In [ ]:
```
cd /content/drive/My Drive/Xray report generation
```

```
/content/drive/My Drive/Xray report generation
```

In [ ]:
```
ls
```

```
 AttentionModel.ipynb                            Medical.csv
 Basic_Model.ipynb                               model_1_plot.png
 brucechou1983_CheXNet_Keras_0.3.0_weights.h5    NLMCXR_png/
 Chexnetmodel/                                  'Old Files'/
'Copy of AttentionModel.ipynb'                   preprocessed.csv
'Copy of SimpleModel.ipynb'                      simple_encoder_decoder_plot.png
 dup_preprocessed.csv                            SimpleModel.ipynb
 ecgen-radiology/                                tokenizer1.pkl
 image_feature_vector1.pkl                       Untitled0.ipynb
 indiana_projections.csv                         Untitled1.ipynb
 logs/
```

In [ ]:
```
import re
import pandas as pd
import matplotlib.pyplot as plt
import string
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import numpy as np
import tensorflow as tf
from nltk.translate.bleu_score import sentence_bleu
```

In [ ]:
```
df = pd.read_csv('Medical.csv')
```

In [ ]:
```
df.head()
```

Out[5]:

	image_name	image_caption	comparison	indication	findings	impression
0	CXR1_1_IM-0001-3001.png,CXR1_1_IM-0001-4001.png	xray chest pa and lateral	none	positive tb test	the cardiac silhouette and mediastinum size ar...	normal chest x
1	CXR10_IM-0002-2001.png,CXR10_IM-0002-1001.png	pa and lateral chest x	chest radiographs	male chest pain	the cardiomediastinal silhouette is within nor...	no acute cardiopulmonary process
2	CXR100_IM-0002-1001.png,CXR100_IM-0002-2001.png	chest v frontallateral pm	none	no indication	both lungs are clear and expanded heart and me...	no active disease
3	CXR1000_IM-0003-1001.png,CXR1000_IM-0003-2001....	pa and lateral chest x	pa and lateral chest radiographs	male	there is increased opacity within the right up...	increased opacity in the right upper lobe with...
4	CXR1001_IM-0004-1001.png,CXR1001_IM-0004-1002.png	chest v frontallateral pm	none	dyspnea subjective fevers arthritis immigrant ...	interstitial markings are diffusely prominent ...	diffuse fibrosis no visible focal acute disease

In [ ]:
```
df.shape
```

Out[6]:
```
(3851, 9)
```

In [ ]:
```
df_projections = pd.read_csv('indiana_projections.csv')
```

In [ ]:
```
df_projections.head()
```

Out[11]:

	uid	filename	projection
0	1	1_IM-0001-4001.dcm.png	Frontal
1	1	1_IM-0001-3001.dcm.png	Lateral
2	2	2_IM-0652-1001.dcm.png	Frontal
3	2	2_IM-0652-2001.dcm.png	Lateral
4	3	3_IM-1384-1001.dcm.png	Frontal

In [ ]:
```
img_path = 'NLMCXR_png/'
```

In [ ]:
```
# to find the matching pattern in a string - https://thepythonguru.com/python-regu
def find_fr_lr_images(li):
  """
  This funct is used to find the lateral images and frontal images from our list o
  """
  img_list = []
  last_imag = ""
  for img in li:
    projections = df_projections[df_projections['filename'].str.contains(re.searc
    if 'Lateral'== projections:
      last_imag = img
    else:
      img_list.append(img)
  return img_list, last_imag
```

In [ ]:
```
#Creating structured data from raw xml files
columns = ["image_1", "image_2", "impression"]
df_1 = pd.DataFrame(columns = columns)
columns = ["image_1", "image_2", "impression"]
df_dup = pd.DataFrame(columns = columns)
no_lateral = 0
for item in tqdm(df.iterrows()):
    l = item[1]['image_name'].split(',')
    if len(l) > 2:
        li, last_img = find_fr_lr_images(l)
        if last_img == "":
            no_lateral +=1
            li, last_img = li[:-1], li[-1]
        for i in li:
            image_1 = i
            image_2 = last_img
            df_1 = df_1.append(pd.Series([image_1, image_2,item[1]['impression']]
    elif len(l) == 2:
        image_1 = l[0]
        image_2 = l[1]
        df_1 = df_1.append(pd.Series([image_1, image_2, item[1]['impression']], i
    elif len(l) == 1:
        #creating duplicate dataframe separately to keep it in all dataset train
        df_dup = df_dup.append(pd.Series([l[0], l[0],item[1]['impression']], inde
print("Total Report without Lateral images {}".format(no_lateral))
```

```
1155it [00:03, 312.59it/s]/usr/local/lib/python3.6/dist-packages/ipykernel_laun
cher.py:10: DeprecationWarning: The truth value of an empty array is ambiguous.
Returning False, but in future this will result in an error. Use `array.size >
0` to check that an array is not empty.
  # Remove the CWD from sys.path while we load stuff.
3851it [00:13, 291.25it/s]

Total Report without Lateral images 1
```

In [ ]:
```
df_1.shape
```

Out[15]:
```
(3532, 3)
```

In [ ]:
```
df_1.head()
```

Out[16]:

	image_1	image_2	impression
0	CXR1_1_IM-0001-3001.png	CXR1_1_IM-0001-4001.png	normal chest x
1	CXR10_IM-0002-2001.png	CXR10_IM-0002-1001.png	no acute cardiopulmonary process
2	CXR100_IM-0002-1001.png	CXR100_IM-0002-2001.png	no active disease
3	CXR1000_IM-0003-1001.png	CXR1000_IM-0003-2001.png	increased opacity in the right upper lobe with...
4	CXR1000_IM-0003-3001.png	CXR1000_IM-0003-2001.png	increased opacity in the right upper lobe with...

