Out[42]:

	ImagePath	MaskPath
0	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
1	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
2	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
3	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
4	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...

In [43]:

```
test_data = pd.read_csv('test_images_dicom.csv')
test_data.head()
```

Out[43]:

	ImageId	ImagePath
0	ID_c68e114ba	siim/test_dicom/ID_c68e114ba.dcm
1	ID_b5a797789	siim/test_dicom/ID_b5a797789.dcm
2	ID_490a04f54	siim/test_dicom/ID_490a04f54.dcm
3	ID_823ca20e1	siim/test_dicom/ID_823ca20e1.dcm
4	ID_5face2763	siim/test_dicom/ID_5face2763.dcm

In [44]:

```
final_test_data = pd.read_csv('test_images_dicom.csv')
final_test_data = final_test_data.drop(['ImageId'], axis=1)
final_test_data['ImagePath'] = 'siim/test_png/' + test_data['ImageId'] + '.png'
# final_test_data['MaskPath'] = 'siim/test_mask_png/' + train_data['ImageId'] + '_mask.png'
# save file for future use
final_test_data.to_csv("final_test_data.csv", index=False)
print("Saved 'final_test_data.csv'")
final_test_data.head()
```

Saved 'final_test_data.csv'

Out[44]:

	ImagePath
0	siim/test_png/ID_c68e114ba.png
1	siim/test_png/ID_b5a797789.png
2	siim/test_png/ID_490a04f54.png
3	siim/test_png/ID_823ca20e1.png
4	siim/test_png/ID_5face2763.png

6. Models

In [46]:

```
final_data = pd.read_csv('final_data.csv')
final_data.head()
```

Out[46]:

	ImagePath	MaskPath
0	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
1	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
2	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
3	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...
4	siim/train_png/1.2.276.0.7230010.3.1.4.8323329...	siim/train_mask_png/1.2.276.0.7230010.3.1.4.83...

In [47]:

```
import tensorflow as tf
import keras
# tf.enable_eager_execution()
import os
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
# from hilbert import hilbertCurve
import imgaug.augmenters as iaa
import numpy as np
import random as rn
```

In [48]:

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler
```

In [49]:

```
class Dataset:

    def __init__(self, images_dir, mask_dir, CLASSES):
        # path to images
        self.images_fps = images_dir.tolist()
        # path to masks
        self.masks_fps = mask_dir.tolist()
        # unique classes (object labels)
        self.class_values = CLASSES

    def __getitem__(self, i):

        # read data
        image = cv2.imread(self.images_fps[i], cv2.IMREAD_UNCHANGED)
        # reshape the image as below
        image = cv2.resize(image, (256, 256))
        image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
        image_mask = cv2.imread(self.masks_fps[i], cv2.IMREAD_UNCHANGED)
        image_mask = cv2.resize(image_mask, (256, 256))
        image_mask = image_mask.reshape(256,256,1)

        # Augmentations
        a = np.random.uniform()
        if a<0.2:
            image = aug2.augment_image(image)
            image_mask = aug2.augment_image(image_mask)
        elif a<0.4:
            image = aug3.augment_image(image)
            image_mask = aug3.augment_image(image_mask)
        elif a<0.6:
            image = aug4.augment_image(image)
            image_mask = aug4.augment_image(image_mask)
        elif a<0.8:
            image = aug5.augment_image(image)
            image_mask = image_mask
        else:
            image = aug6.augment_image(image)
            image_mask = aug6.augment_image(image_mask)

        return image.astype(np.float32), image_mask.astype(np.float32)

    def __len__(self):
        return len(self.images_fps)

class Dataloader(tf.keras.utils.Sequence):

    def __init__(self, dataset, batch_size=1, shuffle=False):
        self.dataset = dataset
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.indexes = np.arange(len(dataset))

    def __getitem__(self, i):

        # collect batch data
        start = i * self.batch_size
        stop = (i + 1) * self.batch_size
        data = []
        for j in range(start, stop):
```

```

        data.append(self.dataset[j])

    batch = [np.stack(samples, axis=0) for samples in zip(*data)]

    return tuple(batch)

def __len__(self):
    return len(self.indexes) // self.batch_size

def on_epoch_end(self):
    if self.shuffle:
        self.indexes = np.random.permutation(self.indexes)

```

In [50]:

```

import imgaug.augmenters as iaa

aug2 = iaa.Fliplr(1)
aug3 = iaa.Flipud(1)
aug4 = iaa.Emboss(alpha=(1), strength=1)
aug5 = iaa.DirectedEdgeDetect(alpha=(0.8), direction=(1.0))
aug6 = iaa.Sharpen(alpha=(1.0), lightness=(1.5))

```

In [51]:

```

from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(final_data, test_size=0.15, random_state=42)

print(X_train.shape)
print(X_test.shape)

```

```

(10239, 2)
(1808, 2)

```

In [52]:

```

# get the unique classes first
CLASSES = [1]
# get the train and test dataset
train_dataset = Dataset(X_train['ImagePath'], X_train['MaskPath'], CLASSES)
test_dataset = Dataset(X_test['ImagePath'], X_test['MaskPath'], CLASSES)

BATCH_SIZE=4

train_dataloader = Dataloader(train_dataset, batch_size=4, shuffle=True)
test_dataloader = Dataloader(test_dataset, batch_size=4, shuffle=True)

print(train_dataloader[0][0].shape)
print(train_dataloader[0][1].shape)
print('***'*5)
print(test_dataloader[0][0].shape)
print(test_dataloader[0][1].shape)
assert train_dataloader[0][0].shape == (BATCH_SIZE, 256, 256, 3)
assert train_dataloader[0][1].shape == (BATCH_SIZE, 256, 256, 1)

```

```

(4, 256, 256, 3)
(4, 256, 256, 1)
*****
(4, 256, 256, 3)
(4, 256, 256, 1)

```

In [53]:

```
def dice_coef(y_true, y_pred, smooth=1.):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (
        K.sum(y_true_f) + K.sum(y_pred_f) + smooth)

def DiceBCELoss(y_true, y_pred):

    #flatten label and prediction tensors
    # inputs = K.flatten(inputs)
    # targets = K.flatten(targets)

    BCE = binary_crossentropy(y_true, y_pred)
    dice_loss = 1 - dice_coef(y_true, y_pred, smooth=1)
    Dice_BCE = BCE + dice_loss

    return Dice_BCE
```

In [54]:

```
from keras.models import Model
from keras.layers import Input
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
from keras import backend as K
```

In [67]:

```
inputs = Input((256, 256, 3))

c1 = Conv2D(8, (3, 3), activation='relu', padding='same') (inputs)
c1 = Conv2D(8, (3, 3), activation='relu', padding='same') (c1)
p1 = MaxPooling2D((2, 2)) (c1)

c2 = Conv2D(16, (3, 3), activation='relu', padding='same') (p1)
c2 = Conv2D(16, (3, 3), activation='relu', padding='same') (c2)
p2 = MaxPooling2D((2, 2)) (c2)

c3 = Conv2D(32, (3, 3), activation='relu', padding='same') (p2)
c3 = Conv2D(32, (3, 3), activation='relu', padding='same') (c3)
p3 = MaxPooling2D((2, 2)) (c3)

c4 = Conv2D(64, (3, 3), activation='relu', padding='same') (p3)
c4 = Conv2D(64, (3, 3), activation='relu', padding='same') (c4)
p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

c5 = Conv2D(64, (3, 3), activation='relu', padding='same') (p4)
c5 = Conv2D(64, (3, 3), activation='relu', padding='same') (c5)
p5 = MaxPooling2D(pool_size=(2, 2)) (c5)

c55 = Conv2D(128, (3, 3), activation='relu', padding='same') (p5)
c55 = Conv2D(128, (3, 3), activation='relu', padding='same') (c55)

u6 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c55)
u6 = concatenate([u6, c5])
c6 = Conv2D(64, (3, 3), activation='relu', padding='same') (u6)
c6 = Conv2D(64, (3, 3), activation='relu', padding='same') (c6)

u71 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c6)
u71 = concatenate([u71, c4])
c71 = Conv2D(32, (3, 3), activation='relu', padding='same') (u71)
c61 = Conv2D(32, (3, 3), activation='relu', padding='same') (c71)

