Import the important modules

In [1]:

```
import pandas as pd # for Dataframe
import numpy as np # for array formation.
import datetime # for datetime.
import string
import random as ran # for randomization
import matplotlib.pyplot as plt # for plotting
import matplotlib.image as mpimg
import re # for regular expression
import nltk # for natural language processing
import tensorflow as tf # for tensorflow
import warnings
warnings.filterwarnings('ignore')
from tqdm import tqdm
from sklearn.utils import shuffle #for randomization in data.
from sklearn.model_selection import train_test_split #split the data
from tensorflow.keras.applications.xception import Xception ,preprocess_input # use pre-trained xc
eption model for image feature extraction.
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer# for tokenization of text data.
from tensorflow.keras.preprocessing.sequence import pad_sequences # padding the text tokenize sequ
ence.
```

Let's import the data files

In [2]:

```
data_frame=pd.read_csv('/content/data_frame.csv')# preprocessed data after extraction from csv fil
es.
df_lateral_frontal=pd.read_csv('/content/df.csv')# data having frontal and lateral images.
df_duplicate_images=pd.read_csv('/content/df_dup.csv')# data having copied images because of
having only one image in xml files.
```

In [3]:

```
print('The shape of data_frame is:',data_frame.shape)# shape of dataframe
print('The shape of df_lateral_frontal is:',df_lateral_frontal.shape) # shape of frontal and
lateral image data.
print('The shape of df_duplicate_images is:',df_duplicate_images.shape)# shape of duplicate image
data.
```

```
The shape of data_frame is: (3851, 4)
The shape of df_lateral_frontal is: (3532, 4)
The shape of df_duplicate_images is: (446, 4)
```

Add start and end in the findings and impressions of data.

In [4]:

```
def remodelling_of_data(x): # for understanding where to start and end of sentences in text data.
  return '<start>'+ ' ' + x + ' ' + '<end>'
```

Remodelling the impression and findings having two images

In [5]:

```
df_lateral_frontal['impression']=df_lateral_frontal['impression'].apply(lambda x : remodelling_of_d
ata(x)) # add <start> and <end> in the text of impression of frontal and lateral image.
df_lateral_frontal['findings']=df_lateral_frontal['findings'].apply(lambda x : remodelling_of_data(
x))# add <start> and <end> in the text of findings of frontal and lateral image.
```

In [6]:

```
df_duplicate_images['impression']=df_duplicate_images['impression'].apply(lambda x : remodelling_of_data(x)) # add <start> and <end> in the text of impression of duplicate image.
df_duplicate_images['findings']=df_duplicate_images['findings'].apply(lambda x : remodelling_of_data(x))  # add <start> and <end> in the text of impression of duplicate image.
```

Let's first start the modelling on the basis of impression generation

In [7]:

```
df_lateral_frontal_imp=df_lateral_frontal[['image_1','image_2','impression']] # just to have the impression in our data and remove the findings from frontal and lateral images.
df_duplicate_images_imp=df_duplicate_images[['image_1','image_2','impression']]# just to have the impression in our data and remove the findings from duplicate images.
```

In [8]:

```
df_lateral_frontal_imp.info()# get the information of dataframe for frontal and lateral images
```

```
<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 [9]:

```
df_duplicate_images_imp.info()# get the information of dataframe for duplicate images.
```

```
<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 [10]:

```
path= []# here we get the image name from the dataframe.
for img in tqdm(data_frame['Path'].str.split(',')): # foe each image name in dataframe
  for i in range(len(img)):# for each image
    path.append(img[i])# append the image name in path of the image.
```

```
100%|██████████| 3851/3851 [00:00<00:00, 696278.33it/s]
```

Xception convolutional neural network

In [11]:

```
model_for_image= Xception(include_top=False, weights='imagenet', pooling='avg') # initialize the xception model
input_layer= model_for_image.input # get the input layer from the model
print('Input for the model:',model_for_image.input)
output_layer = model_for_image.layers[-1].output # get the output layer from the model
print('Output for the model:',model_for_image.output)

model_for_image_features = Model(input_layer, output_layer)# here we pass the input layer and output layer from the model which will obtain the features from the image data.
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 4s 0us/step
Input for the model: Tensor("input_1:0", shape=(None, None, None, 3), dtype=float32)
Output for the model: Tensor("global_average_pooling2d/Mean:0", shape=(None, 2048), dtype=float32)
```

Extract the features from image.

In [12]:

```python
from google.colab import drive# import the drive
drive.mount('/content/drive')
```

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

In [ ]:

```python
path_of_image='/content/drive/My Drive/NLMCXR_png/'# from this path we get our images.
```

In [14]:

```python
tensor_img= [] # here we get the tensor of images
for i in tqdm(path): # for each image name.
  i = tf.io.read_file(path_of_image + str(i))# read the file
  i = tf.image.decode_jpeg(i, channels=3)# decode the jpeg file
  i = tf.image.resize(i, (299,299))# resize the image into 299, 299
  i = preprocess_input(i)# preprocess the image data
  features_of_image = model_for_image_features(tf.constant(i)[None,:]) # apply the xception model
  features_of_image = tf.reshape(features_of_image, (-1, features_of_image.shape[1]))# reshape the features of image
  tensor_img.append(features_of_image)# here we append the tensor of image.
```

```
100%|██████████| 7470/7470 [1:09:11<00:00,  1.80it/s]
```

In [15]:

```python
tensor_img[1].numpy()
```

Out[15]:

```
array([[0.22155306, 0.17472953, 0.0134895 , ..., 0.00808325, 0.08561323,
        0.06069888]], dtype=float32)
```

In [16]:

```python
tensor_img[1]
```

Out[16]:

```
<tf.Tensor: shape=(1, 2048), dtype=float32, numpy=
array([[0.22155306, 0.17472953, 0.0134895 , ..., 0.00808325, 0.08561323,
        0.06069888]], dtype=float32)>
```

Split the data for data_frame having lateral and frontal images

In [17]:

```python
# fixing numpy RS
np.random.seed(42)
# fixing tensorflow RS
tf.random.set_seed(32)
# python RS
ran.seed(12)
```

Split the data

In [18]:

```
# first split the data into train and test.
inp_train, input_test, out_train, output_test = 
train_test_split(df_lateral_frontal[['image_1','image_2']].values, df_lateral_frontal['impression'
].values,test_size=0.1, random_state=15)
```

In [19]:

```
# then split the train data into train and validation.
input_train, input_validation, output_train, output_validation = 
train_test_split(inp_train,out_train,test_size=0.2, random_state=15)
```

In [20]:

```
print('shape of input train is:',input_train.shape)# shape of input train
print('shape of output train is:',output_train.shape)# shape of output train
print('shape of input validation is:',input_validation.shape)# shape of input validation
print('shape of output validation is:',output_validation.shape)# shape of output validation
print('='*80)
print('shape of input test is:',input_test.shape)# shape of input test
print('shape of output test is:',output_test.shape)# shape of output test.
```

```
shape of input train is: (2542, 2)
shape of output train is: (2542,)
shape of input validation is: (636, 2)
shape of output validation is: (636,)
================================================================================
shape of input test is: (354, 2)
shape of output test is: (354,)
```

split the data having duplicate images

In [21]:

```
# split the train and test data of duplicate images
inp_train_dup, input_test_dup, out_train_dup, output_test_dup = train_test_split(df_duplicate_image
s_imp[['image_1','image_2']].values, df_duplicate_images_imp['impression'].values,test_size=0.1, r
andom_state=15)
```

In [22]:

```
# then split the train data into train and validation of duplicate images.
input_train_duplicate, input_validation_duplicate, output_train_duplicate, 
output_validation_duplicate = train_test_split(inp_train_dup,out_train_dup,test_size=0.2, 
random_state=15)
```

In [23]:

```
print('shape of input train is:',input_train_duplicate.shape)# shape of input train duplicate
print('shape of output train is:',output_train_duplicate.shape)# shape of output train duplicate
print('shape of input validation is:',input_validation_duplicate.shape) # shape of input validation
duplicate
print('shape of output validation is:',output_validation_duplicate.shape)# shape of output 
validation duplicate
print('='*80)
print('shape of input test is:',input_test_dup.shape)# shape of input test duplicate
print('shape of output test is:',output_test_dup.shape)# shape of output test duplicate
```

```
shape of input train is: (320, 2)
shape of output train is: (320,)
shape of input validation is: (81, 2)
shape of output validation is: (81,)
================================================================================
shape of input test is: (45, 2)
shape of output test is: (45,)
```

Add the duplicate images data with original frontal and lateral images data in equal splitting proportion

In [24]:

```
train_input= np.append(input_train,input_train_duplicate, axis=0)# here we append the input train a
nd input train duplicate
train_output = np.append(output_train,output_train_duplicate)# here we append the output train and
output train duplicate
print('shape of train input data is {} and train output data is {}'.format(train_input.shape,train_
output.shape))
validation_input = np.append(input_validation,input_validation_duplicate,axis=0)# here we append th
e input validation and input validation duplicate
validation_output = np.append(output_validation,output_validation_duplicate, axis=0)# here we
append the output validation and output validation duplicate
print('shape of validation input data is {} and validation output data is
{}'.format(validation_input.shape,validation_output.shape))
test_input= np.append(input_test,input_test_dup,axis=0)# here we append the input test and input
test dupllicate
test_output= np.append(output_test,output_test_dup,axis=0)# here we append the output test and
output test duplicate.
print('shape of test_input data is {} and test_output data is
{}'.format(test_input.shape,test_output.shape))
```

```
shape of train input data is (2862, 2) and train output data is (2862,)
shape of validation input data is (717, 2) and validation output data is (717,)
shape of test_input data is (399, 2) and test_output data is (399,)
```

for removing biasness we do shuffling in data

In [25]:

```
for k in range(3):
  train_input,train_output= shuffle(train_input,train_output,random_state=15)# here we shuffle
train input and train output data
  validation_input,validation_output= shuffle(validation_input,validation_output,random_state=15)#
here we shuffle the validation input and validation output.
  test_input,test_output= shuffle(test_input,test_output,random_state=15)# here we shuffle the
test input and test output.
```

Lets tokenize the test data.

In [26]:

```
maximum_length_output_sentences= 60 # declare the length of output sentences.

token= Tokenizer(oov_token="<unk>", filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}~ ')# Initialize the
tokenizer
token.fit_on_texts(train_output) # fit the text data.
text_of_train_data= token.texts_to_sequences(train_output) #fit the train text data.
text_of_test_data= token.texts_to_sequences(test_output) # fit the test text data.
text_of_validation_data=token.texts_to_sequences(validation_output) # fit the validation text data
.
dictionary = token.word_index # get the word index

word_to_index= {}  # dictionary to get the words to index
index_to_words = {} # dictionary to get the index to words.

for key, value in dictionary.items(): # for getting key and value in dictionary
  word_to_index[key]=value
  index_to_words[value]=key
```

In [27]:

```
size_of_vocabulary= len(word_to_index)+1 # vocabulary is length of word of index
print('The size of the vocabulary is:',size_of_vocabulary)
```

```
The size of the vocabulary is: 1272
```

In [28]:

```
print('Top 6 words and their index are:') # Top 6 words and their index.
```

```
list(dictionary.items())[:6]
```

```
Top 6 words and their index are:
```

Out[28]:

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

Padding the text sequences

In [29]:

```
train_text_output= pad_sequences(text_of_train_data,maxlen=maximum_length_output_sentences,dtype='
int32',padding='post',truncating='post')# padding of the train text data.
validation_text_output=
pad_sequences(text_of_validation_data,maxlen=maximum_length_output_sentences,dtype='int32',padding
='post',truncating='post')# padding of the validation text data.
test_text_output=
pad_sequences(text_of_test_data,maxlen=maximum_length_output_sentences,dtype='int32',padding='post'
,truncating='post') # padding of the test text data.
```

In [30]:

```
print(' The shape of train_text_output after padding is:',train_text_output.shape)
```

```
 The shape of train_text_output after padding is: (2862, 60)
```

Convert multiple image to tensor and corresponding to their impression

In [31]:

```
def multiple_image(img, imp):# This function convert the multiple image to tensor.
  return tf.convert_to_tensor([tensor_img[path.index(img[0].decode('utf-8'))],
tensor_img[path.index(img[1].decode('utf-8'))]]),imp
```

Prepare the train, validation and test data

train_data

In [32]:

```
train_dataset= tf.data.Dataset.from_tensor_slices((train_input,train_text_output)) # make the trai
n dataset.
```

Here we have to use the mapping to load the numpy files which we have to take parallely

In [33]:

```
train_dataset= train_dataset.map(lambda item1, item2: tf.numpy_function(multiple_image,[item1,item2
], [tf.float32, tf.int32]),num_parallel_calls=tf.data.experimental.AUTOTUNE) # map the multiple im
age with train text data.
```

valiation_data

In [34]:

```
validation_dataset= tf.data.Dataset.from_tensor_slices((validation_input,validation_text_output))
# make the validation dataset.
```

Here we have to use the mapping to load the numpy files which we have to take parallely

In [35]:

```
validation_dataset= validation_dataset.map(lambda item1, item2: tf.numpy_function(multiple_image,[i
tem1,item2],[tf.float32, tf.int32]),num_parallel_calls=tf.data.experimental.AUTOTUNE)# map the mul
tiple image with validation text data.
```

Lets print the first data point of train data

In [36]:

```
for img_tensor, impression_tensor in train_dataset:
  print(img_tensor,impression_tensor)
  break# for printing only one data point we imply the break statement here
```

```
tf.Tensor(
[[[0.41433835 0.57313657 0.1823552  ... 0.         0.18592735 0.03933457]]

 [[0.33634582 0.6412105  0.2646824  ... 0.00730557 0.19422911 0.03461767]]], shape=(2, 1, 2048), d
type=float32) tf.Tensor(
[  2   4  26   5   6  22  52 247   8  17   3   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0], shape=(60,), dtype=int32)
```

In [37]:

```
for img_tensor, impression_tensor in validation_dataset:
  print(img_tensor,impression_tensor)
  break# for printing only one data point we imply the break statement here
```

```
tf.Tensor(
[[[7.2461225e-02 3.9873725e-01 6.9258787e-04 ... 0.0000000e+00
   4.9422745e-02 6.9819771e-02]]

 [[0.0000000e+00 5.9988618e-01 4.8373524e-02 ... 1.6550606e-05
   3.3211779e-02 1.0872577e-02]]], shape=(2, 1, 2048), dtype=float32) tf.Tensor(
[  2  21  23  10  39  69  29  12  14 160  12 634  35 152  41   8 253  11
  25  61 124   1   8  11  21  65  47  67  40  11  56  36  48 306 245  13
  25  19   4  43   4  42   1 207 112   3   0   0   0   0   0   0   0   0
   0   0   0   0   0   0], shape=(60,), dtype=int32)
```

Let's declare the batchsize, buffersize, embedding_dim, units

In [38]:

```
SIZE_OF_BATCH=32  #Batch size
SIZE_OF_BUFFER= 500  #Batch Buffer
DIMENSION_OF_EMBEDDING= 256  #Embedding
UNITS= 512  #units
```

Shuffle and set data according to batchsize

Train dataset

In [39]:

```
train_dataset=train_dataset.shuffle(SIZE_OF_BUFFER).batch(SIZE_OF_BATCH) #train dataset with
shuffling for removing biasness from data.
train_dataset = train_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

validation dataset

In [40]:

```
validation_dataset=validation_dataset.shuffle(SIZE_OF_BUFFER).batch(SIZE_OF_BATCH)# validation
```

```
dataset with shuffling for removing biasness from data.
validation_dataset = validation_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

In [41]:

```
input_layer[:,0] #check the input layer.
```

Out[41]:

```
<tf.Tensor 'strided_slice:0' shape=(None, None, 3) dtype=float32>
```

Lets make the Encoder and Decoder model

Let's make the class of encoder

In [43]:

```
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, Reshape, Concatenate
class encoder(tf.keras.Model):
  def __init__(self,DIMENSION_OF_EMBEDDING):
    super(encoder,self).__init__()
    self.con_1= tf.keras.layers.Conv2D(32, 3, strides=2, activation='relu', padding='same')
    self.con_2= tf.keras.layers.Conv2D(64, 3, strides=2, activation='relu',
padding='same')#convulational layer.
    self.drop_o= tf.keras.layers.Dropout(0.25)
    self.flat=tf.keras.layers.Flatten()
    self.fc = tf.keras.layers.Dense(DIMENSION_OF_EMBEDDING, kernel_initializer=tf.keras.initializer
s.glorot_uniform(seed=45),name='output_layer_of_encoder')# dense layer.

  def call(self, a):
    # this are not for images . it is a one day array which have the shape of input data.
    a = tf.reshape(a, [a.shape[0], a.shape[1],a.shape[2],a.shape[3]])# reshape the data.
    concatination_enc= Concatenate()([a[:,0], a[:,1]])# concatenate.
    a= self.con_1(tf.expand_dims( concatination_enc,1))
    a= self.con_2(a)
    a= self.drop_o(a)
    a=self.flat(a)
    a = self.fc(a)
    a = tf.nn.relu(a)
    return a
```

decoder class

In [44]:

```
class decoder(tf.keras.Model):
  def __init__(self, DIMENSION_OF_EMBEDDING, UNITS, size_of_vocabulary):
    super(decoder,self).__init__()
    self.units = UNITS # units
    self.embedding = tf.keras.layers.Embedding(size_of_vocabulary,DIMENSION_OF_EMBEDDING)#
embedding layer
    self.lstm = tf.keras.layers.LSTM(self.units, return_sequences=True, return_state=True,recurrent
_initializer=tf.keras.initializers.glorot_uniform(seed=45))# LSTM layer
    self.dense = tf.keras.layers.Dense(size_of_vocabulary,kernel_initializer= tf.keras.initializers
.glorot_uniform(seed=45)) # Dense layer

  def call(self, a, features):
    a = self.embedding(a)

    a = tf.concat([a, tf.expand_dims(features,1)], axis= -1)# concatenate
    output, state, _ = self.lstm(a) #lstm

    a = self.dense(output)# dense layer

    return a
```

Loss function

In [45]:

```
OPTIMIZER= tf.keras.optimizers.Adam() #Adam optimizer
object_loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction ='none')# l
oss
object_accuracy = tf.keras.metrics.SparseCategoricalAccuracy()# Accuracy

def function_loss(Actual, predicted): # function for loss calcuation
  function_l= object_loss(Actual, predicted)
  return tf.reduce_mean(function_l)
```

Accuracy Function

In [47]:

```
def Function_Accuracy(Actual, predicted): # function for accuracy calculation.
  function_A= object_accuracy(Actual, predicted)
  return tf.reduce_mean(function_A)
```

Directory and log time

In [48]:

```
recent_time= datetime.datetime.now().strftime("%Y%m%d-%H%M%S") # datetime log directory
directory_log_train= 'content/gradient_tape/'+ recent_time + '/train' # datetime train directory
directory_log_validation= 'content/gradient_tape/'+ recent_time+ '/test' # datetime test directory

train_summary= tf.summary.create_file_writer(directory_log_train)
validation_summary= tf.summary.create_file_writer(directory_log_validation)
```

In [49]:

```
Encoder= encoder(DIMENSION_OF_EMBEDDING) # encoder
Decoder= decoder(DIMENSION_OF_EMBEDDING, UNITS, size_of_vocabulary)# decoder
```

In [50]:

```
!rm -r logs/# clear the previous log directory
```

```
rm: cannot remove 'logs/#': No such file or directory
rm: cannot remove 'clear': No such file or directory
rm: cannot remove 'the': No such file or directory
rm: cannot remove 'previous': No such file or directory
rm: cannot remove 'log': No such file or directory
rm: cannot remove 'directory': No such file or directory
```

training function

In [51]:

```
@tf.function
def training(tensor, target):
  loss= 0 # Initialize the loss.
  acc= 0  # Initialize the accuracy.
  input_dec = tf.expand_dims([token.word_index['<start>']]* target.shape[0], 1)
  with tf.GradientTape() as tape:
    feat= Encoder(tensor) # features
    # teacher forcing is already present in them.
    for i in range(1, target.shape[1]):
      pred= Decoder(input_dec, feat) # predictions
      loss+= function_loss(target[:, i], pred) #loss
      acc+= Function_Accuracy(target[:,i],pred) #accuracy
      # here we use teacher forcing.
      input_dec= tf.expand_dims(target[:, i],1) #input decoder

  loss_tot = (loss / int(target.shape[1])) #total of loss
  acc_tot = (acc / int(target.shape[1])) #total of accuracy
  trainable_variables= Encoder.trainable_variables + Decoder.trainable_variables
  gradients = tape.gradient(loss, trainable_variables) #gradients
  OPTIMIZER.apply_gradients(zip(gradients, trainable_variables)) #optimizer
```

```
  return loss, loss_tot, acc_tot, gradients    #loss , #total loss, # total accuracy
```

validation_function

In [52]:

```
@tf.function
def validation(tensor, target):
  validation_loss= 0 # Intialize the loss.
  validation_acc= 0 # Initialize the accuracy.
  input_dec = tf.expand_dims([token.word_index['<start>']] * target.shape[0], 1)
  with tf.GradientTape() as tape:
    feat= Encoder(tensor) #features
    for i in range(1, target.shape[1]):
      validation_pred= Decoder(input_dec, feat) # validation prediction
      validation_loss+=  function_loss(target[:, i], validation_pred) # validation loss
      validation_acc+= Function_Accuracy(target[:,i], validation_pred) # validation accuracy

      input_dec = tf.expand_dims(target[:,i], 1) #input decoder


  validation_loss_tot = (validation_loss / int(target.shape[1])) # validation loss total
  validation_acc_tot = (validation_acc / int(target.shape[1]))  # validation accuracy total
  trainable_variables= Encoder.trainable_variables + Decoder.trainable_variables

  gradients_validation = tape.gradient(validation_loss, trainable_variables) #gradients

  return validation_loss, validation_loss_tot, validation_acc_tot, gradients_validation
```

Let's do the training of encoder decoder model

In [54]:

```
tf.keras.backend.clear_session()
epochs_used= 10 # number of epochs
train_loss_p=[] # train loss plot
validation_loss_p= [] # validation loss plot
trainable_variables= Encoder.trainable_variables + Decoder.trainable_variables
for ep in range(0,epochs_used): # for each epoch
  print("Intialize the epoch"+str(ep+1)) # for representing the epoch number
  train_tot_loss= 0 # train total loss
  train_tot_acc= 0 # train total accuracy
  validation_tot_loss= 0 # validation total loss
  validation_tot_acc= 0 # validation total accuracy
  print("The training loss batchwise")
  for (batch,(tensor_jpeg, target))in enumerate(train_dataset): # getting each batch, tensor_jpeg and target.
    batch_loss, tot_loss, tot_acc, gradients = training(tensor_jpeg, target) # do the training on the tensor_jpeg, target
    train_tot_loss += tot_loss # total loss
    train_tot_acc += tot_acc # total accuracy

    if batch % 40 == 0:
      print('The Epoch is {} batch is{} loss is {:.4f} and the accuracy is {:.4f}'.format(ep+1, batch, batch_loss / int(target.shape[1]), tot_acc))
  train_loss_p.append(train_tot_loss/ int(len(train_input)//SIZE_OF_BATCH))# append the train total loss plot

  with train_summary.as_default():
    tf.summary.scalar('loss', train_tot_loss/int(len(train_input)// SIZE_OF_BATCH),step=ep) #summary of train loss
    tf.summary.scalar('accuracy', train_tot_acc/int(len(train_input)// SIZE_OF_BATCH), step=ep) #summary of train accuracy.
  with train_summary.as_default():
    for i in range(len(Encoder.trainable_variables)):
      name_temp = Encoder.trainable_variables[i].name
      tf.summary.histogram(name_temp, gradients[i],step=ep)    # here we make the histograms for gradients.
    for i in range(len(Decoder.trainable_variables)):
      name_temp_d = Decoder.trainable_variables[i].name
      tf.summary.histogram(name_temp_d, gradients[i],step=ep)

  print('The validation loss batchwise')
```

```
  for (batch,(tensor_jpeg, target)) in enumerate(validation_dataset): # for each batch,
tensor_jpeg and target.
    validation_batch_loss, validation_t_loss, validation_t_acc, gradients_validation = validation(t
ensor_jpeg, target) # validation of tensor jpeg and target.
    validation_tot_loss += validation_t_loss # validation total loss
    validation_tot_acc += validation_t_acc # validation total accuracy

    if batch % 40 == 0:
      print('The Epoch is {} batch is{} loss is {:.4f} and the accuracy is {:.4f}'.format(ep+1, bat
ch, batch_loss / int(target.shape[1]), validation_t_acc))
  validation_loss_p.append(validation_tot_loss / int(len(validation_input)//SIZE_OF_BATCH)) #append
the validation loss plot.

  with validation_summary.as_default():
      tf.summary.scalar('loss', validation_tot_loss/int(len(validation_input)// SIZE_OF_BATCH),step
=ep)# summary of validation loss.
      tf.summary.scalar('accuracy', validation_tot_acc/int(len(validation_input)// SIZE_OF_BATCH),
step=ep) # summary of validation accuray.

  print(' The epoch is {}, loss is {:.4f}, Accuracy is: {:.4f}, test loss: {:.4f} and test
accuracy: {:.4f}'.format(ep+1, train_tot_loss/ int(len(train_input)// SIZE_OF_BATCH),(train_tot_acc
/ int(len(train_input)//SIZE_OF_BATCH))*100, validation_tot_loss/ int(len(validation_input)// SIZE_
OF_BATCH),(validation_tot_acc/ int(len(validation_input)//SIZE_OF_BATCH))*100))
```

```
Intialize the epoch1
The training loss batchwise
[<tf.Tensor 'gradient_tape/encoder/conv2d_4/Conv2D/Conv2DBackpropFilter:0' shape=(3, 3, 4096, 32)
dtype=float32>, <tf.Tensor 'gradient_tape/encoder/conv2d_4/BiasAdd/BiasAddGrad:0' shape=(32,)
dtype=float32>, <tf.Tensor 'gradient_tape/encoder/conv2d_5/Conv2D/Conv2DBackpropFilter:0' shape=(3
, 3, 32, 64) dtype=float32>, <tf.Tensor 'gradient_tape/encoder/conv2d_5/BiasAdd/BiasAddGrad:0'
shape=(64,) dtype=float32>, <tf.Tensor 'gradient_tape/encoder/output_layer_of_encoder/MatMul_1:0'
shape=(64, 256) dtype=float32>, <tf.Tensor
'gradient_tape/encoder/output_layer_of_encoder/BiasAdd/BiasAddGrad:0' shape=(256,) dtype=float32>,
<tensorflow.python.framework.indexed_slices.IndexedSlices object at 0x7f2df01a9518>, <tf.Tensor 'A
ddN_4:0' shape=(512, 2048) dtype=float32>, <tf.Tensor 'AddN_5:0' shape=(512, 2048) dtype=float32>,
<tf.Tensor 'AddN_6:0' shape=(2048,) dtype=float32>, <tf.Tensor 'AddN_7:0' shape=(512, 1272)
dtype=float32>, <tf.Tensor 'AddN_8:0' shape=(1272,) dtype=float32>]
[<tf.Tensor 'gradient_tape/encoder/conv2d_4/Conv2D/Conv2DBackpropFilter:0' shape=(3, 3, 4096, 32)
dtype=float32>, <tf.Tensor 'gradient_tape/encoder/conv2d_4/BiasAdd/BiasAddGrad:0' shape=(32,)
dtype=float32>, <tf.Tensor 'gradient_tape/encoder/conv2d_5/Conv2D/Conv2DBackpropFilter:0' shape=(3
, 3, 32, 64) dtype=float32>, <tf.Tensor 'gradient_tape/encoder/conv2d_5/BiasAdd/BiasAddGrad:0'
shape=(64,) dtype=float32>, <tf.Tensor 'gradient_tape/encoder/output_layer_of_encoder/MatMul_1:0'
shape=(64, 256) dtype=float32>, <tf.Tensor
'gradient_tape/encoder/output_layer_of_encoder/BiasAdd/BiasAddGrad:0' shape=(256,) dtype=float32>,
<tensorflow.python.framework.indexed_slices.IndexedSlices object at 0x7f2d47e3bf98>, <tf.Tensor 'A
ddN_4:0' shape=(512, 2048) dtype=float32>, <tf.Tensor 'AddN_5:0' shape=(512, 2048) dtype=float32>,
<tf.Tensor 'AddN_6:0' shape=(2048,) dtype=float32>, <tf.Tensor 'AddN_7:0' shape=(512, 1272)
dtype=float32>, <tf.Tensor 'AddN_8:0' shape=(1272,) dtype=float32>]
The Epoch is 1 batch is0 loss is 7.0288 and the accuracy is 0.0062
The Epoch is 1 batch is40 loss is 1.1065 and the accuracy is 0.7986
The Epoch is 1 batch is80 loss is 0.4548 and the accuracy is 0.8081
[<tf.Tensor 'gradient_tape/encoder/conv2d_4/Conv2D/Conv2DBackpropFilter:0' shape=(3, 3, 4096, 32)
dtype=float32>, <tf.Tensor 'gradient_tape/encoder/conv2d_4/BiasAdd/BiasAddGrad:0' shape=(32,)
dtype=float32>, <tf.Tensor 'gradient_tape/encoder/conv2d_5/Conv2D/Conv2DBackpropFilter:0' shape=(3
, 3, 32, 64) dtype=float32>, <tf.Tensor 'gradient_tape/encoder/conv2d_5/BiasAdd/BiasAddGrad:0'
shape=(64,) dtype=float32>, <tf.Tensor 'gradient_tape/encoder/output_layer_of_encoder/MatMul_1:0'
shape=(64, 256) dtype=float32>, <tf.Tensor
'gradient_tape/encoder/output_layer_of_encoder/BiasAdd/BiasAddGrad:0' shape=(256,) dtype=float32>,
<tensorflow.python.framework.indexed_slices.IndexedSlices object at 0x7f2df80ef4e0>, <tf.Tensor 'A
ddN_4:0' shape=(512, 2048) dtype=float32>, <tf.Tensor 'AddN_5:0' shape=(512, 2048) dtype=float32>,
<tf.Tensor 'AddN_6:0' shape=(2048,) dtype=float32>, <tf.Tensor 'AddN_7:0' shape=(512, 1272)
dtype=float32>, <tf.Tensor 'AddN_8:0' shape=(1272,) dtype=float32>]
The validation loss batchwise
The Epoch is 1 batch is0 loss is 0.9814 and the accuracy is 0.8092
 The epoch is 1, loss is 1.7170, Accuracy is: 77.6071, test loss: 0.9360 and test accuracy: 84.624
5
Intialize the epoch2
The training loss batchwise
The Epoch is 2 batch is0 loss is 0.9704 and the accuracy is 0.8107
The Epoch is 2 batch is40 loss is 0.9961 and the accuracy is 0.8068
The Epoch is 2 batch is80 loss is 0.8688 and the accuracy is 0.7985
The validation loss batchwise
The Epoch is 2 batch is0 loss is 1.0573 and the accuracy is 0.7966
 The epoch is 2, loss is 0.8520, Accuracy is: 81.4306, test loss: 0.8742 and test accuracy: 83.089
8
Intialize the epoch3
The training loss batchwise
```

```
The training loss batchwise
The Epoch is 3 batch is0 loss is 0.9809 and the accuracy is 0.7933
The Epoch is 3 batch is40 loss is 0.7115 and the accuracy is 0.7901
The Epoch is 3 batch is80 loss is 0.7518 and the accuracy is 0.7882
The validation loss batchwise
The Epoch is 3 batch is0 loss is 1.1707 and the accuracy is 0.7877
 The epoch is 3, loss is 0.8028, Accuracy is: 79.9099, test loss: 0.8026 and test accuracy: 82.306
6
Intialize the epoch4
The training loss batchwise
The Epoch is 4 batch is0 loss is 0.5399 and the accuracy is 0.7870
The Epoch is 4 batch is40 loss is 0.6531 and the accuracy is 0.7857
The Epoch is 4 batch is80 loss is 0.4637 and the accuracy is 0.7845
The validation loss batchwise
The Epoch is 4 batch is0 loss is 0.4852 and the accuracy is 0.7846
 The epoch is 4, loss is 0.7138, Accuracy is: 79.4466, test loss: 0.7247 and test accuracy: 82.010
6
Intialize the epoch5
The training loss batchwise
The Epoch is 5 batch is0 loss is 0.7377 and the accuracy is 0.7842
The Epoch is 5 batch is40 loss is 0.5390 and the accuracy is 0.7839
The Epoch is 5 batch is80 loss is 0.6735 and the accuracy is 0.7829
The validation loss batchwise
The Epoch is 5 batch is0 loss is 0.4484 and the accuracy is 0.7829
 The epoch is 5, loss is 0.6569, Accuracy is: 79.2310, test loss: 0.6776 and test accuracy: 81.838
3
Intialize the epoch6
The training loss batchwise
The Epoch is 6 batch is0 loss is 0.6834 and the accuracy is 0.7825
The Epoch is 6 batch is40 loss is 0.5509 and the accuracy is 0.7825
The Epoch is 6 batch is80 loss is 0.6512 and the accuracy is 0.7819
The validation loss batchwise
The Epoch is 6 batch is0 loss is 0.4966 and the accuracy is 0.7816
 The epoch is 6, loss is 0.6218, Accuracy is: 79.1026, test loss: 0.6619 and test accuracy: 81.716
2
Intialize the epoch7
The training loss batchwise
The Epoch is 7 batch is0 loss is 0.5038 and the accuracy is 0.7815
The Epoch is 7 batch is40 loss is 0.4240 and the accuracy is 0.7815
The Epoch is 7 batch is80 loss is 0.5980 and the accuracy is 0.7809
The validation loss batchwise
The Epoch is 7 batch is0 loss is 0.9554 and the accuracy is 0.7807
 The epoch is 7, loss is 0.5972, Accuracy is: 79.0008, test loss: 0.6337 and test accuracy: 81.613
2
Intialize the epoch8
The training loss batchwise
The Epoch is 8 batch is0 loss is 0.5318 and the accuracy is 0.7806
The Epoch is 8 batch is40 loss is 0.8152 and the accuracy is 0.7803
The Epoch is 8 batch is80 loss is 0.3146 and the accuracy is 0.7802
The validation loss batchwise
The Epoch is 8 batch is0 loss is 0.6692 and the accuracy is 0.7801
 The epoch is 8, loss is 0.5742, Accuracy is: 78.9161, test loss: 0.6166 and test accuracy: 81.543
9
Intialize the epoch9
The training loss batchwise
The Epoch is 9 batch is0 loss is 0.6423 and the accuracy is 0.7799
The Epoch is 9 batch is40 loss is 0.4253 and the accuracy is 0.7798
The Epoch is 9 batch is80 loss is 0.4039 and the accuracy is 0.7796
The validation loss batchwise
The Epoch is 9 batch is0 loss is 0.4609 and the accuracy is 0.7795
 The epoch is 9, loss is 0.5553, Accuracy is: 78.8550, test loss: 0.6124 and test accuracy: 81.488
4
Intialize the epoch10
The training loss batchwise
The Epoch is 10 batch is0 loss is 0.5056 and the accuracy is 0.7794
The Epoch is 10 batch is40 loss is 0.5865 and the accuracy is 0.7795
The Epoch is 10 batch is80 loss is 0.4503 and the accuracy is 0.7791
The validation loss batchwise
The Epoch is 10 batch is0 loss is 0.4516 and the accuracy is 0.7790
 The epoch is 10, loss is 0.5407, Accuracy is: 78.8121, test loss: 0.5992 and test accuracy: 81.43
03
```

In [55]:

```
%reload_ext tensorboard
```

In [56]:

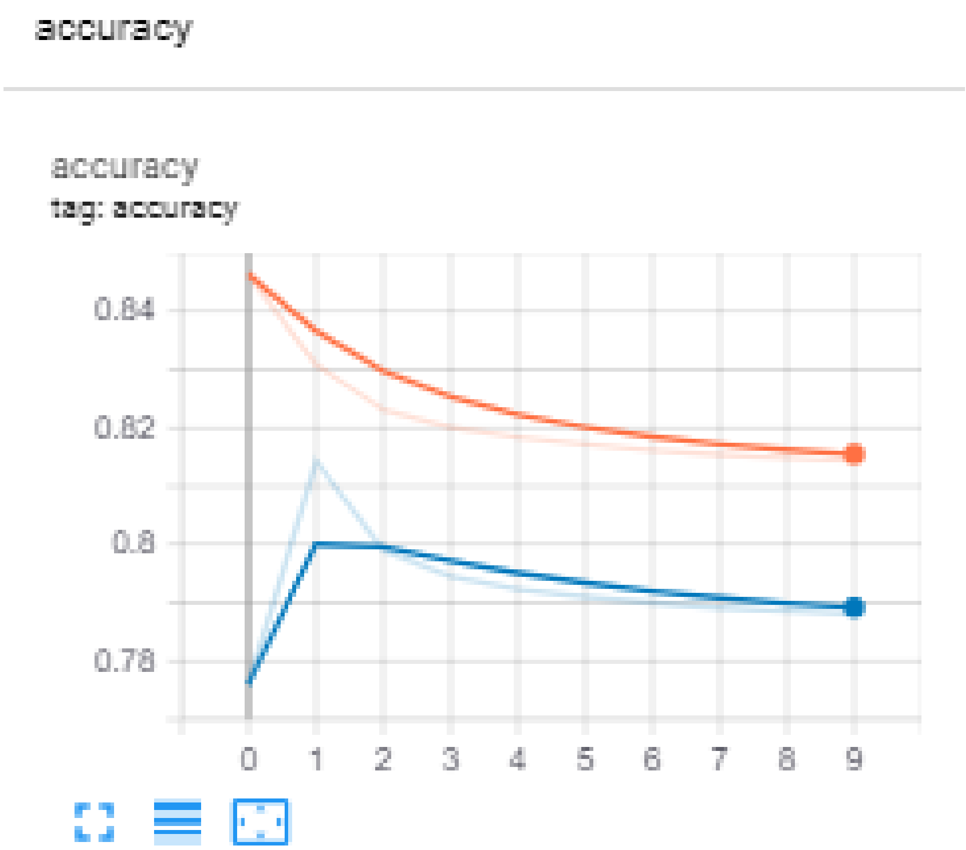
```
tensorboard --logdir=/content/
```

Accuracy plot

In [57]:

```
from IPython.display import Image
Image("/content/Screenshot (101).png", width=600)
```

Out[57]:

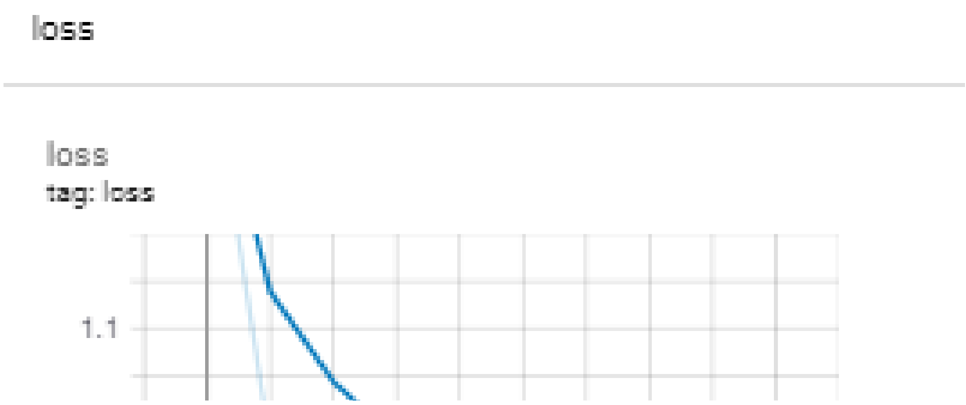


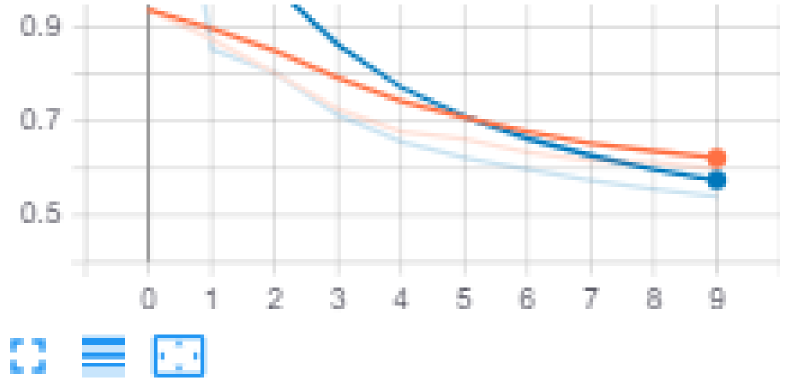
Loss plot

In [58]:

```
from IPython.display import Image
Image("/content/Screenshot (102).png", width=600)
```

Out[58]:



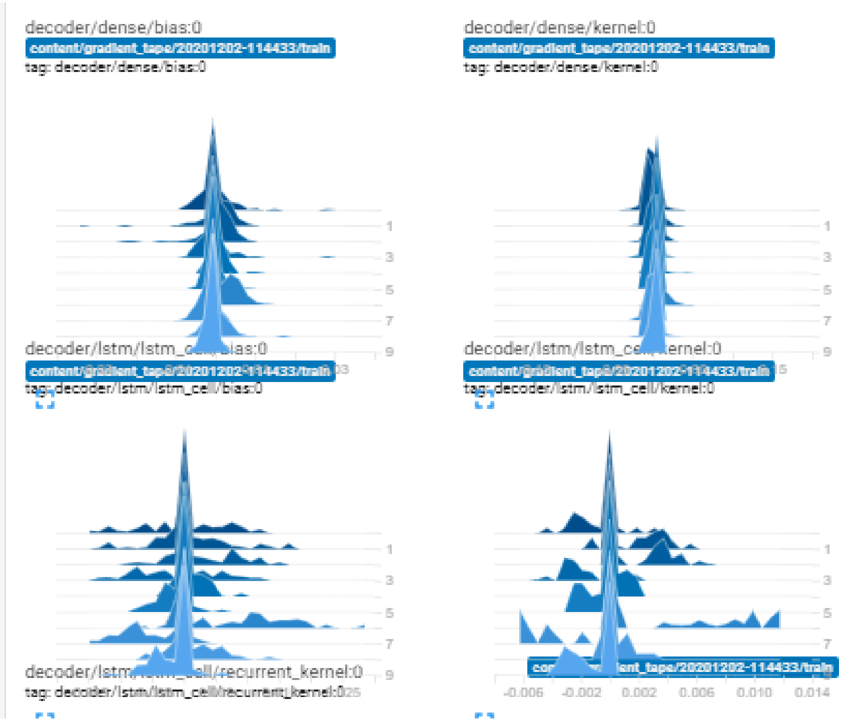


Histogram plots of gradients

In [60]:

```
from IPython.display import Image
Image("/content/Screenshot (103).png", width=1200) # for decoder layers
```

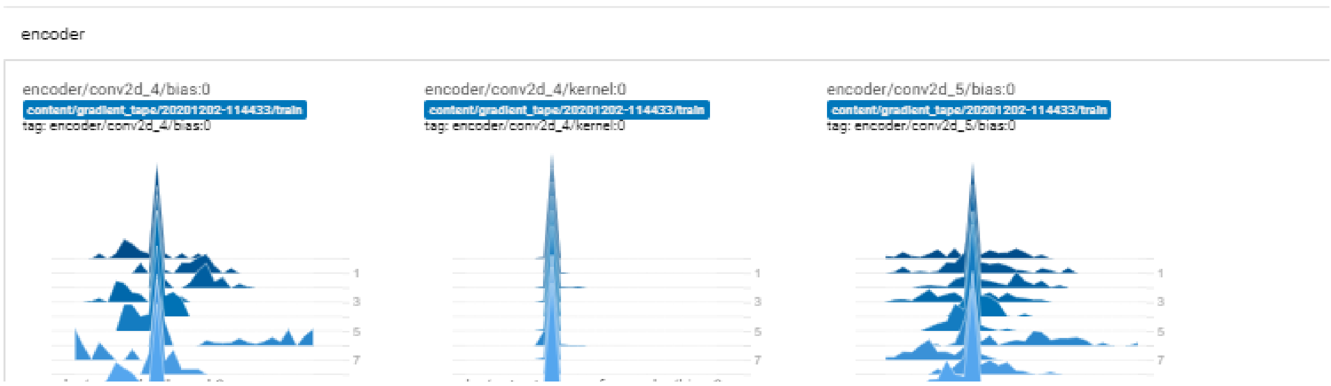
Out[60]:

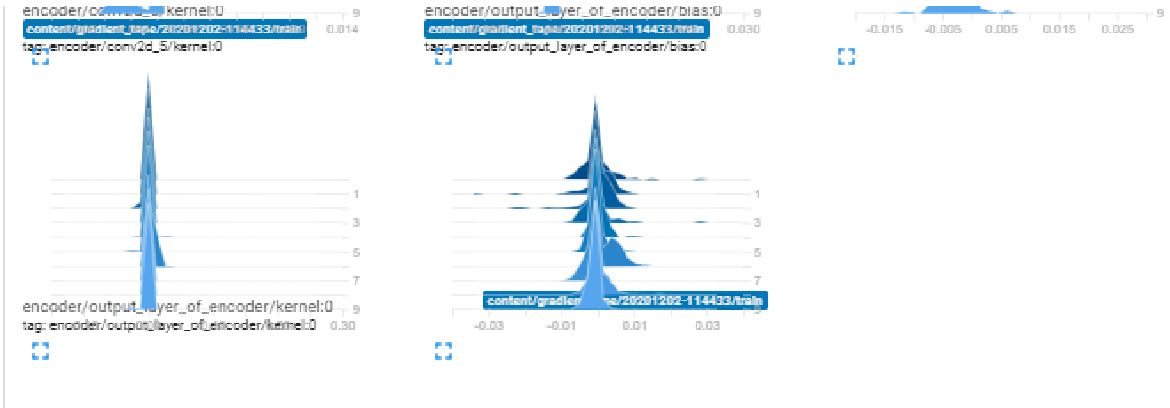


In [61]:

```
from IPython.display import Image
Image("/content/Screenshot (104).png", width=1200) # for encoder layers
```

Out[61]:





Conclusion:

1. Here I observe that results are satisfactory but not that much good
2. Here validation accuracy is not improving but there is a positive sign that loss is converging in this.

Let's evaluate the model

In [62]:

```
def tensor_of_image(path_of_image, name_of_image, model):
  i = tf.io.read_file(path_of_image + str(name_of_image)) # read file
  i = tf.image.decode_jpeg(i, channels=3) # decode the jpeg
  i = tf.image.resize(i, (299,299)) # resize the image
  i = tf.keras.applications.xception.preprocess_input(i) # extract the features with the help of xc
eption model.
  features_of_the_image = model(tf.constant(i)[None, :]) # features of image.
  return features_of_the_image
```

In [63]:

```
# This function is used for the evaluation of model.
def evaluate_model(name_of_image):
  tens_image = tf.convert_to_tensor([tensor_of_image("/content/drive/My Drive/NLMCXR_png/",path[0]
,model_for_image_features), tensor_of_image("/content/drive/My Drive/NLMCXR_png/", path[1],
model_for_image_features)])
  feat_of_image = tf.constant(tens_image)[None, :]
  value_of_feat = Encoder(feat_of_image)
  input_of_decoder = tf.expand_dims([token.word_index['<start>']],1)
  res= []
  txt = ""
  for i in range(maximum_length_output_sentences):
    pred = Decoder(input_of_decoder, value_of_feat) # prediction
    pred = tf.reshape(pred, [pred.shape[0], pred.shape[2]]) # reshape the prediction
    id_of_pred = tf.argmax(pred, axis=1)[0].numpy() # id of prediction
    res.append(token.index_word[id_of_pred]) #result
    txt += " " + token.index_word[id_of_pred] # text
    if token.index_word[id_of_pred] == '<end>':
      return res, txt

    input_of_decoder = tf.expand_dims([id_of_pred],1) #input of decoder
  return res, txt    # return result and text
```

Let's test the image caption

In [64]:

```
# here we aggregate all the functions
def caption_of_image(data_of_image):
  res, txt = evaluate_model(data_of_image) # here we evaluate the model
  fig, axs = plt.subplots(1, len(data_of_image), figsize= (10, 10), tight_layout= True) # subplots
  cnt =0 # Initialize the count
  for i, sp in zip (data_of_image, axs.flatten()):
    i_ =mpimg.imread(path_of_image+i) # read the image
    i_plot = axs[cnt].imshow(i_, cmap= 'bone')
    cnt +=1
  plt.show()
```

```
plt.show()
print('The predicted text is: ', txt)
```

In [65]:

```
print('The Actual text is:',test_output[160]) # Actual text
caption_of_image(test_input[160]) # Predicted image and text
```

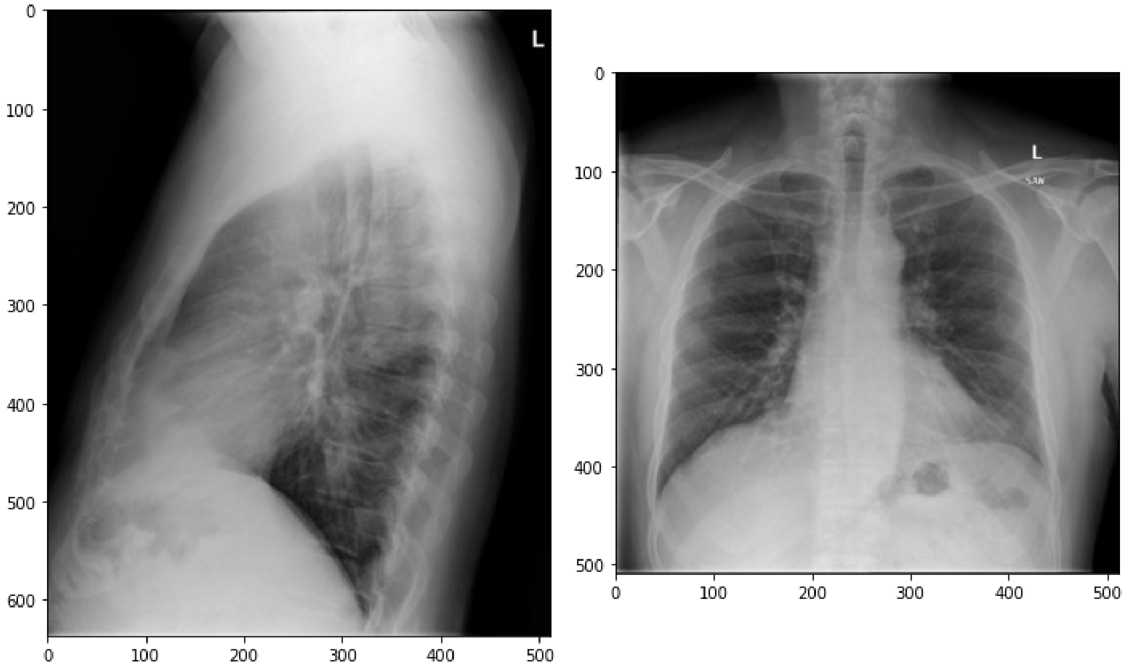
```
The Actual text is: <start> no evidence active disease <end>
```


```
The predicted text is:   no acute cardiopulmonary abnormality <end>
```

In [66]:

```
print('The Actual text is:',test_output[10])
caption_of_image(test_input[10])
```

```
The Actual text is: <start> no acute cardiopulmonary abnormality <end>
```

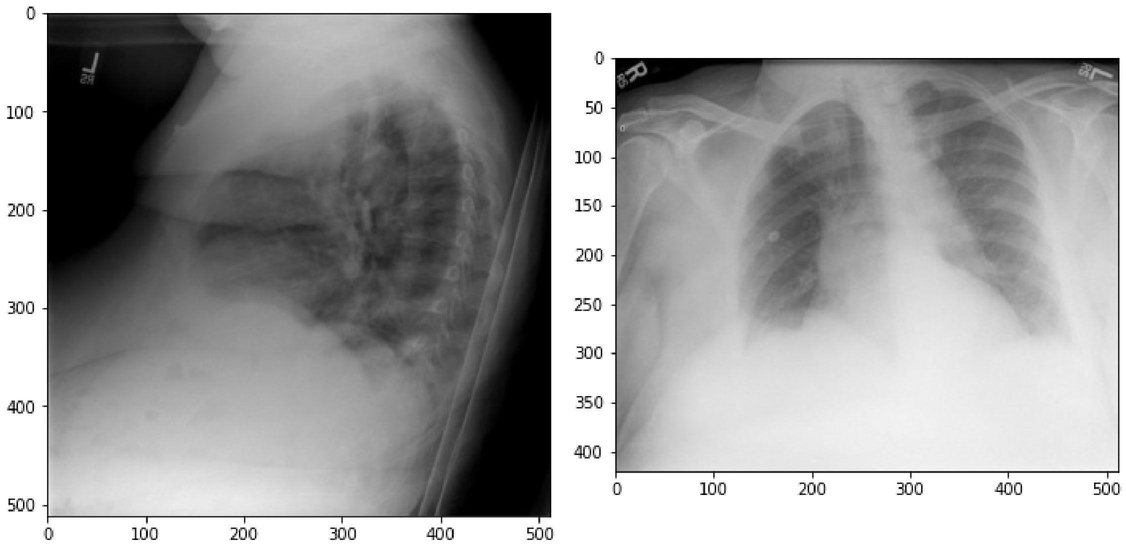



```
The predicted text is:   no acute cardiopulmonary abnormality <end>
```

In [67]:

```
print('The Actual text is:',test_output[366])
caption_of_image(test_input[366])
```

The Actual text is: <start> low volume study without acute cardiopulmonary abnormalities <end>

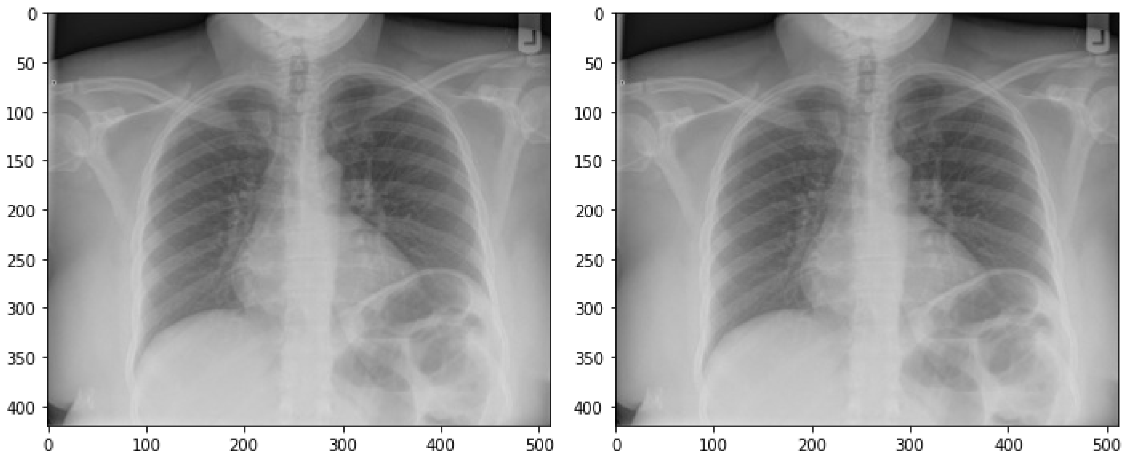


The predicted text is: no acute cardiopulmonary abnormality <end>

In [68]:

```
print('The Actual text is:',test_output[314])
caption_of_image(test_input[314])
```

The Actual text is: <start> elevated left diaphragm no focal airspace disease <end>



The predicted text is: no acute cardiopulmonary abnormality <end>

In [ ]:

```

```