In [ ]:
```
df_dup.shape
```

Out[17]:
```
(446, 3)
```

In [ ]:
```
df_dup.head()
```

Out[18]:

	image_1	image_2	impression
0	CXR1003_IM-0005-2002.png	CXR1003_IM-0005-2002.png	retrocardiac soft tissue density the appearanc...
1	CXR1012_IM-0013-1001.png	CXR1012_IM-0013-1001.png	bibasilar airspace disease and bilateral pleur...
2	CXR1024_IM-0019-1001.png	CXR1024_IM-0019-1001.png	no acute abnormality
3	CXR1026_IM-0021-2002.png	CXR1026_IM-0021-2002.png	no acute cardiopulmonary disease
4	CXR1029_IM-0022-1001.png	CXR1029_IM-0022-1001.png	no pneumonia heart size normal scoliosis

Adding Start and end tokens

In [ ]:
```
def add_start_end_token(data):
    # Combining all the above stundents
    preprocessed_reviews_eng = []

    # tqdm is for printing the status bar
    for sentance in tqdm(data.values):
        sentance = '<start> ' + sentance + ' <end>'
        preprocessed_reviews_eng.append(sentance.strip())
    return preprocessed_reviews_eng
```

In [ ]:
```
df_1['impression'] = add_start_end_token(df_1['impression'].astype(str))
```

```
100%|██████████| 3532/3532 [00:00<00:00, 771697.75it/s]
```

In [ ]:
```
df_dup['impression'] = add_start_end_token(df_dup['impression'].astype(str))
```

```
100%|██████████| 446/446 [00:00<00:00, 396242.23it/s]
```

In [ ]:
```
df_1.head()
```

Out[22]:

	image_1	image_2	impression
0	CXR1_1_IM-0001-3001.png	CXR1_1_IM-0001-4001.png	<start> normal chest x <end>
1	CXR10_IM-0002-2001.png	CXR10_IM-0002-1001.png	<start> no acute cardiopulmonary process <end>
2	CXR100_IM-0002-1001.png	CXR100_IM-0002-2001.png	<start> no active disease <end>
3	CXR1000_IM-0003-1001.png	CXR1000_IM-0003-2001.png	<start> increased opacity in the right upper l...
4	CXR1000_IM-0003-3001.png	CXR1000_IM-0003-2001.png	<start> increased opacity in the right upper l...

In [ ]:
```
df_dup.head()
```

Out[23]:

	image_1	image_2	impression
0	CXR1003_IM-0005-2002.png	CXR1003_IM-0005-2002.png	<start> retrocardiac soft tissue density the a...
1	CXR1012_IM-0013-1001.png	CXR1012_IM-0013-1001.png	<start> bibasilar airspace disease and bilater...
2	CXR1024_IM-0019-1001.png	CXR1024_IM-0019-1001.png	<start> no acute abnormality <end>
3	CXR1026_IM-0021-2002.png	CXR1026_IM-0021-2002.png	<start> no acute cardiopulmonary disease <end>
4	CXR1029_IM-0022-1001.png	CXR1029_IM-0022-1001.png	<start> no pneumonia heart size normal scolios...

In [ ]:
```
df_1.to_csv('preprocessed.csv')
```

In [ ]:
```
df_dup.to_csv('dup_preprocessed.csv')
```

In [ ]:
```
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3532 entries, 0 to 3531
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_1     3532 non-null   object
 1   image_2     3532 non-null   object
 2   impression  3532 non-null   object
dtypes: object(3)
memory usage: 82.9+ KB
```

In [ ]:
```
df_dup.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 446 entries, 0 to 445
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_1     446 non-null    object
 1   image_2     446 non-null    object
 2   impression  446 non-null    object
dtypes: object(3)
memory usage: 10.6+ KB
```

In [ ]:
```
df_1 = pd.read_csv('preprocessed.csv')
```

In [ ]:
```
df_1.shape
```

Out[9]:
```
(3532, 4)
```

In [ ]:
```
df_1.head()
```

Out[10]:

	Unnamed: 0	image_1	image_2	impression
0	0	CXR1_1_IM-0001-3001.png	CXR1_1_IM-0001-4001.png	<start> normal chest x <end>
1	1	CXR10_IM-0002-2001.png	CXR10_IM-0002-1001.png	<start> no acute cardiopulmonary process <end>
2	2	CXR100_IM-0002-1001.png	CXR100_IM-0002-2001.png	<start> no active disease <end>
3	3	CXR1000_IM-0003-1001.png	CXR1000_IM-0003-2001.png	<start> increased opacity in the right upper l...
4	4	CXR1000_IM-0003-3001.png	CXR1000_IM-0003-2001.png	<start> increased opacity in the right upper l...

In [ ]:
```
df_2 = pd.read_csv("dup_preprocessed.csv")
```

In [ ]:
```
df_2.head()
```

Out[12]:

	Unnamed: 0	image_1	image_2	impression
0	0	CXR1003_IM-0005-2002.png	CXR1003_IM-0005-2002.png	<start> retrocardiac soft tissue density the a...
1	1	CXR1012_IM-0013-1001.png	CXR1012_IM-0013-1001.png	<start> bibasilar airspace disease and bilater...
2	2	CXR1024_IM-0019-1001.png	CXR1024_IM-0019-1001.png	<start> no acute abnormality <end>
3	3	CXR1026_IM-0021-2002.png	CXR1026_IM-0021-2002.png	<start> no acute cardiopulmonary disease <end>
4	4	CXR1029_IM-0022-1001.png	CXR1029_IM-0022-1001.png	<start> no pneumonia heart size normal scolios...

In [ ]:
```
image_names = []
for img in tqdm(df['image_name'].str.split(',')):

  for i in range(len(img)):
    image_names.append(img[i])
```

```
100%|██████████| 3851/3851 [00:00<00:00, 894713.60it/s]
```

In [ ]:
```
image_names[0:4]
```

Out[14]:
```
['CXR1_1_IM-0001-3001.png',
 'CXR1_1_IM-0001-4001.png',
 'CXR10_IM-0002-2001.png',
 'CXR10_IM-0002-1001.png']
```

In [ ]:
```
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
```

In [ ]:
```
chex = densenet.DenseNet121(include_top=False,weights=None,input_shape=(224,224,3
X = chex.output
X = Dense(14,activation='sigmoid',name='predictions')(X)
model = Model(inputs=chex.input,outputs=X)
model.load_weights('brucechou1983_CheXNet_Keras_0.3.0_weights.h5')
image_features_model = Model(inputs=model.input,outputs= model.layers[-2].output)
```

In [ ]:
```
# image_features_model.summary()
```

In [ ]:
```
## the input layer shape is (None,None,None,3)
## the output layer is (None,2048)
```

In [ ]:
```
#### This cell takes around 2 hrs to run.
# image_feature_vectors = []
# for img in tqdm(image_names):
#   image = tf.io.read_file(img_path+str(img))
#   image = tf.image.decode_jpeg(image,channels=3)
#   image = tf.image.resize(image,(299,299))
#   image = preprocess_input(image)
#   image_features = image_features_model(tf.constant(image)[None,:]) # here we a
#   image_features = tf.reshape(image_features,[-1,image_features.shape[1]])
#   image_feature_vectors.append(image_features)
```

In [ ]:
```
# print(image_feature_vectors[1])
```

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

In [ ]:
```
# import pickle

# with open('image_feature_vector.pkl','wb') as f:
#   pickle.dump(image_feature_vectors,f)
```

In [ ]:
```
import pickle
with open('image_feature_vector1.pkl','rb') as f:
  image_feature_vector_1 = pickle.load(f)
```

In [ ]:
```
print(image_feature_vector_1[1])
```

```
tf.Tensor(
[[5.0062913e-04 2.0340595e-03 1.2797897e-03 ... 9.1456938e-01
  9.2038012e-01 7.4285048e-01]], shape=(1, 1024), dtype=float32)
```

Train, Validation and Test split

In [ ]:
```
i_train,input_test,o_train,output_test = train_test_split(df_1[['image_1','image_
input_train,input_validate,output_train,output_validate = train_test_split(i_trai
```

In [ ]:
```
input_train.shape,input_validate.shape,input_test.shape,output_train.shape,output_
```

Out[20]:
```
((2542, 2), (636, 2), (354, 2), (2542,), (636,), (354,))
```

In [ ]:
```
i_train_dup,input_test_dup,o_train_dup,output_test_dup = train_test_split(df_2[['
input_train_dup,input_validate_dup,output_train_dup,output_validate_dup = train_t
```

In [ ]:
```
input_train_dup.shape,input_validate_dup.shape,input_test_dup.shape,output_train_
```

Out[22]:
```
((320, 2), (81, 2), (45, 2), (320,), (81,), (45,))
```

Appending the duplicate data and the original data

In [ ]:
```
i_train = np.append(input_train,input_train_dup,axis=0)
o_train = np.append(output_train,output_train_dup,axis=0)
i_validate = np.append(input_validate,input_validate_dup,axis=0)
o_validate = np.append(output_validate,output_validate_dup,axis=0)
i_test = np.append(input_test,input_test_dup,axis=0)
o_test = np.append(output_test,output_test_dup,axis=0)
```

In [ ]:
```
i_train[0]
```

Out[24]:
```
array(['CXR914_IM-2417-1001.png', 'CXR914_IM-2417-3001.png'], dtype=object)
```

In [ ]:
```
#https://datascience.stackexchange.com/questions/24511/why-should-the-data-be-shu
# Here we will be shuffling the data
for i in range(5):
  i_train,o_train = shuffle(i_train,o_train,random_state=15)
  i_validate,o_validate = shuffle(i_validate,o_validate,random_state=15)
  i_test,o_test = shuffle(i_test,o_test,random_state=15)
```

TEXT TOKENIZATION

In [ ]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_len_output = 60

tokenizer = Tokenizer(oov_token="<unk>", filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}~ '
tokenizer.fit_on_texts(o_train)
text_train = tokenizer.texts_to_sequences(o_train)
text_test = tokenizer.texts_to_sequences(o_test)
text_val = tokenizer.texts_to_sequences(o_validate)
dictionary = tokenizer.word_index

word2idx = {}
idx2word = {}
for k, v in dictionary.items():
    word2idx[k] = v
    idx2word[v] = k
```

In [ ]:
```
vocab_size = len(word2idx)+1
vocab_size
```

Out[27]:
```
1339
```

In [ ]:
```
print("===== Top 6 Word and its Index =====")
list(dictionary.items())[:6]
```

```
===== Top 6 Word and its Index =====
```

Out[28]:
```
[('<unk>', 1),
 ('<start>', 2),
 ('<end>', 3),
 ('no', 4),
 ('acute', 5),
 ('cardiopulmonary', 6)]
```

In [ ]:
```
## Padding the text
text_train_output = pad_sequences(text_train,maxlen=60,dtype='int32',padding='pos
text_validation_output = pad_sequences(text_val,maxlen=60,dtype='int32',padding='
text_test_output = pad_sequences(text_test,maxlen=60,dtype='int32',padding='post'
```

In [ ]:
```
text_train_output.shape
```

Out[30]:
```
(2862, 60)
```

Creating Tensorflow Dataset

In [ ]:
```
# converting the numpy image files into tensorformats

def multi_image(img,imp):
  """
  This function will be taking in the imgaes we have and converting into tensor f
  """
  return tf.convert_to_tensor([image_feature_vector_1[image_names.index(img[0].de
```

In [ ]:
```
dataset_train = tf.data.Dataset.from_tensor_slices((i_train,text_train_output))
```

In [ ]:
```
dataset_validation = tf.data.Dataset.from_tensor_slices((i_validate,text_validati
```

In [ ]:
```
dataset_train = dataset_train.map(lambda item1,item2: tf.numpy_function(multi_ima
dataset_validation = dataset_validation.map(lambda item1,item2: tf.numpy_function
```

In [ ]:
```
BATCH_SIZE = 32
BUFFER_SIZE = 500
embedding_dimension = 256
units = 512 # output
```

In [ ]:
```
# Shuffle and Batch
dataset_train = dataset_train.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset_train = dataset_train.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