u7 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c61)
u7 = concatenate([u7, c3])
c7 = Conv2D(32, (3, 3), activation='relu', padding='same') (u7)
c7 = Conv2D(32, (3, 3), activation='relu', padding='same') (c7)

u8 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(16, (3, 3), activation='relu', padding='same') (u8)
c8 = Conv2D(16, (3, 3), activation='relu', padding='same') (c8)

u9 = Conv2DTranspose(8, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(8, (3, 3), activation='relu', padding='same') (u9)
c9 = Conv2D(8, (3, 3), activation='relu', padding='same') (c9)

outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)

model = Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss=DiceBCELoss, metrics=[dice_coef])
model.summary()
```

Model: "functional_5"

Layer (type)	Output Shape	Param #	Connected to
=====			
input_3 (InputLayer)	[(None, 256, 256, 3)]	0	
<hr/>			
conv2d_46 (Conv2D)	(None, 256, 256, 8)	224	input_3[0]
<hr/>			
conv2d_47 (Conv2D)	(None, 256, 256, 8)	584	conv2d_46[0]
<hr/>			
max_pooling2d_10 (MaxPooling2D)	(None, 128, 128, 8)	0	conv2d_47[0]
<hr/>			
conv2d_48 (Conv2D)	(None, 128, 128, 16)	1168	max_pooling2d_10[0]
<hr/>			
conv2d_49 (Conv2D)	(None, 128, 128, 16)	2320	conv2d_48[0]
<hr/>			
max_pooling2d_11 (MaxPooling2D)	(None, 64, 64, 16)	0	conv2d_49[0]
<hr/>			
conv2d_50 (Conv2D)	(None, 64, 64, 32)	4640	max_pooling2d_11[0]
<hr/>			
conv2d_51 (Conv2D)	(None, 64, 64, 32)	9248	conv2d_50[0]
<hr/>			
max_pooling2d_12 (MaxPooling2D)	(None, 32, 32, 32)	0	conv2d_51[0]
<hr/>			
conv2d_52 (Conv2D)	(None, 32, 32, 64)	18496	max_pooling2d_12[0]
<hr/>			
conv2d_53 (Conv2D)	(None, 32, 32, 64)	36928	conv2d_52[0]
<hr/>			
max_pooling2d_13 (MaxPooling2D)	(None, 16, 16, 64)	0	conv2d_53[0]
<hr/>			
conv2d_54 (Conv2D)	(None, 16, 16, 64)	36928	max_pooling2d_13[0]
<hr/>			
conv2d_55 (Conv2D)	(None, 16, 16, 64)	36928	conv2d_54[0]

max_pooling2d_14 (MaxPooling2D) [0][0]	(None, 8, 8, 64)	0	conv2d_55
conv2d_56 (Conv2D) 2d_14[0][0]	(None, 8, 8, 128)	73856	max_pooling
conv2d_57 (Conv2D) [0][0]	(None, 8, 8, 128)	147584	conv2d_56
conv2d_transpose_10 (Conv2DTran [0][0]	(None, 16, 16, 64)	32832	conv2d_57
concatenate_10 (Concatenate) spose_10[0][0]	(None, 16, 16, 128)	0	conv2d_tran conv2d_55
conv2d_58 (Conv2D) _10[0][0]	(None, 16, 16, 64)	73792	concatenate
conv2d_59 (Conv2D) [0][0]	(None, 16, 16, 64)	36928	conv2d_58
conv2d_transpose_11 (Conv2DTran [0][0]	(None, 32, 32, 32)	8224	conv2d_59
concatenate_11 (Concatenate) spose_11[0][0]	(None, 32, 32, 96)	0	conv2d_tran conv2d_53
conv2d_60 (Conv2D) _11[0][0]	(None, 32, 32, 32)	27680	concatenate
conv2d_61 (Conv2D) [0][0]	(None, 32, 32, 32)	9248	conv2d_60
conv2d_transpose_12 (Conv2DTran [0][0]	(None, 64, 64, 32)	4128	conv2d_61
concatenate_12 (Concatenate) spose_12[0][0]	(None, 64, 64, 64)	0	conv2d_tran conv2d_51
conv2d_62 (Conv2D)	(None, 64, 64, 32)	18464	concatenate

_12[0][0]

conv2d_63 (Conv2D) [0][0]	(None, 64, 64, 32)	9248	conv2d_62
conv2d_transpose_13 (Conv2DTran [0][0]	(None, 128, 128, 16)	2064	conv2d_63
concatenate_13 (Concatenate) spose_13[0][0]	(None, 128, 128, 32)	0	conv2d_tran conv2d_49
conv2d_64 (Conv2D) _13[0][0]	(None, 128, 128, 16)	4624	concatenate
conv2d_65 (Conv2D) [0][0]	(None, 128, 128, 16)	2320	conv2d_64
conv2d_transpose_14 (Conv2DTran [0][0]	(None, 256, 256, 8)	520	conv2d_65
concatenate_14 (Concatenate) spose_14[0][0]	(None, 256, 256, 16)	0	conv2d_tran conv2d_47
conv2d_66 (Conv2D) _14[0][0]	(None, 256, 256, 8)	1160	concatenate
conv2d_67 (Conv2D) [0][0]	(None, 256, 256, 8)	584	conv2d_66
conv2d_68 (Conv2D) [0][0]	(None, 256, 256, 1)	9	conv2d_67
=====			
Total params: 600,729			
Trainable params: 600,729			
Non-trainable params: 0			

In [68]:

```
import datetime
# tf.keras.backend.set_image_data_format('channels_last')
# Define optimizer
optim = tf.keras.optimizers.Adam(0.0001)
# Clear any Logs from previous runs
!rm -rf /content/logs/model_1/fit
filepath= "best_vanilla_Unet.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_dice_coef', verbose=1, save_be
# Lrschedule = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.00001
# earllystop = EarlyStopping(monitor='val_dice_coef', min_delta=0.25, patience=5, verbose=1)
# Log_dir="/content/logs/vanilla_Unet/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M
# tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1, w
callback_list = [checkpoint]
```

In [69]:

```
from tensorflow.keras.losses import binary_crossentropy
model.compile(optimizer=optim, loss=DiceBCELoss, metrics=[dice_coef])
history = model.fit_generator(train_dataloader,validation_data=test_dataloader,
                             epochs=30, verbose=1, callbacks=callback_list)
```

Epoch 1/30

2557/2559 [=====>.] - ETA: 0s - loss: 1.0277 - dice
_coef: 0.0866

Epoch 00001: val_dice_coef improved from -inf to 0.35669, saving model to
best_vanilla_Unet.hdf5

2559/2559 [=====] - 68s 26ms/step - loss: 1.0278
- dice_coef: 0.0865 - val_loss: 0.8170 - val_dice_coef: 0.3567

Epoch 2/30

2557/2559 [=====>.] - ETA: 0s - loss: 0.7211 - dice
_coef: 0.3567

Epoch 00002: val_dice_coef did not improve from 0.35669

2559/2559 [=====] - 68s 26ms/step - loss: 0.7206
- dice_coef: 0.3572 - val_loss: 0.7023 - val_dice_coef: 0.3550

Epoch 3/30

2557/2559 [=====>.] - ETA: 0s - loss: 0.7007 - dice
_coef: 0.3611

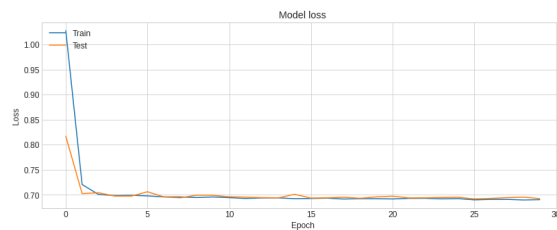
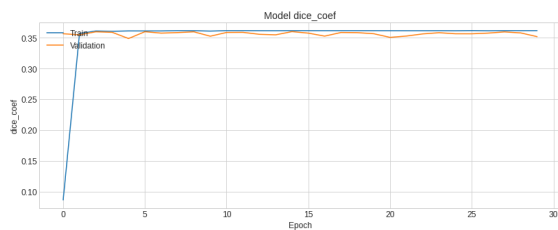
Epoch 00003: val_dice_coef improved from 0.35669 to 0.35985, saving model
to best_vanilla_Unet.hdf5

2559/2559 [=====] - 69s 27ms/step - loss: 0.7006

In [71]:

```
# Plot training & validation iou_score values
plt.figure(figsize=(30, 5))
plt.subplot(121)
plt.plot(history.history['dice_coef'])
plt.plot(history.history['val_dice_coef'])
plt.title('Model dice_coef')
plt.ylabel('dice_coef')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(122)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



- From above graphs I can say that model for train and test sets is working fine but it is not improving score after 0.36014.
- This is may be due to the Unet network architecture is vanilla, and I should try something else than this vanilla structure

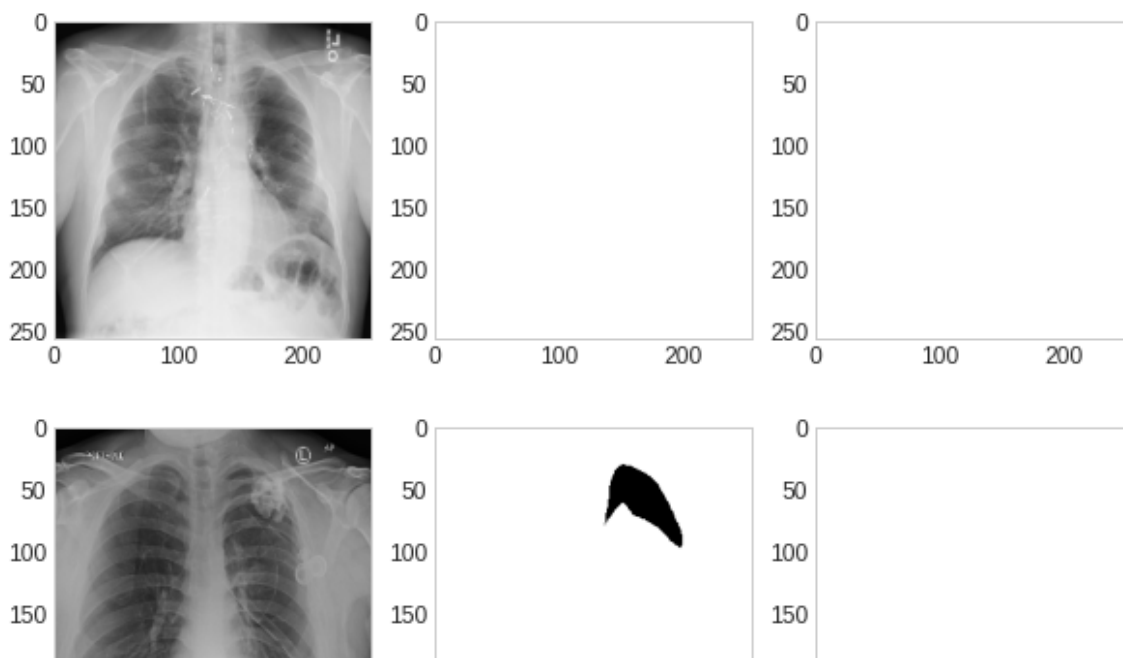
In [73]:

```
for p in range(0,20):
    #original image
    image = cv2.imread(final_data["ImagePath"].tolist()[p], cv2.IMREAD_UNCHANGED)
    image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
    # image = cv2.resize(image, (256,256),interpolation=cv2.INTER_NEAREST)

    #predicted segmentation map
    predicted = model.predict(image[None,:,:,:])
    mask_pred = tf.argmax(predicted,axis=-1)
    #original segmentation map
    image_mask = cv2.imread(final_data["MaskPath"].tolist()[p], cv2.IMREAD_UNCHANGED)
    # image_mask = image_mask.reshape(256,256,1)
    # image_mask = cv2.resize(image_mask, (256,256),interpolation=cv2.INTER_NEAREST)

    plt.figure(figsize=(10,6))
    # plot original image
    plt.subplot(131)
    plt.grid(False)
    # image = cv2.resize(image, (256,256),interpolation=cv2.INTER_NEAREST)
    plt.imshow(image)
    # plot original mask
    plt.subplot(132)
    plt.grid(False)
    plt.imshow(image_mask)
    # plot predicted mask
    plt.subplot(133)
    plt.grid(False)
    plt.imshow(mask_pred[0])

    plt.grid(False)
    plt.show()
```



In []:

