

Library Imports

```
In [2]: !pip install pydicom
```

```
Requirement already satisfied: pydicom in /usr/local/lib/python3.6/dist-packages (2.1.1)
```

```
In [3]: import pandas as pd
import numpy as np
import pydicom
from glob2 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")
```

Download Dataset and Extract

```
In [3]: # https://gist.github.com/jayspeidelL/d10b84b8d3da52df723beacc5b15cb27
!mkdir -p '/root/.kaggle/'
!cp '/content/kaggle.json' '/root/.kaggle/kaggle.json'
# !touch ~/.kaggle/kaggle.json
!chmod 600 ~/.kaggle/kaggle.json
```

```
In [4]: !kaggle datasets download -d seesee/siim-train-test
```

```
Downloading siim-train-test.zip to /content
100% 1.92G/1.92G [00:27<00:00, 71.3MB/s]
100% 1.92G/1.92G [00:27<00:00, 75.9MB/s]
```

```
In [5]: # https://www.kaggle.com/seesee/siim-train-test
# !wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3160.107 Safari/537.36" https://www.kaggle.com/seesee/siim-train-test/siim-train-test.zip
```

```
In [6]: print("Extracting 'siim-train-test.zip'\n")
!unzip -qq '/content/siim-train-test.zip'
print("Done Extracting 'siim-train-test.zip'")
```

```
Extracting 'siim-train-test.zip'
```

```
Done Extracting 'siim-train-test.zip'
```

Move files

```
In [7]: !cp "/content/stage_2_sample_submission.csv" "siim/stage_2_sample_submission.csv"
```

```
In [8]: # 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, 23200.41it/s]
100%|██████████| 3205/3205 [00:00<00:00, 28263.74it/s]

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

Preprocessing

```
In [9]: train_data = pd.read_csv('siim/train-rle.csv', delimiter=',')
train_data.head()
```

Out[9]:

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

Out[10]:

	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 [11]: # 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

```
In [12]: 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.

```
In [13]: 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[13]:

	ImageId	EncodedPixels	ImagePath
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
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/1.2.276.0.7230010.3.1.4.83233...
2	1.2.276.0.7230010.3.1.4.8323329.11028.15178752...		-1 siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
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/1.2.276.0.7230010.3.1.4.83233...
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/1.2.276.0.7230010.3.1.4.83233...

```
In [14]: 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[14]:

	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

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 [15]: train_data = pd.read_csv('train_images_dicom.csv')
train_data.head(2)
```

Out[15]:

	ImageId	EncodedPixels	ImagePath
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
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/1.2.276.0.7230010.3.1.4.83233...

```
In [16]: train_data.shape
```

Out[16]: (12047, 3)

```
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

```
In [18]: test_data.shape
```

Out[18]: (3205, 2)

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

```
In [19]: # 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 [20]: # 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 [21]: 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 [01:35<00:00, 126.25it/s]
```

```
In [22]: 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, 132.37it/s]
```

```
In [23]: print(os.listdir('siim/train_png/')[0])
print(os.listdir('siim/test_png/')[0])

1.2.276.0.7230010.3.1.4.8323329.4380.1517875182.577377.png
ID_5c004b92b.png
```

Mask Creation

```
In [24]: train_data = pd.read_csv('train_images_dicom.csv')
train_data.head(2)
```

	ImageId	EncodedPixels	ImagePath
0	1.2.276.0.7230010.3.1.4.8323329.6904.151787520...	-1	siim/train_dicom/1.2.276.0.7230010.3.1.4.83233...
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/1.2.276.0.7230010.3.1.4.83233...

```
In [25]: 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]] = 255
        current_position += lengths[index]

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

```
In [26]: # 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 [27]: def get_masks(data, destination):
    """
        data : DataFrame with Columns 'ImageId' and 'EncodedPixels'
        destination: Path for saving masks
    """
    print("\nCreating masks...")
    # for each image in the data
    for Id,pix in tqdm(data.values):
        # create filename for mask image
        fname = f"{destination}{Id}.png" #_mask
        # 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% | 0/12047 [00:00<?, ?it/s]

Creating masks...

100% |██████████| 12047/12047 [00:20<00:00, 602.26it/s]

Done!

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

```
In [28]: 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']+'.png' # _mask
# 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[28]:

	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 [29]: test_data = pd.read_csv('test_images_dicom.csv')
test_data.head()
```

Out[29]:

	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 [30]: 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[30]:

	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

Read CSV

```
In [4]: final_data = pd.read_csv('final_data.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()
```

Out[4]:

	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...

TF DataPipeline

```
In [5]: 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
from keras import backend as K
#K.set_image_data_format('channels_first')
```

```
In [6]: if tf.test.is_gpu_available(cuda_only=True):
    print('working on gpu')
else:
    print('warning: working on cpu')
# print(tf.__version__)
# print(keras.__version__)

print("Tensorflow:",tf.__version__)
print("Keras:",keras.__version__)
```

WARNING:tensorflow:From <ipython-input-6-fdbdf1bb57a1>:1: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated and will be removed in a future version.

Instructions for updating:

Use `tf.config.list_physical_devices('GPU')` instead.

working on gpu

Tensorflow: 2.3.0

Keras: 2.4.3

```
In [7]: from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler, ReduceLROnPlateau, Callback
```

```
In [8]: all_mask_fn = glob2.glob('siim/train_mask_png/*')
# all_mask_fn = glob.glob('train/*')

mask_df = pd.DataFrame()
mask_df['file_names'] = all_mask_fn
mask_df['mask_percentage'] = 0
mask_df.set_index('file_names', inplace=True)
for fn in all_mask_fn:
    mask_df.loc[fn, 'mask_percentage'] = np.array(Image.open(fn)).sum()/(256*256*255) #255 is bcz img range is 255

mask_df.reset_index(inplace=True)
mask_df['labels'] = 0
mask_df.loc[mask_df.mask_percentage>0, 'labels'] = 1
sns.distplot(mask_df.mask_percentage.values)
```

```
In [9]: from sklearn.model_selection import train_test_split
X_train, X_val = train_test_split(final_data, stratify=mask_df.labels, test_size=0.15, random_state=42)

print(X_train.shape)
print(X_val.shape)
```

(10239, 2)
(1808, 2)

```
In [10]: # image_shapes = [read_image(x).shape for x in X_train['ImagePath']]
#         # image_mask = read_mask(y)
```

```
In [11]: # image_shapes
```

```
In [12]: # plt.plot(image_shapes)
```

In [13]: # 1. To Load the dataset: image and mask paths
2. Building the TensorFlow Input Data Pipeline using tf.data API

```
import os
import numpy as np
import cv2
from glob2 import glob
import tensorflow as tf
from tensorflow.keras.layers import Concatenate
import imgaug.augmenters as iaa

# create augmentations
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))

def read_image(path):
    """
    This reads image
    -----
    path : image path
    -----
    """
    # image = tf.io.read_file(img_path)
    # image = tf.image.decode_png(image, channels=3)
    # image = tf.image.convert_image_dtype(image, tf.uint8)

    x = tf.io.read_file(path)
    x = tf.image.decode_png(x, channels=3)
    x = tf.image.convert_image_dtype(x, tf.float32)
    # x = x / 255.0
    # x = tf.image.convert_image_dtype(x, tf.float32)

    return x

def read_mask(path):
    """
    This reads mask
    -----
    path : mask path
    -----
    """
    y = tf.io.read_file(path)
    y = tf.image.decode_png(y, channels=1)
    y = tf.image.convert_image_dtype(y, tf.float32)
    # y = y / 255.0
    # x = np.expand_dims(x, axis=-1)
    # y = tf.image.convert_image_dtype(y, tf.float32)
    return y

def preprocess(x, y, augment):
    """
    This function reads images and masks and returns them
    -----
    x : image path
    y : mask path
    -----
    """
    # def f(x, y, augment):
    #     # x = tf.image.decode_png(x, channels=3)
    #     # y = tf.image.decode_png(y, channels=1)

    image = read_image(x)
    image_mask = read_mask(y)
    image = tf.transpose(image, [2,0,1])
    image_mask = tf.transpose(image_mask, [2,0,1])

    # Augmentations
    # a = np.random.uniform()

    # if augment==True:
    #     if a<=0.2 and a>=0.8:
    #         image = image
    #         image_mask = image_mask
    #     elif a>0.2 and a<0.4:
    #         image = aug2.augment_image(image)
    #         image_mask = aug2.augment_image(image_mask)
    #     elif a>0.4 and a<0.6:
    #         image = aug3.augment_image(image)
    #         image_mask = aug3.augment_image(image_mask)
    #     elif a>0.6 and a<0.8:
    #         image = aug4.augment_image(image)
    #         image_mask = aug4.augment_image(image_mask)
    #     elif a>0.8 and a<0.9:
    #         image = aug5.augment_image(image)
    #         image_mask = image_mask
```

```

#     elif a>=0.5 and a<=0.7:
#         image = aug6.augment_image(image)
#         image_mask = aug6.augment_image(image_mask)

# return image, image_mask

# images, masks = tf.numpy_function(f, [x, y, augment], [tf.float32, tf.float32])
# print(type(images))
# images.set_shape([3, 256, 256])
# masks.set_shape([1, 256, 256])
# images = tf.image.convert_image_dtype(images, tf.float32)
# print(masks)
image_mask = Concatenate(axis=0)([image_mask, image_mask])
# masks = tf.image.convert_image_dtype(masks, tf.float32)
# print(masks)
return image, image_mask

def tf_dataset(x, y, batch=8, augment=False):
    """
    This function creates input data pipeline
    -----
    x : all images paths
    y : all masks paths
    batch : batch size
    -----
    """
    augment=augment
    dataset = tf.data.Dataset.from_tensor_slices((x, y))
    dataset = dataset.shuffle(buffer_size=1000)
    dataset = dataset.map(lambda x,y: preprocess(x, y, augment), num_parallel_calls=tf.data.experimental.AUTOTUNE)
    dataset = dataset.batch(batch)
    dataset = dataset.repeat()
    dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)
    return dataset

if __name__ == "__main__":
    tr_images, tr_masks = X_train['ImagePath'], X_train['MaskPath']
    print(f"Images: {len(tr_images)} - Masks: {len(tr_masks)}")
    train_dataset = tf_dataset(tr_images, tr_masks, augment=False, batch=12)

    val_images, val_masks = X_val['ImagePath'], X_val['MaskPath']
    print(f"Images: {len(val_images)} - Masks: {len(val_masks)}")
    val_dataset = tf_dataset(val_images, val_masks, augment=False, batch=12)

    # for x, y in dataset:
    #     x = x[0] * 255
    #     y = y[0] * 255

    #     x = x.numpy()
    #     y = y.numpy()

    #     cv2.imwrite("image.png", x)

    #     y = np.squeeze(y, axis=-1)
    #     cv2.imwrite("mask.png", y)

    #     break

```

Images: 10239 - Masks: 10239
 Images: 1808 - Masks: 1808

In [14]: `print(train_dataset)`
`print(val_dataset)`

<PrefetchDataset shapes: ((None, 3, None, None), (None, 2, None, None)), types: (tf.float32, tf.float32)>
 <PrefetchDataset shapes: ((None, 3, None, None), (None, 2, None, None)), types: (tf.float32, tf.float32)>

```
In [15]: # smooth = 1e-15
# def dice_coef(y_true, y_pred):
#     y_true = tf.keras.layers.Flatten()(y_true)
#     y_pred = tf.keras.layers.Flatten()(y_pred)
#     intersection = tf.reduce_sum(y_true * y_pred)
#     return (2. * intersection + smooth) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) + smooth)

# def dice_loss(y_true, y_pred):
#     return 1.0 - dice_coef(y_true, y_pred)

# def iou(y_true, y_pred):
#     def f(y_true, y_pred):
#         intersection = (y_true * y_pred).sum()
#         union = y_true.sum() + y_pred.sum() - intersection
#         x = (intersection + smooth) / (union + smooth)
#         x = x.astype(np.float32)
#         return x
#     return tf.numpy_function(f, [y_true, y_pred], tf.float32)

# def bce_dice_loss(y_true, y_pred):
#     return binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)
```

Double UNet

```
In [16]: import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model
from tensorflow.keras.applications import *

def squeeze_excite_block(inputs, ratio=8):
    init = inputs
    channel_axis = 1
    filters = init.shape[channel_axis]
    se_shape = (filters, 1, 1)

    se = GlobalAveragePooling2D(data_format='channels_first')(init)
    # se = Reshape(se_shape)(se)
    se = Dense(filters // ratio, activation='relu', kernel_initializer='he_normal', use_bias=False)(se)
    se = Dense(filters, activation='sigmoid', kernel_initializer='he_normal', use_bias=False)(se)
    se = Reshape(se_shape)(se)
    print(se.shape)

    x = Multiply()([init, se])
    return x

def conv_block(inputs, filters):
    x = inputs

    x = Conv2D(filters, (3, 3), padding="same", data_format='channels_first')(x)
    x = BatchNormalization(axis=1)(x)
    x = Activation('relu')(x)

    x = Conv2D(filters, (3, 3), padding="same", data_format='channels_first')(x)
    x = BatchNormalization(axis=1)(x)
    x = Activation('relu')(x)

    x = squeeze_excite_block(x)

    return x

def encoder1(inputs):
    skip_connections = []

    model = VGG19(include_top=False, weights='imagenet', input_tensor=inputs, input_shape=(3, 256, 256))
    # names = ["block1_conv2", "block2_conv2", "block3_conv4", "block4_conv4"]
    names = ["block1_conv2", "block2_conv2"]
    for name in names:
        skip_connections.append(model.get_layer(name).output)

    # output = model.get_layer("block5_conv4").output
    output = model.get_layer("block3_conv4").output
    # print("<<VGG19")
    # for v in range(len(skip_connections)):
    #     print(skip_connections[v].shape)
    # print("VGG19>>")
    return output, skip_connections

def decoder1(inputs, skip_connections):
    # num_filters = [256, 128, 64, 32]
    num_filters = [64, 32]
    skip_connections.reverse()

    x = inputs

    for i, f in enumerate(num_filters):
        # x = UpSampling2D((2, 2), interpolation='bilinear')(x)
        x = UpSampling2D((2, 2), interpolation='bilinear', data_format='channels_first')(x)
        # print(x.shape, skip_connections[i].shape)
        x = Concatenate(axis=1)([x, skip_connections[i]])
        x = conv_block(x, f)

    return x

# def encoder2(inputs):
#     skip_connections = []
#
#     output = DenseNet121(include_top=False, weights='imagenet')(inputs)
#     model = tf.keras.models.Model(inputs, output)
#
#     names = ["input_2", "conv1/relu", "pool2_conv", "pool3_conv"]
#     for name in names:
#         skip_connections.append(model.get_layer(name).output)
#     output = model.get_layer("pool4_conv").output
#
#     return output, skip_connections

def encoder2(inputs):
    # num_filters = [32, 64, 128, 256]
    num_filters = [32, 64]
    skip_connections = []
    x = inputs
```

```

    for i, f in enumerate(num_filters):
        x = conv_block(x, f)
        skip_connections.append(x)
        x = MaxPool2D((2, 2), data_format='channels_first')(x)

    return x, skip_connections

def decoder2(inputs, skip_1, skip_2):
    # num_filters = [256, 128, 64, 32]
    num_filters = [256, 128]
    skip_2.reverse()
    x = inputs

    for i, f in enumerate(num_filters):
        x = UpSampling2D((2, 2), interpolation='bilinear', data_format='channels_first')(x)
        x = Concatenate(axis=1)([x, skip_1[i], skip_2[i]])
        x = conv_block(x, f)

    return x

def output_block(inputs):
    x = Conv2D(1, (1, 1), padding="same", data_format='channels_first')(inputs)
    x = Activation('sigmoid')(x)
    return x

def Upsample(tensor, size):
    """Bilinear upsampling"""
    def _upsample(x, size):
        return tf.image.resize(images=x, size=size)
    return Lambda(lambda x: _upsample(x, size), output_shape=size)(tensor)

def ASPP(x, filter):
    shape = x.shape

    y1 = AveragePooling2D(pool_size=(shape[2], shape[3]), data_format='channels_first')(x)
    y1 = Conv2D(filter, 1, padding="same", data_format='channels_first')(y1)
    y1 = BatchNormalization(axis=1)(y1)
    y1 = Activation("relu")(y1)
    y1 = UpSampling2D((shape[2], shape[3]), interpolation='bilinear', data_format='channels_first')(y1)

    y2 = Conv2D(filter, 1, dilation_rate=1, padding="same", use_bias=False, data_format='channels_first')(x)
    y2 = BatchNormalization(axis=1)(y2)
    y2 = Activation("relu")(y2)

    y3 = Conv2D(filter, 3, dilation_rate=6, padding="same", use_bias=False, data_format='channels_first')(x)
    y3 = BatchNormalization(axis=1)(y3)
    y3 = Activation("relu")(y3)

    y4 = Conv2D(filter, 3, dilation_rate=12, padding="same", use_bias=False, data_format='channels_first')(x)
    y4 = BatchNormalization(axis=1)(y4)
    y4 = Activation("relu")(y4)

    y5 = Conv2D(filter, 3, dilation_rate=18, padding="same", use_bias=False, data_format='channels_first')(x)
    y5 = BatchNormalization(axis=1)(y5)
    y5 = Activation("relu")(y5)

    y = Concatenate(axis=1)([y1, y2, y3, y4, y5])

    y = Conv2D(filter, 1, dilation_rate=1, padding="same", use_bias=False, data_format='channels_first')(y)
    y = BatchNormalization(axis=1)(y)
    y = Activation("relu")(y)

    return y

def build_model(shape):
    inputs = Input(shape)
    x, skip_1 = encoder1(inputs)
    x = ASPP(x, 64)
    x = decoder1(x, skip_1)
    outputs1 = output_block(x)

    x = inputs * outputs1

    x, skip_2 = encoder2(x)
    x = ASPP(x, 64)
    x = decoder2(x, skip_1, skip_2)
    outputs2 = output_block(x)
    outputs = Concatenate(axis=1)([outputs1, outputs2])

    model = Model(inputs, outputs)
    return model

# inp_shape = (256, 256, 3)
# model = build_model(inp_shape)

```

```
In [17]: # K.set_image_data_format('channels_last')
K.set_image_data_format('channels_first')
inp_shape = (3,256,256)
model = build_model(inp_shape)
model.summary()
```

```
(None, 64, 1, 1)
(None, 32, 1, 1)
(None, 32, 1, 1)
(None, 64, 1, 1)
(None, 256, 1, 1)
(None, 128, 1, 1)
Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 3, 256, 256) 0]		
block1_conv1 (Conv2D)	(None, 64, 256, 256) 1792	input_1[0][0]	
block1_conv2 (Conv2D)	(None, 64, 256, 256) 36928	block1_conv1[0][0]	
block1_pool (MaxPooling2D)	(None, 64, 128, 128) 0	block1_conv2[0][0]	
block2_conv1 (Conv2D)	(None, 128, 128, 128) 73856	block1_pool[0][0]	

```
In [18]: model.inputs, model.outputs
```

```
Out[18]: ([<tf.Tensor 'input_1:0' shape=(None, 3, 256, 256) dtype=float32>],
 [<tf.Tensor 'concatenate_6(concat:0' shape=(None, 2, 256, 256) dtype=float32>])
```

Define Loss Function

```
In [ ]: smooth = 1.
def dice_coef(y_true, y_pred):
    y_true = tf.keras.layers.Flatten()(y_true)
    y_pred = tf.keras.layers.Flatten()(y_pred)
    intersection = tf.reduce_sum(y_true * y_pred)
    return (2. * intersection + smooth) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) + smooth)

def dice_loss(y_true, y_pred):
    return 1.0 - dice_coef(y_true, y_pred)

def iou(y_true, y_pred):
    def f(y_true, y_pred):
        intersection = (y_true * y_pred).sum()
        union = y_true.sum() + y_pred.sum() - intersection
        x = (intersection + smooth) / (union + smooth)
        x = x.astype(np.float32)
    return x
    return tf.numpy_function(f, [y_true, y_pred], tf.float32)

def bce_dice_loss(y_true, y_pred):
    return binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)
```

```
In [ ]: import datetime
from tensorflow.keras.callbacks import *
from tensorflow.keras.losses import binary_crossentropy
# 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
# "/content/model_save/model-{epoch:02d}-{val_auc_score:.4f}.hdf5"
# filepath= "best_Double_Unet-{epoch:02d}-{val_dice_coef:.4f}.hdf5"
filepath= "best_Double_Unet.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_dice_coef', verbose=1, save_best_only=True, mode='max')
# lrschedule = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.000001)
# lrschedule = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5)
# earlystop = EarlyStopping(monitor='val_dice_coef', min_delta=0.25, patience=5, verbose=1)
log_dir="/content/logs/Double_Unet/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=0)
callback_list = [checkpoint, tensorboard_callback]
```

Compile and Fit

```
In [ ]: import datetime
from tensorflow.keras.callbacks import *
from tensorflow.keras.losses import binary_crossentropy
# 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
# "/content/model_save/model/weights-{epoch:02d}-{val_auc_score:.4f}.hdf5"
# filepath= "best_Double_Unet-{epoch:02d}-{val_dice_coef:.4f}.hdf5"
filepath= "best_Double_Unet.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_dice_coef', verbose=1, save_best_only=True, mode='max')
# lrschedule = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.000001)
# lrschedule = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5)
# earlystop = EarlyStopping(monitor='val_dice_coef', min_delta=0.25, patience=5, verbose=1)
log_dir="/content/logs/Double_Unet/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=0)
callback_list = [checkpoint, tensorboard_callback]
```

```
In [ ]: train_dataset, val_dataset
```

```
Out[40]: (<PrefetchDataset shapes: ((None, 3, None, None), (None, 2, None, None)), types: (tf.float32, tf.float32)>,
<PrefetchDataset shapes: ((None, 3, None, None), (None, 2, None, None)), types: (tf.float32, tf.float32)>)
```

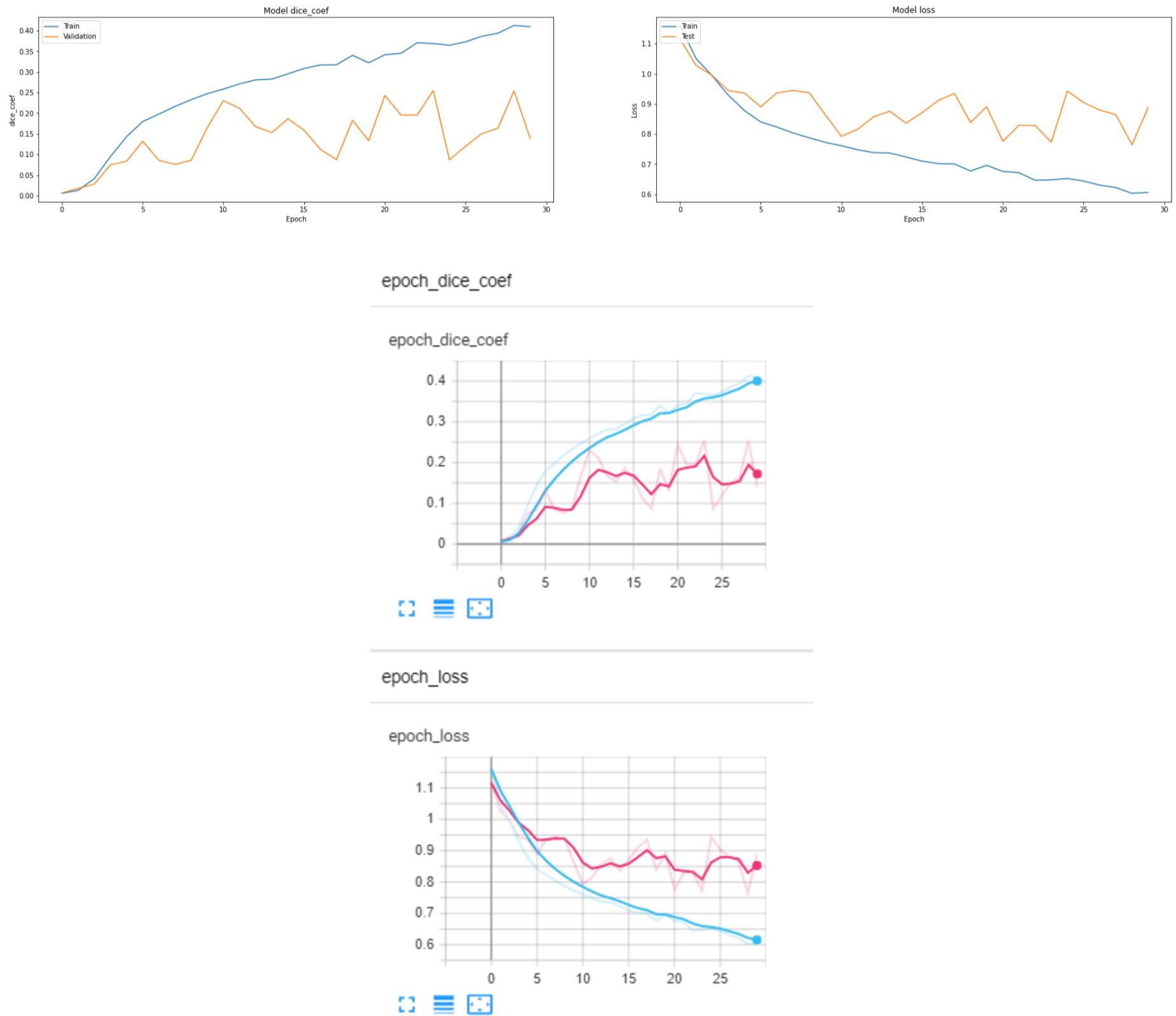
```
In [ ]: # model(np.random.rand(1,3,256,256))
```

```
In [ ]: # true = np.random.rand(1,2, 256, 256).astype(np.float32)
# pred = np.random.rand(1,2,256,256).astype(np.float32)
# bce_dice_loss(true.astype(np.float32),pred.astype(np.float32))
# # binary_crossentropy(true, pred)
```

```
In [ ]: model.compile(optimizer=optim, loss=bce_dice_loss, metrics=[dice_coef])
history = model.fit(train_dataset,validation_data=val_dataset,
                    epochs=30, steps_per_epoch=10239//12, validation_steps=1808//12, verbose=1,
                    callbacks=callback_list)
```

```
Epoch 1/30
 2/853 [........................] - ETA: 8:35 - loss: 1.6445 - dice_coef: 0.0037WARNING:tensorflow:Callback method `on_train_batch_end` is slow compared to the batch time (batch time: 0.2340s vs `on_train_batch_end` time: 0.5841s). Check your callbacks.
853/853 [=====] - ETA: 0s - loss: 1.1607 - dice_coef: 0.0057
Epoch 00001: val_dice_coef improved from -inf to 0.00651, saving model to best_Double_Unet.hdf5
853/853 [=====] - 574s 673ms/step - loss: 1.1607 - dice_coef: 0.0057 - val_loss: 1.1165 - val_dice_coef: 0.0065
Epoch 2/30
853/853 [=====] - ETA: 0s - loss: 1.0506 - dice_coef: 0.0129
Epoch 00002: val_dice_coef improved from 0.00651 to 0.01761, saving model to best_Double_Unet.hdf5
853/853 [=====] - 573s 671ms/step - loss: 1.0506 - dice_coef: 0.0129 - val_loss: 1.0276 - val_dice_coef: 0.0176
Epoch 3/30
853/853 [=====] - ETA: 0s - loss: 0.9934 - dice_coef: 0.0412
Epoch 00003: val_dice_coef improved from 0.01761 to 0.02821, saving model to best_Double_Unet.hdf5
853/853 [=====] - 573s 672ms/step - loss: 0.9934 - dice_coef: 0.0412 - val_loss: 0.9945 - val_dice_coef: 0.0282
Epoch 4/30
853/853 [=====] - ETA: 0s - loss: 0.9936 - dice_coef: 0.0412
```

```
In [ ]: # 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()
```



Inference

```
In [46]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [ ]: # !cp '/content/best_Double_Unet.hdf5' '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet.hdf5'
```

```
In [ ]: # !cp -r '/content/logs' '/content/drive/MyDrive/27_Case_study_2/double_unet_logs'
```

```
In [ ]: # %Load_ext tensorboard
```

```
In [ ]: # %tensorboard --logdir="/content/Logs/Double_Unet/fit/"
```

```
In [ ]: model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet.hdf5' )
```

```
In [ ]: X_val.columns
```

```
Out[18]: Index(['ImagePath', 'MaskPath'], dtype='object')
```

```
In [22]: def read_image(path):
    """
    This reads image
    -----
    path : image path
    -----
    """
    # image = tf.io.read_file(img_path)
    # image = tf.image.decode_png(image, channels=3)
    # image = tf.image.convert_image_dtype(image, tf.uint8)

    x = tf.io.read_file(path)
    x = tf.image.decode_png(x, channels=3)
    x = tf.image.convert_image_dtype(x, tf.float32)
    # x = x / 255.0
    # x = tf.image.convert_image_dtype(x, tf.float32)

    return x

def read_mask(path):
    """
    This reads mask
    -----
    path : mask path
    -----
    """
    y = tf.io.read_file(path)
    y = tf.image.decode_png(y, channels=1)
    y = tf.image.convert_image_dtype(y, tf.float32)
    # y = y / 255.0
    # x = np.expand_dims(x, axis=-1)
    # y = tf.image.convert_image_dtype(y, tf.float32)
    return y
```

```
In [23]: def Predict(x, y):
    """
    This function predicts the mask for given input image and plots it.
    -----
    x      : image path
    y      : original mask path
    return: the original image, original mask and predicted mask
    -----
    """
    # read the original image
    image_orig = read_image(x)
    # read the original mask
    image_mask_orig = read_mask(y)
    # reshape image and mask as first channel image format
    image = tf.transpose(image_orig, [2,0,1])
    image_mask = tf.transpose(image_mask_orig, [2,0,1])
    # predict the mask using trained model
    predict_mask = model.predict(tf.expand_dims(image, axis=0))
    predict_mask = tf.transpose(predict_mask, [0,2,3,1])
    # return the original image, original mask and predicted mask
    return image_orig, image_mask_orig, predict_mask
```

```
In [ ]: mask_df.columns
```

```
Out[25]: Index(['file_names', 'mask_percentage', 'labels'], dtype='object')
```

```
In [ ]: mask_df['labels'].value_counts()
```

```
Out[26]: 0    9378
1    2669
Name: labels, dtype: int64
```

```
In [ ]: len(final_data), len(mask_df)
```

```
Out[27]: (12047, 12047)
```

```
In [ ]: # new_test = final_data.copy()
# new_test['labels'] = mask_df['Labels']
# new_test = new_test[new_test['Labels']==1]
# new_test.head()
```

```
In [ ]: # plot the original image
    # plt.imshow(image_orig, cmap="bone")
    # plt.imshow(image_mask_orig[:, :, 0], cmap='Greens', alpha = 0.3)
    # plt.imshow(np.squeeze(predict_mask[:, :, :, 1]), cmap='Reds', alpha = 0.3)
    # plt.title("Pneumothorax")
    # plt.show()
```

```
In [ ]: max_images = 48
grid_width = 6
grid_height = int(max_images / grid_width)
fig, axs = plt.subplots(grid_height, grid_width, figsize=(grid_width*4, grid_height*4))
fig.subplots_adjust(bottom = 0.05)

for i in range(max_images):
    img = final_data['ImagePath'].values[i]
    msk = final_data['MaskPath'].values[i]
    image_orig, image_mask_orig, predict_mask = Predict(img, msk)

    ax = axs[int(i / grid_width), i % grid_width]
    ax.imshow(image_orig)
    ax.imshow(image_mask_orig[:, :, 0], cmap='Greens', alpha = 0.3)
    ax.imshow(np.squeeze(predict_mask[:, :, :, 1]), cmap='Reds', alpha = 0.3)
    # ax.axis('off')
plt.suptitle("Red: Predicted Mask, Green: Original Mask", fontsize=20)
plt.show()
```

Let's analyze the outputs for all datapoints

```
In [24]: model.load_weights( '/content/drive/MyDrive/27_Case_study_2/best_Double_Unet.hdf5' )
print('loaded saved model {best_Double_Unet.hdf5}')

loaded saved model {best_Double_Unet.hdf5}
```

```
In [25]: smooth = 1.
def dice_coef(y_true, y_pred):
    y_true = tf.keras.layers.Flatten()(y_true)
    y_pred = tf.keras.layers.Flatten()(y_pred)
    intersection = tf.reduce_sum(y_true * y_pred)
    return (2. * intersection + smooth) / (tf.reduce_sum(y_true) + tf.reduce_sum(y_pred) + smooth)

def dice_loss(y_true, y_pred):
    return 1.0 - dice_coef(y_true, y_pred)

def iou(y_true, y_pred):
    def f(y_true, y_pred):
        intersection = (y_true * y_pred).sum()
        union = y_true.sum() + y_pred.sum() - intersection
        x = (intersection + smooth) / (union + smooth)
        x = x.astype(np.float32)
        return x
    return tf.numpy_function(f, [y_true, y_pred], tf.float32)

from tensorflow.keras.losses import binary_crossentropy

def bce_dice_loss(y_true, y_pred):
    return binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)
```

```
In [ ]:
```

```
In [26]: dice_coefs = []
for i in tqdm(range(len(final_data))):
    img = final_data['ImagePath'].values[i]
    msk = final_data['MaskPath'].values[i]
    image_orig, image_mask_orig, predict_mask = Predict(img, msk)
    image_mask_orig = np.expand_dims(image_mask_orig, axis=0)
    image_mask_orig = Concatenate(axis=0)([image_mask_orig, image_mask_orig])
    y_true = K.flatten(image_mask_orig)
    y_pred = K.flatten(predict_mask)
    d_c = dice_coef(y_true, y_pred)
    # print(d_c)
    dice_coefs.append(d_c)

print("Done")
```

100%|██████████| 12047/12047 [14:23<00:00, 13.95it/s]

Done

```
In [30]: from IPython.display import Audio
sound_file = 'https://www.soundjay.com/button/beep-01a.mp3'

Audio(sound_file, autoplay=True)
```

Out[30]:
0:00 / 0:00

```
In [32]: dice_coefs = [dc.numpy() for dc in dice_coefs]
dice_coefs[0]
```

Out[32]: 0.24002662

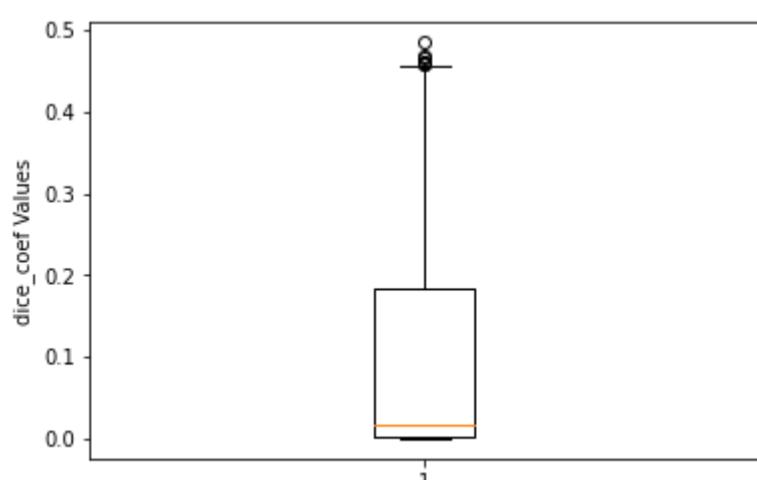
```
In [62]: output_df = final_data.copy()
output_df['dice_coef'] = dice_coefs
output_df.head()
```

	ImagePath	MaskPath	dice_coef
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...	0.240027
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...	0.000143
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...	0.074667
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...	0.000566
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...	0.008678

```
In [63]: output_df['dice_coef'].describe()
```

```
Out[63]: count    12047.000000
mean      0.099794
std       0.135103
min      0.000074
25%      0.002315
50%      0.018253
75%      0.184656
max      0.484696
Name: dice_coef, dtype: float64
```

```
In [64]: plt.boxplot(dice_coefs)
plt.ylabel("dice_coef Values")
plt.show()
```



- The above description about dice coefficient shows that there is Max value is 0.484696 and the 75th percentile is 0.184656. I will consider the score greater than equal to 0.184656 as Max score

- 50th percentile is 0.018253. I will consider score greater than 0.018253 and less than 0.184656 as Medium score
- And for the score less than or equal to 0.018253 I will consider Low score

```
In [65]: low_score_images = output_df[output_df['dice_coef']<=0.018253]
low_score_images = low_score_images.sort_values(by=['dice_coef'], ascending=True)
print(low_score_images.shape)
low_score_images.head()

(6025, 3)
```

Out[65]:

	ImagePath	MaskPath	dice_coef
9230	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...	0.000074
4205	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...	0.000074
5488	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...	0.000076
3670	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...	0.000085
541	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...	0.000089

```
In [66]: medium_score_images = output_df.query('dice_coef > 0.018253 & dice_coef < 0.184656')
medium_score_images = medium_score_images.sort_values(by=['dice_coef'], ascending=False)
print(medium_score_images.shape)
medium_score_images.head()

(3010, 3)
```

Out[66]:

	ImagePath	MaskPath	dice_coef
976	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...	0.184586
839	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...	0.184408
4460	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...	0.184360
10463	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...	0.184129
1190	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...	0.183970

```
In [67]: high_score_images = output_df[output_df['dice_coef']>=0.184656]
high_score_images = high_score_images.sort_values(by=['dice_coef'], ascending=False)
print(high_score_images.shape)
high_score_images.head()

(3012, 3)
```

Out[67]:

	ImagePath	MaskPath	dice_coef
9157	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...	0.484696
7619	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...	0.468473
9008	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...	0.466120
9033	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...	0.461297
2653	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...	0.459939

```
In [68]: total_count = len(low_score_images) + len(medium_score_images) + len(high_score_images)
print(total_count)
```

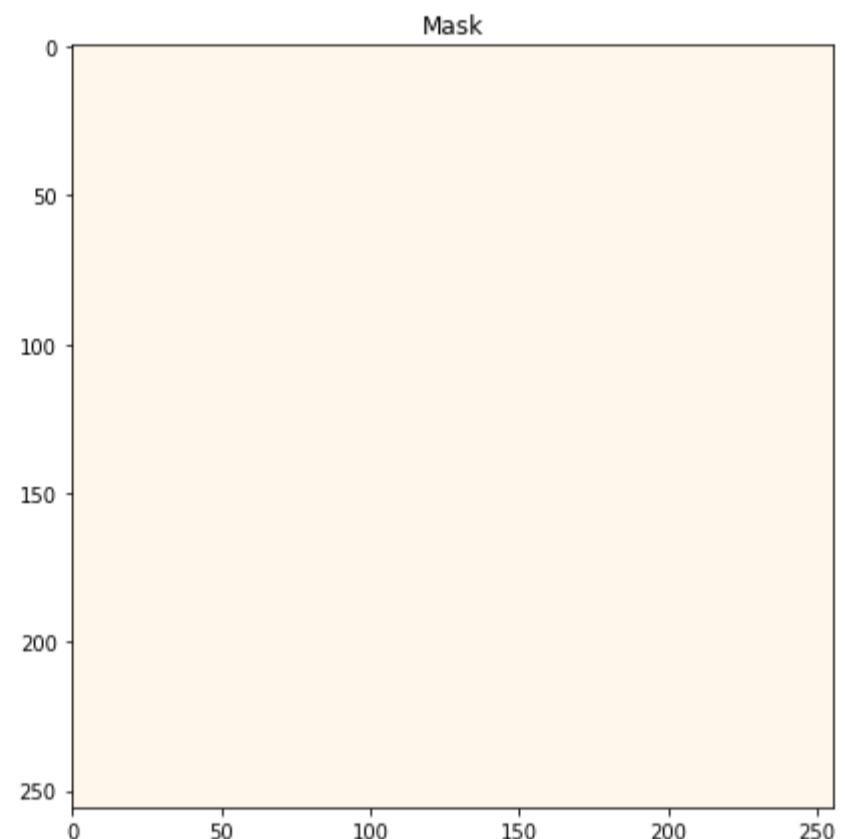
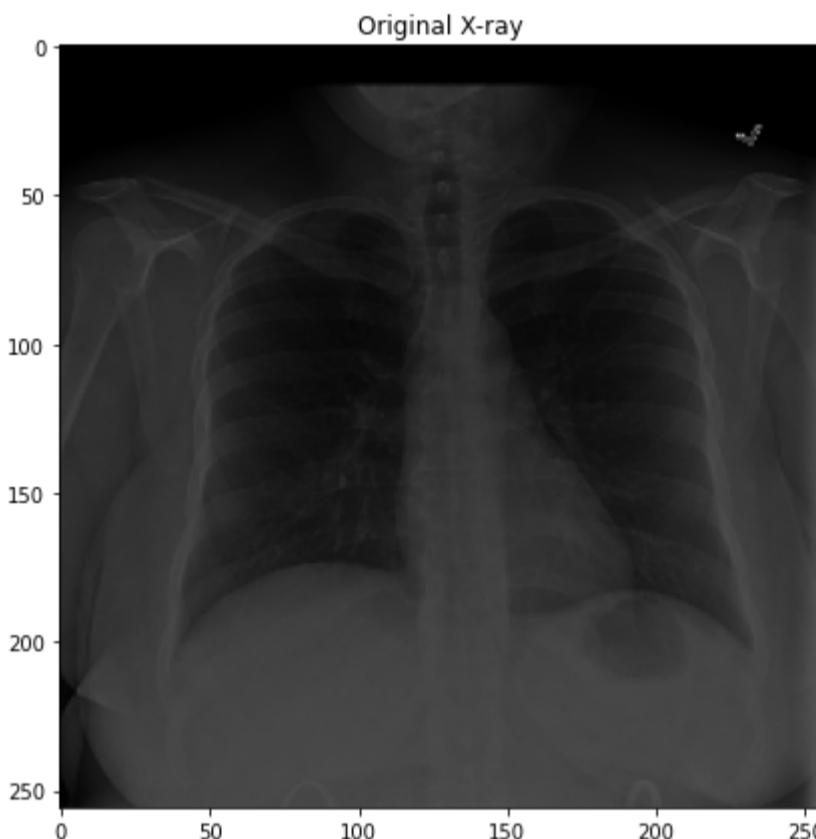
12047

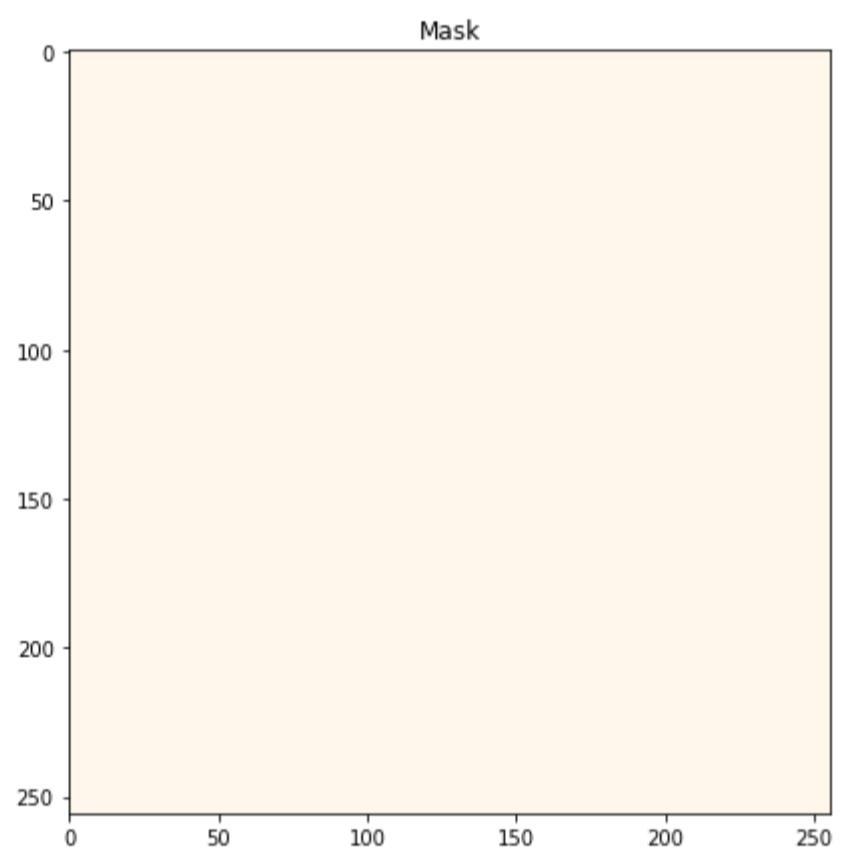
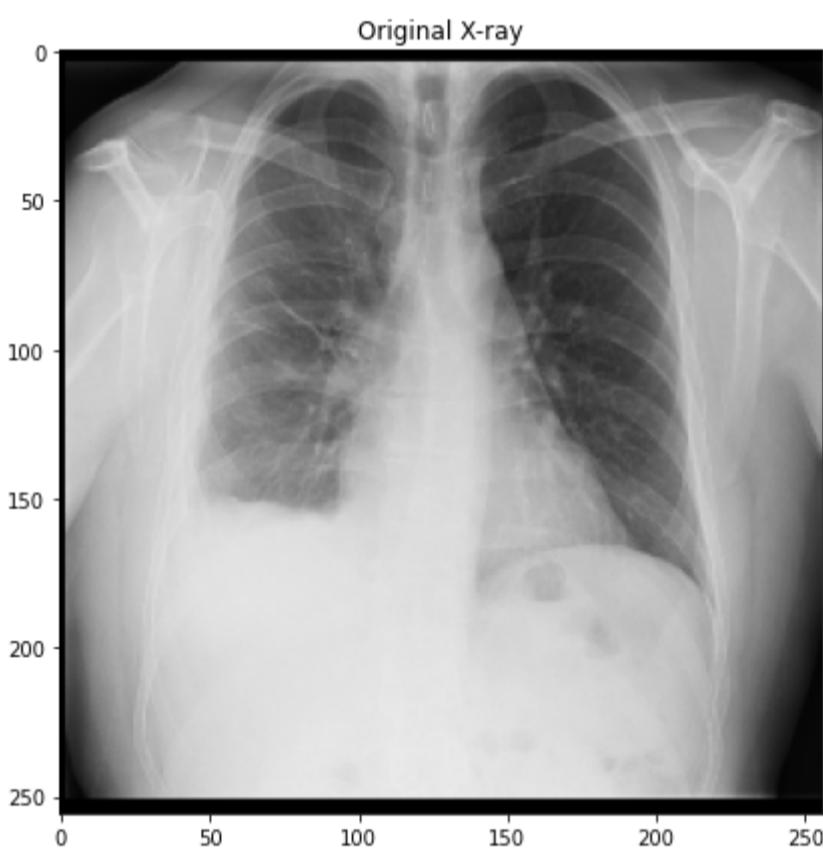
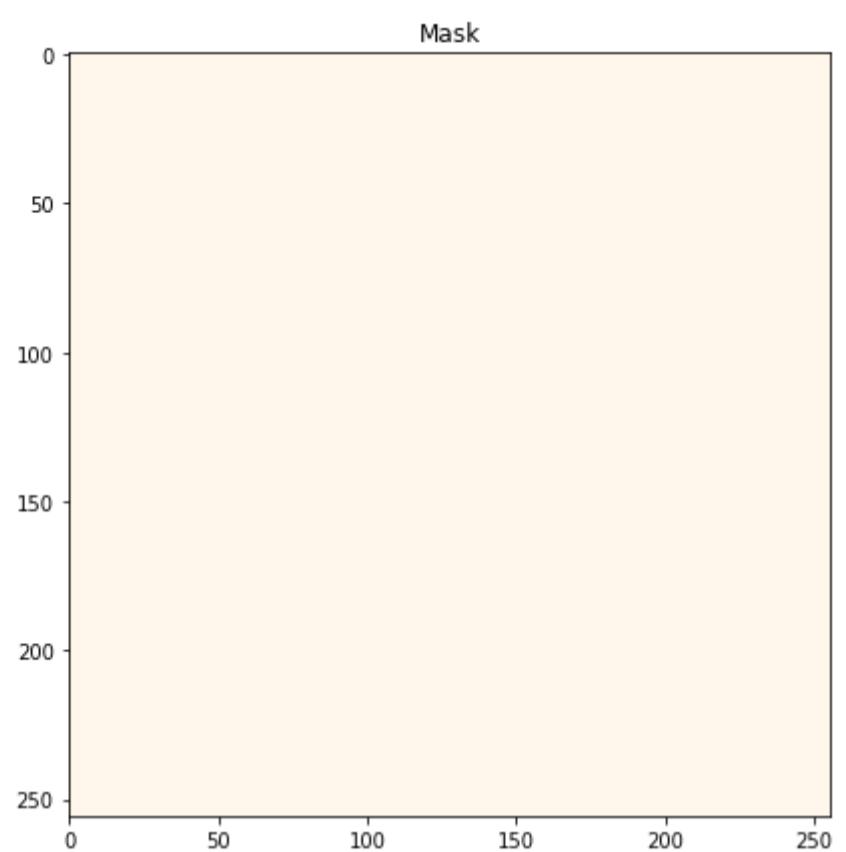
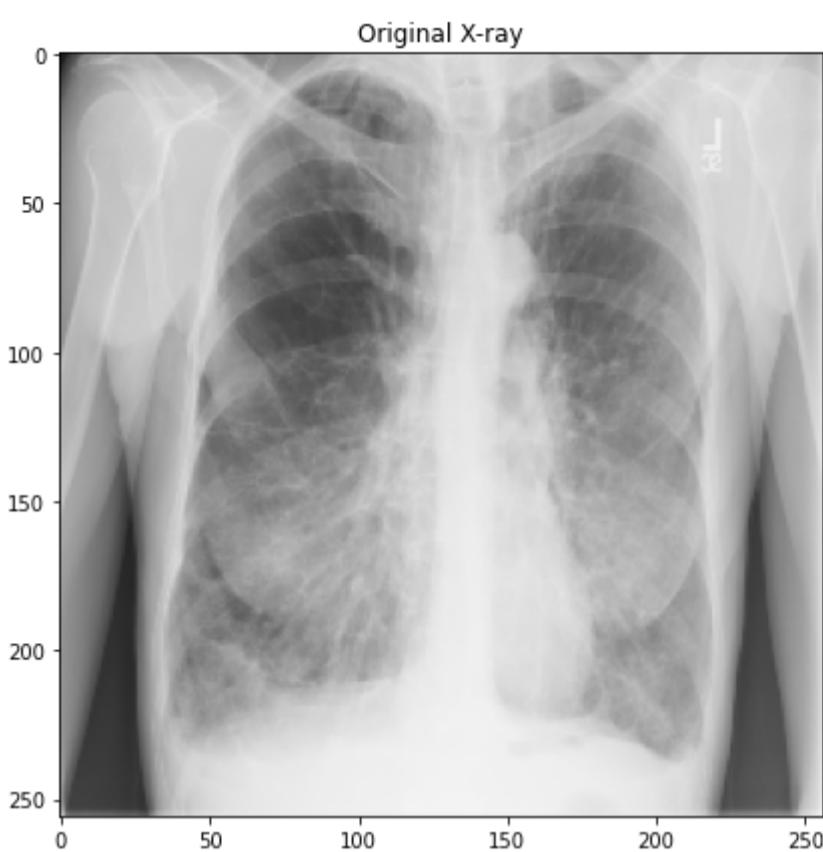
I have now splitted the images best on the high, low and medium scores. Now let's plot some of the images from each score group.

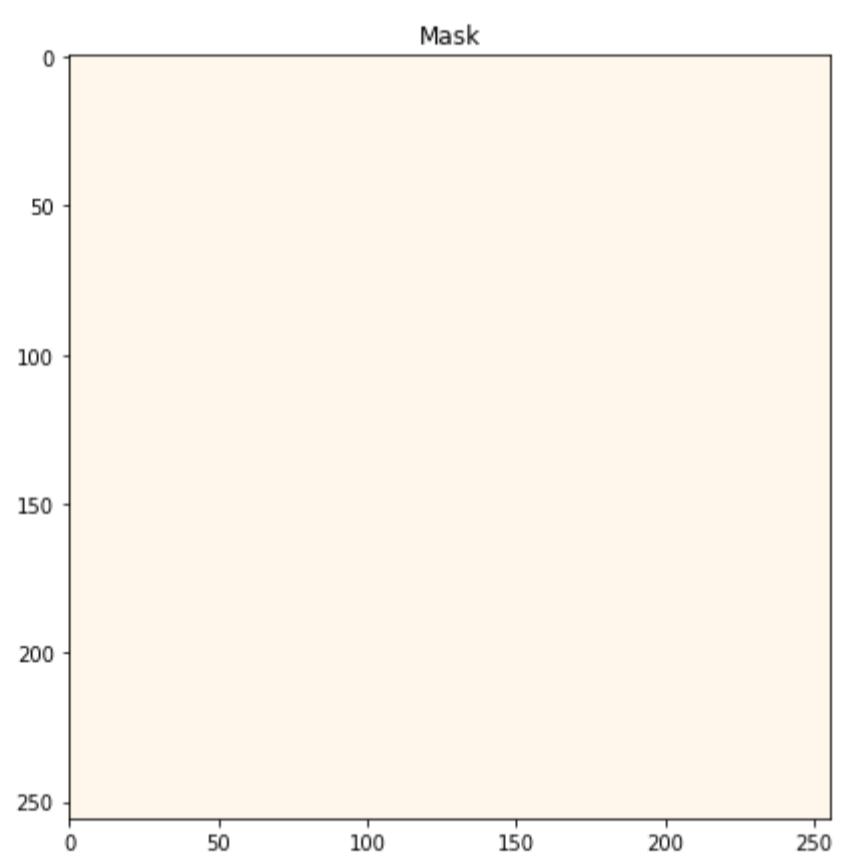
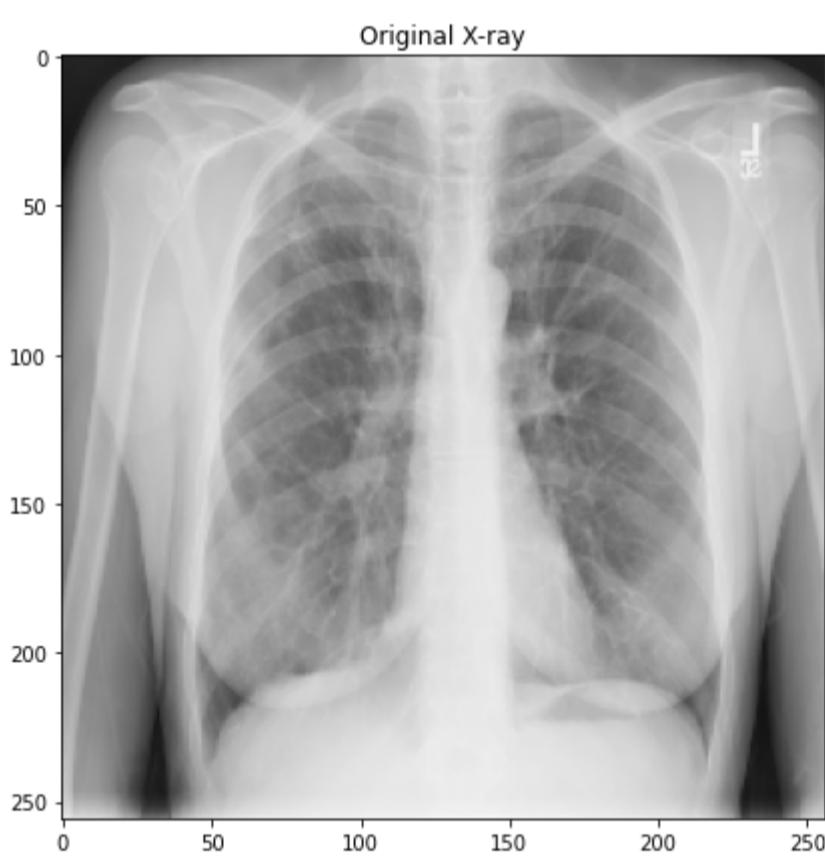
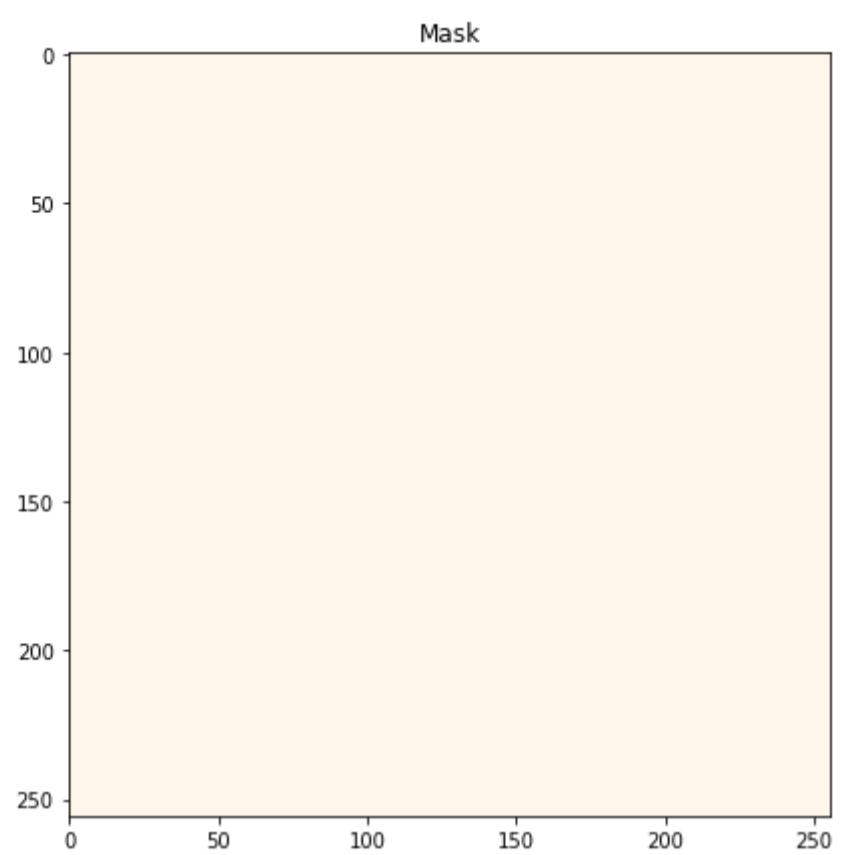
High Score images

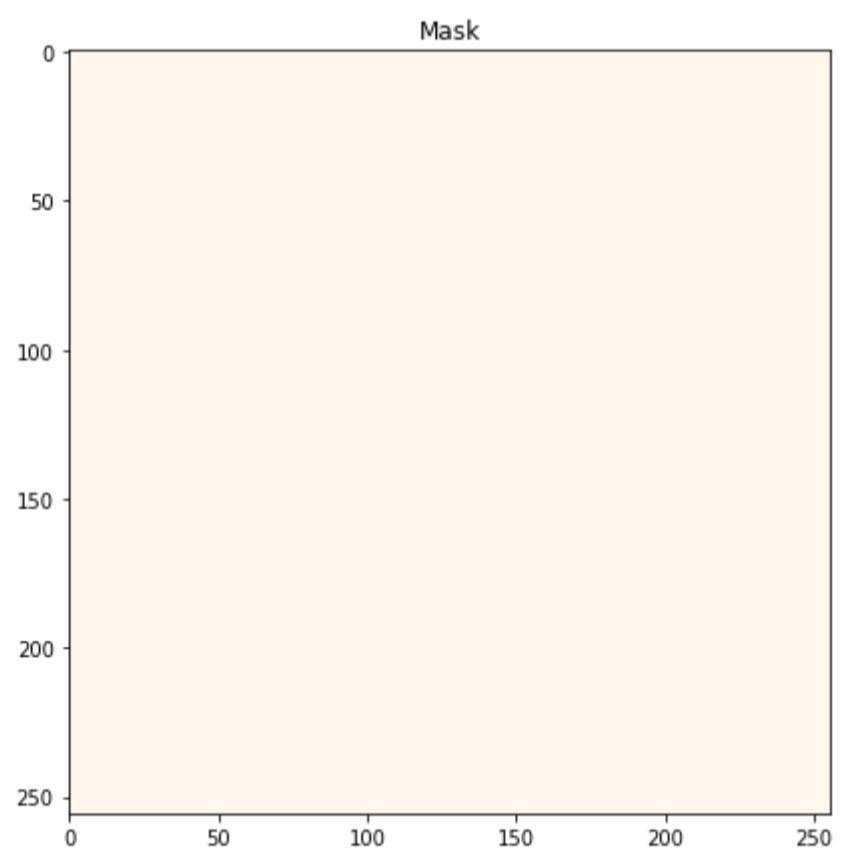
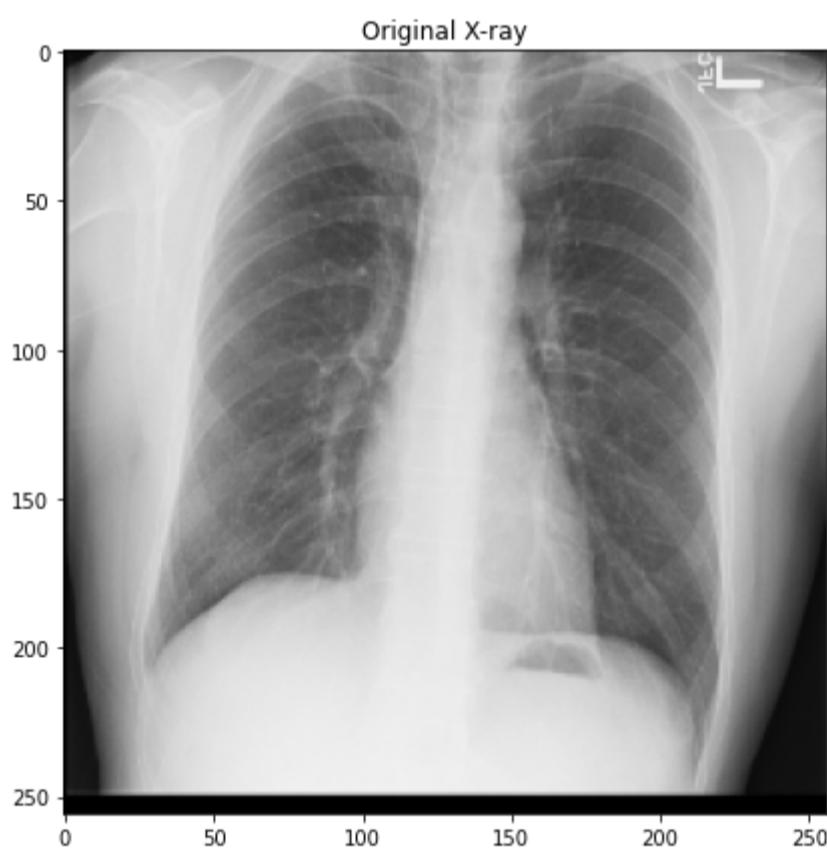
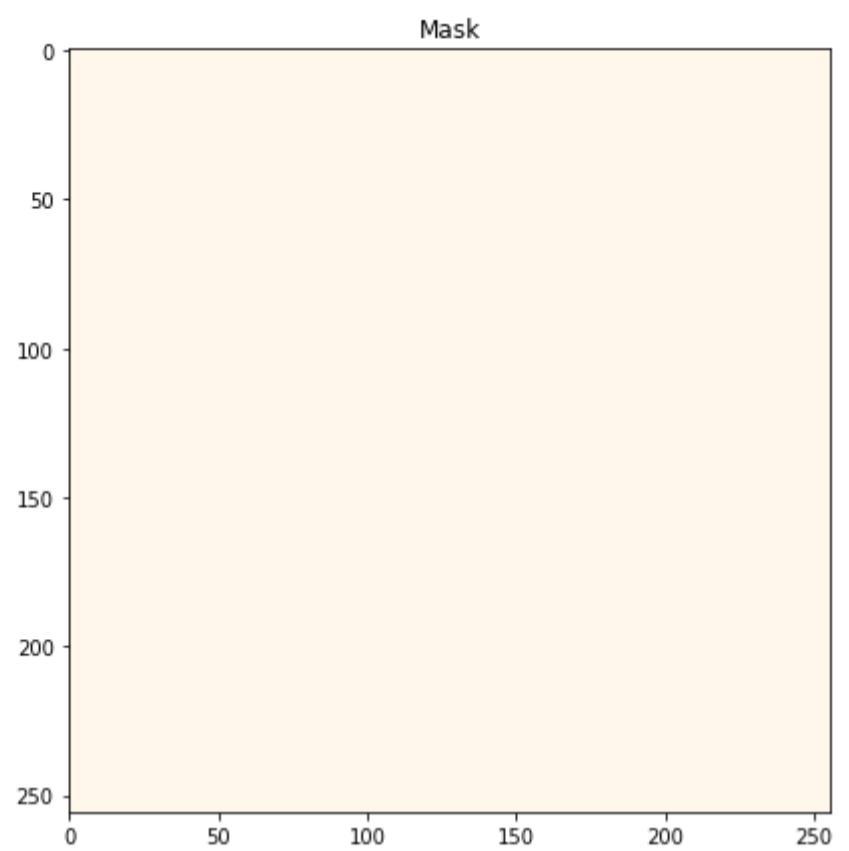
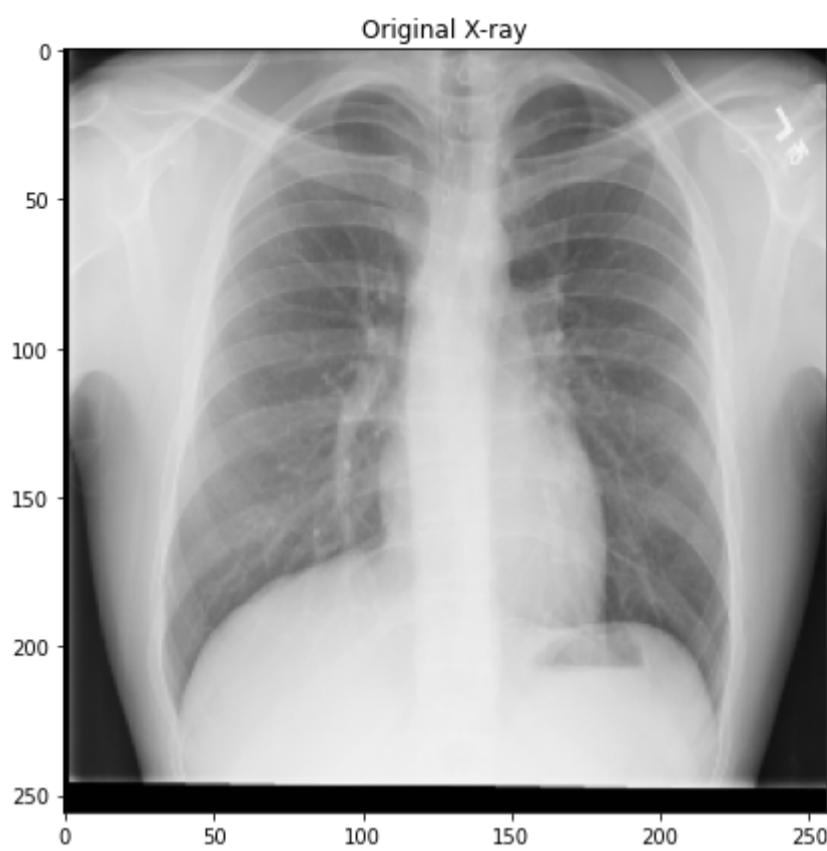
```
In [69]: max_images = 12
```

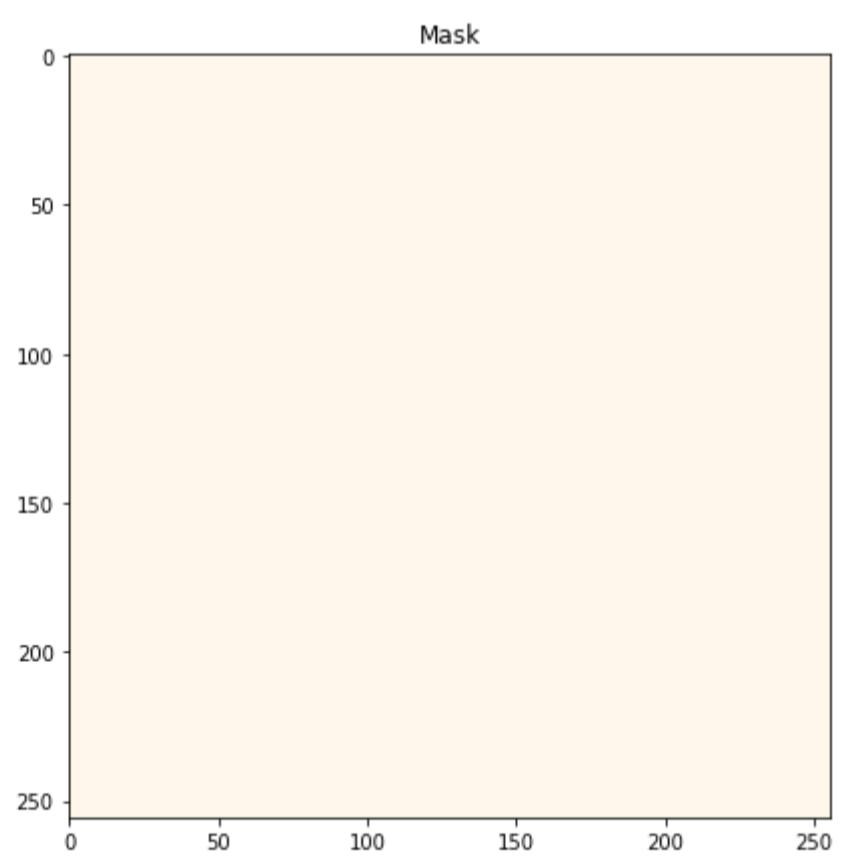
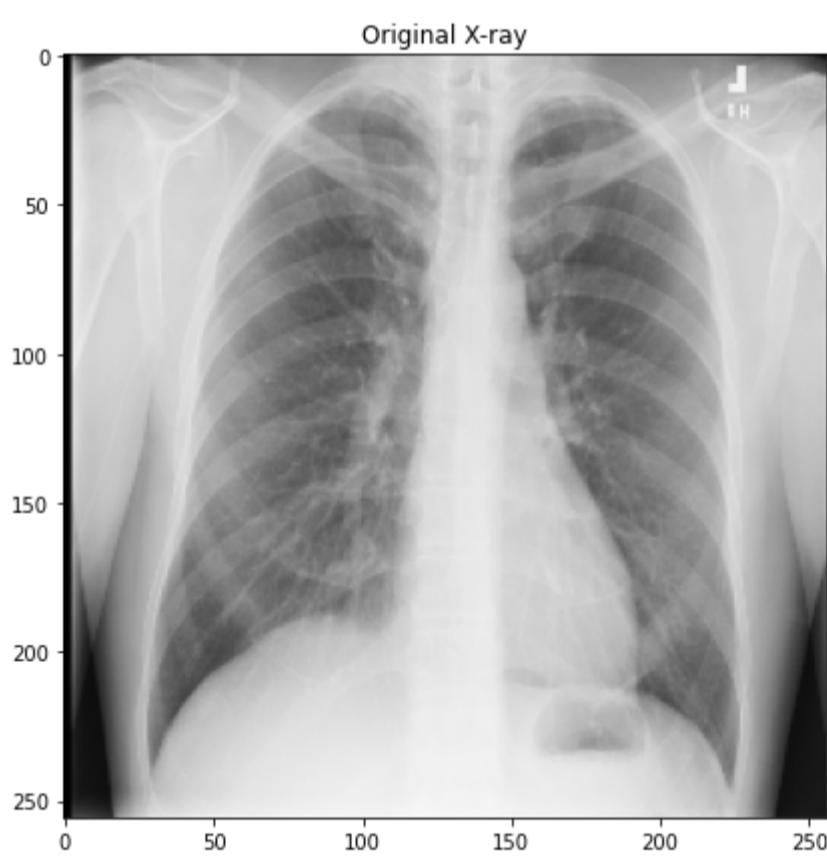
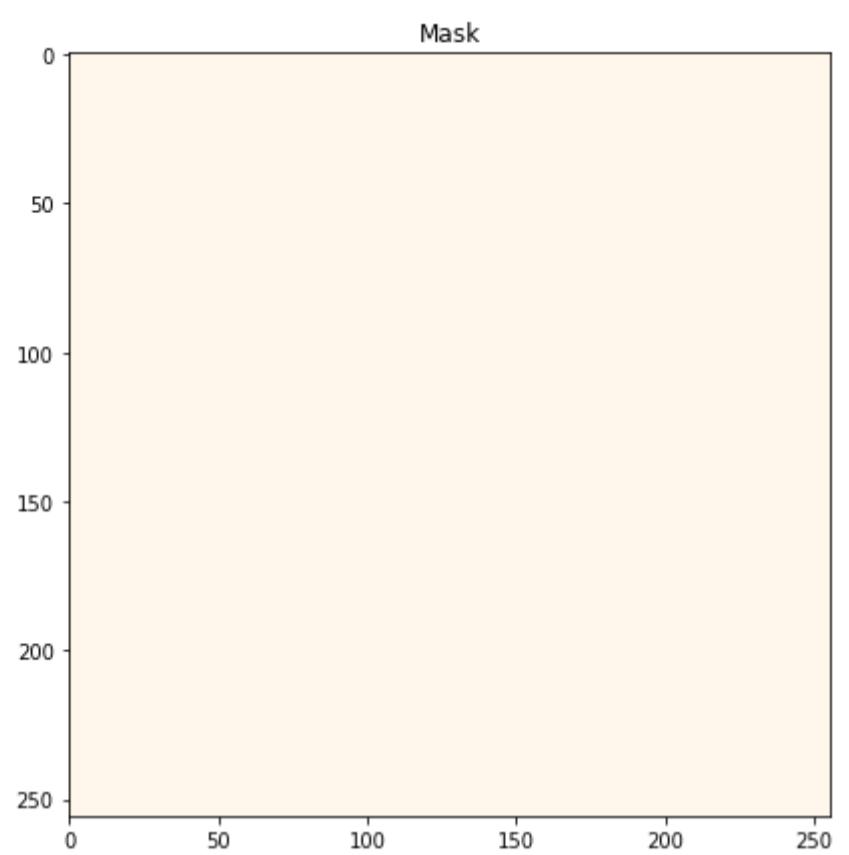
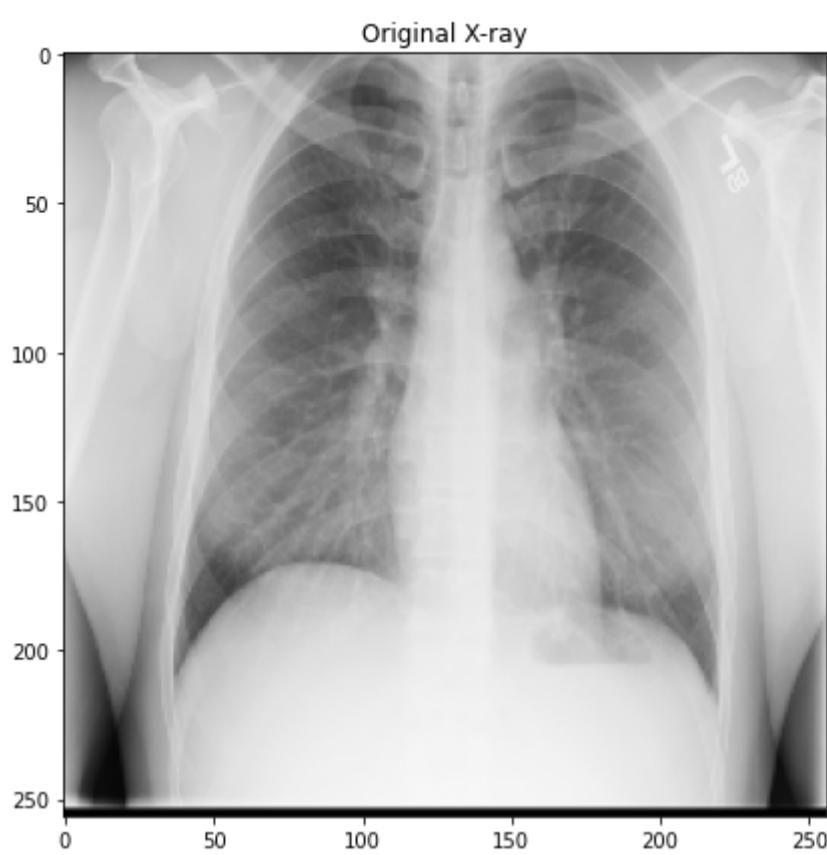
```
for i in range(max_images):
    img = high_score_images['ImagePath'].values[i]
    msk = high_score_images['MaskPath'].values[i]
    # read the original image
    image_orig = read_image(img)
    # read the original mask
    image_mask_orig = read_mask(msk)
    # plot Original X-ray
    fig = plt.figure(figsize=(15, 10))
    a = fig.add_subplot(1, 2, 1)
    plt.imshow(image_orig, cmap='bone')
    a.set_title(f"Original X-ray")
    plt.grid(False)
    # plt.axis("off")
    # Plotting mask only
    a = fig.add_subplot(1, 2, 2)
    plt.imshow(np.squeeze(image_mask_orig), cmap='OrRd', alpha=1)
    a.set_title("Mask")
    plt.grid(False)
    # plt.axis("off")
    # break
```

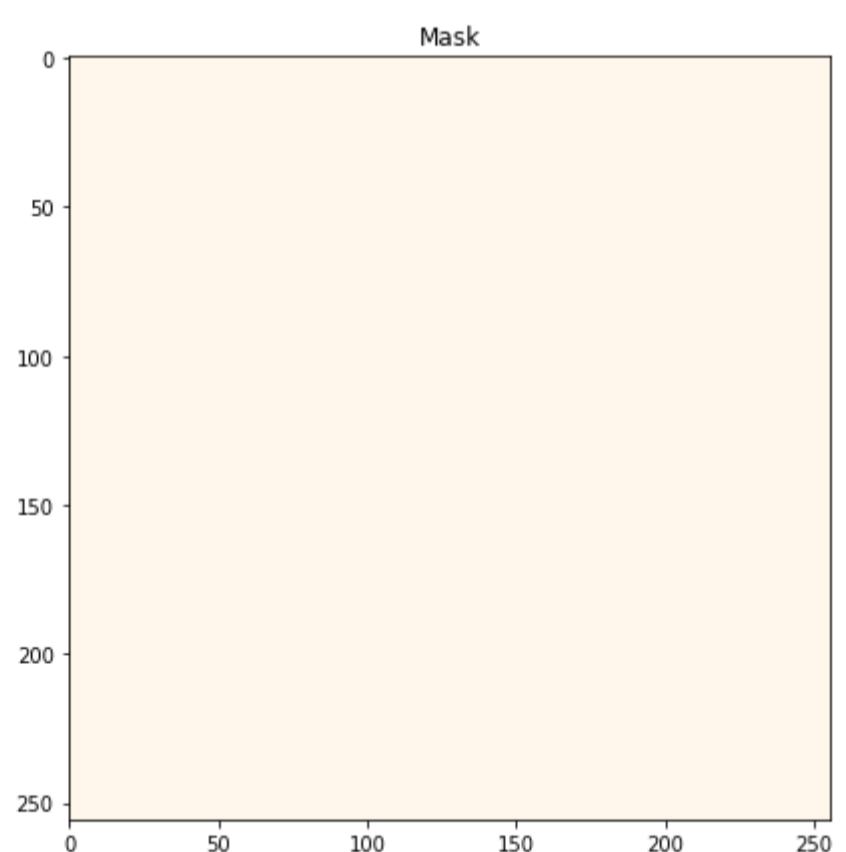
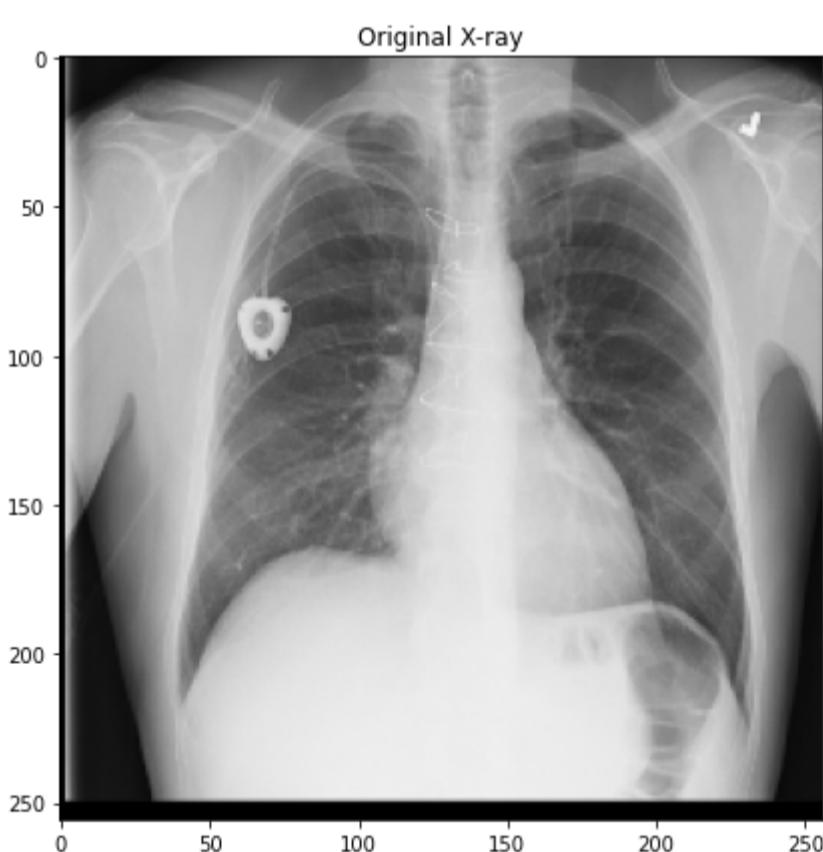
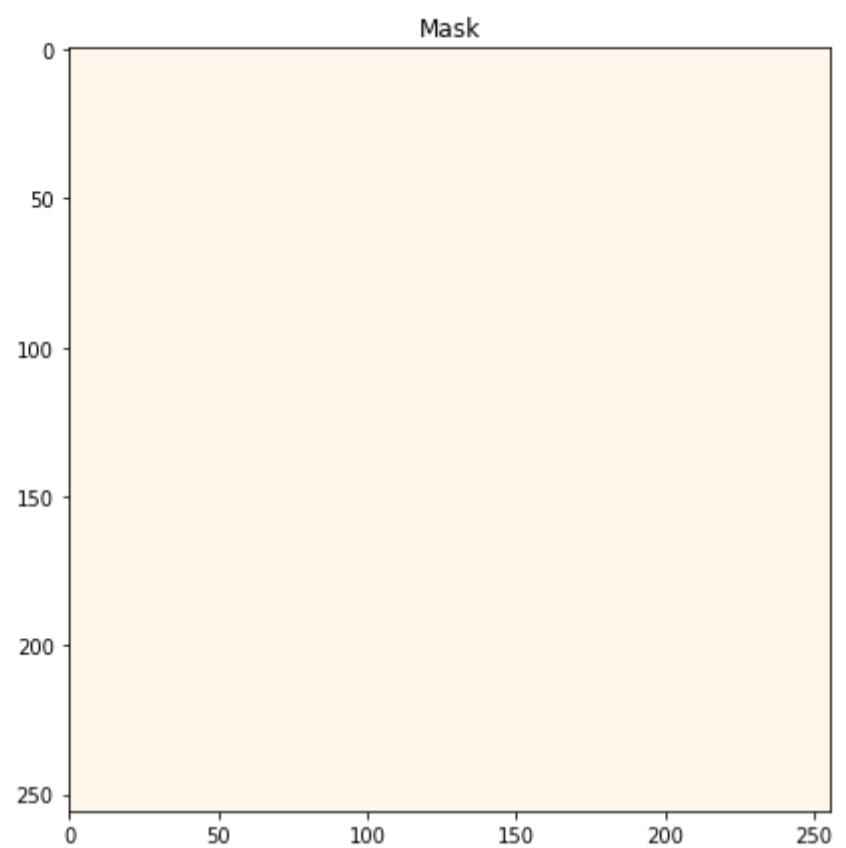
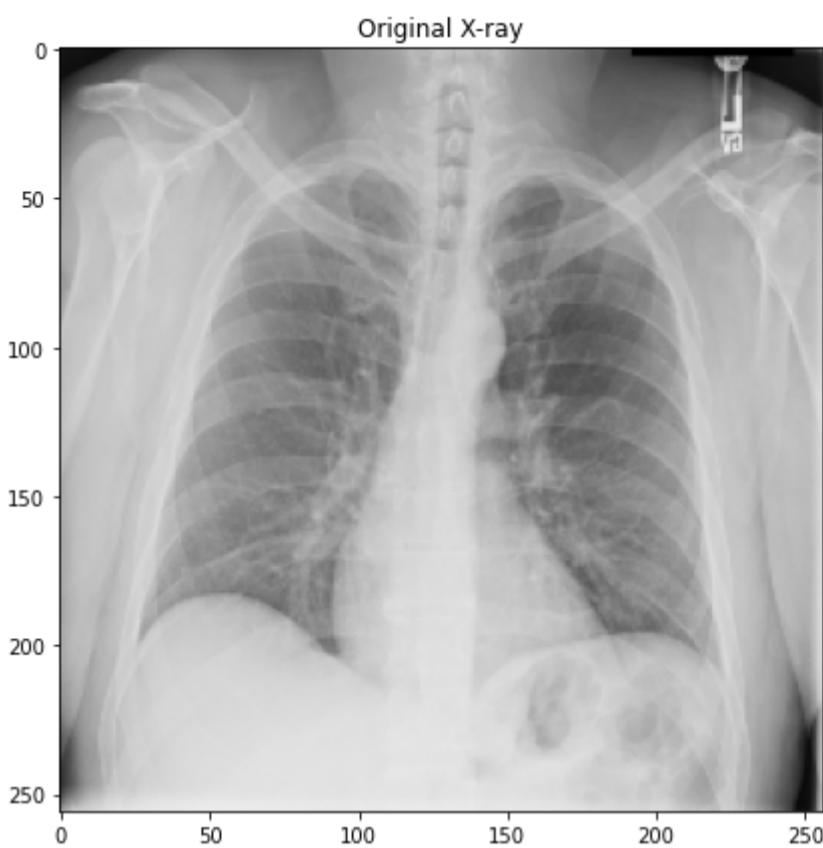


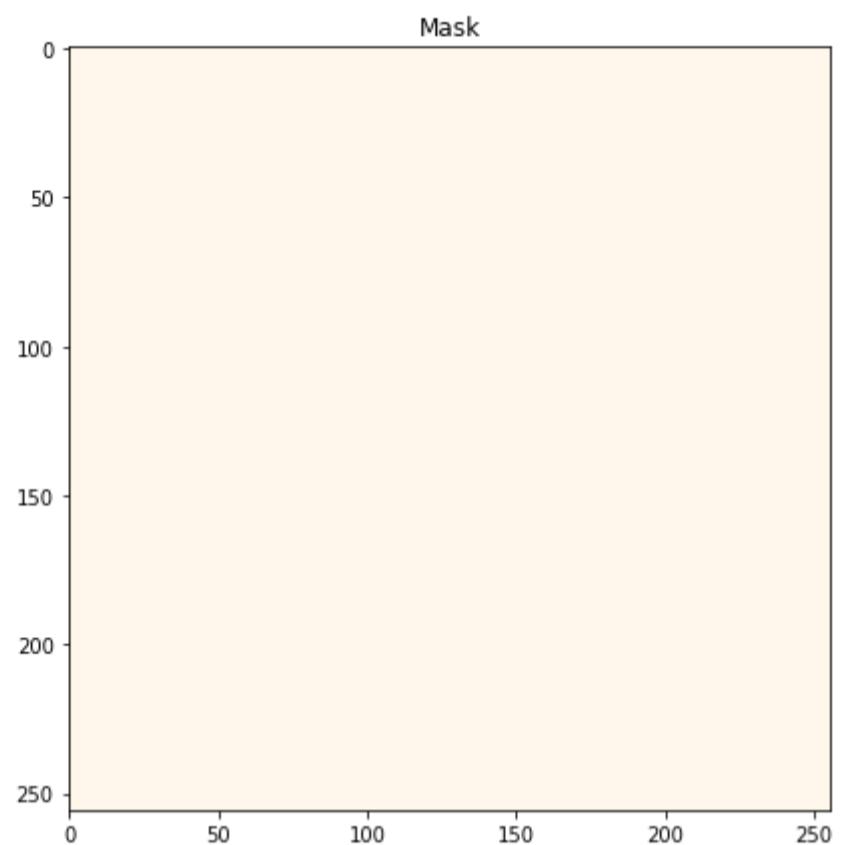
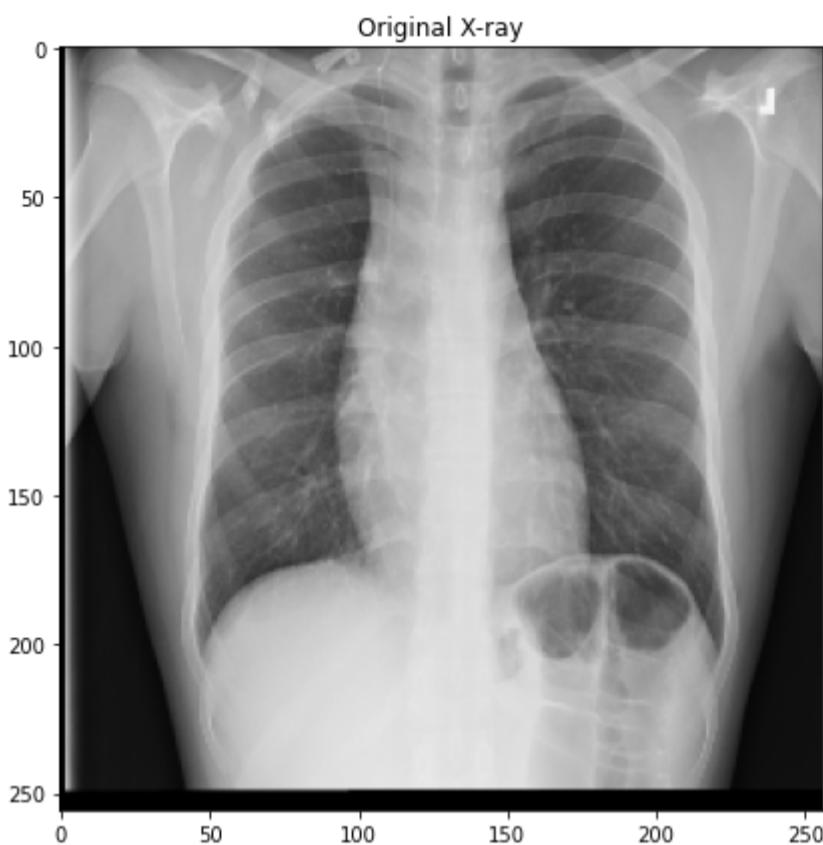












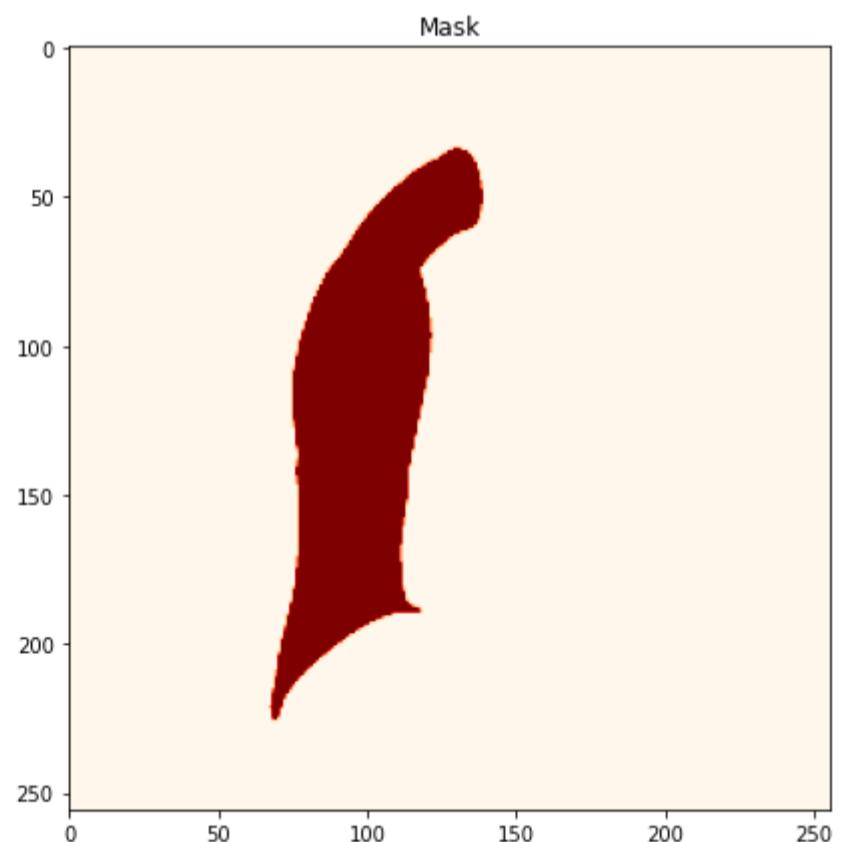
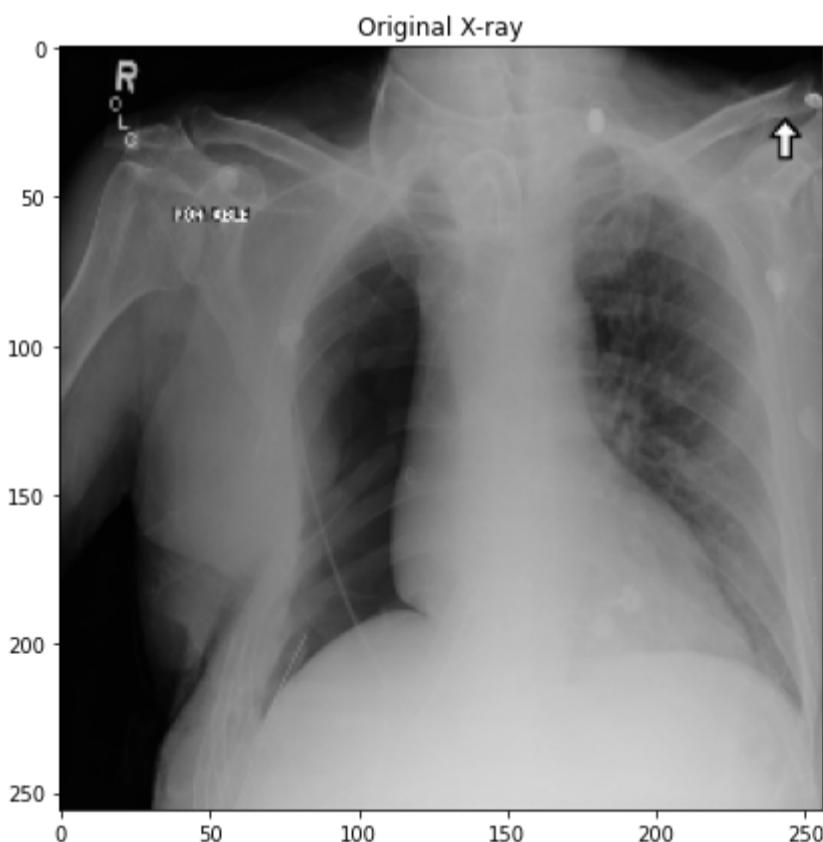
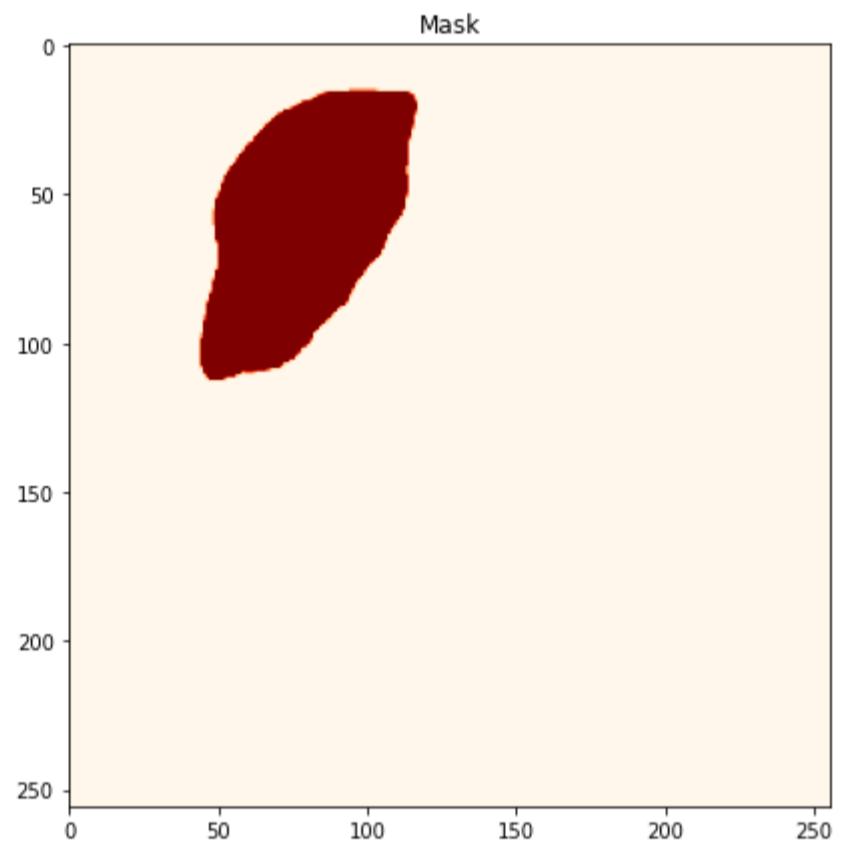
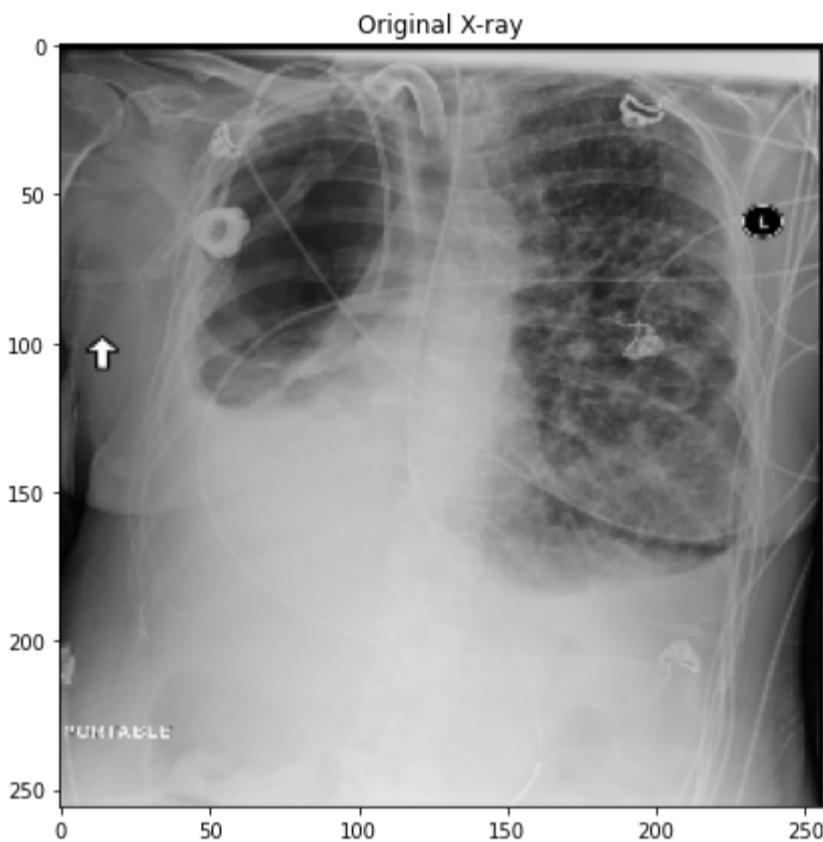
Low Score images

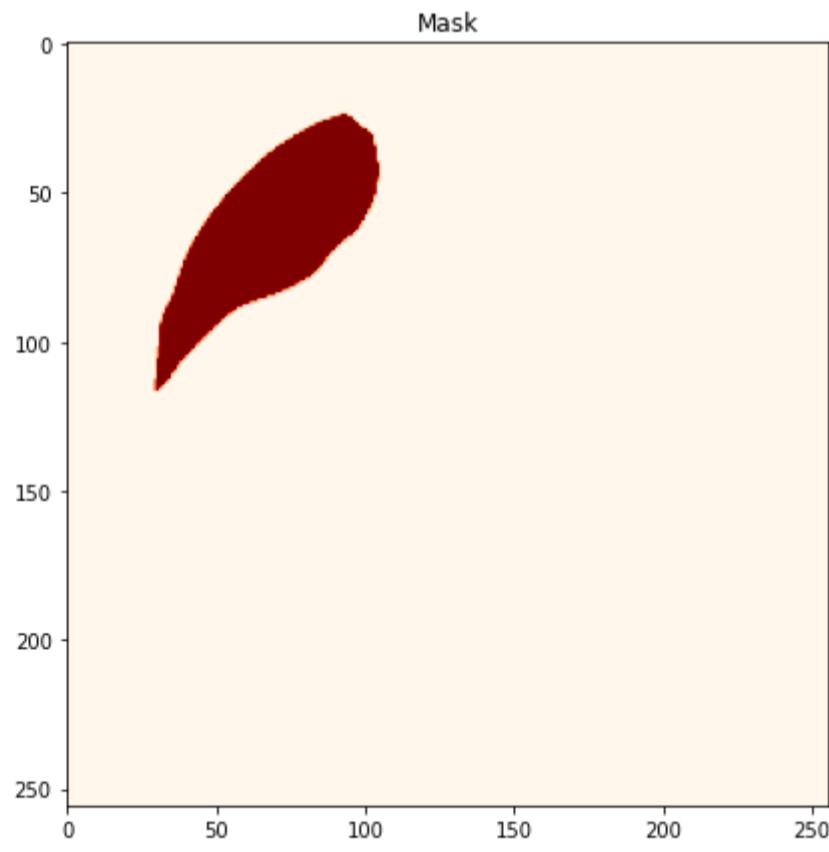
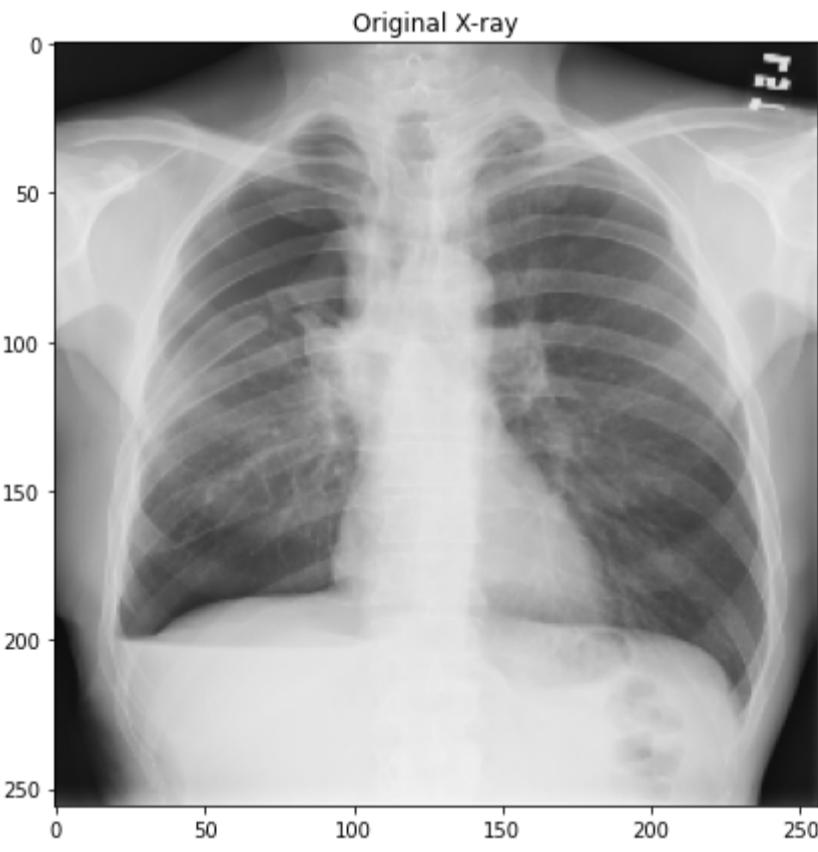
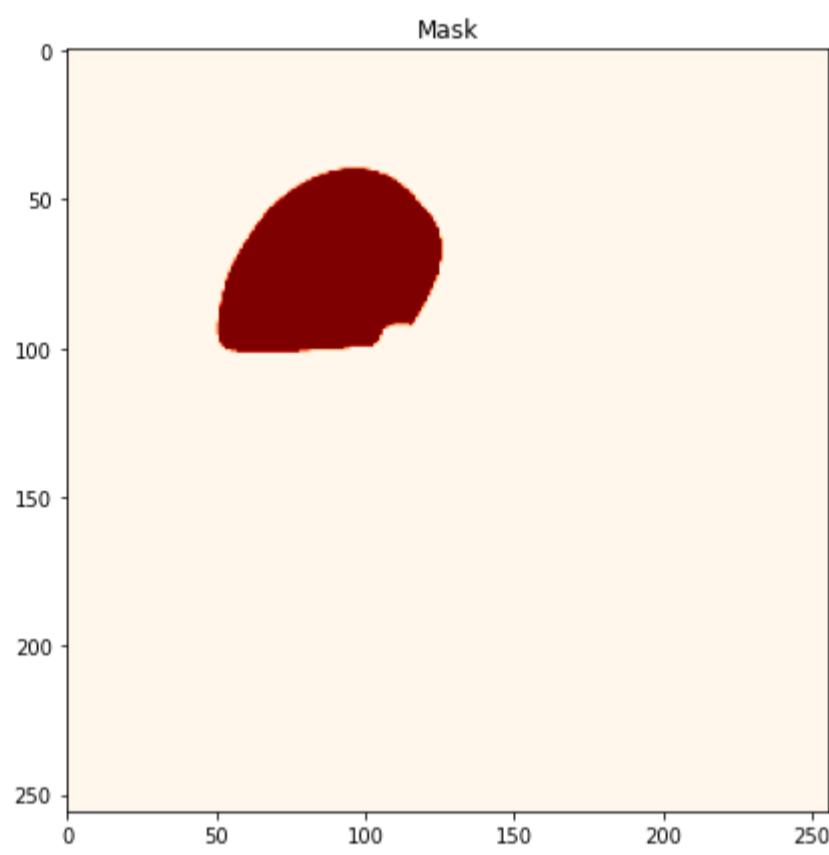
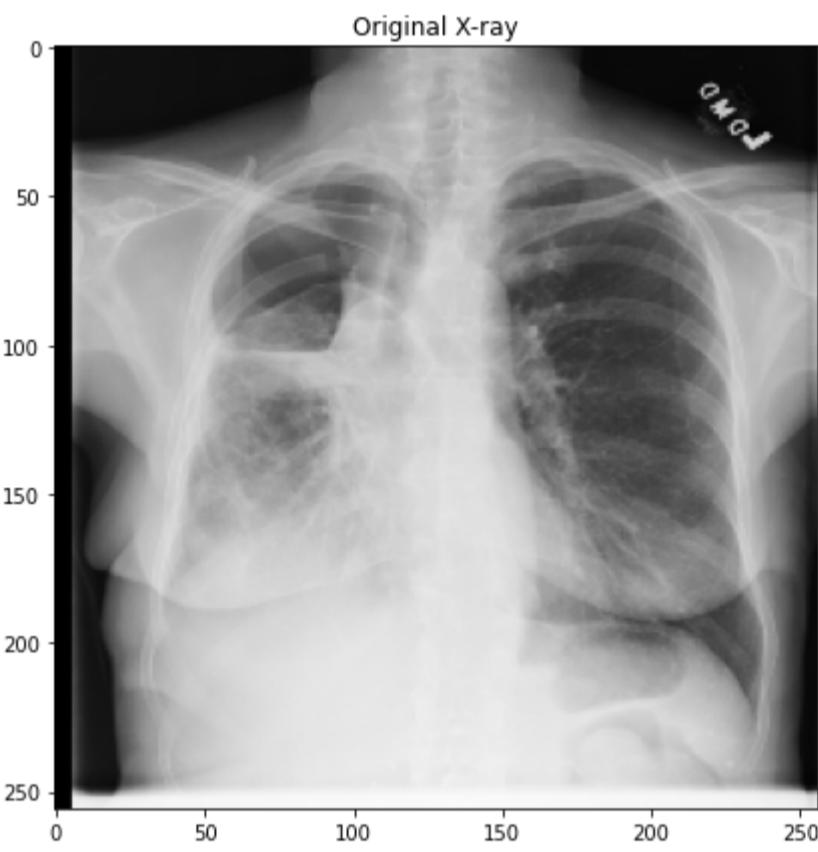
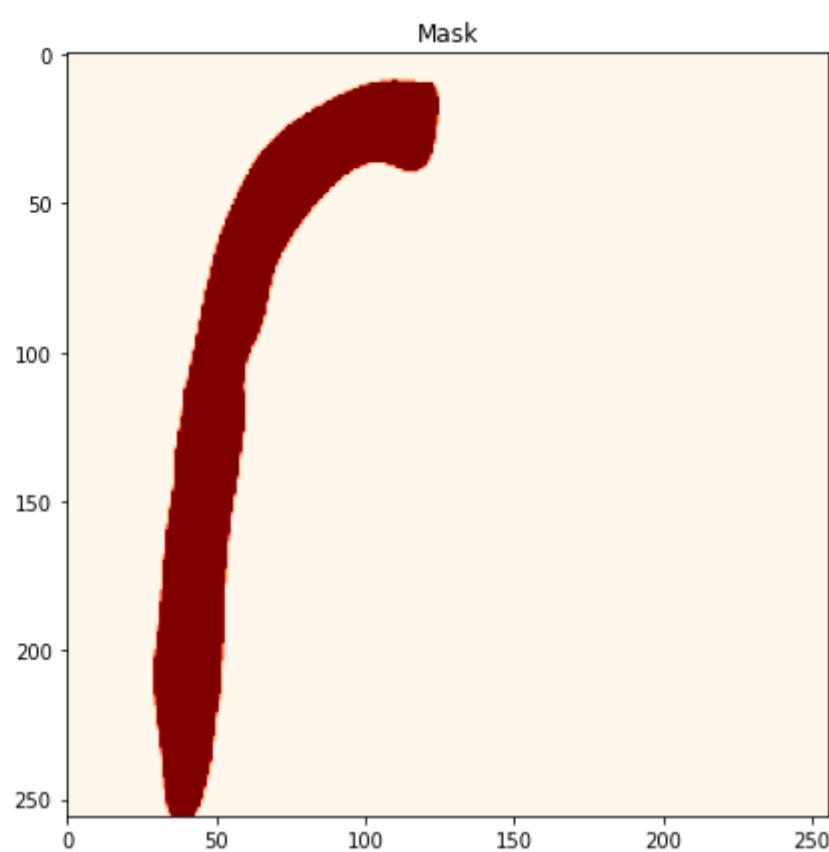
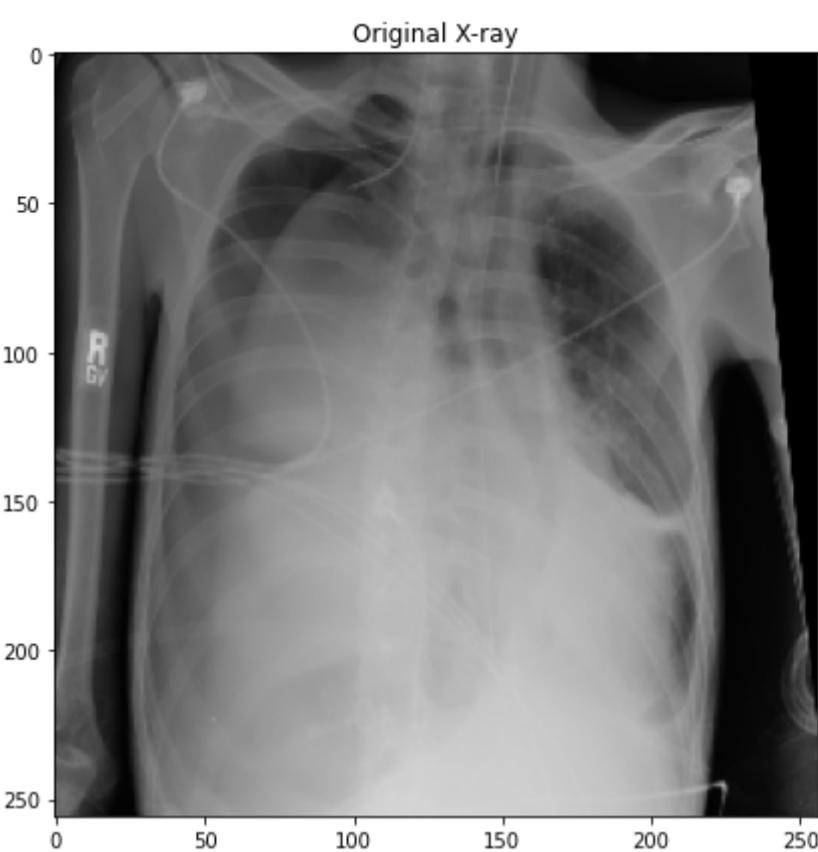
In [70]:

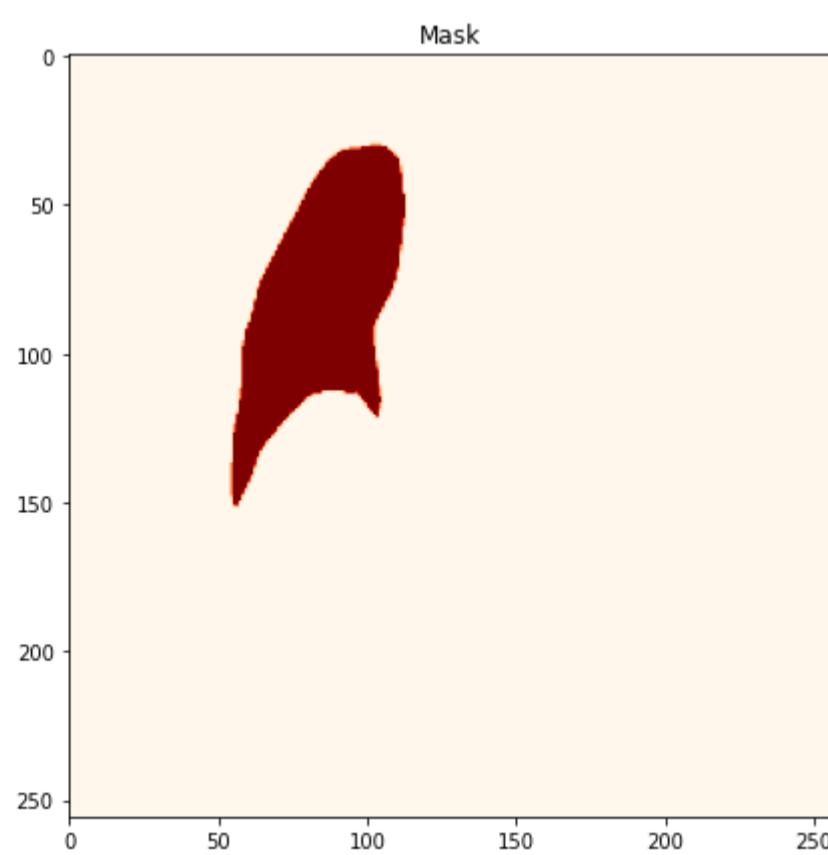
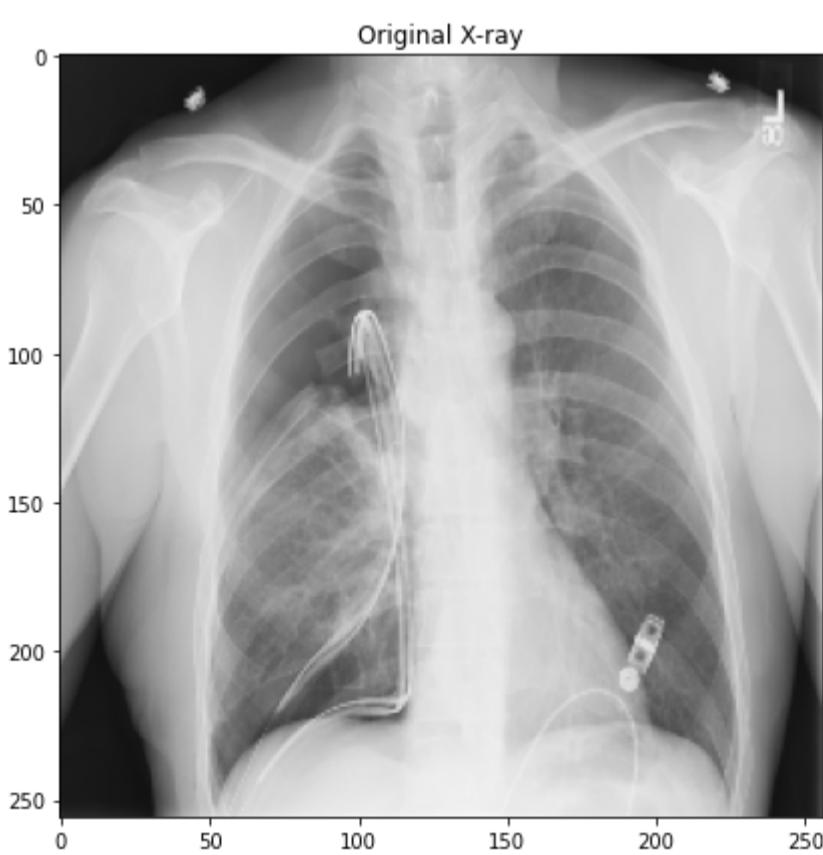
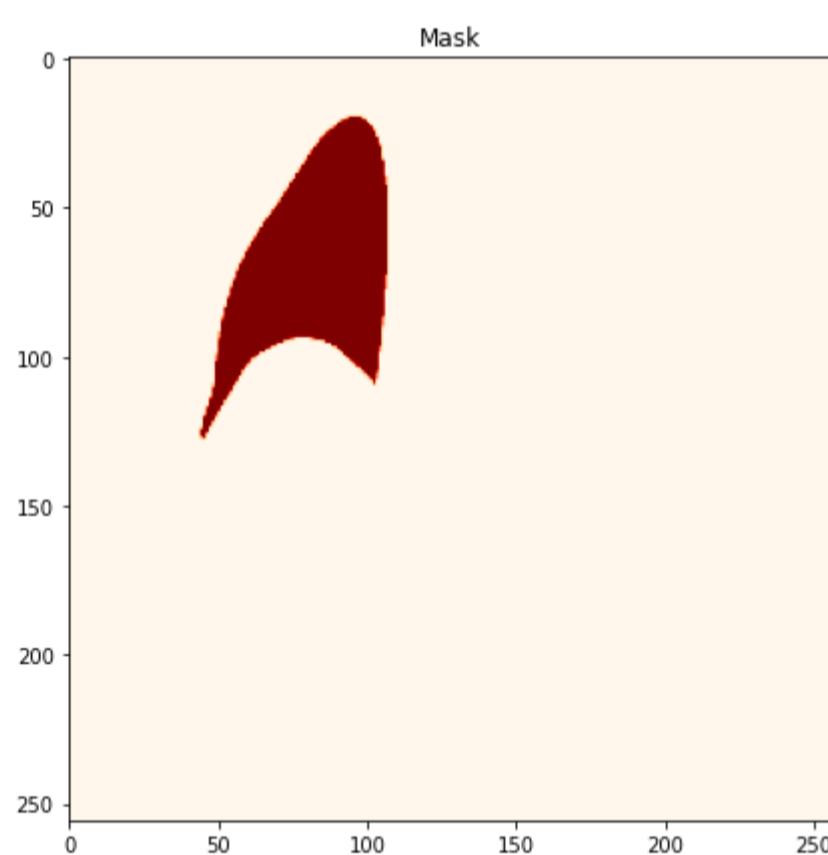
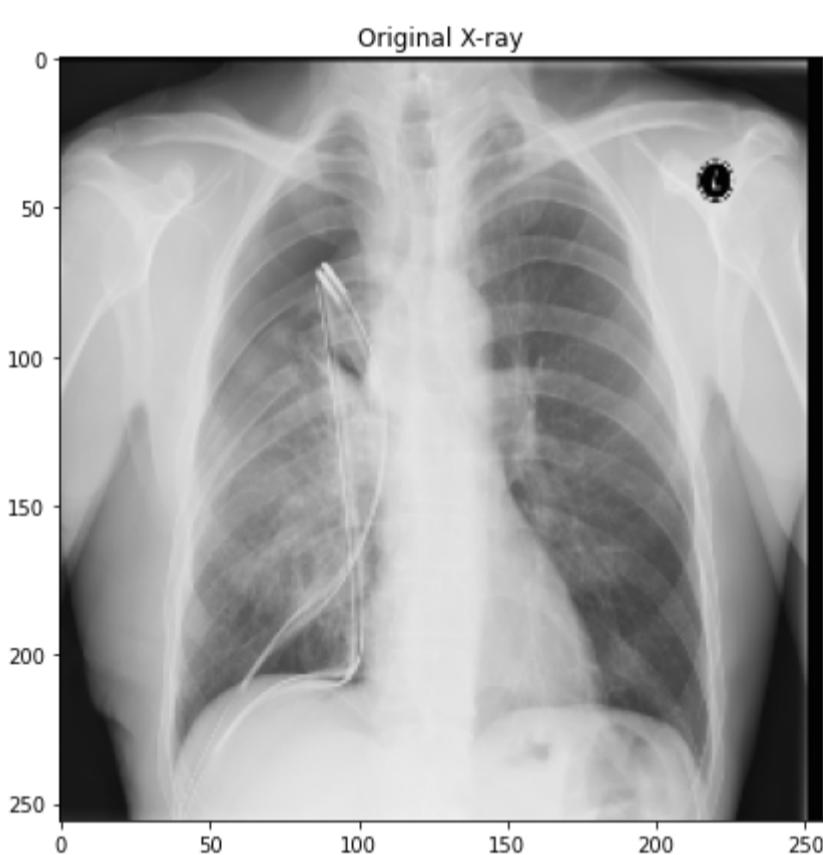
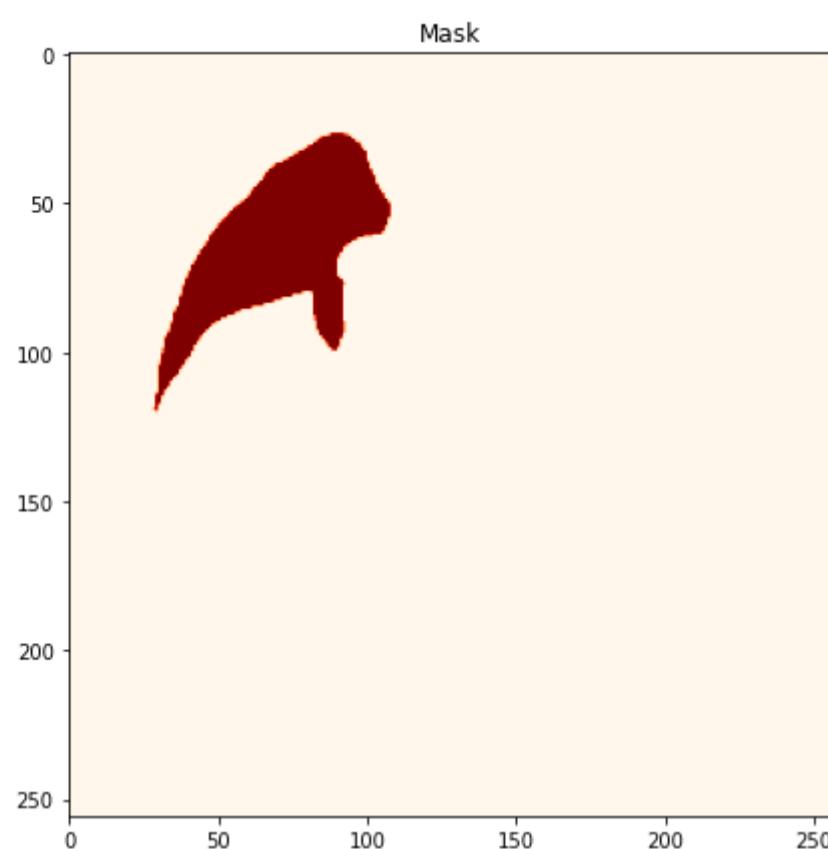
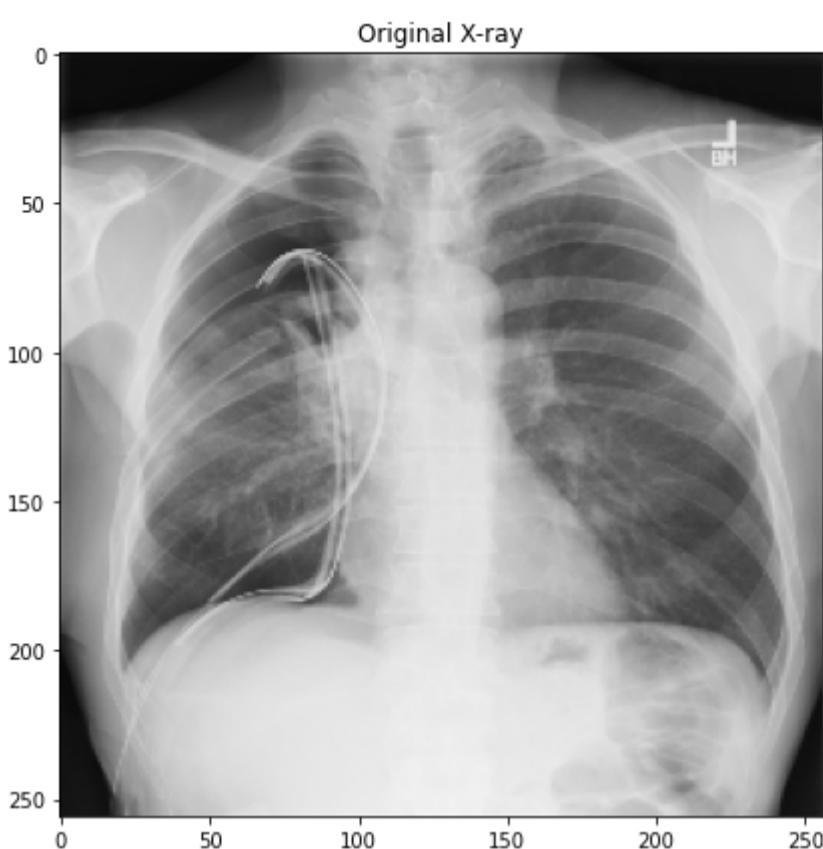
```
max_images = 12

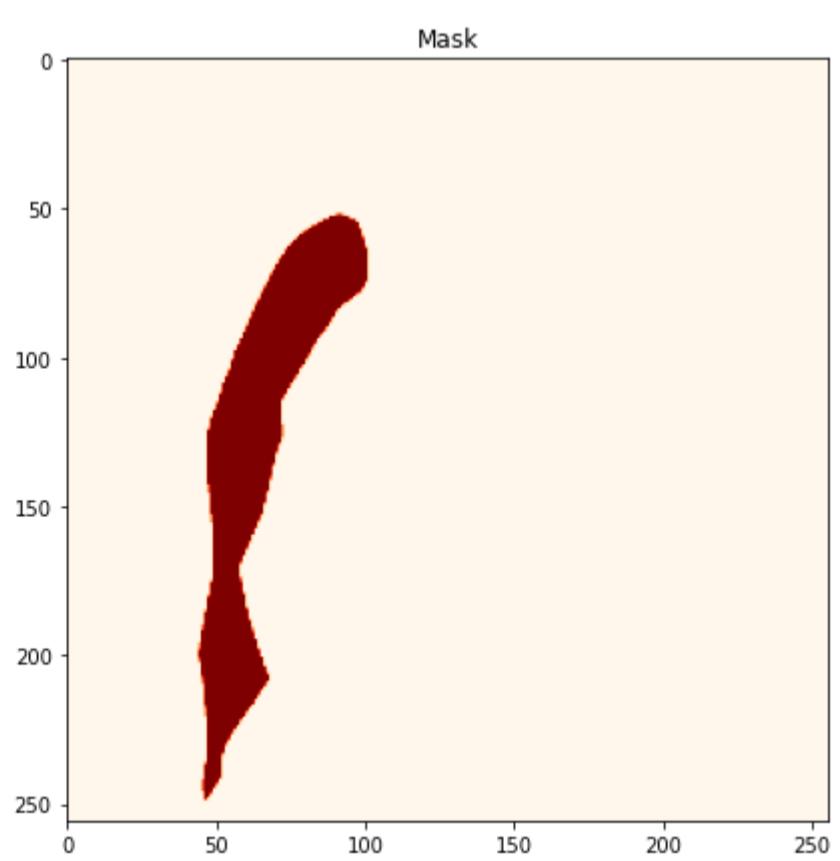
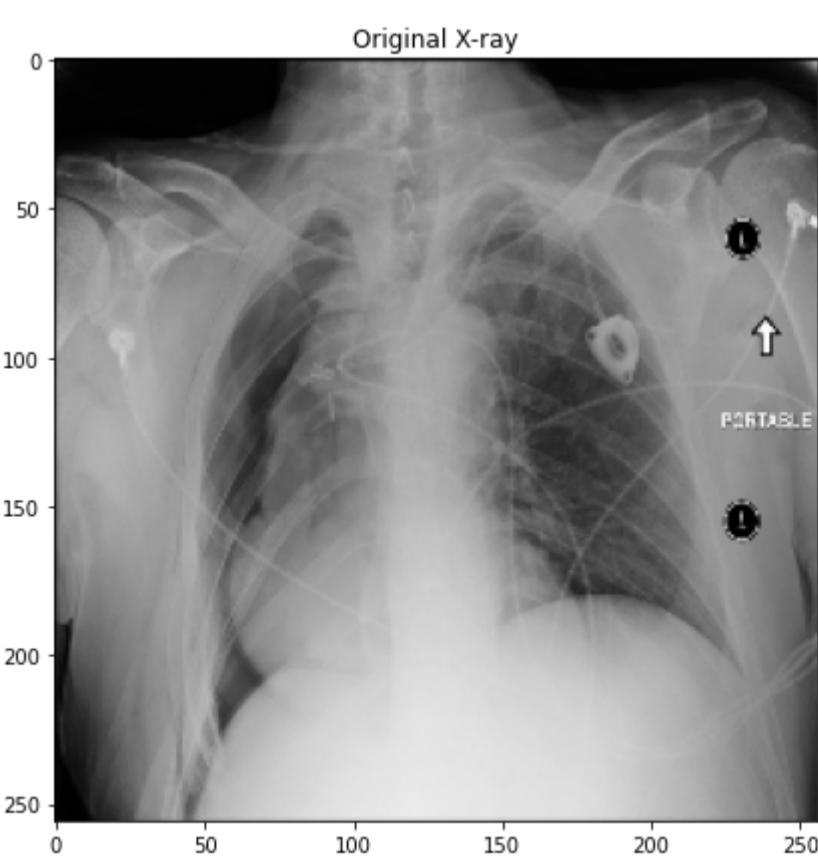
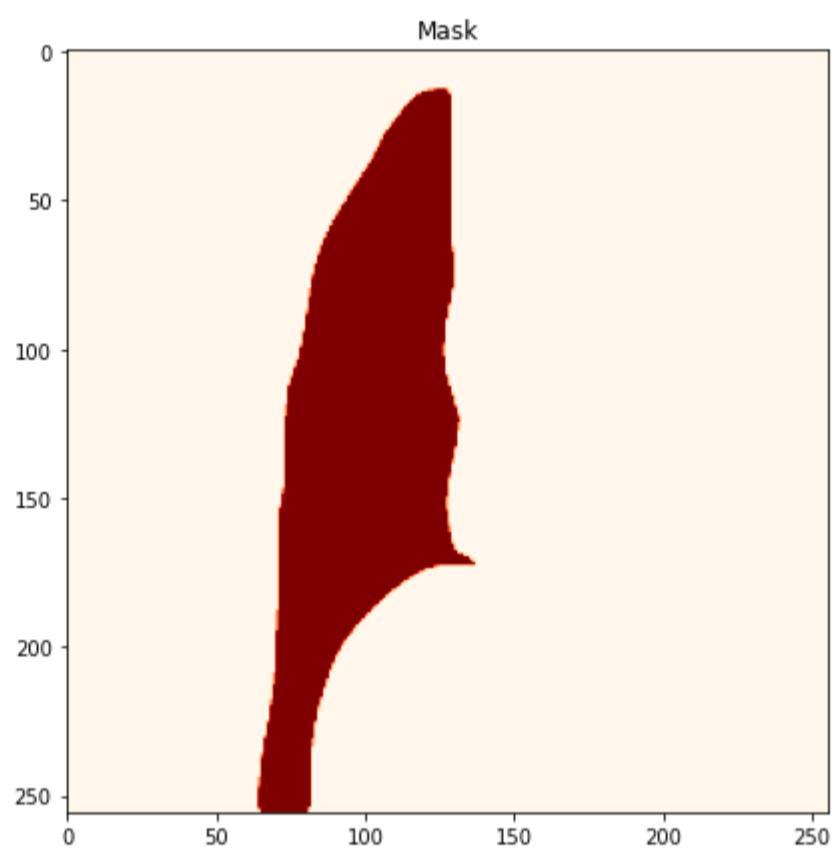
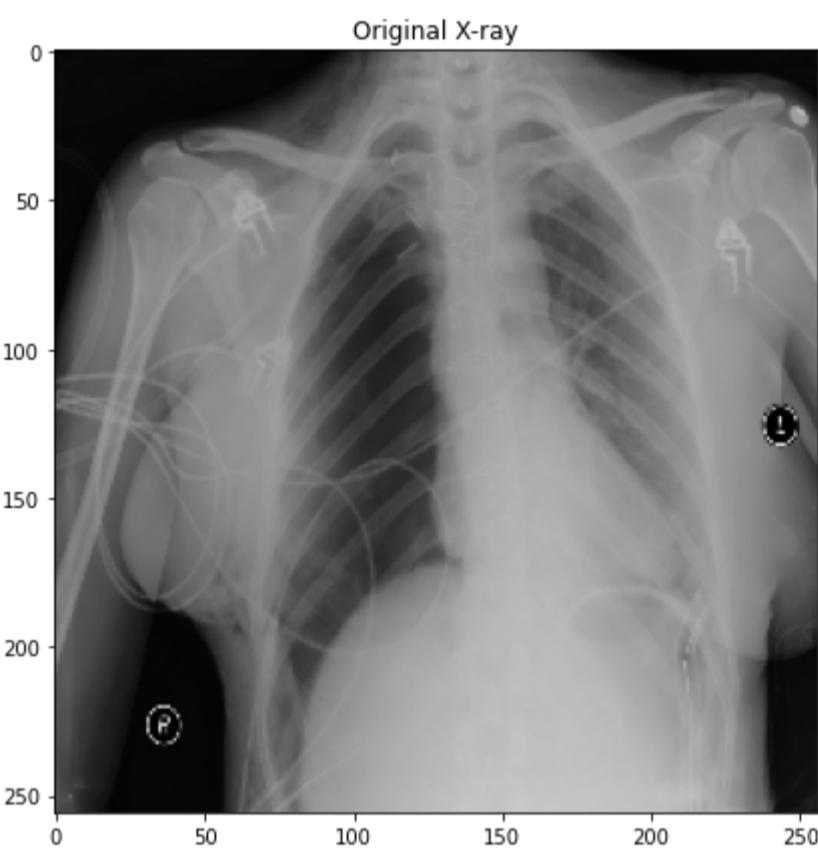
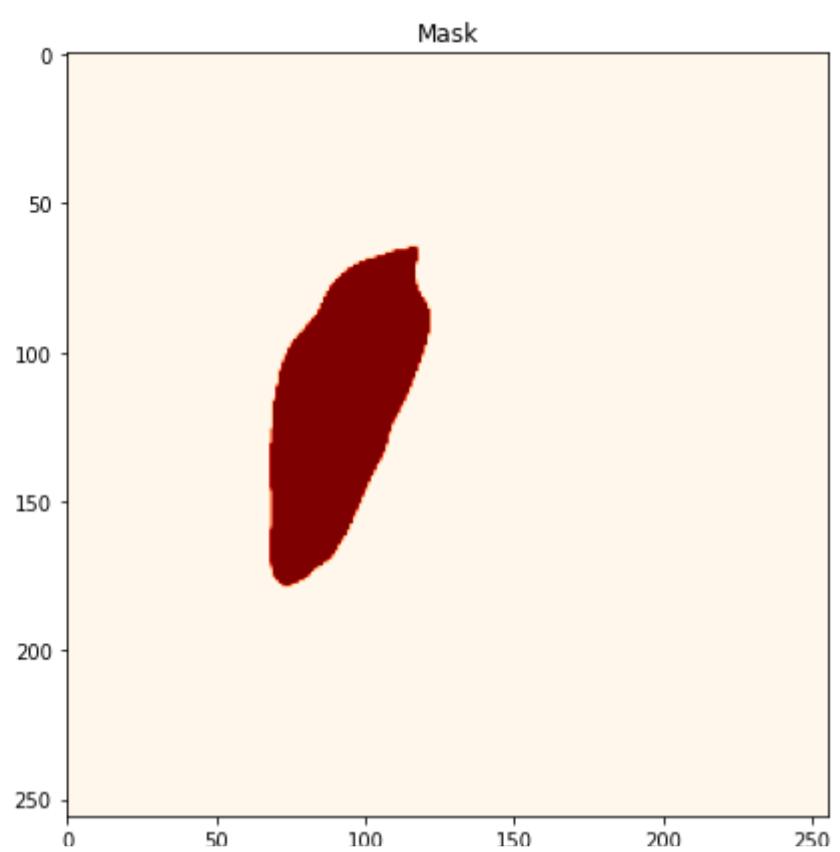
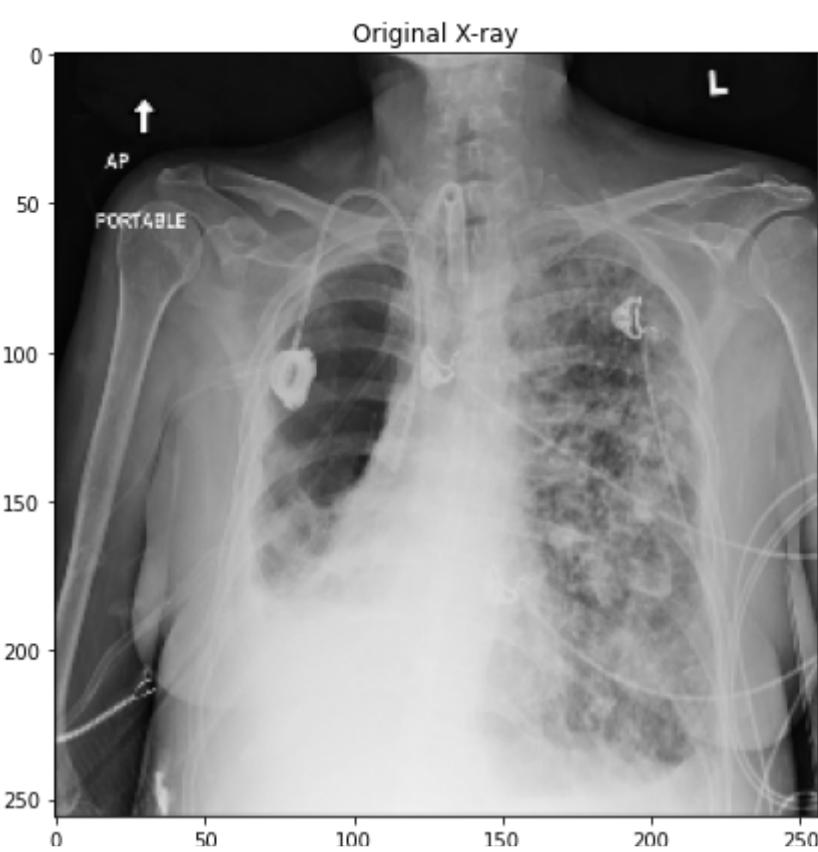
for i in range(max_images):
    img = low_score_images['ImagePath'].values[i]
    msk = low_score_images['MaskPath'].values[i]
    # read the original image
    image_orig = read_image(img)
    # read the original mask
    image_mask_orig = read_mask(msk)
    # plot Original X-ray
    fig = plt.figure(figsize=(15, 10))
    a = fig.add_subplot(1, 2, 1)
    plt.imshow(image_orig, cmap='bone')
    a.set_title(f"Original X-ray")
    plt.grid(False)
    # plt.axis("off")