#Shuffle and Batch
dataset_validation = dataset_validation.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset_validation = dataset_validation.prefetch(buffer_size=tf.data.experimental
```

In [ ]:
```
# input_shape is (2,1,1024)
# vocab_size = 1225
```

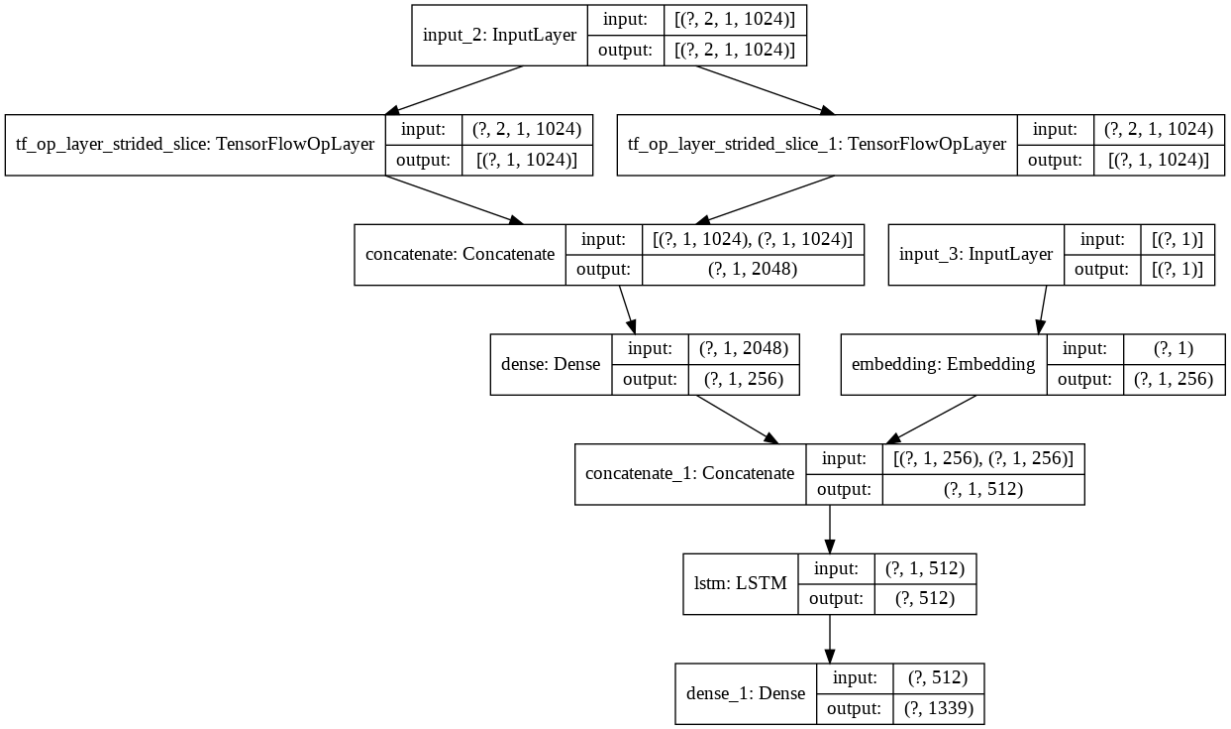
In [ ]:
```
vocab_size = 1339
input_layer = tf.keras.layers.Input(shape=(2,1,1024))
encoder_concat = tf.keras.layers.Concatenate()([input_layer[:,0], input_layer[:,1
encoder_out = tf.keras.layers.Dense(embedding_dimension)(encoder_concat)
input_layer_text = tf.keras.layers.Input(shape=(1,))
x = tf.keras.layers.Embedding(vocab_size, embedding_dimension)(input_layer_text)
x = tf.keras.layers.Concatenate()([x, encoder_out])
x = tf.keras.layers.LSTM(units)(x)
x = tf.keras.layers.Dense(vocab_size)(x)
model = tf.keras.Model([input_layer, input_layer_text],x)
model.summary()
```

```
Model: "functional_5"
__________________________________________________________________________________
_________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
input_2 (InputLayer)            [(None, 2, 1, 1024)] 0
__________________________________________________________________________________
_________________
tf_op_layer_strided_slice (Tens [(None, 1, 1024)]    0           input_2[0][0]
__________________________________________________________________________________
_________________
tf_op_layer_strided_slice_1 (Te [(None, 1, 1024)]    0           input_2[0][0]
__________________________________________________________________________________
_________________
input_3 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________
_________________
concatenate (Concatenate)       (None, 1, 2048)      0           tf_op_layer_st
rided_slice[0][0]
                                                                 tf_op_layer_st
rided_slice_1[0][0]
__________________________________________________________________________________
_________________
embedding (Embedding)           (None, 1, 256)       342784      input_3[0][0]
__________________________________________________________________________________
_________________
dense (Dense)                   (None, 1, 256)       524544      concatenate[0]
[0]
__________________________________________________________________________________
_________________
concatenate_1 (Concatenate)     (None, 1, 512)       0           embedding[0]
[0]
                                                                 dense[0][0]
__________________________________________________________________________________
_________________
lstm (LSTM)                     (None, 512)          2099200     concatenate_1
[0][0]
__________________________________________________________________________________
_________________
dense_1 (Dense)                 (None, 1339)         686907      lstm[0][0]
==================================================================================
==================
Total params: 3,653,435
```

```
Trainable params: 3,653,435
Non-trainable params: 0
_________________________________________________________________
_________________
```

In [ ]:
```
tf.keras.utils.plot_model(model,to_file='simple_encoder_decoder_plot.png',show_sh
```

Out[40]:



In [ ]:
```
class Encoder(tf.keras.Model):
    def __init__(self, embedding_dim):
        super(Encoder, self).__init__()
        self.fc = tf.keras.layers.Dense(embedding_dim, kernel_initializer=tf.kera
                  name="encoder_output_layer")

    def call(self, x):
        x = tf.reshape(x, [x.shape[0], x.shape[1], x.shape[3]])
        encoder_concat = tf.keras.layers.Concatenate()([x[:,0], x[:,1]])
        x = self.fc(encoder_concat)
        x = tf.nn.relu(x)
        return x
```

In [ ]:
```
class Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(Decoder, self).__init__()
        self.units = units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.lstm = tf.keras.layers.LSTM(self.units,
                                         return_sequences=True,
                                         return_state=True,
                                         recurrent_initializer=tf.keras.initializer
        self.dense = tf.keras.layers.Dense(vocab_size, kernel_initializer=tf.kera

    def call(self, x, features):
        #input x = input word teach forcing
        #input features = encoder image features
        x = self.embedding(x)
        x = tf.concat([x, tf.expand_dims(features,1)], axis=-1)
        output, state, _ = self.lstm(x)
        x = self.dense(output)
        return x
```

Encoder Decoder Model

In [ ]:
```
# modelling loss and accuracy for gradient tape based on this https://www.tensorf

optimizer = tf.keras.optimizers.Adam()
loss_obj  = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduc
acc_obj   = tf.keras.metrics.SparseCategoricalAccuracy()

def loss_func(real,pred):
  # getting the mask
  mask = tf.math.logical_not(tf.math.equal(real,0))

