Using ChexNet pretrained model to extarct features from images to be used in transfer learning for Encoder Decoder model

In [1]:
```
import tensorflow as tf
from tensorflow.keras.applications import densenet
from tensorflow.keras.applications.densenet import preprocess_input
from tensorflow.keras.layers import Dense, Dropout, Input, Conv2D
from tensorflow.keras.models import Model
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import os
import cv2
import tensorflow as tf
import re
import pickle
from PIL import Image
from skimage.transform import resize
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
from tqdm import tqdm
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import time
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, LSTM, Input, Embedding, Conv2D, C
import random
import datetime
from nltk.translate.bleu_score import sentence_bleu
```

In [37]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
train_data = pd.read_csv('/content/Final_Train_Data.csv')
test_data = pd.read_csv('/content/Final_Test_Data.csv')
cv_data = pd.read_csv('/content/Final_CV_Data.csv')
```

In [3]:
```
print('train data shape : ',train_data.shape)
train_data.head(2)
```

```
train data shape :  (2764, 4)
```

Out[3]:

	Person_id	Image1	Image2	Report
0	NLMCXR_png/CXR1082_IM-0058	NLMCXR_png/CXR1082_IM-0058-1001.png	NLMCXR_png/CXR1082_IM-0058-1001.png	startseq stable cardiomegaly . stable tortuos...
1	NLMCXR_png/CXR1883_IM-0572_0	NLMCXR_png/CXR1883_IM-0572-1001.png	NLMCXR_png/CXR1883_IM-0572-2001.png	startseq frontal and lateral views the chest s...

In [4]:
```
print('test data shape : ',test_data.shape)
test_data.head(2)
```

```
test data shape :  (389, 4)
```

Out[4]:

	Person_id	Image1	Image2	Report
0	NLMCXR_png/CXR2828_IM-1247_0	NLMCXR_png/CXR2828_IM-1247-1001.png	NLMCXR_png/CXR2828_IM-1247-2001.png	startseq heart size and mediastinal contour ar...
1	NLMCXR_png/CXR2777_IM-1217_0	NLMCXR_png/CXR2777_IM-1217-1001.png	NLMCXR_png/CXR2777_IM-1217-2001.png	startseq and lateral views the chest were obta...

In [5]:
```
print('CV data shape : ',cv_data.shape)
cv_data.head(2)
```

```
CV data shape :  (554, 4)
```

Out[5]:

	Person_id	Image1	Image2	Report
0	NLMCXR_png/CXR3072_IM-1433_0	NLMCXR_png/CXR3072_IM-1433-1001.png	NLMCXR_png/CXR3072_IM-1433-2001.png	startseq leftsided medication injection has it...
1	NLMCXR_png/CXR1864_IM-0558_0	NLMCXR_png/CXR1864_IM-0558-1001.png	NLMCXR_png/CXR1864_IM-0558-3001.png	startseq heart size borderline enlarged . no ...

In [6]:
```
chexNet = densenet.DenseNet121(include_top=False, weights = None,   input_sh
X = chexNet.output
X = Dense(14, activation="sigmoid", name="predictions")(X)
model = Model(inputs=chexNet.input, outputs=X)
```

In [7]:
```
! gdown "https://openi.nlm.nih.gov/imgs/collections/NLMCXR_png.tgz"
```

```
Downloading...
From: https://openi.nlm.nih.gov/imgs/collections/NLMCXR_png.tgz (https://openi.nlm.nih.gov/imgs/collections/NLMCXR_png.tgz)
To: /content/NLMCXR_png.tgz
100% 1.36G/1.36G [00:32<00:00, 42.4MB/s]
```

In [8]:
```
import shutil
shutil.unpack_archive("/content/NLMCXR_png.tgz","/content/NLMCXR_png")
```

In [9]:
```
! gdown "https://drive.google.com/u/0/uc?id=19BllaOvs2x5PLV_vlWMy4i8LapLb2j6
```

```
Downloading...
From: https://drive.google.com/u/0/uc?id=19BllaOvs2x5PLV_vlWMy4i8LapLb2j6b&export=download (https://drive.google.com/u/0/uc?id=19BllaOvs2x5PLV_vlWMy4i8LapLb2j6b&export=download)
To: /content/brucechou1983_CheXNet_Keras_0.3.0_weights.h5
29.1MB [00:00, 178MB/s]
```

In [10]:
```
#loadind pretrained weights for ChexNet model
model.load_weights('brucechou1983_CheXNet_Keras_0.3.0_weights.h5')
```

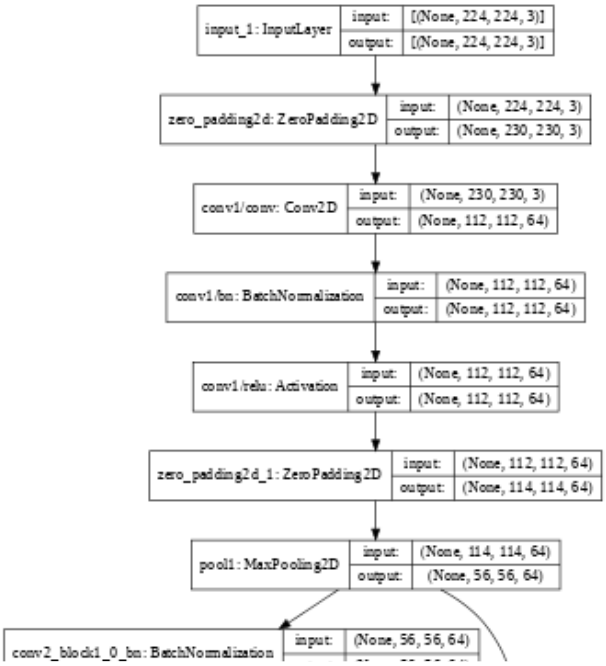
In [11]:
```
chexNet = Model(inputs = model.input, outputs = model.layers[-2].output)
```

In [12]:
```
chexNet.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)      0
_________________________________________________________________
zero_padding2d (ZeroPadding2D)  (None, 230, 230, 3)  0
[0]
_________________________________________________________________
conv1/conv (Conv2D)          (None, 112, 112, 64)      9408
2d[0][0]
_________________________________________________________________
conv1/bn (BatchNormalization)  (None, 112, 112, 64)    256
[0][0]
```

In [13]:
```
1 tf.keras.utils.plot_model(chexNet,show_shapes=True, dpi = 42)
```

Out[13]:

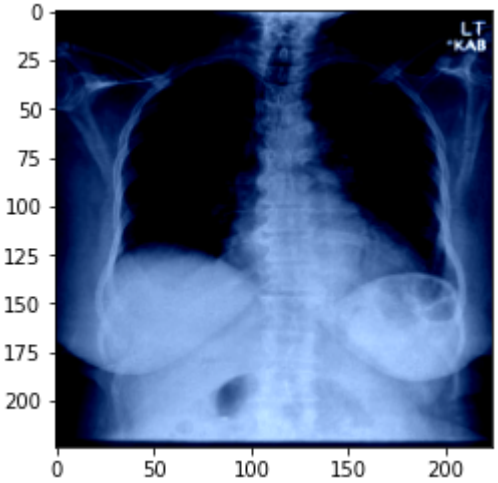


In [14]:
```
1 def load_image(img_name):
2     image = Image.open(img_name)
3     X = np.asarray(image.convert("RGB"))
4     X = np.asarray(X)
5     X = preprocess_input(X)
6     X = resize(X, (224,224,3))
7     X = np.expand_dims(X, axis=0)
8     X = np.asarray(X)
9     return X
```

In [15]:
```
1 img = load_image('NLMCXR_png/CXR1082_IM-0058-1001.png')
2 plt.imshow(img[0])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
```

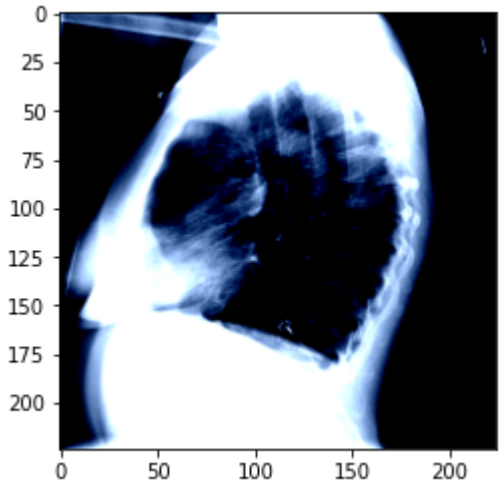
Out[15]:
```
<matplotlib.image.AxesImage at 0x7fd32ae259b0>
```



In [16]:
```
1 img = load_image('NLMCXR_png/CXR2828_IM-1247-2001.png')
2 plt.imshow(img[0])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
```

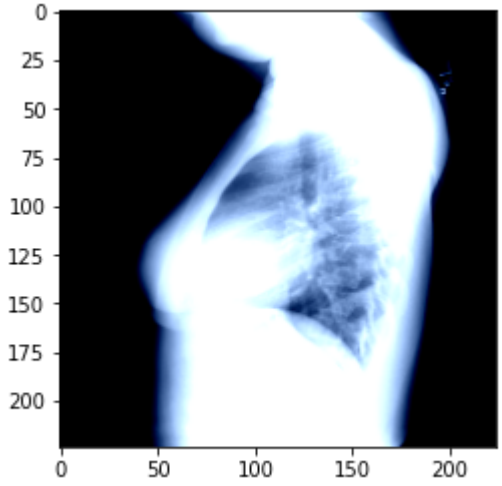
Out[16]: <matplotlib.image.AxesImage at 0x7fd328579da0>



In [17]:
```
1 img = load_image('NLMCXR_png/CXR1_1_IM-0001-3001.png')
2 plt.imshow(img[0])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
```

Out[17]: <matplotlib.image.AxesImage at 0x7fd32ae5c358>



In [18]:
```
Xnet_features = {}
for key, img1, img2, finding in tqdm(train_data.values):
    i1 = load_image(img1)
    img1_features = chexNet.predict(i1)
    i2 = load_image(img2)
    img2_features = chexNet.predict(i2)
    input_ = np.concatenate((img1_features, img2_features), axis=1)
    Xnet_features[key] = input_

for key, img1, img2, finding in tqdm(test_data.values):
    i1 = load_image(img1)
    img1_features = chexNet.predict(i1)
    i2 = load_image(img2)
    img2_features = chexNet.predict(i2)
    input_ = np.concatenate((img1_features, img2_features), axis=1)
    Xnet_features[key] = input_

for key, img1, img2, finding in tqdm(cv_data.values):
    i1 = load_image(img1)
    img1_features = chexNet.predict(i1)
    i2 = load_image(img2)
    img2_features = chexNet.predict(i2)
    input_ = np.concatenate((img1_features, img2_features), axis=1)
    Xnet_features[key] = input_
```

```
100%|██████████| 2764/2764 [09:12<00:00,  5.01it/s]
100%|██████████| 389/389 [01:14<00:00,  5.19it/s]
100%|██████████| 554/554 [01:47<00:00,  5.17it/s]
```

In [38]:
```
Xnet_features['NLMCXR_png/CXR236_IM-0924_0'].shape
```

Out[38]: (1, 2048)

In [39]:
```
# save the file for future use
f = open('Image_features_ecoder_decoder.pickle','wb')
pickle.dump(Xnet_features, f)
f.close()
```

In [40]:
```
len(Xnet_features)
```

Out[40]: 3707

Encoder Decoder Model

In [41]:
```
X_train = train_data['Person_id']
X_test = test_data['Person_id']
X_cv = cv_data['Person_id']
y_train = train_data['Report']
y_test =  test_data['Report']
y_cv =  cv_data['Report']
```

In [42]:
```
cheXnet_Features = Xnet_features
```

In [43]:
```
tokenizer = Tokenizer(filters='!"#$%&()*+,-/:;<=>?@[\\]^_`{|}~\t\n')
tokenizer.fit_on_texts(y_train.values)
```

In [44]:
```
pading_size = 153   # Max Length
```

In [45]:
```
vocab_size = len(tokenizer.word_index.keys()) + 1
```

In [46]:
```
f = open('/content/drive/MyDrive/glove_vectors','rb') # 300d glove vectors
glove_vectors = pickle.load(f)
f.close()
```

In [47]:
```
embedding_matrix = np.zeros((vocab_size,300))
for word, i in tokenizer.word_index.items():
    if word in glove_vectors.keys():
        vec = glove_vectors[word]
        embedding_matrix[i] = vec
    else:
        continue
```

In [48]:
```
BATCH_SIZE = 12
```

In [49]:
```
def load_image(id_, report):
    '''Loads the Image Features with their corresponding Ids'''
    img_feature = cheXnet_Features[id_.decode('utf-8')][0]
    return img_feature, report
```

In [50]:
```
def dataset_generator(img_name, caption):

    dataset = tf.data.Dataset.from_tensor_slices((img_name, caption))

  # Use map to load the numpy files in parallel
    dataset = dataset.map(lambda item1, item2: tf.numpy_function(load_image,
                          num_parallel_calls=tf.data.experimental.AUTOTUNE)

  # Shuffle and batch
    dataset = dataset.shuffle(500).batch(BATCH_SIZE).prefetch(buffer_size=tf
    return dataset
```

In [51]:
```
train_generator = dataset_generator(X_train, y_train)
cv_generator = dataset_generator(X_cv, y_cv)
```

In [52]:
```
def bytes_to_string(arr):
    '''The generator gives provides data in bytes. This function converts th
    for i in range(len(arr)):
        arr[i] = arr[i].decode('utf-8')
    return arr
```

In [53]:

```
def convert(images, reports):
    '''This function takes the batch of data and converts them into a new da
    imgs = []
    in_reports = []
    out_reports = []
    for i in range(len(images)):
        sequence = [tokenizer.word_index[e] for e in reports[i].split() if e
      #  print(sequence)
        for j in range(1,len(sequence)):

            in_seq = sequence[:j]
            out_seq = sequence[j]
            out_seq = tf.keras.utils.to_categorical(out_seq, num_classes=voc

            imgs.append(images[i])
          #  print(in_seq)
            in_reports.append(in_seq)
          # print(out_seq)
            out_reports.append(out_seq)

    return np.array(imgs), np.array(in_reports), np.array(out_reports)
```

In [54]:

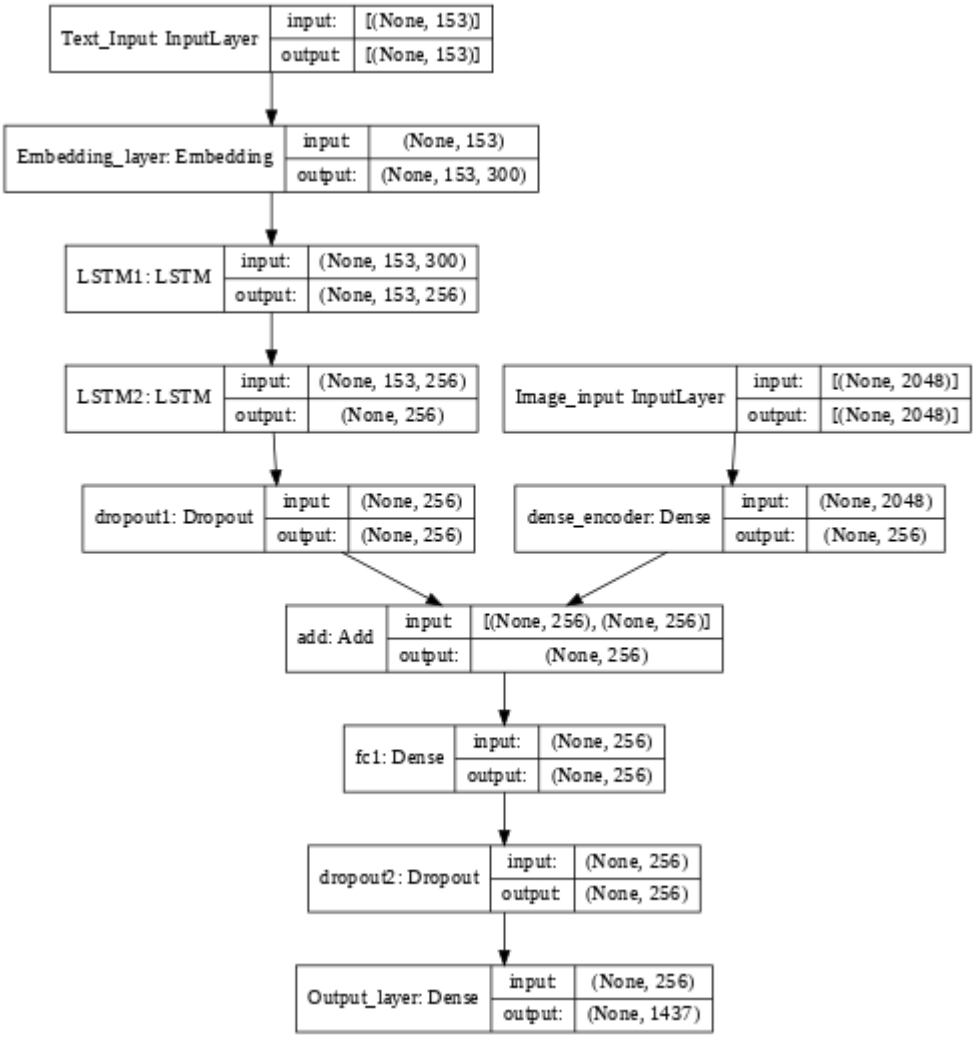
```
input1 = Input(shape=(2048), name='Image_input')
dense1 = Dense(256, kernel_initializer=tf.keras.initializers.glorot_uniform(

input2 = Input(shape=(153), name='Text_Input')
embedding_layer = Embedding(input_dim = vocab_size, output_dim = 300, input_
                weights=[embedding_matrix], name="Embedding_layer")
emb = embedding_layer(input2)

LSTM1 = LSTM(units=256, activation='tanh', recurrent_activation='sigmoid', u
            kernel_initializer=tf.keras.initializers.glorot_uniform(seed=23)
            recurrent_initializer=tf.keras.initializers.orthogonal(seed=7),
            bias_initializer=tf.keras.initializers.zeros(), return_sequences
#LSTM1_output = LSTM1(emb)

LSTM2 = LSTM(units=256, activation='tanh', recurrent_activation='sigmoid', u
            kernel_initializer=tf.keras.initializers.glorot_uniform(seed=23)
            recurrent_initializer=tf.keras.initializers.orthogonal(seed=7),
            bias_initializer=tf.keras.initializers.zeros(), name="LSTM2")
LSTM2_output = LSTM2(LSTM1)

dropout1 = Dropout(0.5, name='dropout1')(LSTM2_output)

dec =  tf.keras.layers.Add()([dense1, dropout1])

fc1 = Dense(256, activation='relu', kernel_initializer=tf.keras.initializers
fc1_output = fc1(dec)
dropout2 = Dropout(0.4, name='dropout2')(fc1_output)
output_layer = Dense(vocab_size, activation='softmax', name='Output_layer')
output = output_layer(dropout2)

encoder_decoder = Model(inputs = [input1, input2], outputs = output)
encoder_decoder.summary()
```

```
Model: "model_2"
__________________________________________________________________________
_____________________
Layer (type)                 Output Shape              Param #     Connected to
==========================================================================
====================
Text_Input (InputLayer)      [(None, 153)]             0
__________________________________________________________________________
_____________________
Embedding_layer (Embedding)  (None, 153, 300)          431100      Text_Input[0]
[0]
__________________________________________________________________________
_____________________
LSTM1 (LSTM)                 (None, 153, 256)          570368      Embedding_laye
r[0][0]
__________________________________________________________________________
_____________________
Image_input (InputLayer)     [(None, 2048)]            0
__________________________________________________________________________
_____________________
LSTM2 (LSTM)                 (None, 256)               525312      LSTM1[0][0]
__________________________________________________________________________
_____________________
dense_encoder (Dense)        (None, 256)               524544      Image_input[0]
```

```
[0]
__________________________________________________________________________________________
____________________
dropout1 (Dropout)              (None, 256)          0           LSTM2[0][0]
__________________________________________________________________________________________
____________________
add (Add)                       (None, 256)          0           dense_encoder
[0][0]
                                                                 dropout1[0][0]
__________________________________________________________________________________________
____________________
fc1 (Dense)                     (None, 256)          65792       add[0][0]
__________________________________________________________________________________________
____________________
dropout2 (Dropout)              (None, 256)          0           fc1[0][0]
__________________________________________________________________________________________
____________________
Output_layer (Dense)            (None, 1437)         369309      dropout2[0][0]
==========================================================================================
==================
Total params: 2,486,425
Trainable params: 2,055,325
Non-trainable params: 431,100
__________________________________________________________________________________________
____________________
```

In [55]:
```
1 tf.keras.utils.plot_model(encoder_decoder,show_shapes=True, dpi = 52)
```

Out[55]:



In [56]:
```
loss_function = tf.keras.losses.CategoricalCrossentropy(from_logits=False, r

def maskedLoss(y_true, y_pred):
    #getting mask value
    mask = tf.math.logical_not(tf.math.equal(y_true, 0))

    #calculating the loss
    loss_ = loss_function(y_true, y_pred)

    #converting mask dtype to loss_ dtype
    mask = tf.cast(mask, dtype=loss_.dtype)

    #applying the mask to loss
    loss_ = loss_*mask

    #getting mean over all the values
    loss_ = tf.reduce_mean(loss_)
    return loss_
```

In [57]:
```
optimizer = tf.keras.optimizers.Adam(0.001)
encoder_decoder.compile(optimizer, loss = maskedLoss)

current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'Tensorboard/logs_m1/fit3/' + current_time + '/train'
val_log_dir = 'Tensorboard/logs_m1/fit3/' + current_time + '/test'
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
val_summary_writer = tf.summary.create_file_writer(val_log_dir)
```

In [58]:

```
#training for 20 epochs
epoch_train_loss = []
epoch_val_loss = []

for epoch in range(20):
    print('EPOCH : ',epoch+1)
    start = time.time()
    batch_loss_tr = 0
    batch_loss_vl = 0

    for img, report in train_generator:

        r1 = bytes_to_string(report.numpy())
        img_input, rep_input, output_word = convert(img.numpy(), r1)
        rep_input = pad_sequences(rep_input, maxlen=153, padding='post')
        results = encoder_decoder.train_on_batch([img_input, rep_input], out

        batch_loss_tr += results

    train_loss = batch_loss_tr/(X_train.shape[0]//14)
 #   print('Saving Tensorboard')
    with train_summary_writer.as_default():
        tf.summary.scalar('loss', train_loss, step = epoch)

    for img, report in cv_generator:

        r1 = bytes_to_string(report.numpy())
        img_input, rep_input, output_word = convert(img.numpy(), r1)
        rep_input = pad_sequences(rep_input, maxlen=153, padding='post')
        results = encoder_decoder.test_on_batch([img_input, rep_input], outp
        batch_loss_vl += results

    val_loss = batch_loss_vl/(X_cv.shape[0]//14)

    with val_summary_writer.as_default():
        tf.summary.scalar('loss', val_loss, step = epoch)

    epoch_train_loss.append(train_loss)

    epoch_val_loss.append(val_loss)

    print('Training Loss: {},  Val Loss: {}'.format(train_loss, val_loss))
    print('Time Taken for this Epoch : {} sec'.format(time.time()-start))
    encoder_decoder.save_weights('encoder_decoder_epoch_'+ str(epoch+1) + '.
    print('-----------------------------------------------------------------
```

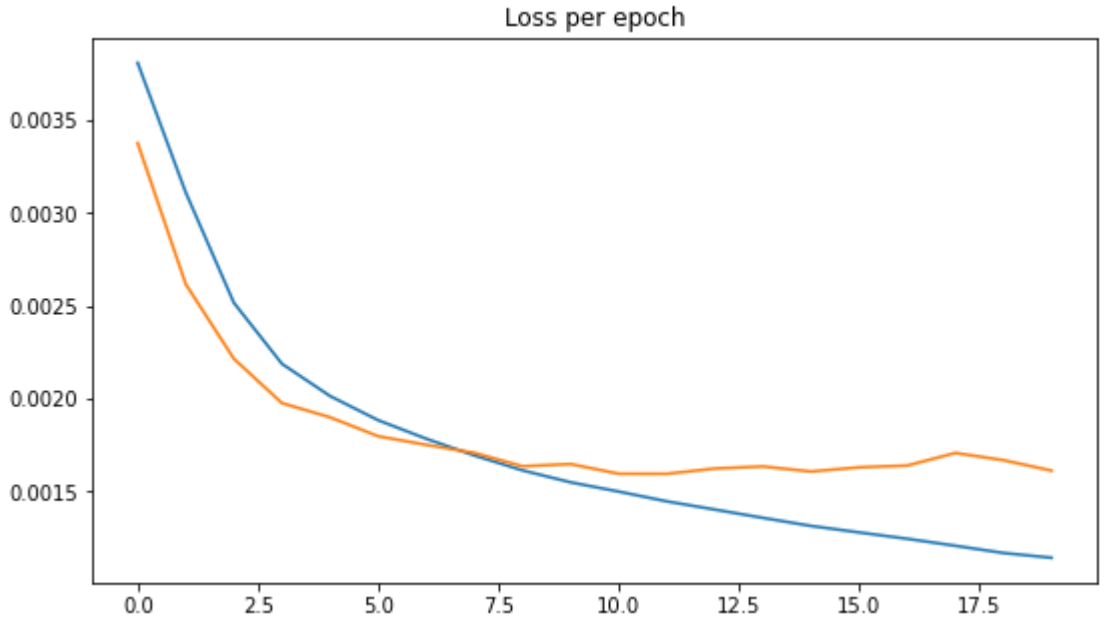
```
EPOCH :  1
Training Loss: 0.003811859837756847,  Val Loss: 0.003378847691540917
Time Taken for this Epoch : 35.43555426597595 sec
--------------------------------------------------------------------------------
-------------
EPOCH :  2
Training Loss: 0.0031122732233861226,  Val Loss: 0.002617753128736065
Time Taken for this Epoch : 19.199682235717773 sec
--------------------------------------------------------------------------------
-------------
```

```
EPOCH :  3
Training Loss: 0.00251724414390318,  Val Loss: 0.002215469253058426
Time Taken for this Epoch : 19.386552333831787 sec
--------------------------------------------------------------------------------
------------
EPOCH :  4
Training Loss: 0.0021881484385192166,  Val Loss: 0.0019762548498618296
Time Taken for this Epoch : 19.365970134735107 sec
--------------------------------------------------------------------------------
------------
EPOCH :  5
Training Loss: 0.0020142961671661,  Val Loss: 0.0018996467652659004
Time Taken for this Epoch : 19.194966077804565 sec
--------------------------------------------------------------------------------
------------
EPOCH :  6
Training Loss: 0.0018839153897272528,  Val Loss: 0.0017975640202609773
Time Taken for this Epoch : 19.46571397781372 sec
--------------------------------------------------------------------------------
------------
EPOCH :  7
Training Loss: 0.001783981510357657,  Val Loss: 0.0017513166769789772
Time Taken for this Epoch : 19.38042116165161 sec
--------------------------------------------------------------------------------
------------
EPOCH :  8
Training Loss: 0.0016937019819923238,  Val Loss: 0.0017077396131263902
Time Taken for this Epoch : 19.50852060317993 sec
--------------------------------------------------------------------------------
------------
EPOCH :  9
Training Loss: 0.001613419305599356,  Val Loss: 0.0016357599942682262
Time Taken for this Epoch : 19.49079155921936 sec
--------------------------------------------------------------------------------
------------
EPOCH :  10
Training Loss: 0.0015490241192144148,  Val Loss: 0.0016473947057070641
Time Taken for this Epoch : 19.211782217025757 sec
--------------------------------------------------------------------------------
------------
EPOCH :  11
Training Loss: 0.0014991406853627509,  Val Loss: 0.001595342458667568
Time Taken for this Epoch : 19.37200903892517 sec
--------------------------------------------------------------------------------
------------
EPOCH :  12
Training Loss: 0.0014462974835494486,  Val Loss: 0.0015947213978506625
Time Taken for this Epoch : 19.645337104797363 sec
--------------------------------------------------------------------------------
------------
EPOCH :  13
Training Loss: 0.0014019837510786232,  Val Loss: 0.0016233775875149055
Time Taken for this Epoch : 19.886524438858032 sec
--------------------------------------------------------------------------------
------------
EPOCH :  14
Training Loss: 0.0013571550870470248,  Val Loss: 0.001634487801661285
```

```
Time Taken for this Epoch : 19.65162968635559 sec
--------------------------------------------------------------------------------
-------------
EPOCH :  15
Training Loss: 0.0013137239747713816,  Val Loss: 0.0016073409209732348
Time Taken for this Epoch : 19.594081163406372 sec
--------------------------------------------------------------------------------
-------------
EPOCH :  16
Training Loss: 0.0012786779239499554,  Val Loss: 0.0016298411825958353
Time Taken for this Epoch : 19.6272873878479 sec
--------------------------------------------------------------------------------
-------------
EPOCH :  17
Training Loss: 0.0012443640519307008,  Val Loss: 0.0016394440839902903
Time Taken for this Epoch : 19.607653617858887 sec
--------------------------------------------------------------------------------
-------------
EPOCH :  18
Training Loss: 0.0012067323393982641,  Val Loss: 0.0017072825441853358
Time Taken for this Epoch : 19.699045658111572 sec
--------------------------------------------------------------------------------
-------------
EPOCH :  19
Training Loss: 0.001167800024401179,  Val Loss: 0.0016684900157344646
Time Taken for this Epoch : 19.323747873306274 sec
--------------------------------------------------------------------------------
-------------
EPOCH :  20
Training Loss: 0.0011420611438086152,  Val Loss: 0.0016120212064542545
Time Taken for this Epoch : 19.82920217514038 sec
--------------------------------------------------------------------------------
-------------
```

In [59]:
```
plt.figure(figsize=(9,5))

plt.plot(epoch_train_loss)
plt.plot(epoch_val_loss)

plt.title('Loss per epoch')
```

Out[59]: Text(0.5, 1.0, 'Loss per epoch')



In [83]:
```
encoder_decoder.load_weights("/content/encoder_decoder_epoch_5.h5")
```

In [84]:
```
# encoder
encoder_input = encoder_decoder.input[0]
encoder_output = encoder_decoder.get_layer('dense_encoder').output
encoder_model = Model(encoder_input, encoder_output)

# decoder#
text_input = encoder_decoder.input[1]
enc_output = Input(shape=(256,), name='Enc_Output')
text_output = encoder_decoder.get_layer('LSTM2').output
add1 = tf.keras.layers.Add()([text_output, enc_output])
fc_1 = fc1(add1)
decoder_output = output_layer(fc_1)

decoder_model = Model(inputs = [text_input, enc_output], outputs = decoder_o
```

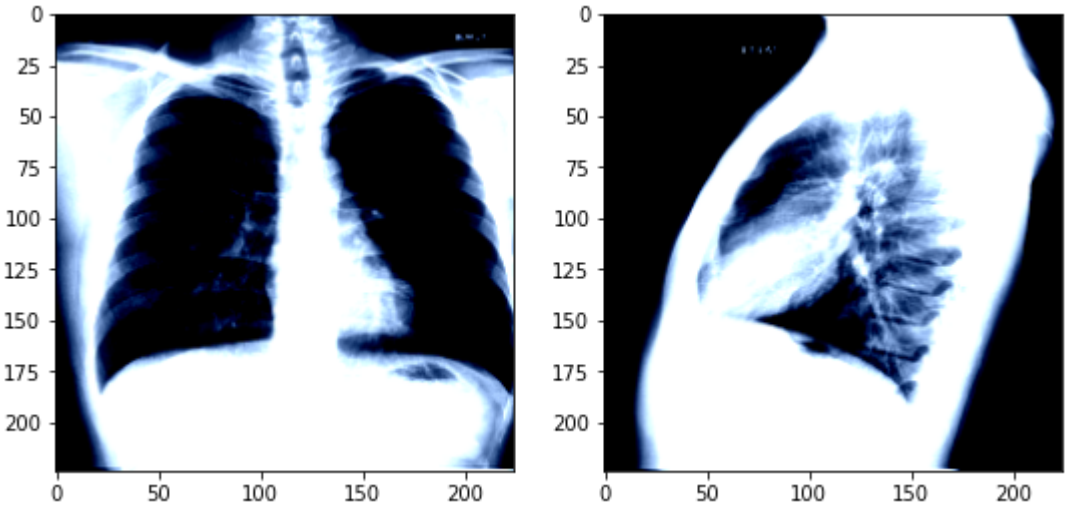
In [85]:
```
def greedysearch(img):
    image = cheXnet_Features[img]
    input_ = 'startseq'
    image_features = encoder_model.predict(image)

    result = []
    for i in range(153):
        input_tok = [tokenizer.word_index[w] for w in input_.split()]
        input_padded = pad_sequences([input_tok], 153, padding='post')
        predictions = decoder_model.predict([input_padded, image_features])
        arg = np.argmax(predictions)
        if arg != 7:   # endseq
            result.append(tokenizer.index_word[arg])
            input_ = input_ + ' ' + tokenizer.index_word[arg]
        else:
            break
    rep = ' '.join(e for e in result)
    return rep
```

In [86]:
```
def load_image(img_name):
    image = Image.open(img_name)
    X = np.asarray(image.convert("RGB"))
    X = np.asarray(X)
    X = preprocess_input(X)
    X = resize(X, (224,224,3))
    X = np.expand_dims(X, axis=0)
    X = np.asarray(X)
    return X
```

In [87]:
```
def get_result(idx=0):

  plt.figure(figsize=(9,5))

  pre_Report = greedysearch(cv_data['Person_id'][idx]) # result after 20 epo
  print('---------------------------------------------------------------------
  print("Predicted Report : ",pre_Report)
  print('---------------------------------------------------------------------
  print("Actual Report : ",cv_data['Report'][idx])

  plt.subplot(121)
  img = load_image(cv_data['Image1'][idx])
  plt.imshow(img[0])

  plt.subplot(122)
  img = load_image(cv_data['Image2'][idx])
  plt.imshow(img[0])
```

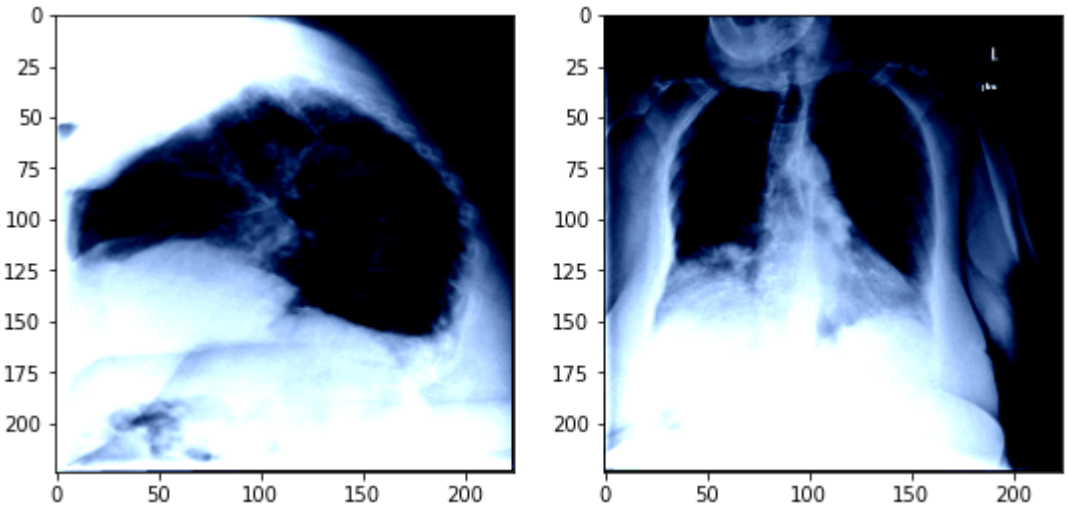
In [88]:
```
1 get_result(2)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

--------------------------------------------------------------------------------
----------------------
Predicted Report :  the heart size normal . the mediastinum unremarkable . the
lungs are clear .
--------------------------------------------------------------------------------
----------------------
Actual Report :  startseq no pneumothora .  no large pleural effusions .  heart
size normal .  no acute focal space opacities .  endseq
```



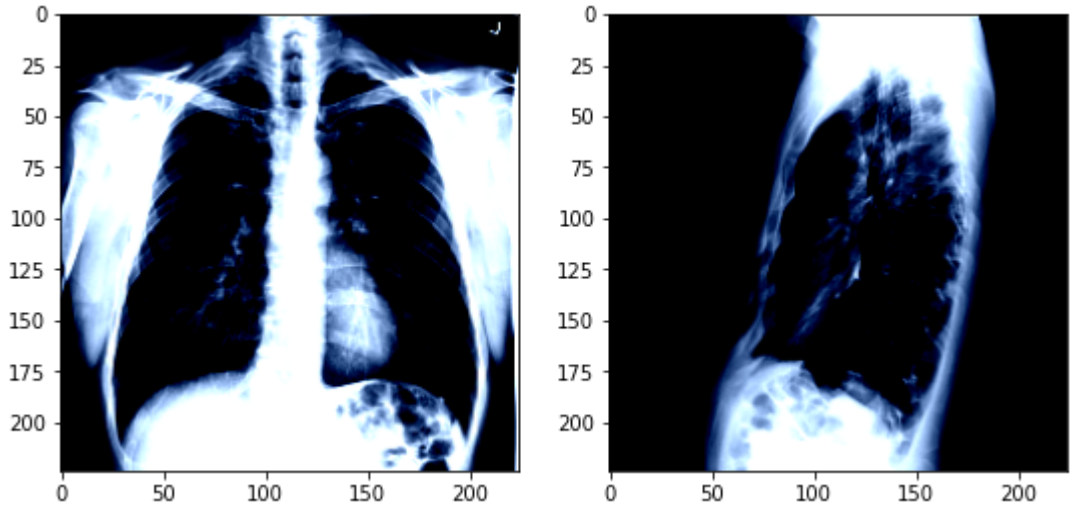
In [89]:
```
1 get_result(98)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

--------------------------------------------------------------------------------
----------------------
Predicted Report :  the heart size normal . the mediastinal contour within norm
al limits . the lungs are free focal airspace disease . no pleural effusion pne
umothora .
--------------------------------------------------------------------------------
----------------------
Actual Report :  startseq the heart mildly enlarged .  pulmonary vascularity in
creased .  there again mild elevation the right hemidiaphragm .  air space dise
ase andor atelectasis noted right lung base .  there also streaky opacity the l
eft base .  the costophrenic are blunted .  endseq
```



In [90]:
```
get_result(48)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

-----------------------------------------------------------------------------
----------------------
Predicted Report :  the heart size and mediastinal contours are within normal l
imits . no focal consolidation pleural pleural effusion .
-----------------------------------------------------------------------------
----------------------
Actual Report :  startseq the heart normal size .  the mediastinum unremarkable
.  there again biapical scarring .  small stable calcified left lower lobe gran
uloma .  the lungs are otherwise clear .  endseq
```



Blue Score

In [91]:
```
def rem_fullstops(text):
    '''Removes punctuations'''
    punctuations = '''.''' # full stop is not removed
    new_text = []
    for char in text:
        if char in punctuations:
            text = text.replace(char, "")
    new_text.append(' '.join(e for e in text.split()))
    return new_text[0]
```

CV

In [ ]:
```
from tqdm.notebook import tqdm

bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []
for img, rep in tqdm(zip(X_cv.values, y_cv.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep = greedysearch(img)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
```

```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
=''))) 
```

In [ ]:
```
a = sum(bleu1)/X_cv.shape[0]
b = sum(bleu2)/X_cv.shape[0]
c = sum(bleu3)/X_cv.shape[0]
d = sum(bleu4)/X_cv.shape[0]

print('Bleu1 Score: ',a)
print('Bleu2 Score: ',b)
print('Bleu3 Score: ',c)
print('Bleu4 Score: ',d)

print('-----------------------------------------------------------------------')
print("Avg Blue score:",(a+b+c+d)/4)
```

```
Bleu1 Score:  0.1813011714955598
Bleu2 Score:  0.17041689497849832
Bleu3 Score:  0.17754108502201618
Bleu4 Score:  0.19533480354978888
-----------------------------------------------------------------------
Avg Blue score: 0.18114848876146578
```

Test

In [ ]:
```
bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []
for img, rep in tqdm(zip(X_test.values, y_test.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep = greedysearch(img)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
```

```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
='')))
```

In [ ]:
```
a = sum(bleu1)/X_test.shape[0]
b = sum(bleu2)/X_test.shape[0]
c = sum(bleu3)/X_test.shape[0]
d = sum(bleu4)/X_test.shape[0]

print('Bleu1 Score: ',a)
print('Bleu2 Score: ',b)
print('Bleu3 Score: ',c)
print('Bleu4 Score: ',d)

print('-----------------------------------------------------------------------
print("Avg Blue score:",(a+b+c+d)/4)
```

```
Bleu1 Score:  0.16357241534275888
Bleu2 Score:  0.15045014851595584
Bleu3 Score:  0.15754604224530216
Bleu4 Score:  0.17542506415900994
----------------------------------------------------------------------
Avg Blue score: 0.1617484175657567
```

Beamsearch

In [92]:

```
# beam_width = 2
def beamsearch(image, beam_width = 2):

    start = [tokenizer.word_index['startseq']]

    sequences = [[start, 0]]

    img_features = cheXnet_Features[image]
    img_features = encoder_model.predict(img_features)
    finished_seq = []

    for i in range(153):
        all_candidates = []
        new_seq = []
        for s in sequences:

            text_input = pad_sequences([s[0]], 153, padding='post')
            predictions = decoder_model.predict([text_input,img_features])
            top_words = np.argsort(predictions[0])[-beam_width:]
            seq, score = s

            for t in top_words:
                candidates = [seq + [t], score - np.log(predictions[0][t])]
                all_candidates.append(candidates)

        sequences = sorted(all_candidates, key = lambda l: l[1])[:beam_width
        # checks for 'endseq' in each seq in the beam
        count = 0
        for seq,score in sequences:
            if seq[len(seq)-1] == tokenizer.word_index['endseq']:
                score = score/len(seq)   # normalized
                finished_seq.append([seq, score])
                count+=1
            else:
                new_seq.append([seq, score])
        beam_width -= count
        sequences = new_seq

        # if all the sequences reaches its end before 155 timesteps
        if not sequences:
            break
        else:
            continue

    sequences = finished_seq[-1]
    rep = sequences[0]
    score = sequences[1]
    temp = []
    rep.pop(0)
    for word in rep:
        if word != tokenizer.word_index['endseq']:
            temp.append(tokenizer.index_word[word])
        else:
            break
    rep = ' '.join(e for e in temp)

```

```
57      return rep, score
```

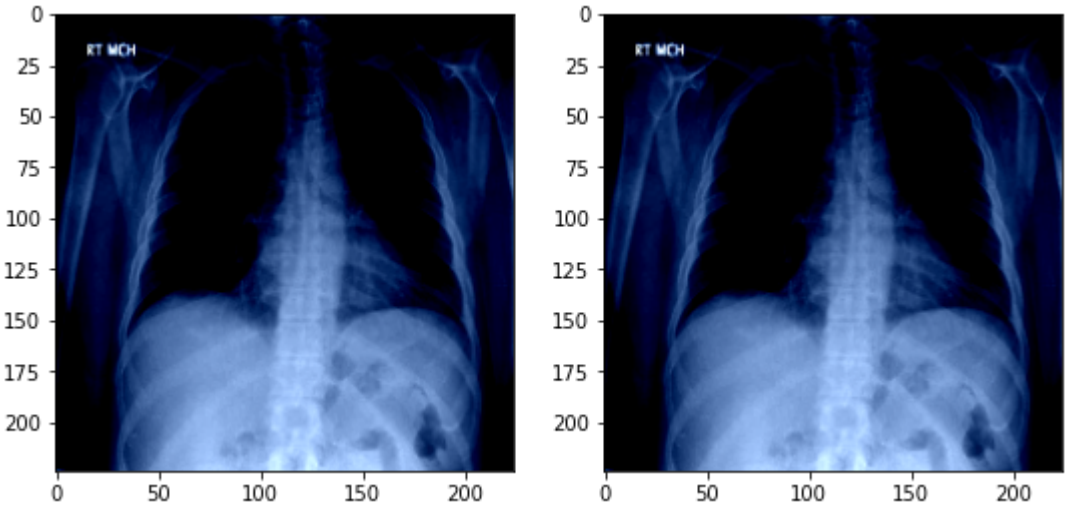
In [93]:

```
def get_result(beam_width,idx=0):

  plt.figure(figsize=(9,5))

  pre_Report,Score = beamsearch(cv_data['Person_id'][idx],beam_width) # resu
  print('---------------------------------------------------------------------
  print("Predicted Report : ",pre_Report)
  print('Score is :',Score)
  print('---------------------------------------------------------------------
  print("Actual Report : ",cv_data['Report'][idx])

  plt.subplot(121)
  img = load_image(cv_data['Image1'][idx])
  plt.imshow(img[0])

  plt.subplot(122)
  img = load_image(cv_data['Image2'][idx])
  plt.imshow(img[0])
```

In [94]:

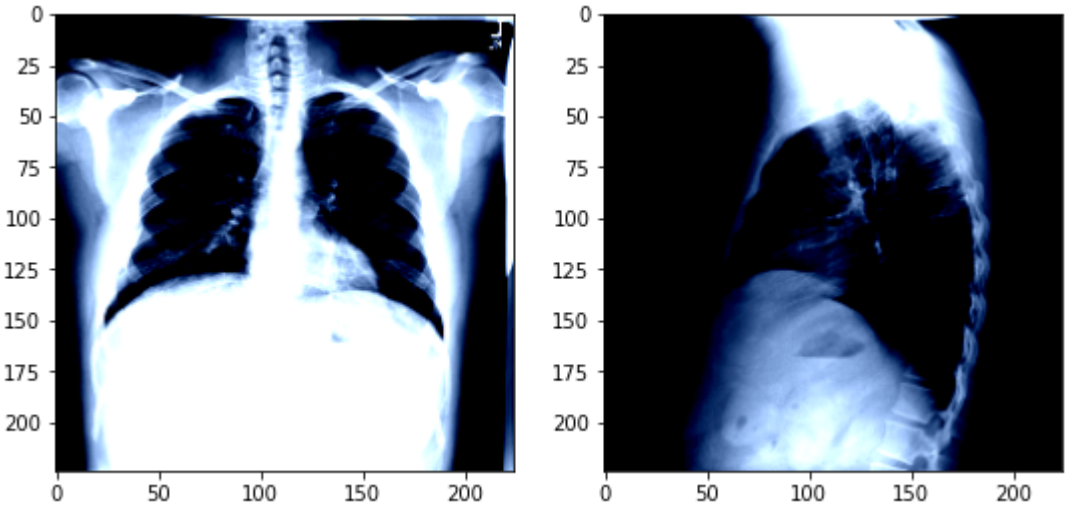
```
# beam_width = 2
get_result(2,5)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

---------------------------------------------------------------------------
-----------------------
Predicted Report :  the heart size normal . the mediastinal unremarkable . the
lungs are free focal airspace disease . no pleural effusion pneumothora .
Score is : 0.715110859911268
---------------------------------------------------------------------------
-----------------------
Actual Report :  startseq normal heart size and mediastinal contours .  low lun
g volumes with no significant airspace consolidation .  no pleural effusion pne
umothora .  visualized osseous structures are unremarkable appearance .  endseq
```



In [95]:
```
# beam_width = 2
get_result(2,7)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).


--------------------------------------------------------------------------------
-----------------------
Predicted Report :  the heart size normal . the mediastinum contours are within
normal limits . the lungs are free focal airspace disease . no pleural effusion
pneumothora .
Score is : 0.5355977995028037
--------------------------------------------------------------------------------
-----------------------
Actual Report :  startseq the heart normal size .  the mediastinum unremarkable
.  the lungs are clear .  endseq
```



In [ ]:

```
from tqdm.notebook import tqdm
# beam_width = 2
'''CV'''

bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []
for img, rep in tqdm(zip(X_cv.values, y_cv.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep,score = beamsearch(img,2)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0

a = sum(bleu1)/X_test.shape[0]
b = sum(bleu2)/X_test.shape[0]
c = sum(bleu3)/X_test.shape[0]
d = sum(bleu4)/X_test.shape[0]

print('CV Bleu1 Score: ',a)
print('CV Bleu2 Score: ',b)
print('CV Bleu3 Score: ',c)
print('CV Bleu4 Score: ',d)

print('-----------------------------------------------------------------------
print("Avg CV Blue score:",(a+b+c+d)/4)

print('=======================================================================

# beam_width = 2
'''TEST'''
bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []

for img, rep in tqdm(zip(X_test.values, y_test.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep,score = beamsearch(img,2)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
```

```
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0

a = sum(bleu1)/X_test.shape[0]
b = sum(bleu2)/X_test.shape[0]
c = sum(bleu3)/X_test.shape[0]
d = sum(bleu4)/X_test.shape[0]

print('Test Bleu1 Score: ',a)
print('Test Bleu2 Score: ',b)
print('Test Bleu3 Score: ',c)
print('Test Bleu4 Score: ',d)

print('--------------------------------------------------------------------')
print("Avg Test Blue score:",(a+b+c+d)/4)
```

```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
=''')))


CV Bleu1 Score:  0.4431255936227122
CV Bleu2 Score:  0.3098561134547433
CV Bleu3 Score:  0.31468296277522284
CV Bleu4 Score:  0.35990415454093605
--------------------------------------------------------------------
Avg CV Blue score: 0.3568922060984036
====================================================================================
======

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
=''')))


Test Bleu1 Score:  0.2912680411096469
Test Bleu2 Score:  0.207034308388247
Test Bleu3 Score:  0.21311129742783588
Test Bleu4 Score:  0.24675493283561065
--------------------------------------------------------------------
Avg Test Blue score: 0.2395421449403351
```

In [ ]:

```
from tqdm.notebook import tqdm
# beam_width = 5
'''CV'''

bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []
for img, rep in tqdm(zip(X_cv.values, y_cv.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep,score = beamsearch(img,5)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0

a = sum(bleu1)/X_test.shape[0]
b = sum(bleu2)/X_test.shape[0]
c = sum(bleu3)/X_test.shape[0]
d = sum(bleu4)/X_test.shape[0]

print('CV Bleu1 Score: ',a)
print('CV Bleu2 Score: ',b)
print('CV Bleu3 Score: ',c)
print('CV Bleu4 Score: ',d)

print('-----------------------------------------------------------------------
print("Avg CV Blue score:",(a+b+c+d)/4)

print('=======================================================================

# beam_width = 5
'''TEST'''
bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []

for img, rep in tqdm(zip(X_test.values, y_test.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep,score = beamsearch(img,5)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
```

```
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0

a = sum(bleu1)/X_test.shape[0]
b = sum(bleu2)/X_test.shape[0]
c = sum(bleu3)/X_test.shape[0]
d = sum(bleu4)/X_test.shape[0]

print('Test Bleu1 Score: ',a)
print('Test Bleu2 Score: ',b)
print('Test Bleu3 Score: ',c)
print('Test Bleu4 Score: ',d)

print('----------------------------------------------------------------------
print("Avg Test Blue score:",(a+b+c+d)/4)
```

```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
='')))


CV Bleu1 Score:  0.49315195463081307
CV Bleu2 Score:  0.3364148008701202
CV Bleu3 Score:  0.32786218875600814
CV Bleu4 Score:  0.3742228380850862
----------------------------------------------------------------------
Avg CV Blue score: 0.3829129455855069
==========================================================================
======

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
='')))


Test Bleu1 Score:  0.3314679715747508
Test Bleu2 Score:  0.2287347207212481
Test Bleu3 Score:  0.22166958182138657
Test Bleu4 Score:  0.2585134869156888
----------------------------------------------------------------------
Avg Test Blue score: 0.26009644025826856
```

In [ ]:

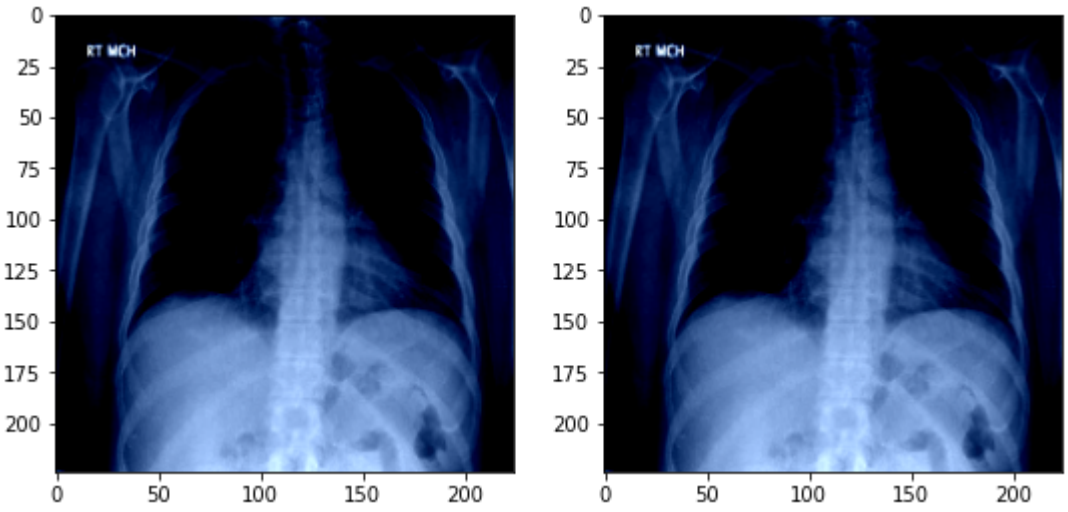
```
from tqdm.notebook import tqdm
# beam_width = 7
'''CV'''

bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []
for img, rep in tqdm(zip(X_cv.values, y_cv.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep,score = beamsearch(img,7)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0

a = sum(bleu1)/X_test.shape[0]
b = sum(bleu2)/X_test.shape[0]
c = sum(bleu3)/X_test.shape[0]
d = sum(bleu4)/X_test.shape[0]

print('CV Bleu1 Score: ',a)
print('CV Bleu2 Score: ',b)
print('CV Bleu3 Score: ',c)
print('CV Bleu4 Score: ',d)

print('-----------------------------------------------------------------------
print("Avg CV Blue score:",(a+b+c+d)/4)

print('=======================================================================

# beam_width = 7
'''TEST'''
bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []

for img, rep in tqdm(zip(X_test.values, y_test.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep,score = beamsearch(img,7)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
```

```
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0

a = sum(bleu1)/X_test.shape[0]
b = sum(bleu2)/X_test.shape[0]
c = sum(bleu3)/X_test.shape[0]
d = sum(bleu4)/X_test.shape[0]

print('Test Bleu1 Score: ',a)
print('Test Bleu2 Score: ',b)
print('Test Bleu3 Score: ',c)
print('Test Bleu4 Score: ',d)

print('--------------------------------------------------------------------')
print("Avg Test Blue score:",(a+b+c+d)/4)
```

```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
=''')))


CV Bleu1 Score:  0.5045278287455154
CV Bleu2 Score:  0.3486129728173558
CV Bleu3 Score:  0.3340529472544908
CV Bleu4 Score:  0.37893574981149486
--------------------------------------------------------------------
Avg CV Blue score: 0.39153237465721424
====================================================================================
======

HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
=''')))


Test Bleu1 Score:  0.3406031120390907
Test Bleu2 Score:  0.2344795461556287
Test Bleu3 Score:  0.2263127304699709
Test Bleu4 Score:  0.26361966375328694
--------------------------------------------------------------------
Avg Test Blue score: 0.26625376310449433
```

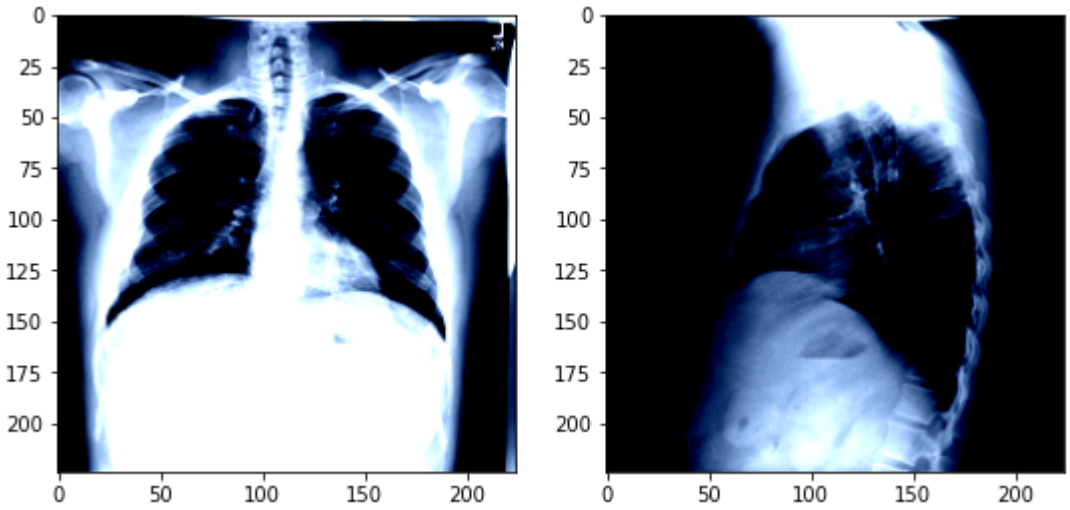
In [ ]:
```
1 get_result(5,5)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

--------------------------------------------------------------------------------
----------------------
Predicted Report :  the heart size and pulmonary vascularity are within normal
limits . the lungs are free focal airspace disease . no pleural effusion pneumo
thora seen . no acute bony abnormality .
Score is : 0.4401072240580106
--------------------------------------------------------------------------------
----------------------
Actual Report :  startseq normal heart size and mediastinal contours .  low lun
g volumes with no significant airspace consolidation .  no pleural effusion pne
umothora .  visualized osseous structures are unremarkable appearance .  endseq
```



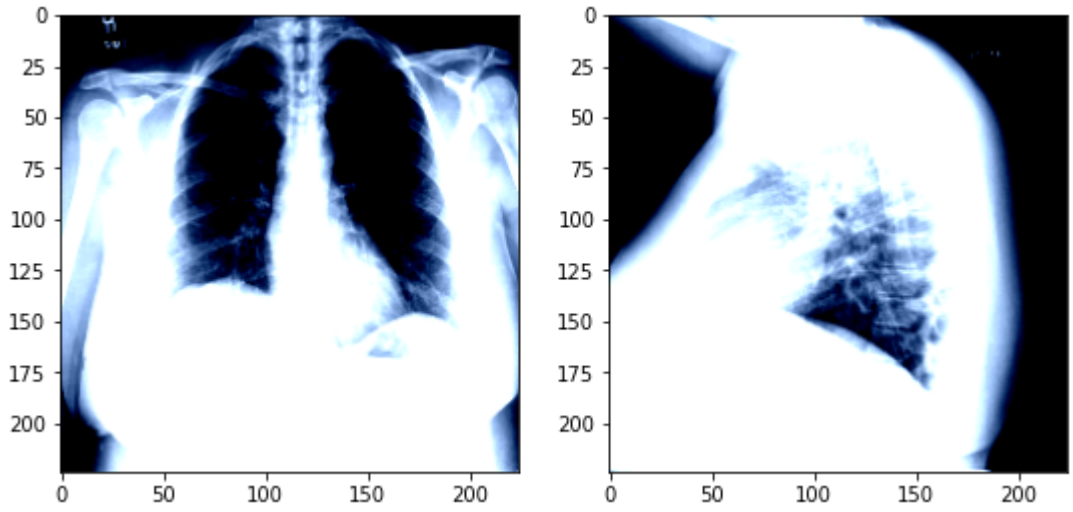
In [96]:
```
1 get_result(5,7)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

--------------------------------------------------------------------------------
----------------------
Predicted Report :  the heart size and mediastinal contours are within normal l
imits . the lungs are free focal airspace disease . no pleural effusion pneumot
hora .
Score is : 0.5155239434870265
--------------------------------------------------------------------------------
----------------------
Actual Report :  startseq the heart normal size .  the mediastinum unremarkable
.  the lungs are clear .  endseq
```



In [97]:
```
1 get_result(5,21)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

--------------------------------------------------------------------------------
----------------------
Predicted Report :  the heart normal size . the mediastinum unremarkable . the
lungs are free focal airspace disease . no pleural effusion pneumothora .
Score is : 0.5601838477887213
--------------------------------------------------------------------------------
----------------------
Actual Report :  startseq the cardiac silhouette upper mediastinum and pulmonar
y vasculature are within normal limits .  there no acute air space infiltrate p
leural effusion pneumothora .  endseq
```



Using ChexNet pretrained model to extarct features from images to be used in transfer learning for Attention model

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import os
from tqdm import tqdm
import tensorflow as tf
import cv2
import pickle
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import time
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, LSTM, Input, Embedding, Conv2D,Co
import random
import datetime
from nltk.translate.bleu_score import sentence_bleu
from math import log
```

In [ ]:
```
train_dataset = pd.read_csv('/content/Final_Train_Data.csv')
cv_dataset = pd.read_csv('/content/Final_CV_Data.csv')
test_dataset = pd.read_csv('/content/Final_Test_Data.csv')
```

In [ ]:
```
chexNet = chex = densenet.DenseNet121(include_top=False, weights = None, inp
X = chexNet.output
X = Dense(14, activation="sigmoid", name="predictions")(X)
chexNet = Model(inputs=chexNet.input, outputs=X)
```

In [ ]:
```
chexNet.load_weights('/content/brucechou1983_CheXNet_Keras_0.3.0_weights.h5'
```

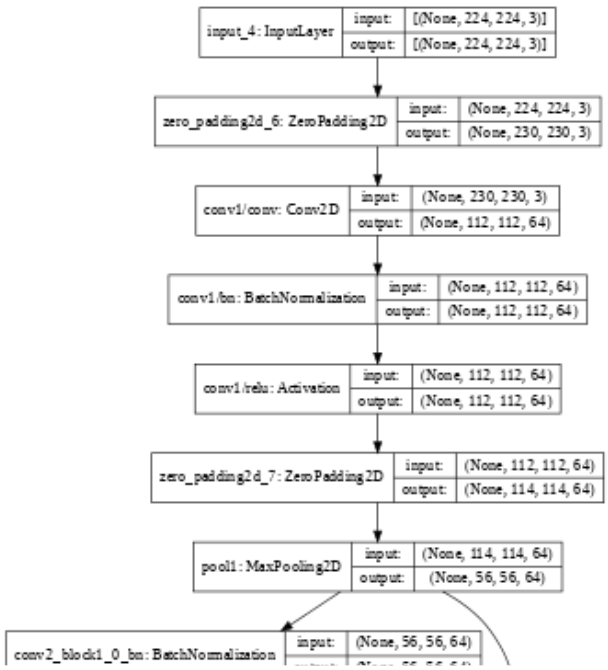
In [ ]:
```
model = Model(inputs = chexNet.input, outputs = chexNet.layers[-2].output)
```

In [ ]:
```
model.summary()
```

```
Model: "model_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 224, 224, 3)      0
_________________________________________________________________
zero_padding2d_6 (ZeroPadding2D (None, 230, 230, 3)  0
[0]
_________________________________________________________________
conv1/conv (Conv2D)          (None, 112, 112, 64)      9408
2d_6[0][0]
_________________________________________________________________
conv1/bn (BatchNormalization)   (None, 112, 112, 64)      256
[0][0]
```

In [ ]:
```
1 tf.keras.utils.plot_model(model,show_shapes=True, dpi = 42)
```

Out[239]:

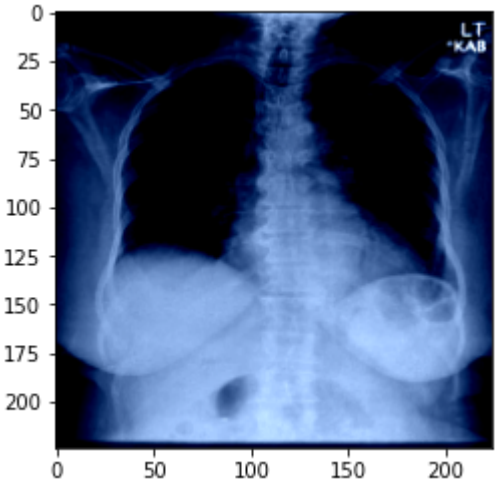


In [ ]:
```
1 def load_image(img_name):
2     image = Image.open(img_name)
3     X = np.asarray(image.convert("RGB"))
4     X = np.asarray(X)
5     X = preprocess_input(X)
6     X = resize(X, (224,224,3))
7     X = np.expand_dims(X, axis=0)
8     X = np.asarray(X)
9     return X
```

In [ ]:
```
1 img = load_image('NLMCXR_png/CXR1082_IM-0058-1001.png')
2 plt.imshow(img[0])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
```

Out[241]:
```
<matplotlib.image.AxesImage at 0x7fe6328dfc18>
```



In [ ]:
```
from tqdm import tqdm_notebook as tqdm

def image_features(train, test, cv):
    Xnet_features_attention = {}

    for key, img1, img2, finding in tqdm(train.values):
        i1 = load_image(img1)
        img1_features = model.predict(i1)

        i2 = load_image(img2)
        img2_features = model.predict(i2)

        input_ = np.concatenate((img1_features, img2_features), axis=2)
        input_ = tf.reshape(input_, (input_.shape[0], -1, input_.shape[-1]))

        Xnet_features_attention[key] = input_

    for key, img1, img2, finding in tqdm(test.values):
        i1 = load_image(img1)
        img1_features = model.predict(i1)

        i2 = load_image(img2)
        img2_features = model.predict(i2)

        input_ = np.concatenate((img1_features, img2_features), axis=2)
        input_ = tf.reshape(input_, (input_.shape[0], -1, input_.shape[-1]))

        Xnet_features_attention[key] = input_

    for key, img1, img2, finding in tqdm(cv.values):
        i1 = load_image(img1)
        img1_features = model.predict(i1)

        i2 = load_image(img2)
        img2_features = model.predict(i2)

        input_ = np.concatenate((img1_features, img2_features), axis=2)
        input_ = tf.reshape(input_, (input_.shape[0], -1, input_.shape[-1]))

        Xnet_features_attention[key] = input_

    return Xnet_features_attention
```

In [ ]:
```
Xnet_features_attention = image_features(train_dataset, test_dataset, cv_dat
```

```
HBox(children=(FloatProgress(value=0.0, max=2764.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, max=389.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, max=554.0), HTML(value='')))
```

In [ ]:
```
546 % 14
```

Out[244]:
```
0
```

In [ ]:
```
Xnet_features_attention['NLMCXR_png/CXR1796_IM-0517_0'].shape
```

Out[245]:
```
TensorShape([1, 98, 1024])
```

In [ ]:
```
# save the file for future use
f = open('Image_features_for_attention_model.pickle','wb')
pickle.dump(Xnet_features_attention, f)
f.close()
```

In [ ]:
```
X_train = train_dataset['Person_id'][:2758]
X_test = test_dataset['Person_id'][:378]
X_cv = cv_dataset['Person_id'][:546]
y_train = train_dataset['Report'][:2758]
y_test = test_dataset['Report'][:378]
y_cv = cv_dataset['Report'][:546]
```

In [ ]:
```
max_capt_len = 153
pad_size = max_capt_len

tokenizer = Tokenizer(filters='!"#$%&()*+,-/:;<=>?@[\\]^_`{|}~\t\n')
tokenizer.fit_on_texts(y_train.values)

train_rep_tok = tokenizer.texts_to_sequences(y_train)
cv_rep_tok = tokenizer.texts_to_sequences(y_cv)
test_rep_tok = tokenizer.texts_to_sequences(y_test)

train_rep_padded = pad_sequences(train_rep_tok, maxlen=153, padding='post')
cv_rep_padded = pad_sequences(cv_rep_tok, maxlen=153, padding='post')
test_rep_padded = pad_sequences(test_rep_tok, maxlen=153, padding='post')

tokenizer.word_index['<pad>'] = 0
tokenizer.index_word[0] = '<pad>'
```

In [ ]:
```
BATCH_SIZE = 14
BUFFER_SIZE = 500
```

In [ ]:
```
a = Xnet_features_attention['NLMCXR_png/CXR1796_IM-0517_0'][0]
a.shape
```

Out[250]:
```
TensorShape([98, 1024])
```

In [ ]:
```
def load_image(id_, report):
    '''Loads the Image Features with their corresponding Ids'''
    img_feature = Xnet_features_attention[id_.decode('utf-8')][0]
    return img_feature, report
```

In [ ]:
```
def dataset_generator(img_name_train,reps):

  dataset = tf.data.Dataset.from_tensor_slices((img_name_train, reps))
  # Use map to load the numpy files in parallel
  dataset = dataset.map(lambda item1, item2: tf.numpy_function(load_image, [
                        num_parallel_calls=tf.data.experimental.AUTOTUNE)

  # Shuffle and batch
  dataset = dataset.shuffle(500).batch(BATCH_SIZE).prefetch(buffer_size=tf.d
  return dataset
```

In [ ]:
```
cv_rep_padded.shape
```

Out[253]: (546, 153)

In [ ]:
```
train_generator = dataset_generator(X_train.values, train_rep_padded)
cv_generator = dataset_generator(X_cv.values, cv_rep_padded)
```

In [ ]:
```
f = open('/content/drive/MyDrive/glove_vectors','rb') # 300d glove vectors
glove_vectors = pickle.load(f)
f.close()
```

In [ ]:
```
vocab_size = len(tokenizer.word_index.keys()) + 1

embedding_matrix = np.zeros((vocab_size,300))
for word, i in tokenizer.word_index.items():
    if word in glove_vectors.keys():
        vec = glove_vectors[word]
        embedding_matrix[i] = vec
    else:
        continue
```

In [ ]:
```
class Encoder(tf.keras.layers.Layer):
    def __init__(self, units):
        super(Encoder, self).__init__()
        self.units = units

    def build(self, input_shape):
        self.maxpool = tf.keras.layers.MaxPool1D()
        self.dense = Dense(self.units, kernel_initializer=tf.keras.initializ

    def call(self, input_, training=True):

        x = self.maxpool(input_)
        x = self.dense(x)

        return x

    def get_states(self, bs):

        return tf.zeros((bs, self.units))
```

In [ ]:

```
class Decoder(tf.keras.layers.Layer):

    def __init__(self, vocab_size, input_length, dec_units, att_units):
        super(Decoder, self).__init__()
        self.vocab_size = vocab_size
        self.input_length = input_length
        self.dec_units = dec_units
        self.att_units = att_units
        self.onestep_decoder = OneStepDecoder(self.vocab_size, self.att_unit
    @tf.function
    def call(self, dec_input, hidden_state, enc_output):
        all_outputs = tf.TensorArray(tf.float32, dec_input.shape[1], name='o

        for timestep in range(dec_input.shape[1]):

            output, hidden_state, attention_weights = self.onestep_decoder(d
                                                                           h


            all_outputs = all_outputs.write(timestep, output)

        all_outputs = tf.transpose(all_outputs.stack(), [1,0,2])
        return all_outputs

        return tf.zeros((bs, self.units))
```

In [ ]:
```
class OneStepDecoder(tf.keras.layers.Layer):
    def __init__(self, vocab_size, att_units, dec_units):
        super(OneStepDecoder, self).__init__()
        self.vocab_size = vocab_size
        self.att_units = att_units
        self.dec_units = dec_units

    def build(self, input_shape):
        self.embedding = Embedding(self.vocab_size, output_dim=300, input_le
                                   weights = [embedding_matrix],
                                   name="embedding_layer_decoder")
        self.gru = GRU(self.dec_units, return_sequences=True, return_state=T
        self.fc = Dense(self.vocab_size)

        self.V = Dense(1)
        self.W = Dense(self.att_units)
        self.U = Dense(self.att_units)

    def call(self, dec_input, hidden_state, enc_output):

        hidden_with_time = tf.expand_dims(hidden_state, 1)
        attention_weights = self.V(tf.nn.tanh(self.U(enc_output) + self.W(hi
        attention_weights = tf.nn.softmax(attention_weights, 1)
        context_vector = attention_weights * enc_output
        context_vector = tf.reduce_sum(context_vector, axis=1)

        x = self.embedding(dec_input)
        x = tf.concat([tf.expand_dims(context_vector, axis=1),x], axis=-1)
        output, h_state = self.gru(x, initial_state = hidden_state)
        output = tf.reshape(output, (-1, output.shape[2]))
        x = self.fc(output)

        return x, h_state, attention_weights
```

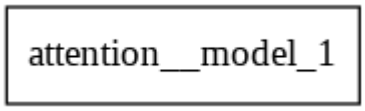
In [ ]:
```
class Attention_Model(tf.keras.Model):
    def __init__(self, vocab, units, max_capt_len, att_units, batch_size):
        super(Attention_Model, self).__init__()
        self.batch_size = batch_size
        self.encoder = Encoder(units)
        self.decoder = Decoder(vocab_size, max_capt_len, units, att_units)

    def call(self, data):
        enc_input, dec_input = data[0], data[1]

        enc_output = self.encoder(enc_input)
        enc_state = self.encoder.get_states(self.batch_size)
        dec_output = self.decoder(dec_input, enc_state, enc_output)

        return dec_output
```

In [ ]:
```
units = 256
att_units = 10
```

In [ ]:
```
model_atten = Attention_Model(vocab_size, units, max_capt_len, att_units, BA
```

In [ ]:
```
1 tf.keras.utils.plot_model(model_atten,show_shapes=True,expand_nested=True)
```

Out[263]:



In [ ]:
```
1  optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
2  loss_function = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Tr
3
4  def maskedLoss(y_true, y_pred):
5      #getting mask value
6      mask = tf.math.logical_not(tf.math.equal(y_true, 0))
7
8      #calculating the loss
9      loss_ = loss_function(y_true, y_pred)
10
11     #converting mask dtype to loss_ dtype
12     mask = tf.cast(mask, dtype=loss_.dtype)
13
14     #applying the mask to loss
15     loss_ = loss_*mask
16
17     #getting mean over all the values
18     loss_ = tf.reduce_mean(loss_)
19     return loss_
```

In [ ]:
```
1 model_atten.compile(optimizer=optimizer, loss=maskedLoss)
```

In [ ]:
```
1 EPOCHS = 10
```

In [ ]:
```
1 current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
2 train_log_dir = 'Tensorboard/attention_OneStep/fit2/' + current_time + '/tra
3 val_log_dir = 'Tensorboard/attention_OneStep/fit2/' + current_time + '/test'
4 train_summary_writer = tf.summary.create_file_writer(train_log_dir)
5 val_summary_writer = tf.summary.create_file_writer(val_log_dir)
```

In [ ]:

```
for img, rep in cv_generator:
  print(img.shape,rep.shape)
```

```
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
(14, 98, 1024) (14, 153)
```

In [ ]:

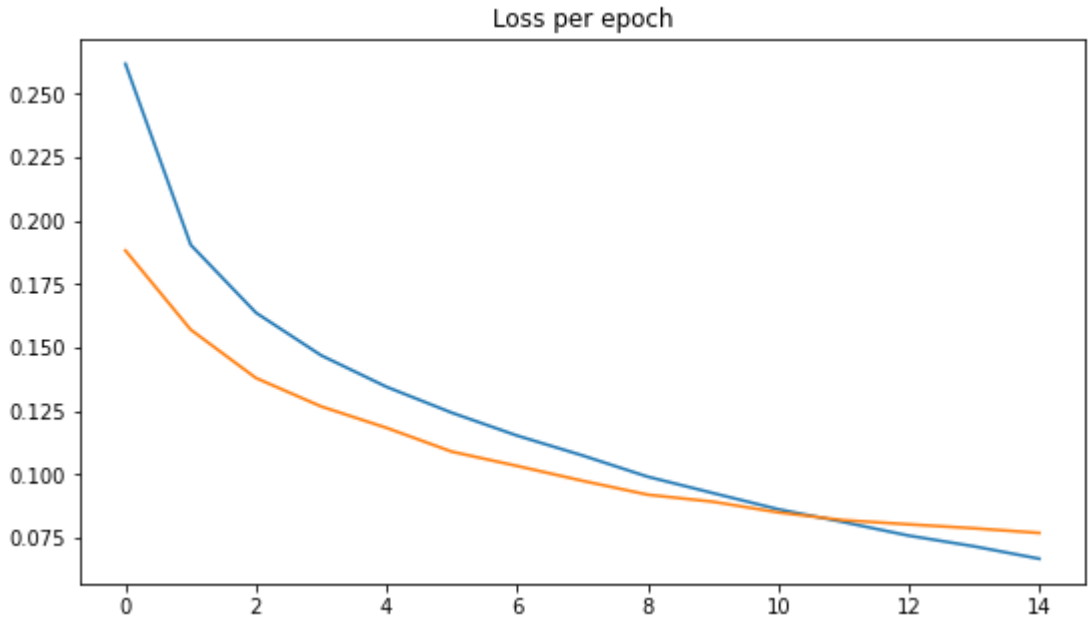
```
epoch_train_loss = []
epoch_val_loss = []

for epoch in range(15):
    start = time.time()
    print("EPOCH: ", epoch+1)
    batch_loss_tr = 0
    batch_loss_val = 0

    for img, rep in train_generator:
        res = model_atten.train_on_batch([img, rep[:,:-1]], rep[:,1:])
        batch_loss_tr += res

    train_loss = batch_loss_tr/(X_train.shape[0]/BATCH_SIZE)

    with train_summary_writer.as_default():
        tf.summary.scalar('loss', train_loss, step = epoch)

    for img, rep in cv_generator:
        res = model_atten.test_on_batch([img, rep[:,:-1]], rep[:,1:])
        batch_loss_val += res

    val_loss = batch_loss_val/(X_cv.shape[0]/BATCH_SIZE)

    with val_summary_writer.as_default():
        tf.summary.scalar('loss', val_loss, step = epoch)

    epoch_train_loss.append(train_loss)

    epoch_val_loss.append(val_loss)

    print('Training Loss: {},  Validation Loss: {}'.format(train_loss, val_l
    print('Time Taken for this Epoch : {} sec'.format(time.time()-start))
    model_atten.save_weights('epoch_'+ str(epoch+1) + '.h5')
    print('-----------------------------------------------------------------
```

```
EPOCH:  1
Training Loss: 0.2619428846058507,  Validation Loss: 0.18833483965733114
Time Taken for this Epoch : 209.3699083328247 sec
--------------------------------------------------------------------------------
--
EPOCH:  2
Training Loss: 0.1905702367651886,  Validation Loss: 0.157130245023813
Time Taken for this Epoch : 59.84448742866516 sec
--------------------------------------------------------------------------------
--
EPOCH:  3
Training Loss: 0.16380256350119102,  Validation Loss: 0.13806124986746374
Time Taken for this Epoch : 59.694589376449585 sec
--------------------------------------------------------------------------------
--
EPOCH:  4
Training Loss: 0.14696644736349884,  Validation Loss: 0.126835676913078
Time Taken for this Epoch : 59.67362976074219 sec
--------------------------------------------------------------------------------
--
```

```
EPOCH:  5
Training Loss: 0.13457499096538814,  Validation Loss: 0.11841330218773621
Time Taken for this Epoch : 59.96831274032593 sec
--------------------------------------------------------------------------------
--
EPOCH:  6
Training Loss: 0.12437395441350597,  Validation Loss: 0.10902612522626534
Time Taken for this Epoch : 59.69680881500244 sec
--------------------------------------------------------------------------------
--
EPOCH:  7
Training Loss: 0.11535636889677363,  Validation Loss: 0.103392177571853
Time Taken for this Epoch : 59.83068513870239 sec
--------------------------------------------------------------------------------
--
EPOCH:  8
Training Loss: 0.10754140804942489,  Validation Loss: 0.09755306738691452
Time Taken for this Epoch : 59.558510541915894 sec
--------------------------------------------------------------------------------
--
EPOCH:  9
Training Loss: 0.09915354566147484,  Validation Loss: 0.09206054455194718
Time Taken for this Epoch : 59.586930990219116 sec
--------------------------------------------------------------------------------
--
EPOCH:  10
Training Loss: 0.09269946219806138,  Validation Loss: 0.08928571230708024
Time Taken for this Epoch : 59.588436126708984 sec
--------------------------------------------------------------------------------
--
EPOCH:  11
Training Loss: 0.08631703974253635,  Validation Loss: 0.08507981437903184
Time Taken for this Epoch : 59.298696994781494 sec
--------------------------------------------------------------------------------
--
EPOCH:  12
Training Loss: 0.08125368146424367,  Validation Loss: 0.08199318937766246
Time Taken for this Epoch : 59.68211483955383 sec
--------------------------------------------------------------------------------
--
EPOCH:  13
Training Loss: 0.07591960616031576,  Validation Loss: 0.08036750822495191
Time Taken for this Epoch : 59.57699203491211 sec
--------------------------------------------------------------------------------
--
EPOCH:  14
Training Loss: 0.07169463908120158,  Validation Loss: 0.07880976184820518
Time Taken for this Epoch : 59.416261196136475 sec
--------------------------------------------------------------------------------
--
EPOCH:  15
Training Loss: 0.06679856850940565,  Validation Loss: 0.07698093421566181
Time Taken for this Epoch : 59.586071491241455 sec
--------------------------------------------------------------------------------
--
```

In [ ]:
```
plt.figure(figsize=(9,5))

plt.plot(epoch_train_loss)
plt.plot(epoch_val_loss)

plt.title('Loss per epoch')
```

Out[272]: Text(0.5, 1.0, 'Loss per epoch')



In [ ]:
```
model_atten.summary()
```

```
Model: "attention__model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_1 (Encoder)          multiple                  262400
_________________________________________________________________
decoder_1 (Decoder)          multiple                  1430712
=================================================================
Total params: 1,693,112
Trainable params: 1,693,112
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
def beam_search(image, beam_width = 3):
    """Beam search implementaion takes images as input"""
    image_features = Xnet_features_attention[image]

    features_val = encoder(image_features)
    start = [tokenizer.word_index["<start>"]]
    dec_word = [[start, 0.0]]
    finished_cap = []
    while len(dec_word[0][0]) < max_doc_length_x:
        temp = []
        new_cap = []
        for s in dec_word:

            predictions, hidden,attention_weights = decoder(tf.cast(tf.expan
            predictions = tf.reshape(predictions, [predictions.shape[0],pred

            word_preds = np.argsort(predictions[0])[-beam_width:]
            cap,score = s
            for w in word_preds:
                candidates = [cap + [w], score - log(predictions[0][w])]
                temp.append(candidates)
        dec_word = sorted(temp, key = lambda l: l[1])[:beam_width]
        count = 0
        for cap,score in dec_word:
          if cap[len(cap)-1] == tokenizer.word_index['<end>']:
            score = score/len(cap)
            finished_cap.append([cap, score])
            count+=1
          else:
            new_cap.append([cap, score])
        beam_width -= count
        dec_word = new_cap
        # if all the dec_word reaches its end before all timesteps
        if not dec_word:
          break
        else:
          continue
    dec_word = finished_cap[-1]
    text = dec_word[0]
    score = dec_word[1]
    result = []
    text.pop(0)
    for word in text:
      if word != tokenizer.word_index['<end>']:
        result.append(tokenizer.index_word[word])
      else:
        break
    text = ' '.join(e for e in result)

    return result, text
```

In [ ]:

```
def inference(inputs):

    in_ = len(inputs.split()) - 1
    inputs = Xnet_features_attention[inputs]
    enc_state = tf.zeros((1, 256))
    enc_output = model_atten.layers[0](inputs)
    input_state = enc_state
    pred = []
    cur_vec = np.array([tokenizer.word_index['startseq']]).reshape(-1,1)

    for i in range(153):

        inf_output, input_state, attention_weights = model_atten.layers[1].o

        cur_vec = np.reshape(np.argmax(inf_output), (1, 1))
        if cur_vec[0][0] != 0:
            pred.append(cur_vec)
        else:
            break

    output = ' '.join([tokenizer.index_word[e[0][0]] for e in pred if e[0][0
    return output
```

CV

In [ ]:

```
from tqdm.notebook import tqdm

bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []
for img, rep in tqdm(zip(X_cv.values, y_cv.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep = inference(img)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0

a = sum(bleu1)/X_cv.shape[0]
b = sum(bleu2)/X_cv.shape[0]
c = sum(bleu3)/X_cv.shape[0]
d = sum(bleu4)/X_cv.shape[0]

print('Bleu1 Score: ',a)
print('Bleu2 Score: ',b)
print('Bleu3 Score: ',c)
print('Bleu4 Score: ',d)

print('----------------------------------------------------------------------
print("Avg Blue score:",(a+b+c+d)/4)
```

```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
='')))


Bleu1 Score:  0.18160201949602198
Bleu2 Score:  0.17133661156441182
Bleu3 Score:  0.18028301971740157
Bleu4 Score:  0.19899543673720677
----------------------------------------------------------------------
Avg Blue score: 0.18305427187876053
```

Test

In [ ]:

```
bleu1 = []
bleu2 = []
bleu3 = []
bleu4 = []
for img, rep in tqdm(zip(X_test.values, y_test.values)):

    rep = rem_fullstops(rep)
    rep = rep.split()[1:]
    rep = rep[:len(rep)-1]
    rep = ' '.join(e for e in rep)

    pred_rep = inference(img)
    pred_rep = rem_fullstops(pred_rep)

    bleu1.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (1
    bleu2.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu3.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0
    bleu4.append(sentence_bleu([rep.split()], pred_rep.split(), weights = (0

a = sum(bleu1)/X_cv.shape[0]
b = sum(bleu2)/X_cv.shape[0]
c = sum(bleu3)/X_cv.shape[0]
d = sum(bleu4)/X_cv.shape[0]

print('Bleu1 Score: ',a)
print('Bleu2 Score: ',b)
print('Bleu3 Score: ',c)
print('Bleu4 Score: ',d)

print('----------------------------------------------------------------------')
print("Avg Blue score:",(a+b+c+d)/4)
```

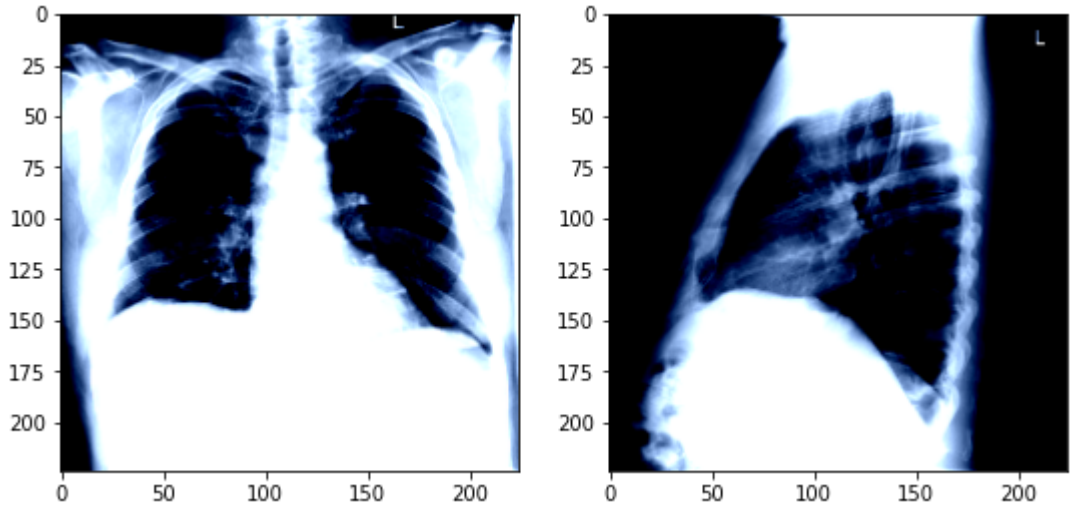
```
HBox(children=(FloatProgress(value=1.0, bar_style='info', max=1.0), HTML(value
='')))


Bleu1 Score:  0.11383215029474758
Bleu2 Score:  0.10556084807355325
Bleu3 Score:  0.11101751407061891
Bleu4 Score:  0.1234984414656531
----------------------------------------------------------------------
Avg Blue score: 0.1134772384761432
```

In [ ]:

```
def load_image(img_name):
    image = Image.open(img_name)
    X = np.asarray(image.convert("RGB"))
    X = np.asarray(X)
    X = preprocess_input(X)
    X = resize(X, (224,224,3))
    X = np.expand_dims(X, axis=0)
    X = np.asarray(X)
    return X

def get_result_attention_model(idx):
  plt.figure(figsize=(9,5))

  pre_Report = inference(cv_dataset['Person_id'][idx]) # result after 20 epo
  print('--------------------------------------------------------------------')
  print("Predicted Report : ",pre_Report)
  print('--------------------------------------------------------------------')
  print("Actual Report : ",cv_dataset['Report'][idx])

  plt.subplot(121)
  img = load_image(cv_dataset['Image1'][idx])
  plt.imshow(img[0])

  plt.subplot(122)
  img = load_image(cv_dataset['Image2'][idx])
  plt.imshow(img[0])
```

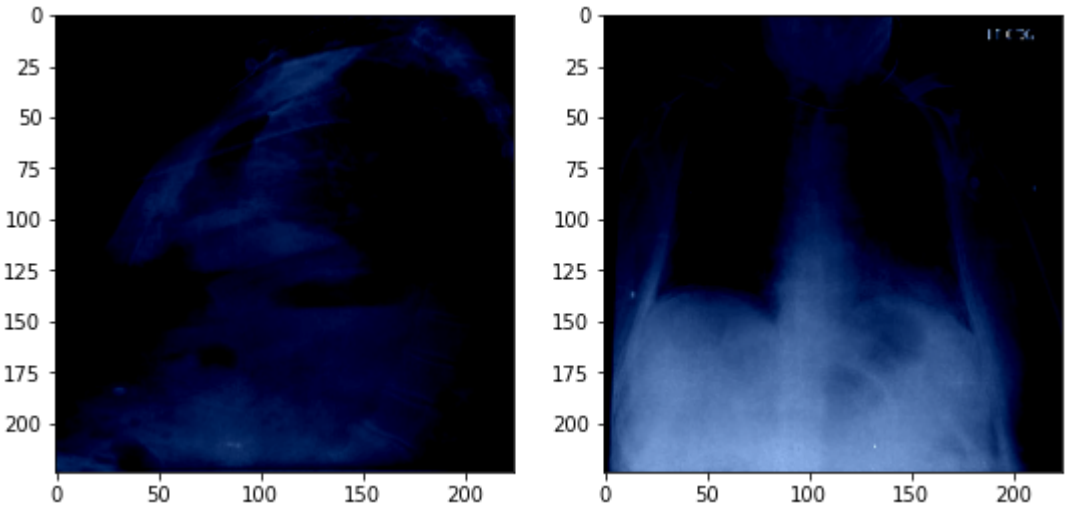
In [ ]:
```
1 get_result_attention_model(72)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

--------------------------------------------------------------------------------
----------------------
Predicted Report :  the heart normal size . the mediastinum unremarkable . the
lungs are clear .
--------------------------------------------------------------------------------
----------------------
Actual Report :  startseq the heart normal size .  the mediastinum unremarkable
.  the lungs are clear .  endseq
```



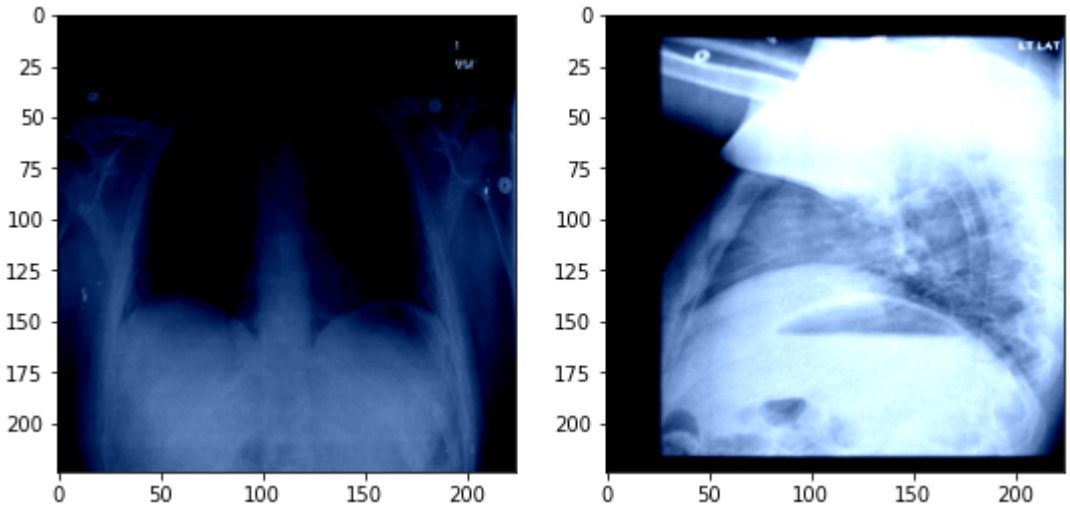
In [ ]:
```
1 get_result_attention_model(399)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

------------------------------------------------------------------------------
----------------------
Predicted Report :  the heart size and pulmonary vascularity appear within norm
al limits . there no focal airspace opacity pleural effusion pneumothora . the
are intact .
------------------------------------------------------------------------------
----------------------
Actual Report :  startseq low lung volumes with bibasilar subsegmental atelecta
sis .  no focal consolidations pleural effusions pneumothoraces .  cardiomedias
tinal silhouette within normal limits .  degenerative changes the thoracic spin
e .  endseq
```



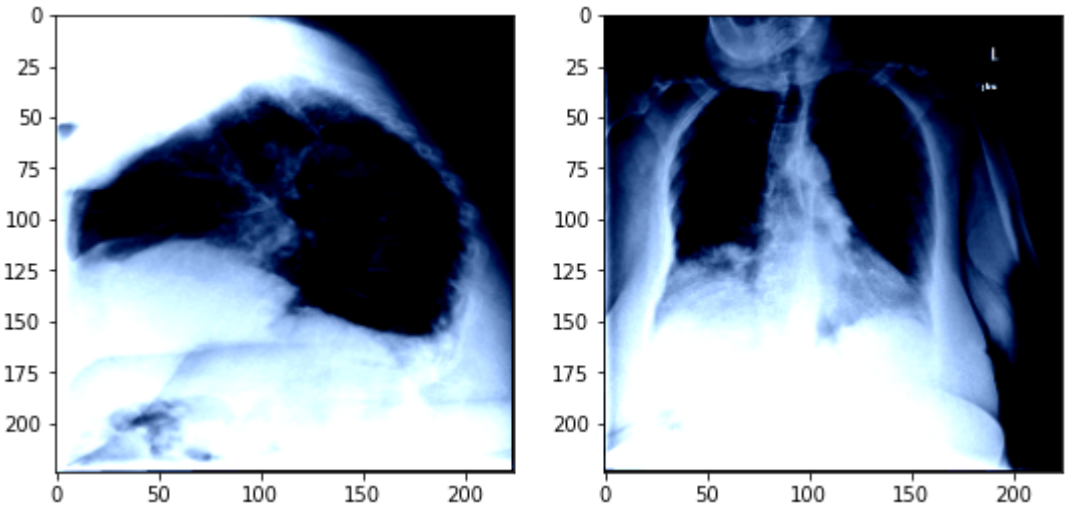
In [ ]:
```
1 get_result_attention_model(34)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

-----------------------------------------------------------------------------
-----------------------
Predicted Report :  the heart normal size . the mediastinum unremarkable . the
lungs are clear .
-----------------------------------------------------------------------------
-----------------------
Actual Report :  startseq there are low lung volumes .  the heart size and uppe
r mediastinum have normal appearance .  there no pulmonary vascular congestion
.  there minimal right basilar atelectasis .  there no large effusion pneumotho
ra .  the osseous structures appear intact .  endseq
```



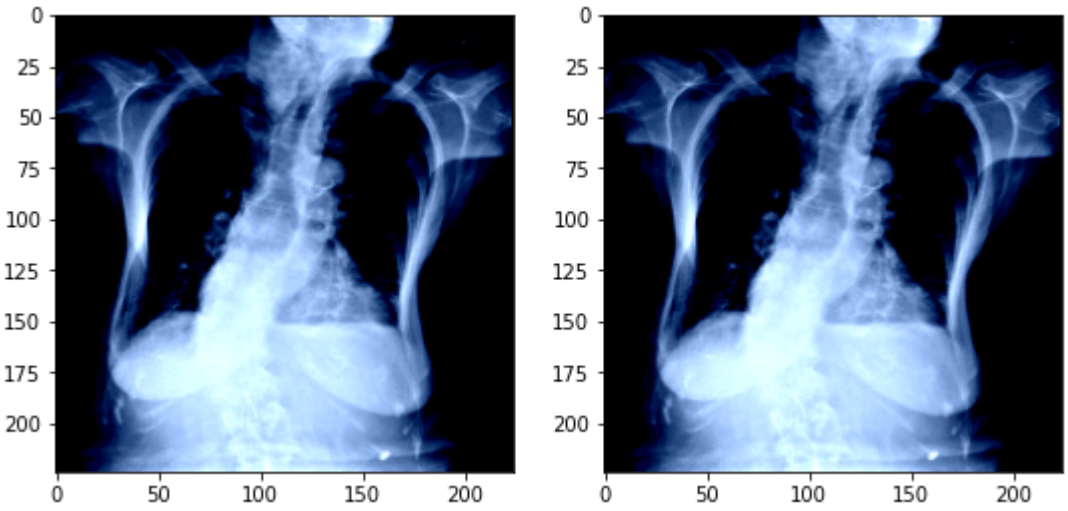
In [ ]:
```
1 get_result_attention_model(98)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

------------------------------------------------------------------------------
----------------------
Predicted Report :  the heart size and pulmonary vascularity appear within norm
al limits . there no focal airspace opacity pleural effusion pneumothora . the
are intact .
------------------------------------------------------------------------------
----------------------
Actual Report :  startseq the heart mildly enlarged .  pulmonary vascularity in
creased .  there again mild elevation the right hemidiaphragm .  air space dise
ase andor atelectasis noted right lung base .  there also streaky opacity the l
eft base .  the costophrenic are blunted .  endseq
```



In [ ]:
```
1 get_result_attention_model(13)
```

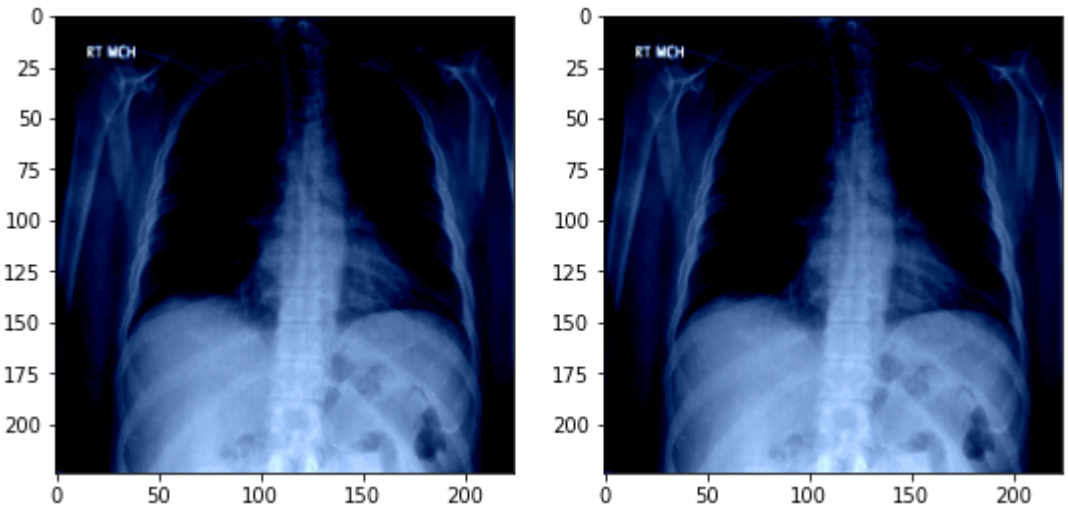
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

------------------------------------------------------------------------------
----------------------
Predicted Report :  the heart size and pulmonary vascularity appear within norm
al limits . there no pleural effusion pneumothora . there are no focal air spac
e opacity suggest pneumonia . there are no acute bony abnormalities .
------------------------------------------------------------------------------
----------------------
Actual Report :  startseq stable appearance the cardiomediastinal silhouette .
the aorta calcified and tortuous .  there detroscoliosis the thoracolumbar spin
e .  multiple thoracic deformities appear unchanged .  there no displaced rib f
racture identified .  there no pneumothora large pleural effusion .  stable cha
nges chronic lung disease with flattening the left hemidiaphragm .  there mild
right basilar airspace disease which may represent atelectasis versus infiltrat
e .  endseq
```



In [ ]:
```
def get_result(beam_width,idx=0):

  plt.figure(figsize=(9,5))

  pre_Report,Score = beamsearch(cv_data['Person_id'][idx],beam_width) # resu
  print('---------------------------------------------------------------------
  print("Predicted Report : ",pre_Report)
  print('Score is :',Score)
  print('---------------------------------------------------------------------
  print("Actual Report : ",cv_data['Report'][idx])

  plt.subplot(121)
  img = load_image(cv_data['Image1'][idx])
  plt.imshow(img[0])

  plt.subplot(122)
  img = load_image(cv_data['Image2'][idx])
  plt.imshow(img[0])
```

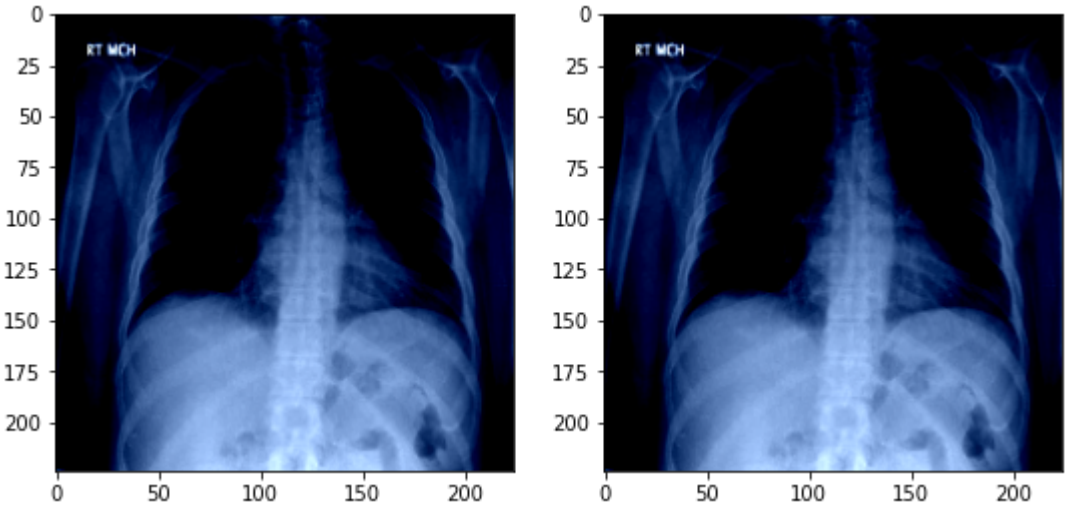
In [ ]:
```
#beam width = 10
get_result(10,5)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

---------------------------------------------------------------------------
-----------------------
Predicted Report :  the lungs are clear bilaterally . specifically no evidence
focal consolidation pneumothora pleural effusion . cardio mediastinal silhouett
e unremarkable . visualized osseous structures the thora are without acute abno
rmality . there are no acute bony abnormality .
Score is : 0.40586483738241863
---------------------------------------------------------------------------
-----------------------
Actual Report :  startseq normal heart size and mediastinal contours .  low lun
g volumes with no significant airspace consolidation .  no pleural effusion pne
umothora .  visualized osseous structures are unremarkable appearance .  endseq
```



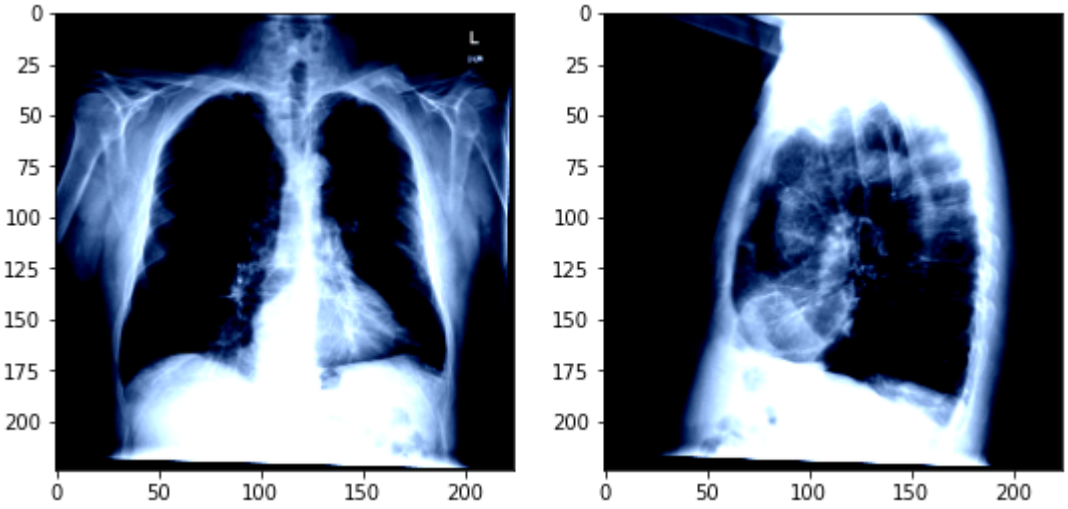
In [ ]:
```
#beam width = 10
get_result(5,5)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).


--------------------------------------------------------------------------------
----------------------
Predicted Report :  the cardiomediastinal silhouette within normal limits for s
ize and contour . the lungs are normally inflated without evidence focal airspa
ce disease pleural effusion pneumothora . visualized osseous structures are unr
emarkable .
Score is : 0.3412864783304306
--------------------------------------------------------------------------------
----------------------
Actual Report :  startseq normal heart size and mediastinal contours .  low lun
g volumes with no significant airspace consolidation .  no pleural effusion pne
umothora .  visualized osseous structures are unremarkable appearance .  endseq
```



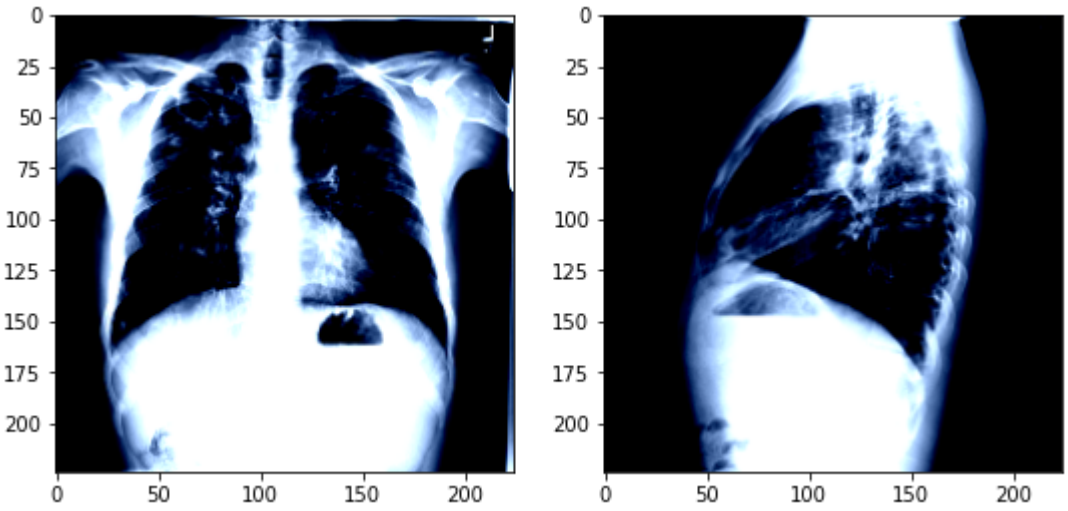
In [ ]:
```
#beam width = 10
get_result(10,11)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

--------------------------------------------------------------------------------
-----------------------
Predicted Report :  the lungs are clear bilaterally . specifically no evidence
focal consolidation pneumothora pleural effusion . cardio mediastinal silhouett
e unremarkable . visualized osseous structures the thora are without acute nega
tive .
Score is : 0.3476788840998779
--------------------------------------------------------------------------------
-----------------------
Actual Report :  startseq heart size and mediastinal contour within normal limi
ts .  aortic atherosclerotic calcifications .  emphysematous changes .  nodular
densities projecting over right anterior fifth and ribs .  no focal airspace co
nsolidation pneumothora large pleural effusion .  no acute osseous abnormality
.  endseq
```



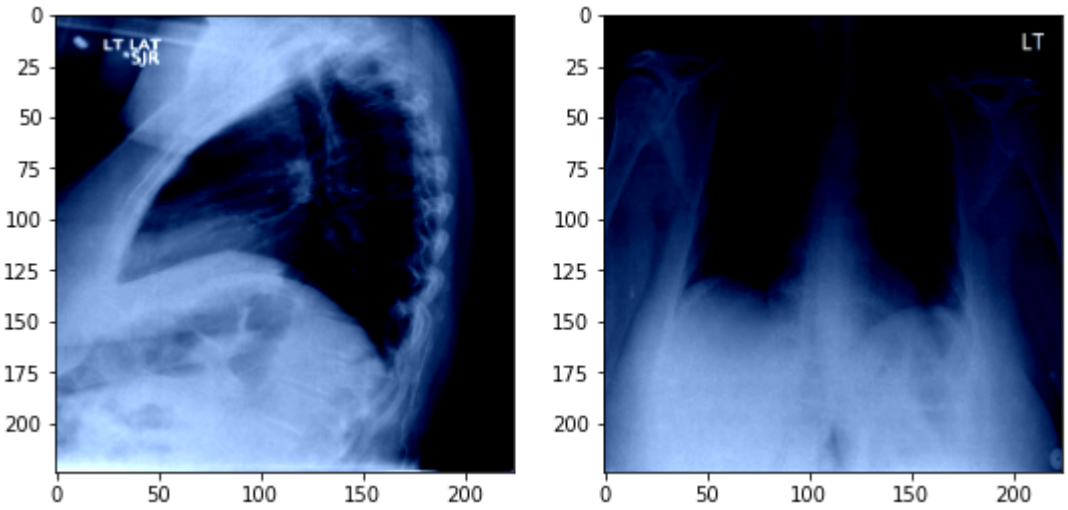
In [ ]:
```
#beam width = 10
get_result(10,39)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

--------------------------------------------------------------------------------
-----------------------
Predicted Report :  the heart size and pulmonary vascularity appear within norm
al limits . the lungs are free focal airspace disease . no pleural effusion pne
umothora seen . there are no acute bony findings .
Score is : 0.4011605606193531
--------------------------------------------------------------------------------
-----------------------
Actual Report :  startseq heart size within normal limits .  prominent intersti
tial and nodular opacities are increased since comparison eam .  there nodular
opacity the right costophrenic increased since comparison eamination .  cystic
lesion the right upper lobe appears similar prior eamination .  no pleural effu
sion pneumothora .  endseq
```



In [ ]:
```
#beam width = 10
get_result(10,117)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for flo
ats or [0..255] for integers).

--------------------------------------------------------------------------------
----------------------
Predicted Report :  the cardiomediastinal silhouette and pulmonary vasculature
are within normal limits size . the lungs are clear focal airspace disease pneu
mothora pleural effusion . there are no acute bony abnormality .
Score is : 0.32011180980862264
--------------------------------------------------------------------------------
----------------------
Actual Report :  startseq no focal areas consolidation .  no suspicious pulmona
ry opacities .  heart size within normal limits .  no pleural effusions .  no e
vidence pneumothora .  osseous structures intact .  endseq
```



In [5]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Data","Avg_blue_score"]
x.add_row(["Encoder Decoder", "CV", 0.18114848876146578])
x.add_row(["","",""])
x.add_row(["", "Test", 0.1617484175657567])
x.add_row(["-----------------", "------------------","------------------"])


x.add_row(["Attention Mechanism", "CV",  0.199915143372483])
x.add_row(["","",""])
x.add_row(["", "Test", 0.12827193524412095])

print(x)
```

```
+--------------------+--------------------+---------------------+
|       Model        |        Data        |    Avg_blue_score   |
+--------------------+--------------------+---------------------+
|   Encoder Decoder  |         CV         | 0.18114848876146578 |
|                    |                    |                     |
|                    |        Test        |  0.1617484175657567 |
|  -----------------  | ------------------ |  ------------------ |
| Attention Mechanism |         CV         |  0.199915143372483  |
|                    |                    |                     |
|                    |        Test        | 0.12827193524412095 |
+--------------------+--------------------+---------------------+
```

Observations :

1. With less epochs attention model is predicting same result.
2. When number of epochs increased attention model is now performing better
3. Attention model prediction using greedy search and beam search seems to be same. So to reduce the time complexity of the model it is wise to use greedy search for predicting output.
4. Analyzing the performnce of both encoder decoder model and attention model,we can go for encoder decoder model as its is result is quite better using the beam search algorithm also the model complexity less.

Future work:

1. we can use EfficientNetB7 pretrained model to extract features from the images and then implement out model on top of it.
2. Collecting more data samples in order to make this model more powerful is highly required.Our present model is not so accurate due to fewer dataset at hand.