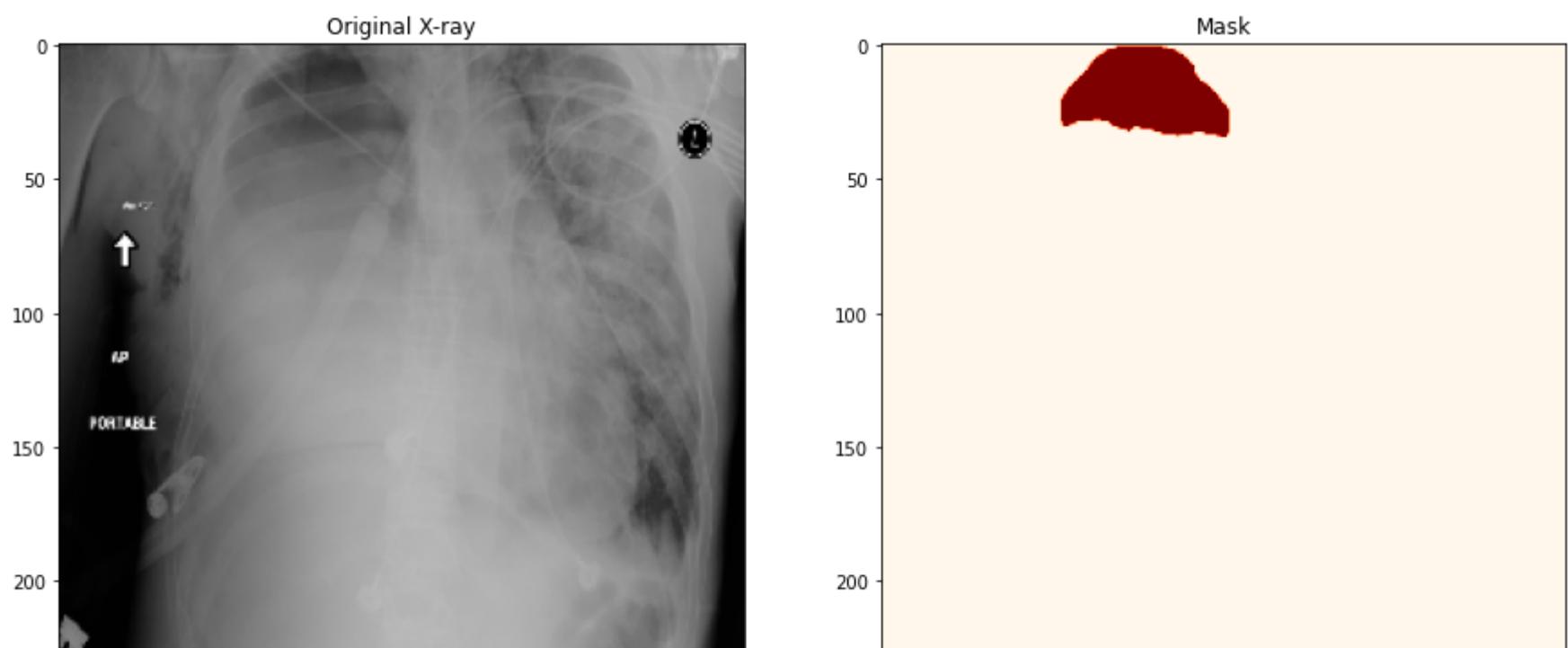
    # Plotting mask only
    a = fig.add_subplot(1, 2, 2)
    plt.imshow(np.squeeze(image_mask_orig), cmap='OrRd', alpha=1)
    a.set_title("Mask")
    plt.grid(False)
    # plt.axis("off")
    # break
```









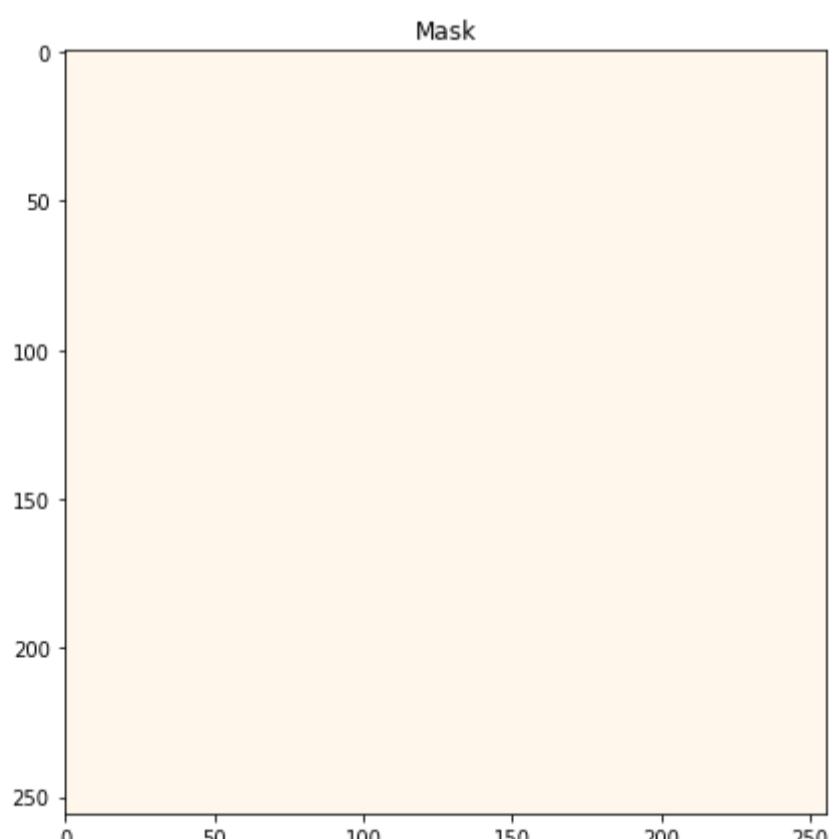
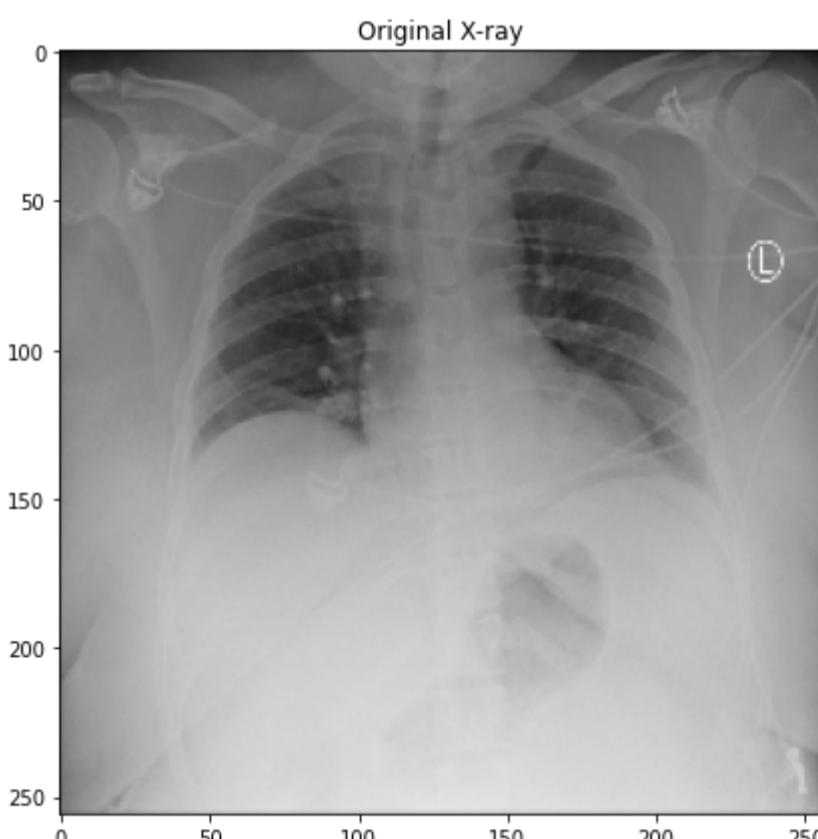
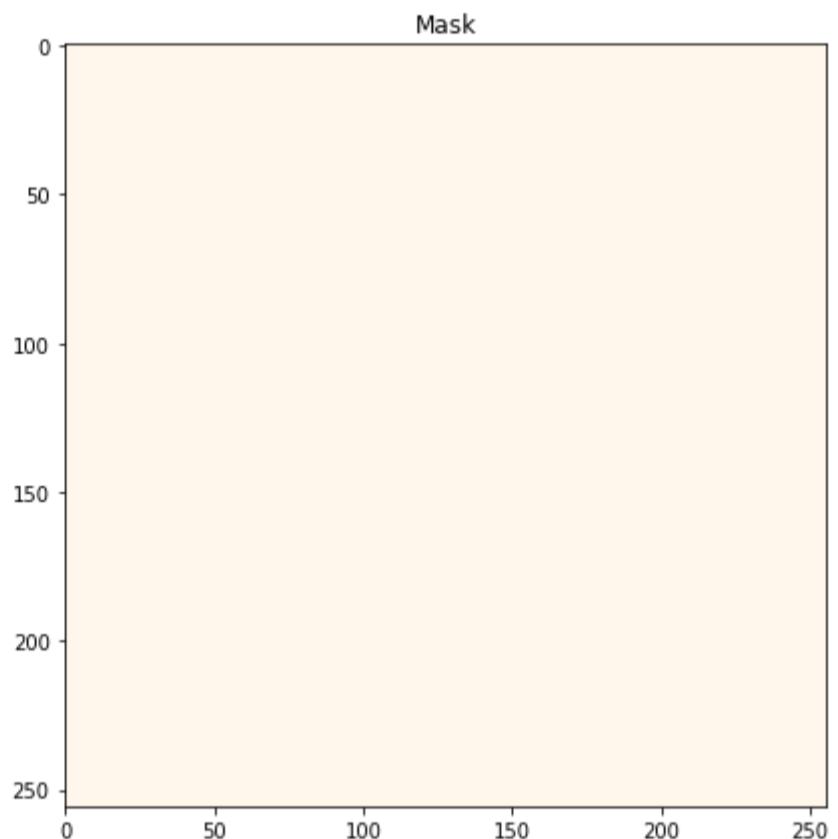
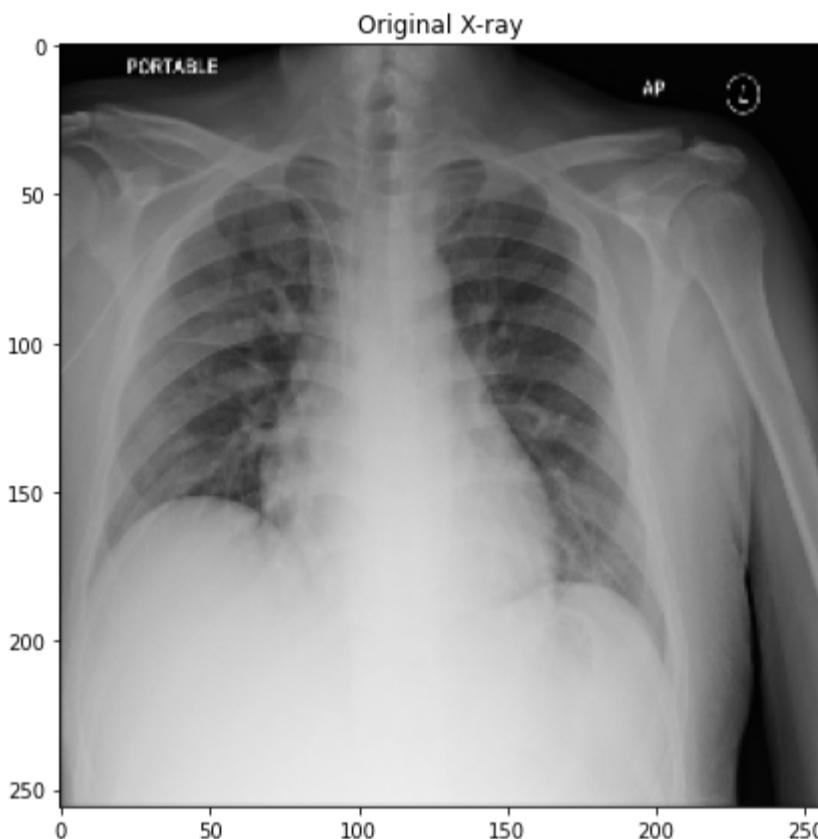


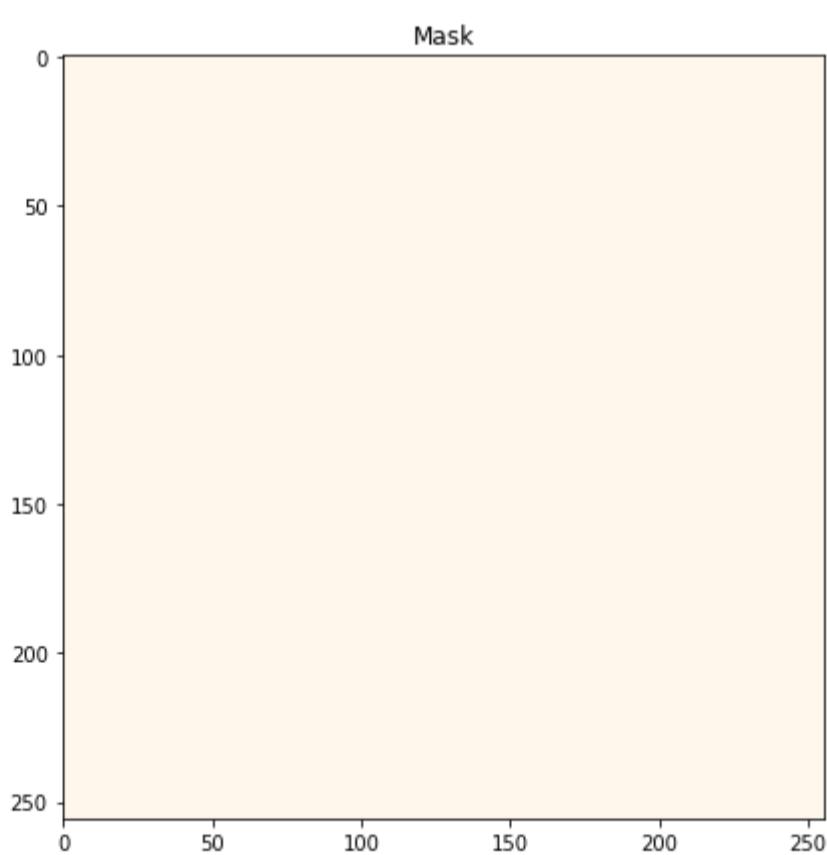
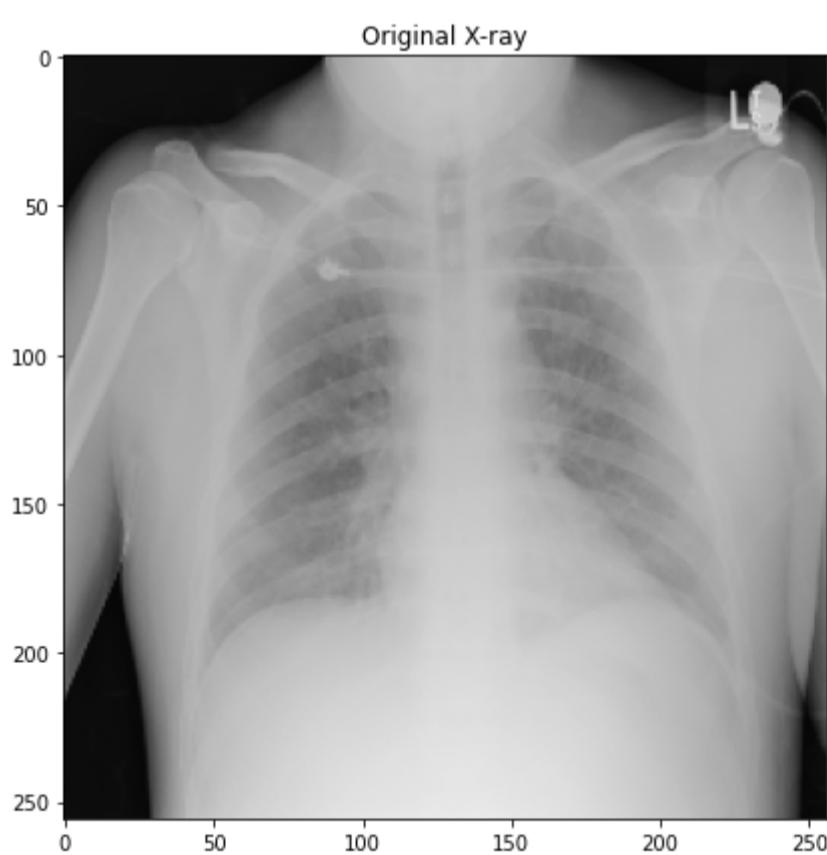
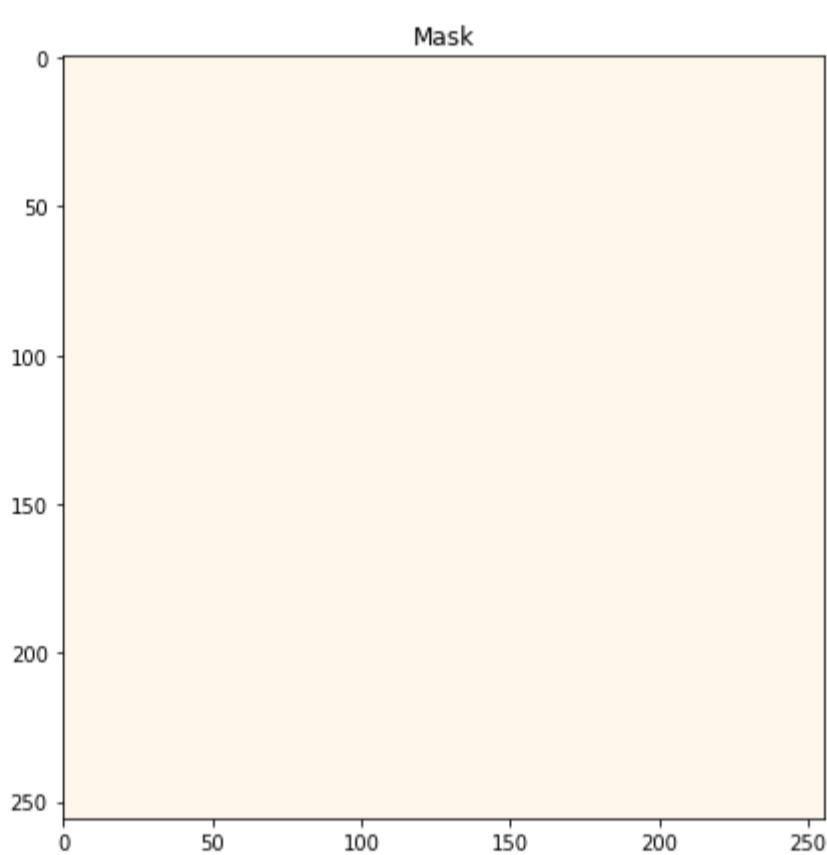
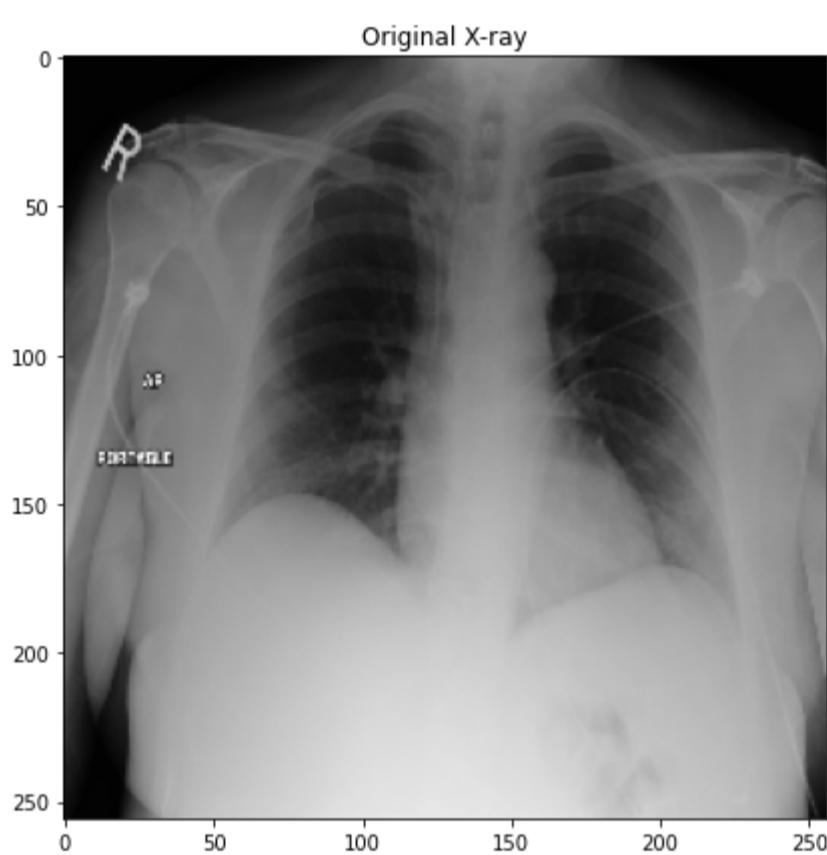
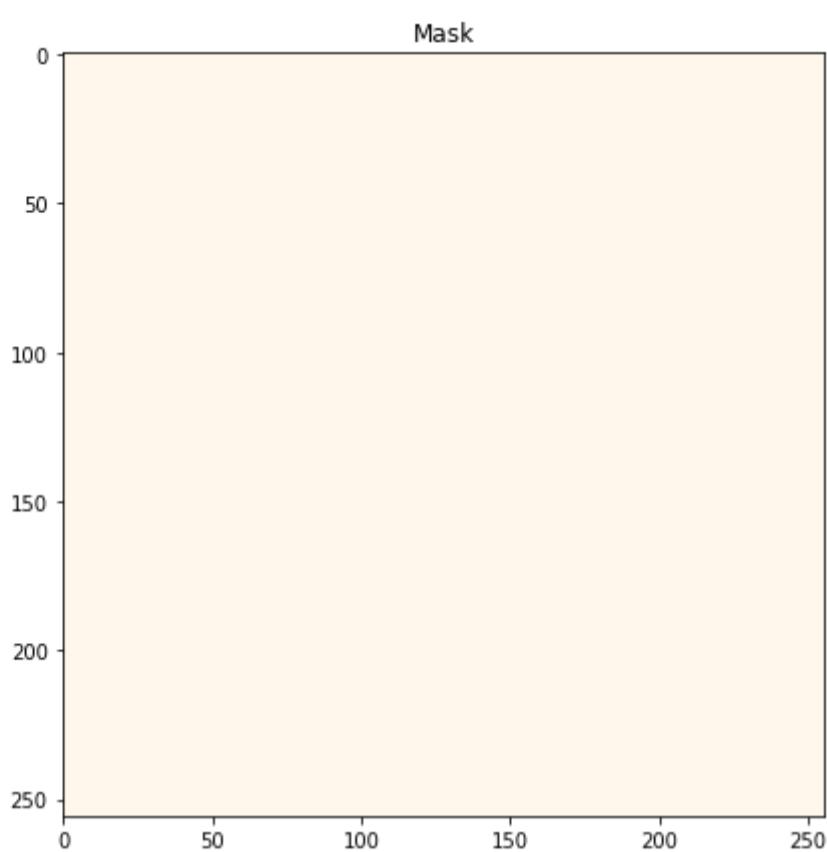
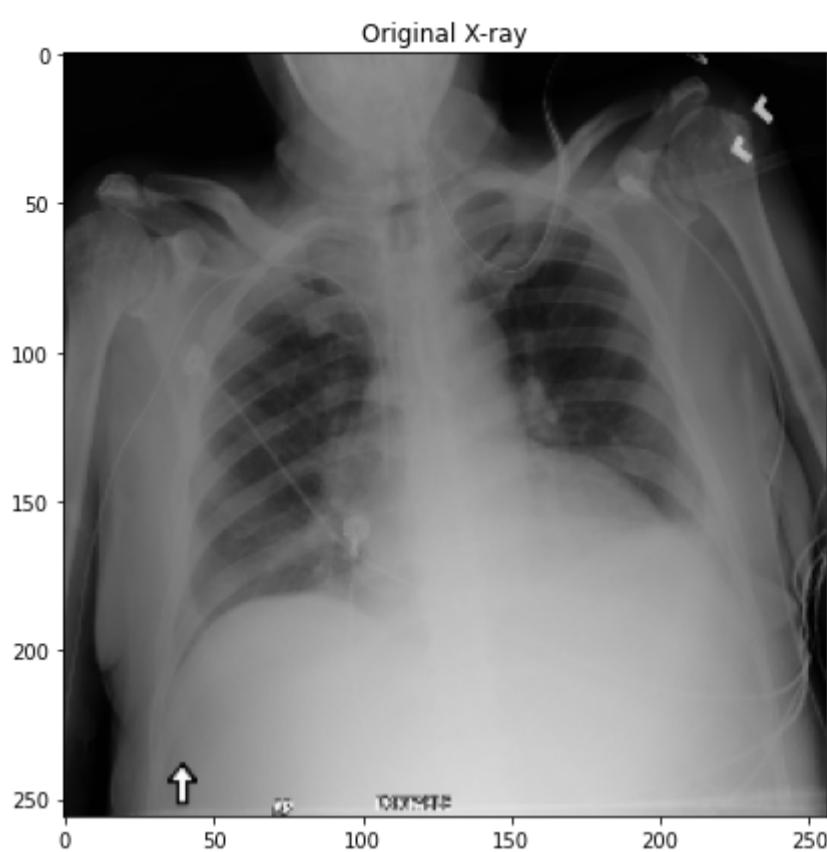
Medium Score images

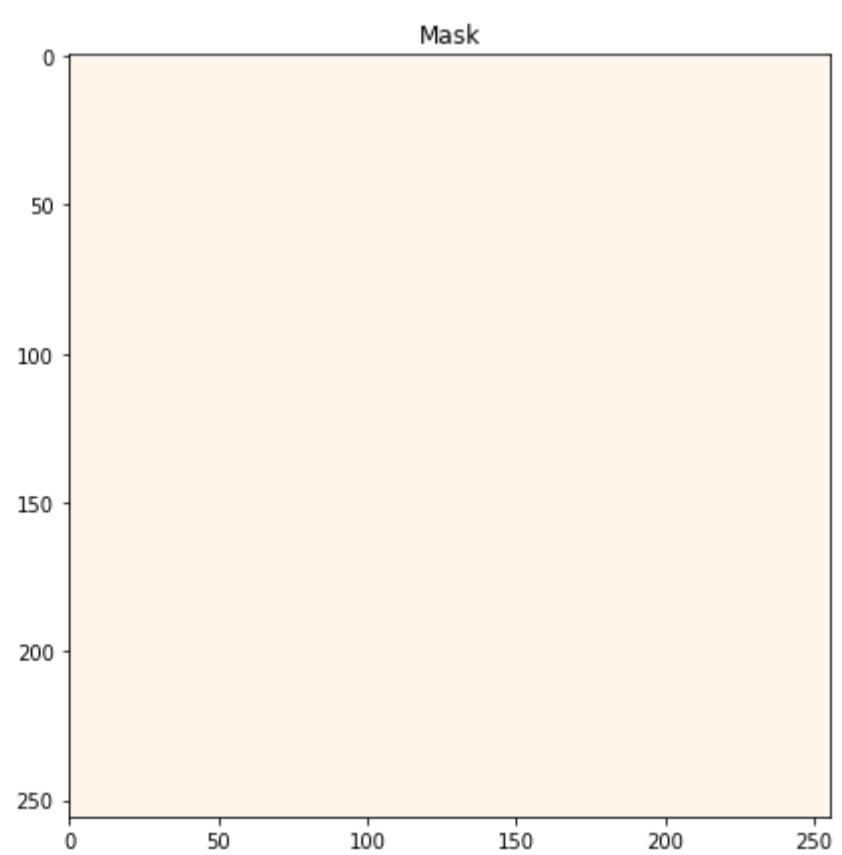
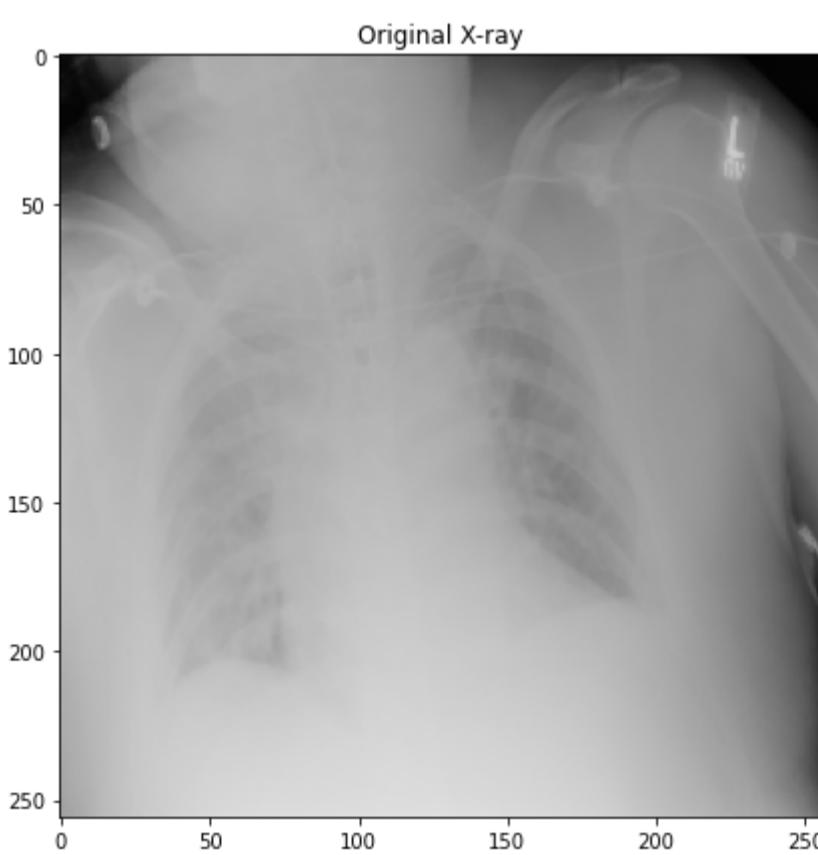
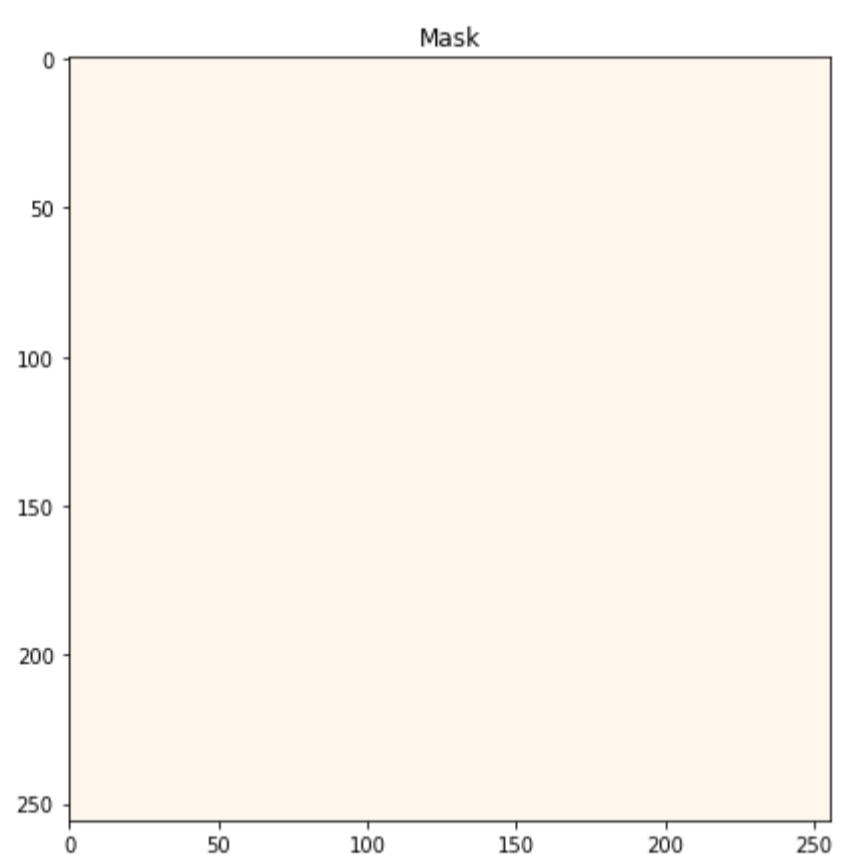
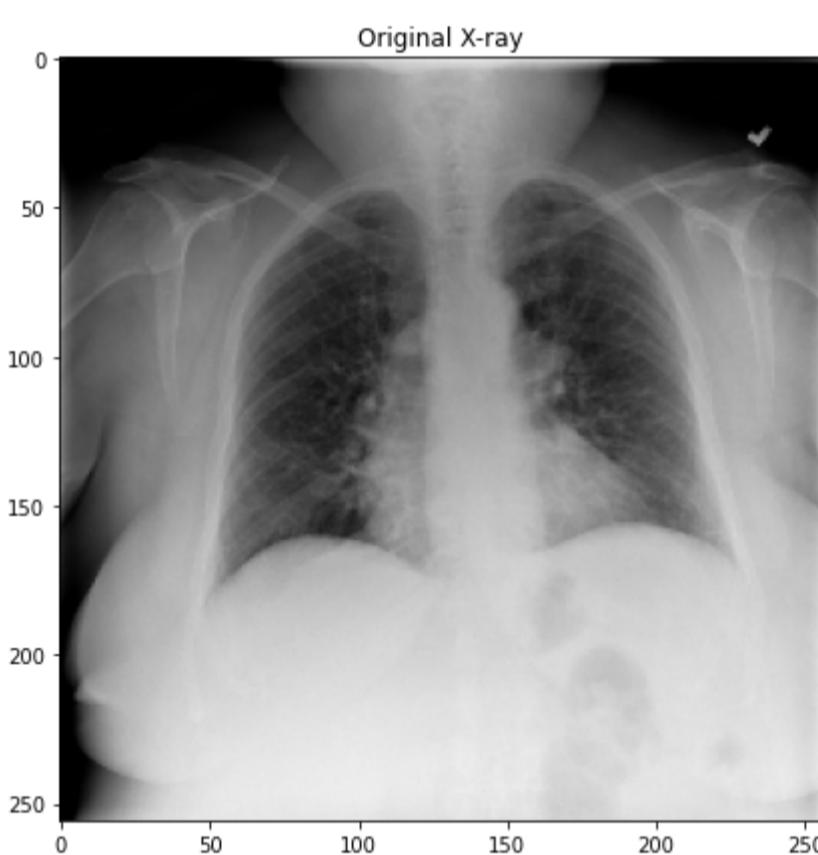
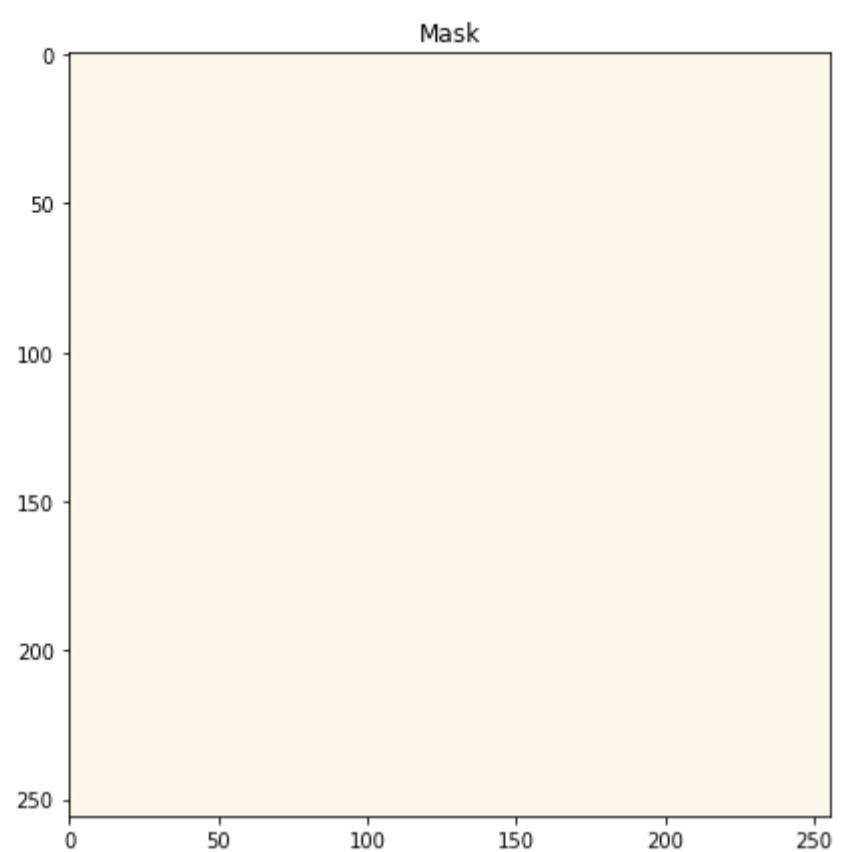
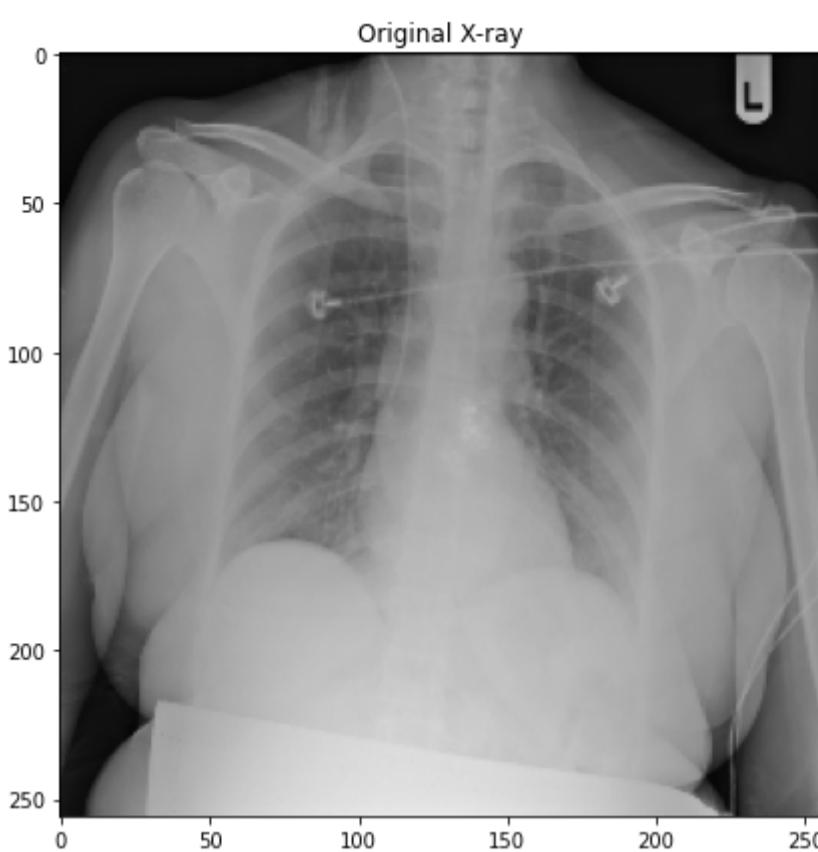
In [71]:

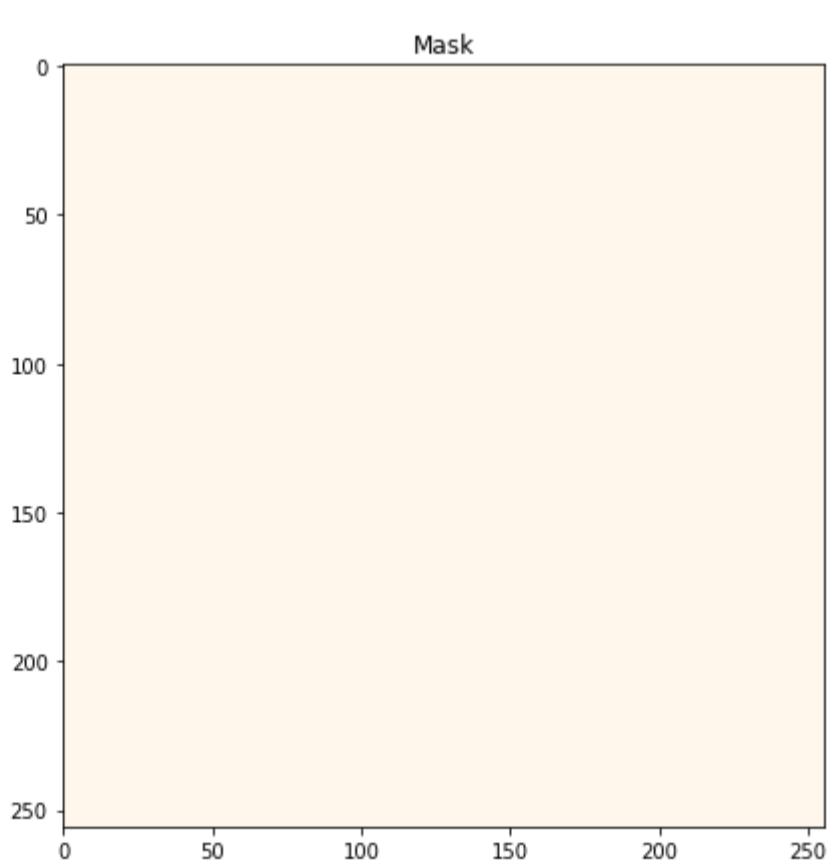
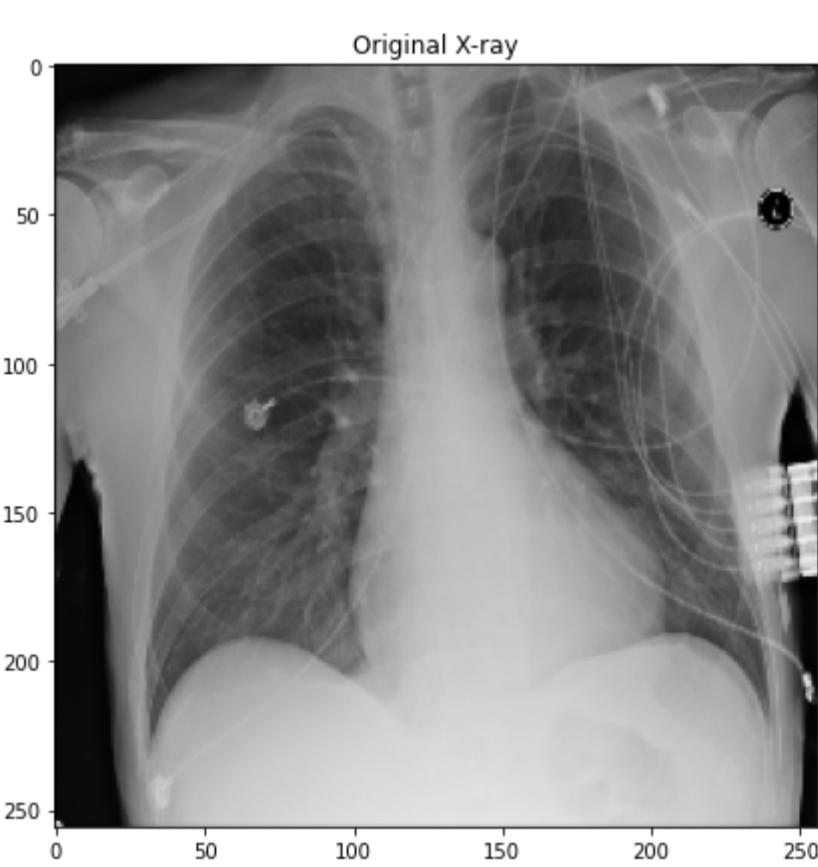
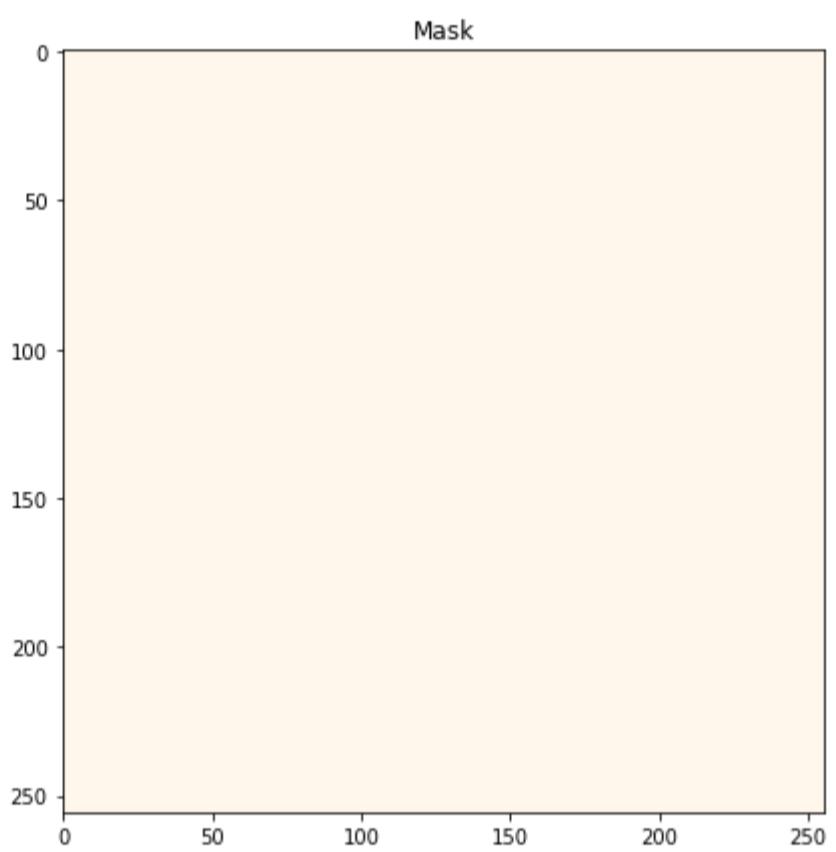
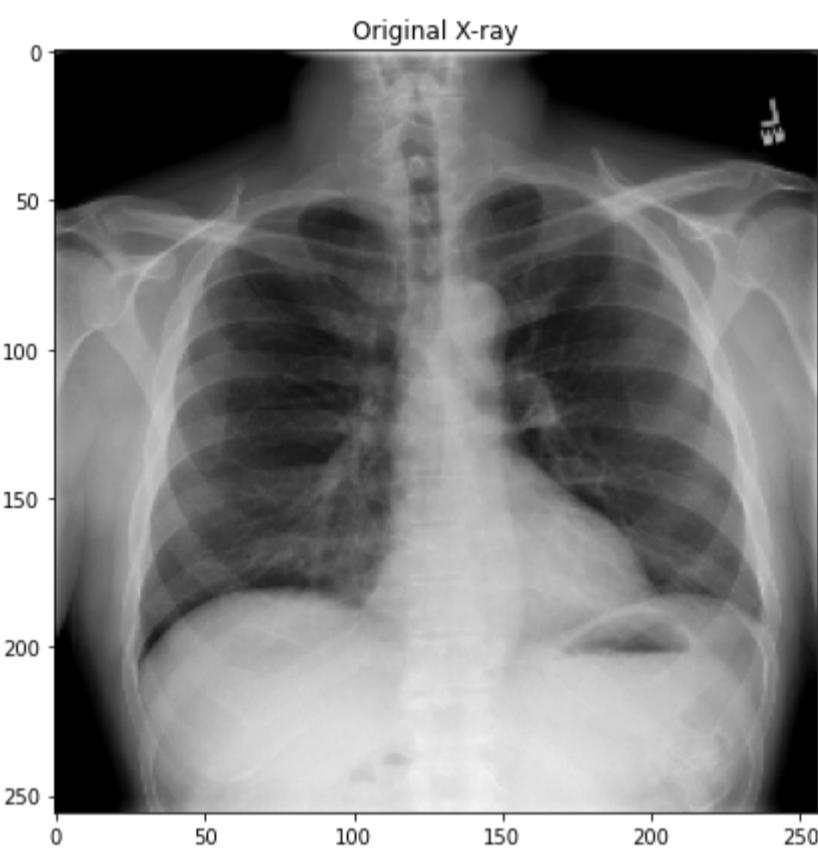
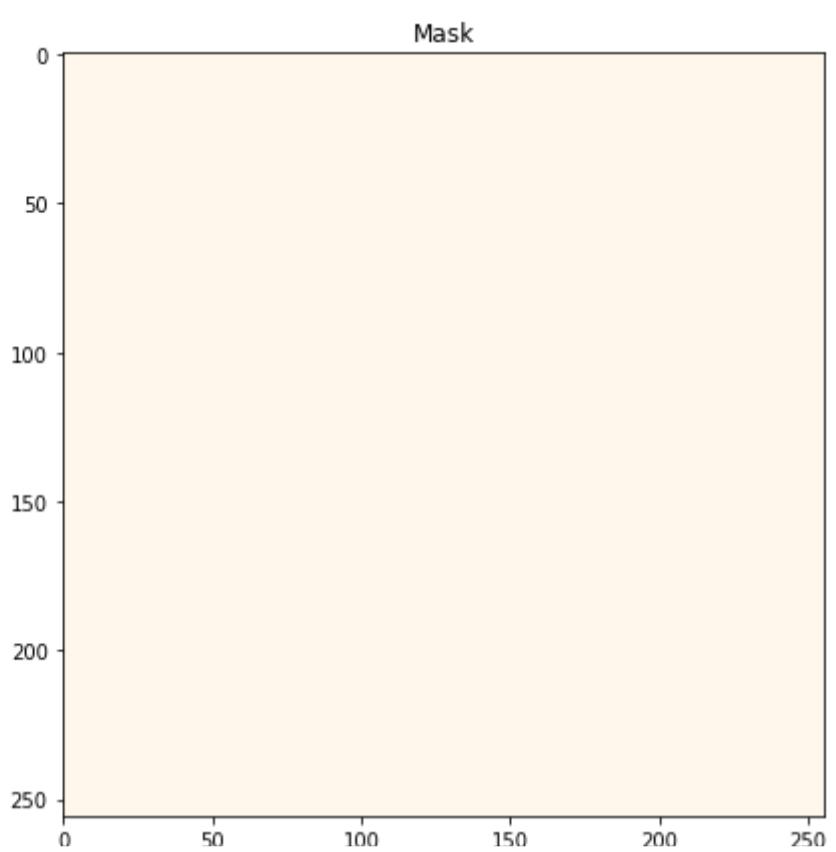
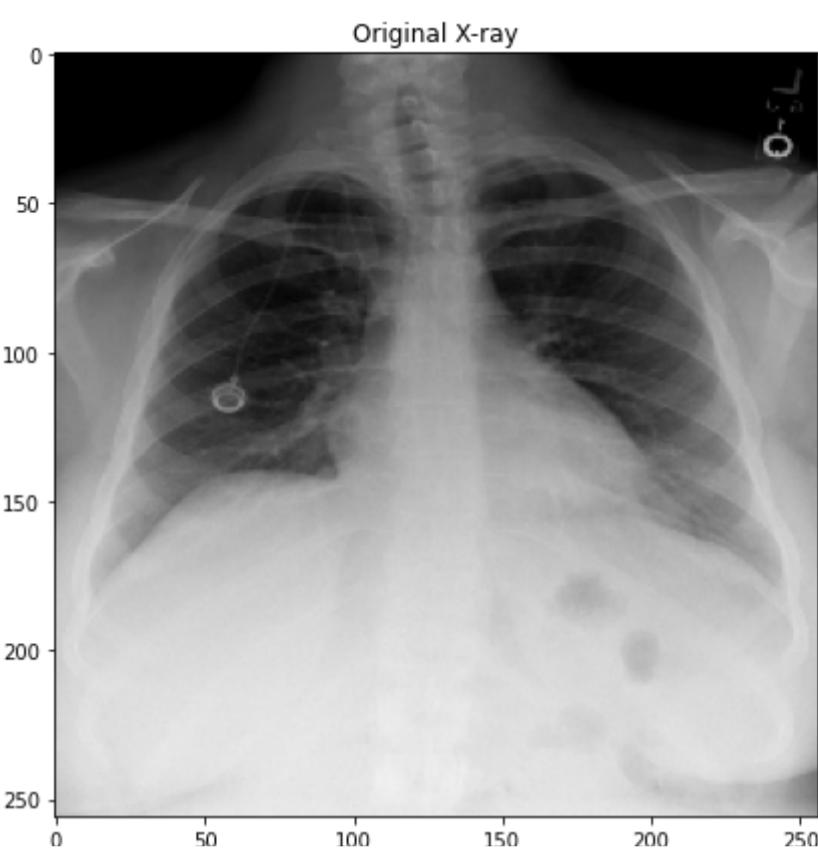
```
max_images = 12

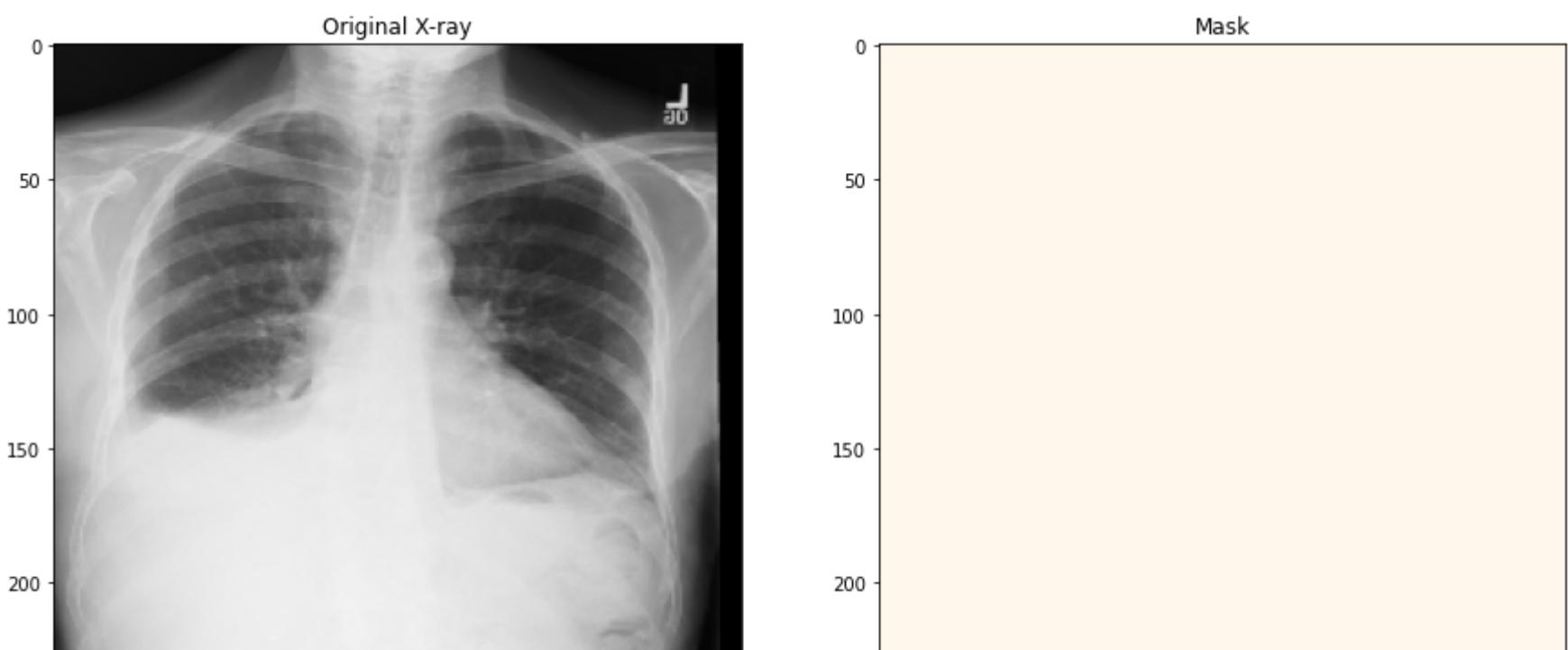
for i in range(max_images):
    img = medium_score_images['ImagePath'].values[i]
    msk = medium_score_images['MaskPath'].values[i]
    # read the original image
    image_orig = read_image(img)
    # read the original mask
    image_mask_orig = read_mask(msk)
    # plot Original X-ray
    fig = plt.figure(figsize=(15, 10))
    a = fig.add_subplot(1, 2, 1)
    plt.imshow(image_orig, cmap='bone')
    a.set_title(f"Original X-ray")
    plt.grid(False)
    # plt.axis("off")
    # Plotting mask only
    a = fig.add_subplot(1, 2, 2)
    plt.imshow(np.squeeze(image_mask_orig), cmap='OrRd', alpha=1)
    a.set_title("Mask")
    plt.grid(False)
    # plt.axis("off")
    # break
```











Observations:

- [1] The model is giving High score for empty masks that is the case for most images as I can see in above plotted images for high score.
- [2] And it is predicting medium score also for empty masks
- [3] Low score is for actual masks which are filled with some pneumothorax patches