  # calculate the loss
  loss_ = loss_obj(real,pred)

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

  # applying the mask to loss
  loss_ = loss_* mask
  loss_ = tf.reduce_mean(loss_)

  return loss_


def acc_func(real,pred):
  acc_f = acc_obj(real,pred)
  return tf.reduce_mean(acc_f)
```

In [ ]:
```
import datetime
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'logs/Basic_gradient_tape/' + current_time +'/train'
validation_log_dir = 'logs/Basic_gradient_tape/'+ current_time +'/test'
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
validattion_summary_writer = tf.summary.create_file_writer(validation_log_dir)
```

In [ ]:
```
encoder = Encoder(embedding_dimension)
decoder = Decoder(embedding_dimension,units,vocab_size)
```

In [ ]:
```
@tf.function
def train_step(tensor, target):
    loss = 0
    accuracy = 0
    dec_input = tf.expand_dims([tokenizer.word_index['<start>']] * target.shape[0
    with tf.GradientTape() as tape:
        features = encoder(tensor)
        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions = decoder(dec_input, features)
            loss += loss_func(target[:, i], predictions)
            accuracy += acc_func(target[:, i], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i],1)
            #print("decoder input teacher", dec_input.shape)
    total_loss = (loss / int(target.shape[1]))
    total_acc = (accuracy / int(target.shape[1]))
    trainable_variables = encoder.trainable_variables + decoder.trainable_variabl

    gradients = tape.gradient(loss, trainable_variables)

    optimizer.apply_gradients(zip(gradients, trainable_variables))

    return loss, total_loss, total_acc

#validation function
@tf.function
def val_step(tensor, target):
    loss_val = 0
    accuracy_val = 0
    dec_input = tf.expand_dims([tokenizer.word_index['<start>']] * target.shape[0
    with tf.GradientTape() as tape:
        features = encoder(tensor)
        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions_val = decoder(dec_input, features)
            loss_val += loss_func(target[:, i], predictions_val)
            accuracy_val += acc_func(target[:, i], predictions_val)
            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i],1)
            #print("decoder input teacher", dec_input)
    total_loss_val = (loss_val / int(target.shape[1]))
    total_acc_val = (accuracy_val / int(target.shape[1]))
    return loss_val, total_loss_val, total_acc_val
```

MODEL Training

In [ ]:
```
tf.keras.backend.clear_session()
EPOCHS = 10
loss_plot_train = []
loss_plot_val = []
for epoch in range(0, EPOCHS):
    print("====== Start Epoch " +str(epoch + 1)+ " ========")

    total_loss_train = 0
    total_acc_train = 0
    total_loss_val = 0
    total_acc_val = 0
    print('Batchwise Train loss')
    for (batch, (jpg_tensor, target)) in enumerate(dataset_train):
        batch_loss, t_loss, t_acc = train_step(jpg_tensor, target)
        total_loss_train += t_loss
        total_acc_train += t_acc

        if batch % 40 == 0:
            print ('Epoch {} Batch {} Loss {:.4f} acc {:.4f}'.format(
              epoch + 1, batch, batch_loss / int(target.shape[1]), t_acc))

    loss_plot_train.append(total_loss_train / int(len(i_train) // BATCH_SIZE))
    with train_summary_writer.as_default():
        tf.summary.scalar('loss', total_loss_train/ int(len(i_train) // BATCH_SIZ
        tf.summary.scalar('accuracy', total_acc_train/ int(len(i_train) // BATCH_

    print('Batchwise validation loss')
    for (batch, (jpg_tensor, target)) in enumerate(dataset_validation):
        batch_loss_val, t_loss_val, t_acc_val = val_step(jpg_tensor, target)
        total_loss_val += t_loss_val
        total_acc_val += t_acc_val
        if batch % 40 == 0:
            print ('Epoch {} Batch {} Loss {:.4f} acc {:.4f}'.format(
              epoch + 1, batch, batch_loss_val / int(target.shape[1]), t_acc_val)

    with validattion_summary_writer.as_default():
        tf.summary.scalar('loss', total_loss_val/int(len(i_validate) // BATCH_SIZ
        tf.summary.scalar('accuracy', total_acc_val/int(len(i_validate) // BATCH_

    template = 'Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test Accuracy: {
    print (template.format(epoch+1,
                           total_loss_train/ int(len(i_train) // BATCH_SIZE),
                           (total_acc_train/ int(len(i_train) // BATCH_SIZE))*10
                           total_loss_val/int(len(i_validate) // BATCH_SIZE),
                           (total_acc_val/int(len(i_validate) // BATCH_SIZE))*10
```

```
====== Start Epoch 1 ========
Batchwise Train loss
Epoch 1 Batch 0 Loss 1.0213 acc 0.0004
Epoch 1 Batch 40 Loss 0.5824 acc 0.7893
Epoch 1 Batch 80 Loss 0.6593 acc 0.7930
Batchwise validation loss
Epoch 1 Batch 0 Loss 0.3677 acc 0.7922
Epoch 1, Loss: 0.6054882407188416, Accuracy: 76.15751647949219, Test Loss: 0.51
```

```
41300559043884, Test Accuracy: 82.40718078613281
====== Start Epoch 2 ========
Batchwise Train loss
Epoch 2 Batch 0 Loss 0.5559 acc 0.7840
Epoch 2 Batch 40 Loss 0.4391 acc 0.7782
Epoch 2 Batch 80 Loss 0.3942 acc 0.7741
Batchwise validation loss
Epoch 2 Batch 0 Loss 0.4417 acc 0.7728
Epoch 2, Loss: 0.4169798791408539, Accuracy: 78.60909271240234, Test Loss: 0.39
965489506721497, Test Accuracy: 80.63554382324219
====== Start Epoch 3 ========
Batchwise Train loss
Epoch 3 Batch 0 Loss 0.3044 acc 0.7703
Epoch 3 Batch 40 Loss 0.2925 acc 0.7681
Epoch 3 Batch 80 Loss 0.2865 acc 0.7662
Batchwise validation loss
Epoch 3 Batch 0 Loss 0.3120 acc 0.7661
Epoch 3, Loss: 0.33620578050613403, Accuracy: 77.6750259399414, Test Loss: 0.35
80491840839386, Test Accuracy: 80.03739929199219
====== Start Epoch 4 ========
Batchwise Train loss
Epoch 4 Batch 0 Loss 0.2925 acc 0.7647
Epoch 4 Batch 40 Loss 0.3061 acc 0.7643
Epoch 4 Batch 80 Loss 0.3507 acc 0.7629
Batchwise validation loss
Epoch 4 Batch 0 Loss 0.2451 acc 0.7629
Epoch 4, Loss: 0.3041955232620239, Accuracy: 77.23392486572266, Test Loss: 0.33
000192046165466, Test Accuracy: 79.68699645996094
====== Start Epoch 5 ========
Batchwise Train loss
Epoch 5 Batch 0 Loss 0.2505 acc 0.7619
Epoch 5 Batch 40 Loss 0.3086 acc 0.7613
Epoch 5 Batch 80 Loss 0.2852 acc 0.7609
Batchwise validation loss
Epoch 5 Batch 0 Loss 0.2808 acc 0.7608
Epoch 5, Loss: 0.2833741307258606, Accuracy: 76.99518585205078, Test Loss: 0.31
85734152793884, Test Accuracy: 79.48360443115234
====== Start Epoch 6 ========
Batchwise Train loss
Epoch 6 Batch 0 Loss 0.3131 acc 0.7600
Epoch 6 Batch 40 Loss 0.3046 acc 0.7601
Epoch 6 Batch 80 Loss 0.1760 acc 0.7595
Batchwise validation loss
Epoch 6 Batch 0 Loss 0.3143 acc 0.7594
Epoch 6, Loss: 0.27113163471221924, Accuracy: 76.8564453125, Test Loss: 0.31030
49397468567, Test Accuracy: 79.36990356445312
====== Start Epoch 7 ========
Batchwise Train loss
Epoch 7 Batch 0 Loss 0.3416 acc 0.7588
Epoch 7 Batch 40 Loss 0.3698 acc 0.7588
Epoch 7 Batch 80 Loss 0.2490 acc 0.7584
Batchwise validation loss
Epoch 7 Batch 0 Loss 0.3573 acc 0.7584
Epoch 7, Loss: 0.26025623083114624, Accuracy: 76.72484588623047, Test Loss: 0.3
112887442111969, Test Accuracy: 79.25816345214844
====== Start Epoch 8 ========
Batchwise Train loss
```

```
Epoch 8 Batch 0 Loss 0.1634 acc 0.7581
Epoch 8 Batch 40 Loss 0.2906 acc 0.7582
Epoch 8 Batch 80 Loss 0.3026 acc 0.7582
Batchwise validation loss
Epoch 8 Batch 0 Loss 0.4076 acc 0.7578
Epoch 8, Loss: 0.25657811760902405, Accuracy: 76.65787506103516, Test Loss: 0.3
164512813091278, Test Accuracy: 79.20201110839844
====== Start Epoch 9 ========
Batchwise Train loss
Epoch 9 Batch 0 Loss 0.2662 acc 0.7575
Epoch 9 Batch 40 Loss 0.2277 acc 0.7575
Epoch 9 Batch 80 Loss 0.1490 acc 0.7573
Batchwise validation loss
Epoch 9 Batch 0 Loss 0.3120 acc 0.7573
Epoch 9, Loss: 0.24627408385276794, Accuracy: 76.60527801513672, Test Loss: 0.3
0513450503349304, Test Accuracy: 79.16907501220703
====== Start Epoch 10 ========
Batchwise Train loss
Epoch 10 Batch 0 Loss 0.2724 acc 0.7570
Epoch 10 Batch 40 Loss 0.2121 acc 0.7570
Epoch 10 Batch 80 Loss 0.1736 acc 0.7567
Batchwise validation loss
Epoch 10 Batch 0 Loss 0.3214 acc 0.7568
Epoch 10, Loss: 0.24162065982818604, Accuracy: 76.54540252685547, Test Loss: 0.
2941944897174835, Test Accuracy: 79.09961700439453
```

In [ ]:
```
%load_ext tensorboard
```

In [ ]:
```
tensorboard --logdir=logs/
```

```
Output hidden; open in https://colab.research.google.com (https://colab.researc
h.google.com) to view.
```

Evaluation

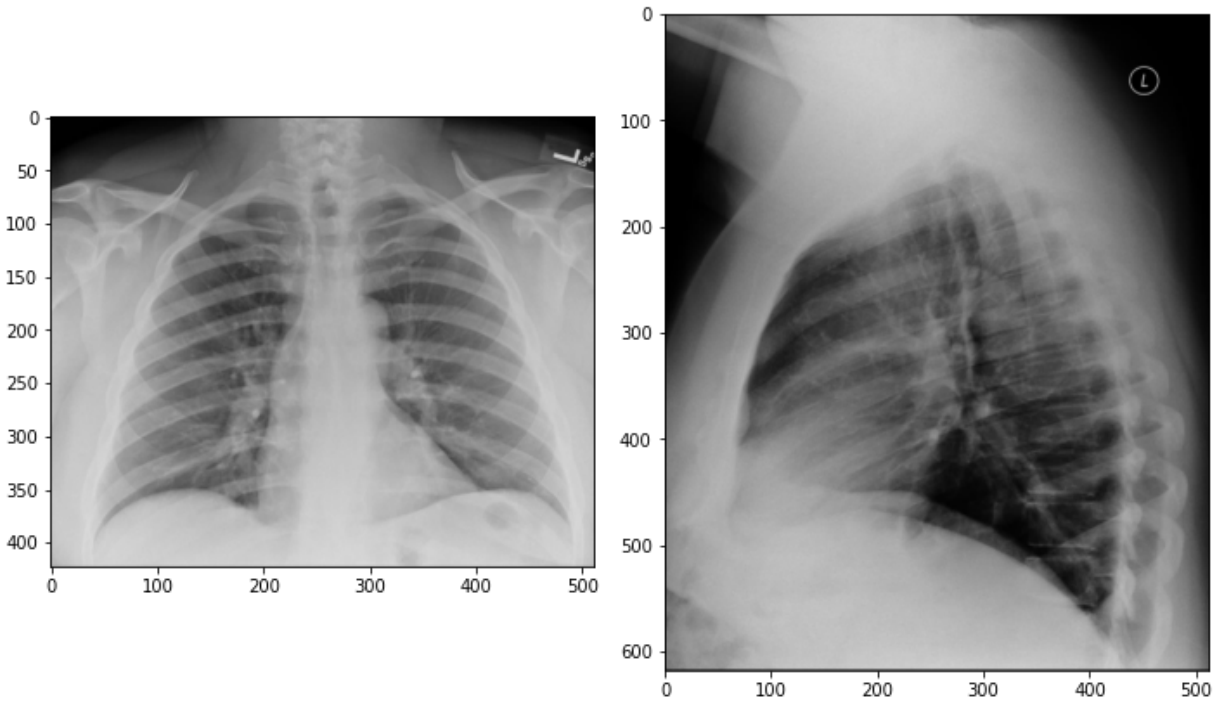
In [ ]:
```
from tensorflow.keras.applications.densenet import preprocess_input
def get_img_tensor(image_path,img_name,model_image):
  img = tf.io.read_file(image_path+str(img_name))
  img = tf.image.decode_jpeg(img,channels=3)
  img = tf.image.resize(img,(224,224))
  img = preprocess_input(img)
  img_features = model_image(tf.constant(img)[None,:])
  return img_features
```

In [ ]:
```
def evaluate(img_name):
  img_tensor = tf.convert_to_tensor([get_img_tensor(img_path,img_name[0], image_f
                                     get_img_tensor(img_path,img_name[1], image_
  img_features = tf.constant(img_tensor)[None,:]
  feature_val  = encoder(img_features)
  dec_input = tf.expand_dims([tokenizer.word_index['<start>']],1)
  result    = []
  text      = ""
  max_length_output = 60
  for i in range(max_length_output):
    predictions = decoder(dec_input,feature_val)# the output shape is 1X1X1225

    predictions = tf.reshape(predictions,[predictions.shape[0],predictions.shape[
    # print(predictions)
    # break

    predicted_id =tf.argmax(tf.math.log(predictions),1)[0].numpy()

    result.append(tokenizer.index_word[predicted_id])
    text += " " + tokenizer.index_word[predicted_id]
    if tokenizer.index_word[predicted_id] == '<end>':
      return result,text

    dec_input = tf.expand_dims([predicted_id],1)
  return result,text
```

In [ ]:
```
import matplotlib.image as mpimg
from nltk.translate.bleu_score import sentence_bleu
def test_img_cap(img_data):
  result,text  = evaluate(img_data)
  fig,axs = plt.subplots(1,len(img_data),figsize=(10,10),tight_layout=True)
  count = 0
  for img,subplot in zip(img_data,axs.flatten()):
    img_ = mpimg.imread(img_path+img)
    imgplot = axs[count].imshow(img_,cmap='bone')
    count +=1
  plt.show()

  print("Predicted:",text)
```

In [ ]:
```
print("Actual",o_test[164])
test_img_cap(i_test[164])
```

```
Actual <start> no acute pulmonary disease <end>
```

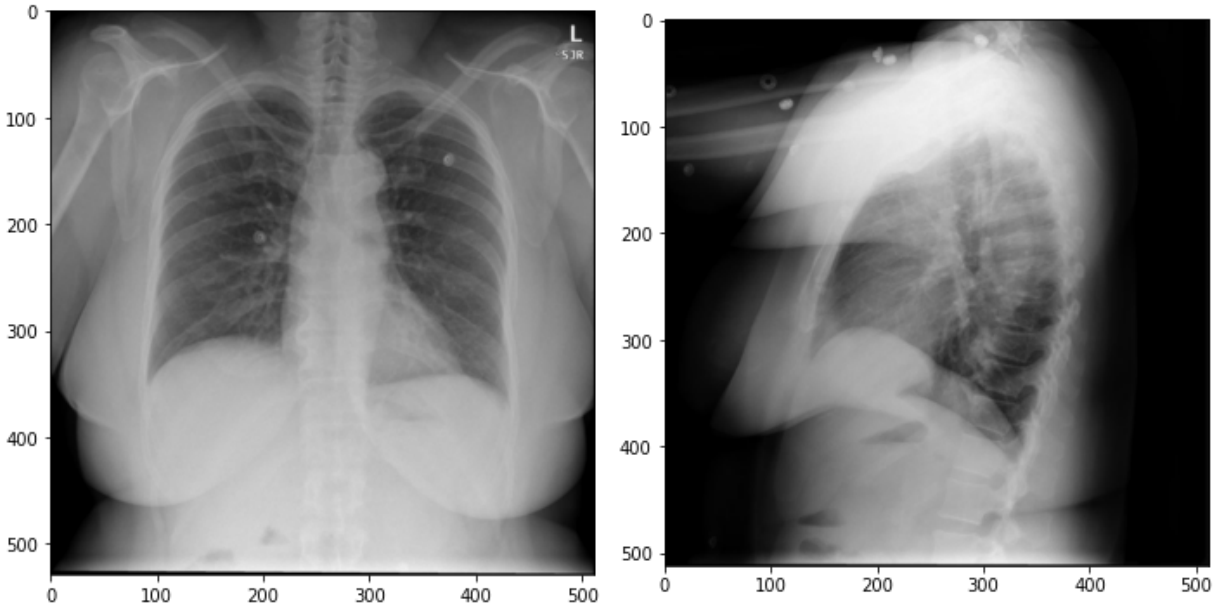



```
Predicted:  no acute cardiopulmonary abnormality <end>
```

In [ ]:
```

```

In [ ]:
```
print("Actual",o_test[64])
test_img_cap(i_test[64])
```

```
Actual <start> no acute cardiopulmonary disease <end>
```

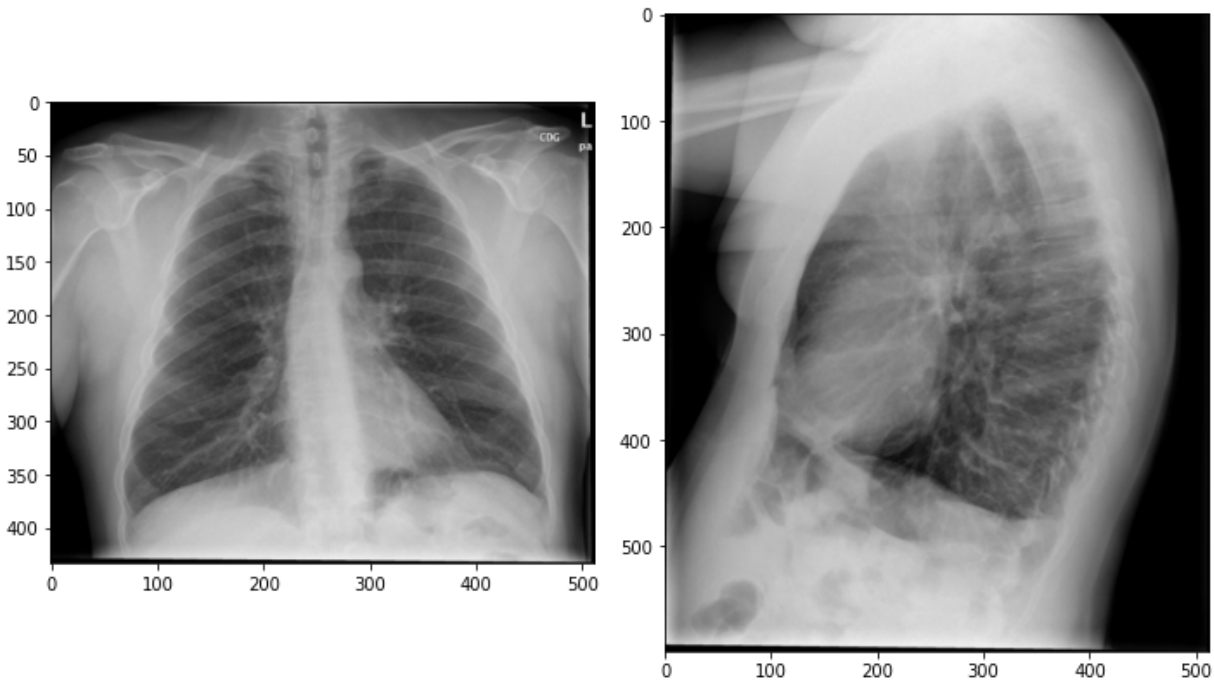



```
Predicted:  no acute cardiopulmonary abnormality <end>
```

In [ ]:

In [ ]:
```
print("Actual",o_test[66])
test_img_cap(i_test[66])
```

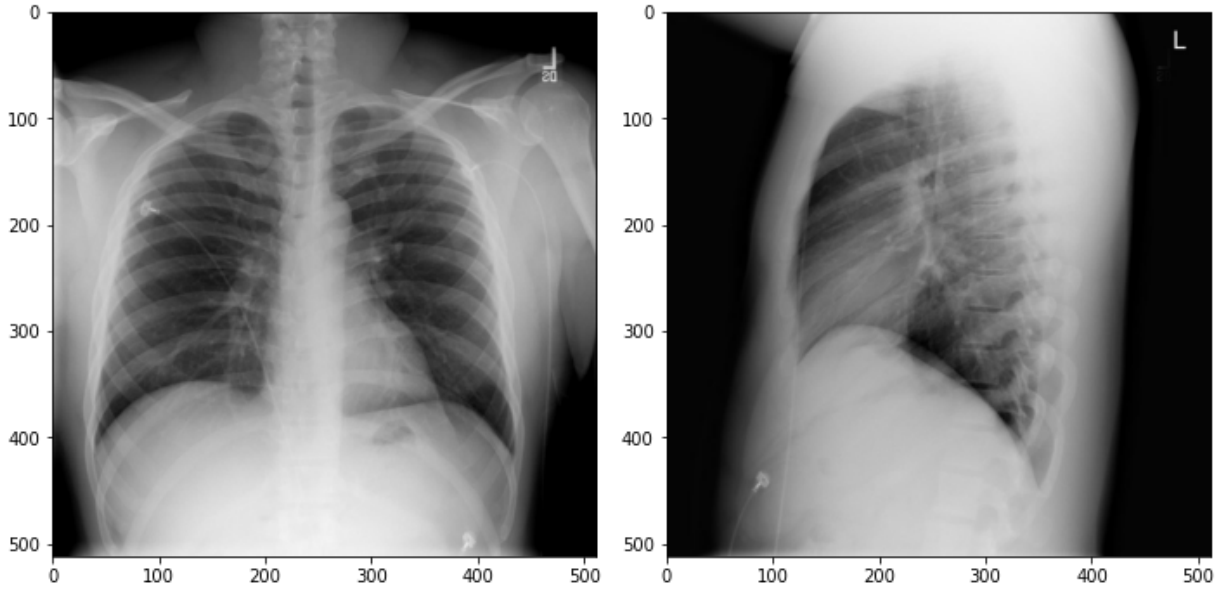
```
Actual <start> heart size is normal and lungs are clear stable mm right midlung
perform granuloma <end>
```




```
Predicted:  no acute cardiopulmonary abnormality <end>
```

In [ ]:
```
print("Actual",o_test[266])
test_img_cap(i_test[266])
```

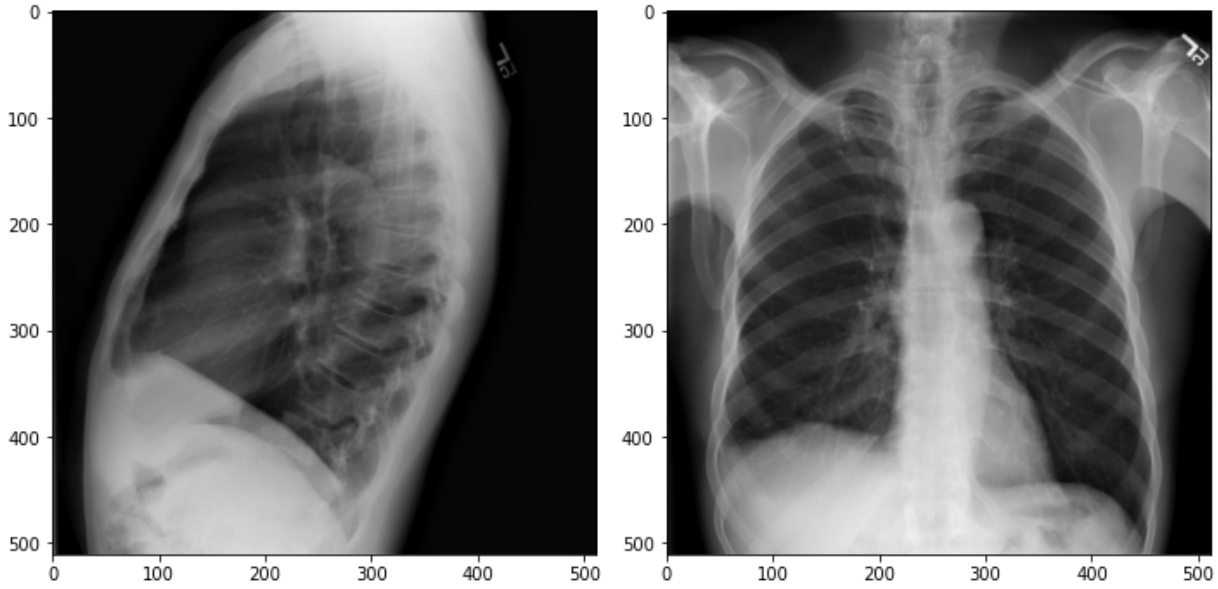
```
Actual <start> no acute active cardiac pulmonary pleural disease <end>
```




```
Predicted:  no acute cardiopulmonary abnormality <end>
```

In [ ]:
```
print("Actual",o_test[29])
test_img_cap(i_test[29])
```

```
Actual <start> no acute cardiopulmonary disease <end>
```




```
Predicted:  no acute cardiopulmonary abnormality <end>
```

In [ ]: