Final_Model

July 3, 2020

[4]:
```
import pandas as pd
import numpy as np
import random as rn
import string
import matplotlib.pyplot as plt
from tqdm import tqdm
%matplotlib inline
import re
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import nltk
import warnings
warnings.filterwarnings('ignore')
from IPython.display import Image
```

[5]:
```
import tensorflow as tf
```

[ ]:
```
tf.__version__
```

[ ]:
```
'2.2.0'
```

[ ]:
```
data = pd.read_csv("data.csv")
data.head()
```

[ ]:
```
                                          image_name  … image_count
0    CXR1_1_IM-0001-3001.png,CXR1_1_IM-0001-4001.png  …           2
1      CXR10_IM-0002-1001.png,CXR10_IM-0002-2001.png  …           2
2    CXR100_IM-0002-1001.png,CXR100_IM-0002-2001.png  …           2
3  CXR1000_IM-0003-2001.png,CXR1000_IM-0003-1001…  …           3
4  CXR1001_IM-0004-1002.png,CXR1001_IM-0004-1001.png  …           2

[5 rows x 9 columns]
```

[ ]:
```
data_projecttions = pd.read_csv("data_projections.csv")
data_projecttions.head()
```

[ ]:
```
   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
```

[ ]:
```
image_path = "img/"
```

1 Constructing data point

limiting the data point to 2 images per data point, if we have 5 images, its 4+1 (all image + last image) so make it as 4 data points as below

if i have 5 images then
1 image + 5th image
2nd image + 5th image
3rd image + 5th image
4th image + 5th image
4 data point

like wise for other data point,

if i have 3 images then
1st + 3rd
2nd + 3rd
2 data point

if i have 4 images then
1st + 4th
2nd + 4th
3rd + 4th
3 data point

[ ]:
```
data.shape
```

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

[ ]:
```
data['image_count'].value_counts()
```

[ ]:
```
2    3208
1     446
3     181
4      15
5       1
Name: image_count, dtype: int64
```

Validate output:
2 images = 3208

3 images = 181*2*
4 images = 153
5 images = 1*4
Total = 3619

Total data point_

we should create duplicate dataframe separately to keep it in all dataset train test validate sets
1 images = 446
3619+446 = 4065

[ ]:
```
def find_Fr_la(li):
    """Function to find the lateral anf frontal images
    Returns All frontal as list and Lateral as last image"""
    img_list = []
    last_img = ""
    for i in li:
        projection = data_projecttions[data_projecttions['filename'].str.
 ↪contains(re.search(r"\d.*\_IM-\d.*\.", i).group())]['projection'].values
        if "Lateral" == projection:
            last_img = i
        else:
            img_list.append(i)
    return img_list, last_img
```

[ ]:
```
#Creating structured data from raw xml files
columns = ["image_1", "image_2", "impression"]
df = pd.DataFrame(columns = columns)
columns = ["image_1", "image_2", "impression"]
df_dup = pd.DataFrame(columns = columns)
no_lateral = 0
for item in tqdm(data.iterrows()):
    l = item[1]['image_name'].split(',')
    if len(l) > 2:
        li, last_img = find_Fr_la(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 = df.append(pd.Series([image_1, image_2, item[1]['impression']],␣
 ↪index = columns), ignore_index = True)
    elif len(l) == 2:
        image_1 = l[0]
        image_2 = l[1]
        df = df.append(pd.Series([image_1, image_2, item[1]['impression']],␣
 ↪index = columns), ignore_index = True)
```

```
    elif len(l) == 1:
        #creating duplicate dataframe separately to keep it in all dataset␣
↪train test validate
        df_dup = df_dup.append(pd.Series([l[0], l[0], item[1]['impression']],␣
↪index = columns), ignore_index = True)
print("Total Report without Lateral images {}".format(no_lateral))
```

```
3851it [00:13, 295.13it/s]

Total Report without Lateral images 1
```

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

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

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

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

[ ]:
```
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
```

[ ]:
```
df['impression'] = add_start_end_token(df['impression'])
df_dup['impression'] = add_start_end_token(df_dup['impression'])
```

```
100%|          | 3532/3532 [00:00<00:00, 798785.82it/s]
100%|          | 446/446 [00:00<00:00, 404204.75it/s]
```

[ ]:
```
df[['image_1','image_2', 'impression']].head()
```

[ ]:
```
                    image_1  …
impression
0   CXR1_1_IM-0001-3001.png  …                       <start> normal chest x
<end>
1    CXR10_IM-0002-1001.png  …     <start> no acute cardiopulmonary process
<end>
2   CXR100_IM-0002-1001.png  …                    <start> no active disease
<end>
3  CXR1000_IM-0003-1001.png  …  <start> increased opacity in the right upper
```

```
l…
4  CXR1000_IM-0003-3001.png  …  <start> increased opacity in the right upper
l…

 [5 rows x 3 columns]
```

[ ]:
```python
df.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
```

[ ]:
```python
df_dup[['image_1','image_2', 'impression']].head()
```

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

 [5 rows x 3 columns]
```

[ ]:
```python
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
```

[ ]:
```
image_name = []
for img in tqdm(data['image_name'].str.split(',')):
    for i in range(len(img)):
        image_name.append(img[i])
```

```
100%|      | 3851/3851 [00:00<00:00, 1058054.81it/s]
```

[ ]:
```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import InceptionV3,␣
 ↪preprocess_input
from tensorflow.keras.models import Model
```

[ ]:
```
#loads the pretrained weights
image_features_model = InceptionV3(include_top=False, weights='imagenet',␣
 ↪pooling='avg', input_shape=(299,299,3))
image_features_model.load_weights("trained_weights-07-0.9102.hdf5")
```

[ ]:
```
img_tensor = []
#creates image feature vector
for img in tqdm(image_name):
    img = tf.io.read_file(image_path + str(img))
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = preprocess_input(img)
    img_features = image_features_model(tf.constant(img)[None, :])
    img_tensor.append(img_features)
```

```
100%|      | 7470/7470 [14:41<00:00,  8.48it/s]
```

2 Train Test and Validation split

[ ]:
```
##fixing numpy RS
np.random.seed(42)
##fixing tensorflow RS
tf.random.set_seed(32)
##python RS
rn.seed(12)
```

[ ]:
```
i_train, input_test, o_train, output_test =␣
 ↪train_test_split(df[['image_1','image_2']].values, df['impression'].values,␣
 ↪test_size=0.1, random_state=15)
input_train, input_val, output_train, output_val = train_test_split(i_train,␣
 ↪o_train, test_size=0.2, random_state=15)
input_train.shape, output_train.shape, input_val.shape, output_val.shape,␣
 ↪input_test.shape, output_test.shape
```

[ ]: ((2542, 2), (2542,), (636, 2), (636,), (354, 2), (354,))

- Train test and validation split for duplicate dataframe

[ ]:
```
i_train_dup, input_test_dup, o_train_dup, output_test_dup =␣
↪train_test_split(df_dup[['image_1','image_2']].values, df_dup['impression'].
↪values, test_size=0.1, random_state=15)
input_train_dup, input_val_dup, output_train_dup, output_val_dup =␣
↪train_test_split(i_train_dup, o_train_dup, test_size=0.2, random_state=15)
input_train_dup.shape, output_train_dup.shape, input_val_dup.shape,␣
↪output_val_dup.shape, input_test_dup.shape, output_test_dup.shape
```

[ ]: ((320, 2), (320,), (81, 2), (81,), (45, 2), (45,))

- Append duplicate data equally with train, test, and validation dataset

[ ]:
```
in_train = np.append(input_train, input_train_dup, axis=0)
out_train = np.append(output_train, output_train_dup, axis=0)
in_val = np.append(input_val, input_val_dup, axis=0)
out_val = np.append(output_val, output_val_dup, axis=0)
in_test = np.append(input_test, input_test_dup, axis=0)
out_test = np.append(output_test, output_test_dup, axis=0)
print("===== Final data point shape =====")
in_train.shape, out_train.shape, in_val.shape, out_val.shape, in_test.shape,␣
↪out_test.shape
```

```
===== Final data point shape =====
```

[ ]: ((2862, 2), (2862,), (717, 2), (717,), (399, 2), (399,))

Shuffle the data point

[ ]:
```
# Shuffle captions and image_names together
# Set a random state
for i in range(3):
    in_train, out_train = shuffle(in_train, out_train, random_state=15)
    in_val, out_val = shuffle(in_val, out_val, random_state=15)
    in_test, out_test = shuffle(in_test, out_test, random_state=15)
```

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

max_len_output = 80


tokenizer = Tokenizer(oov_token="<unk>", filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}~␣
↪')
tokenizer.fit_on_texts(out_train)
```

```
text_train = tokenizer.texts_to_sequences(out_train)
text_test = tokenizer.texts_to_sequences(out_test)
text_val = tokenizer.texts_to_sequences(out_val)
dictionary = tokenizer.word_index

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

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

[ ]: 1339

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

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

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

[ ]:
```
text_output_train = pad_sequences(text_train, maxlen=max_len_output,␣
 ↪dtype='int32', padding='post', truncating='post')
text_output_val = pad_sequences(text_val, maxlen=max_len_output, dtype='int32',␣
 ↪padding='post', truncating='post')
text_output_test = pad_sequences(text_test, maxlen=max_len_output,␣
 ↪dtype='int32', padding='post', truncating='post')
```

[ ]:
```
text_output_train.shape
```

[ ]: (2862, 80)

[ ]:
```
def multi_image(img, imp):
    return tf.convert_to_tensor([img_tensor[image_name.index(img[0].
 ↪decode('utf-8'))], img_tensor[image_name.index(img[1].decode('utf-8'))]]),␣
 ↪imp
```

[ ]:
```
in_train[0], text_output_train[0]
```

[ ]:
```
(array(['CXR1391_IM-0250-1001.png', 'CXR1391_IM-0250-5001.png'],
       dtype=object),
 array([ 2, 67, 28, 74, 88, 22,  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,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0], dtype=int32))
```

[ ]:
```
dataset_train = tf.data.Dataset.from_tensor_slices((in_train,␣
 ↪text_output_train))

# Use map to load the numpy files in parallel
dataset_train = dataset_train.map(lambda item1, item2: tf.numpy_function(
          multi_image, [item1, item2], [tf.float32, tf.int32]),
          num_parallel_calls=tf.data.experimental.AUTOTUNE)

dataset_val = tf.data.Dataset.from_tensor_slices((in_val, text_output_val))

# Use map to load the numpy files in parallel
dataset_val = dataset_val.map(lambda item1, item2: tf.numpy_function(
          multi_image, [item1, item2], [tf.float32, tf.int32]),
          num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

[ ]:
```
BATCH_SIZE = 32
BUFFER_SIZE = 1000
embedding_dim = 256
units = 128
```

[ ]:
```
# Shuffle and batch Train
dataset_train = dataset_train.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset_train = dataset_train.prefetch(buffer_size=tf.data.experimental.
 ↪AUTOTUNE)
# Shuffle and batch Validation
dataset_val = dataset_val.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset_val = dataset_val.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

[ ]:
```
class Encoder(tf.keras.Model):
    def __init__(self, embedding_dim):
        super(Encoder, self).__init__()
        self.dense = tf.keras.layers.Dense(embedding_dim, activation='relu',␣
 ↪kernel_initializer=tf.keras.initializers.glorot_uniform(seed=45))
        self.repeatVector = tf.keras.layers.RepeatVector(max_len_output)
        self.concat = tf.keras.layers.Concatenate()
    def call(self, x):
        # CNN two input Images concatenate to get single vector
        # Concatenating 2 images
        # Input x shape (batch_size, 2,None,2048)
```

```
        # x1 shape (batch_size, None, 2048)
        # x2 shape (batch_size, None, 2048)
        encoder_concat = self.concat([x[:,0], x[:,1]])
        x = self.dense(encoder_concat)
        x = tf.nn.relu(x)
        #converting 3d tensor [None, 1, units] to 2d tensor [None, units] for␣
↪repeat layer
        x = self.repeatVector(tf.reshape(x,[x.shape[0], x.shape[2]]))
        return x
```

[ ]:
```
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,␣
↪input_length=max_len_output)
        self.w_1 = tf.keras.layers.Dense(units, activation='relu')
        self.w_2 = tf.keras.layers.Dense(units, activation='relu')
        # Bidirectional LSTM
        self.bilstm_1 = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM \
                                        (self.units, dropout=0.3,␣
↪return_sequences=True, return_state=True, \
                                         #kernel_regularizer=l2(0.001),␣
↪recurrent_regularizer=l2(0.001), \
                                         recurrent_activation='relu',␣
↪recurrent_initializer= \
                                         tf.keras.initializers.
↪glorot_uniform(seed=26)))
        self.bilstm_2 = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM\
                                        (self.units, dropout=0.2,␣
↪return_sequences=True, return_state=True, \
                                         #kernel_regularizer=l2(0.001),␣
↪recurrent_regularizer=l2(0.001), \
                                         recurrent_activation='relu',␣
↪recurrent_initializer= \
                                         tf.keras.initializers.
↪glorot_uniform(seed=26)))
        self.dense_1 = tf.keras.layers.Dense(self.units, activation='relu',␣
↪kernel_initializer=tf.keras.initializers.glorot_uniform(seed=45))
        self.dense_2 = tf.keras.layers.Dense(vocab_size, kernel_initializer=tf.
↪keras.initializers.glorot_uniform(seed=45))
        #Additive Attention
        self.attention = tf.keras.layers.AdditiveAttention(self.units)
        self.concat = tf.keras.layers.Concatenate()
        self.flatten = tf.keras.layers.Flatten()
    def call(self, x):
```

```
        # x = [dec_input, features, hidden] [decoder_input_word_tensor,␣
↪encoder_output, hidden_state_previous]
        embedded_layer = self.embedding(x[0])
        x_con = self.concat([embedded_layer, x[1]])
        bi_lstm = self.bilstm_1(x_con)
        lstm, forward_h, forward_c, backward_h, backward_c = self.
↪bilstm_2(bi_lstm)
        state = self.concat([forward_h, backward_h])
        state = self.concat([state,x[2]])
        state = self.w_1(state)
        lstm = self.w_2(lstm)
        additive = self.attention([lstm,state])
        #decoder_1_1/additive_attention_1/Identity_1:0, shape=(None, max_len,␣
↪128)
        #Reshaping to (None, max_len * units)
        output = self.flatten(additive)
        output = self.dense_1(output)
        output = self.dense_2(output)
        # output will be (None, 1339)
        return output, state
```

[ ]:
```
encoder = Encoder(embedding_dim)
```

[ ]:
```
decoder = Decoder(embedding_dim, units, vocab_size)
```

```
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernel since it doesn't meet
the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernel since it doesn't meet
the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernel since it doesn't meet
the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernel since it doesn't meet
the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernel since it doesn't meet
the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernel since it doesn't meet
the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
```

[ ]:
```
optimizer = tf.keras.optimizers.Adam()
loss_obj = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')
```

```
acc_obj = tf.keras.metrics.SparseCategoricalAccuracy()

def loss_func(real, pred):
    """Loss calculation"""
    loss_f = loss_obj(real, pred)
    return tf.reduce_mean(loss_f)


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

[ ]:
```
!rm -r logs/
```

[ ]:
```
import datetime
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'logs/' + current_time + '/train'
val_log_dir = 'logs/' + 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)
```

[ ]:
```
@tf.function
def train_step(tensor, target):
    """Subclassed model training step"""
    loss = 0
    accuracy = 0
    # initializing the hidden state for each batch
    hidden =  tf.zeros((target.shape[0], units))
    dec_input = tf.expand_dims([tokenizer.word_index['<start>']] * target.
 ↪shape[0], 1)
    actual, predicted = list(), list()
    with tf.GradientTape() as tape:
        features = encoder(tensor)
        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions, hidden = decoder([dec_input, features, hidden])
            loss += loss_func(target[:, i], predictions)
            accuracy += acc_func(target[:, i], predictions)
            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i], 1)
    total_loss = (loss / int(target.shape[1]))
    total_acc = (accuracy / int(target.shape[1]))

    trainable_variables = encoder.trainable_variables + decoder.
 ↪trainable_variables
```

```
    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):
    """Subclassed model validation step"""
    loss_val = 0
    accuracy_val = 0
    # initializing the hidden state for each batch
    hidden =  tf.zeros((target.shape[0], units))
    dec_input = tf.expand_dims([tokenizer.word_index['<start>']] * target.
 ↪shape[0], 1)
    with tf.GradientTape() as tape:
        features = encoder(tensor)
        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions_val, hidden = decoder([dec_input, features, hidden])
            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)
    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
```

[ ]:
```
EPOCHS = 50
val_loss_history = []
count = 0
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('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))
    with train_summary_writer.as_default():
        tf.summary.scalar('loss', total_loss_train/ int(len(in_train) //␣
↪BATCH_SIZE), step=epoch)
        tf.summary.scalar('accuracy', total_acc_train/ int(len(in_train) //␣
↪BATCH_SIZE), step=epoch)

    #validation
    print('Validation loss')
    for (batch, (jpg_tensor, target)) in enumerate(dataset_val):
        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 val_summary_writer.as_default():
        tf.summary.scalar('loss', total_loss_val/int(len(in_val) //␣
↪BATCH_SIZE), step=epoch)
        tf.summary.scalar('accuracy', total_acc_val/int(len(in_val) //␣
↪BATCH_SIZE), step=epoch)

    print ('Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test Accuracy: {}'.
↪format(epoch+1,
                           total_loss_train/ int(len(in_train) // BATCH_SIZE),
                           (total_acc_train/ int(len(in_train) //␣
↪BATCH_SIZE))*100,
                           total_loss_val/int(len(in_val) // BATCH_SIZE),
                           (total_acc_val/int(len(in_val) // BATCH_SIZE))*100))

    val_loss_history.append(total_loss_val/int(len(in_val) // BATCH_SIZE))
    if epoch > 6:
        if count >= 4:
            print("\n\nEarlyStopping Invoked!!! Stopping the training\n\n")
            break
        else:
            if val_loss_history[epoch-1] < val_loss_history[epoch]:
                print("\n******count++ Increased******\n")
                count +=1
```

```
====== Start Epoch 1 ========
Train loss
Epoch 1 Batch 0 Loss 7.1097 acc 0.0000
Epoch 1 Batch 40 Loss 0.7348 acc 0.8282
```

```
Epoch 1 Batch 80 Loss 0.9673 acc 0.8511
Validation loss
Epoch 1 Batch 0 Loss 0.9144 acc 0.8545
Epoch 1, Loss: 1.3000187873840332, Accuracy: 80.83170318603516, Test Loss:
0.9424629211425781, Test Accuracy: 89.50749206542969
====== Start Epoch 2 ========
Train loss
Epoch 2 Batch 0 Loss 1.0024 acc 0.8567
Epoch 2 Batch 40 Loss 1.2531 acc 0.8599
Epoch 2 Batch 80 Loss 0.4921 acc 0.8624
Validation loss
Epoch 2 Batch 0 Loss 0.6302 acc 0.8635
Epoch 2, Loss: 0.8713120222091675, Accuracy: 86.97420501708984, Test Loss:
0.7736465930938721, Test Accuracy: 90.29264831542969
====== Start Epoch 3 ========
Train loss
Epoch 3 Batch 0 Loss 0.7500 acc 0.8638
Epoch 3 Batch 40 Loss 0.6370 acc 0.8661
Epoch 3 Batch 80 Loss 0.4971 acc 0.8680
Validation loss
Epoch 3 Batch 0 Loss 0.4858 acc 0.8685
Epoch 3, Loss: 0.6340424418449402, Accuracy: 87.58219146728516, Test Loss:
0.6784430742263794, Test Accuracy: 90.82454681396484
====== Start Epoch 4 ========
Train loss
Epoch 4 Batch 0 Loss 0.6093 acc 0.8690
Epoch 4 Batch 40 Loss 0.5876 acc 0.8699
Epoch 4 Batch 80 Loss 0.6561 acc 0.8718
Validation loss
Epoch 4 Batch 0 Loss 0.4393 acc 0.8722
Epoch 4, Loss: 0.5907294154167175, Accuracy: 88.01396179199219, Test Loss:
0.6547850966453552, Test Accuracy: 91.20196533203125
====== Start Epoch 5 ========
Train loss
Epoch 5 Batch 0 Loss 0.5242 acc 0.8725
Epoch 5 Batch 40 Loss 0.6156 acc 0.8734
Epoch 5 Batch 80 Loss 0.3255 acc 0.8748
Validation loss
Epoch 5 Batch 0 Loss 0.6553 acc 0.8750
Epoch 5, Loss: 0.5610045790672302, Accuracy: 88.34941864013672, Test Loss:
0.6239184141159058, Test Accuracy: 91.4938735961914
====== Start Epoch 6 ========
Train loss
Epoch 6 Batch 0 Loss 0.6868 acc 0.8753
Epoch 6 Batch 40 Loss 0.7803 acc 0.8760
Epoch 6 Batch 80 Loss 0.6151 acc 0.8771
Validation loss
Epoch 6 Batch 0 Loss 0.5967 acc 0.8774
```

```
Epoch 6, Loss: 0.5359786152839661, Accuracy: 88.60169982910156, Test Loss:
0.6147091388702393, Test Accuracy: 91.74430084228516
====== Start Epoch 7 ========
Train loss
Epoch 7 Batch 0 Loss 0.3569 acc 0.8777
Epoch 7 Batch 40 Loss 0.5505 acc 0.8784
Epoch 7 Batch 80 Loss 0.6732 acc 0.8792
Validation loss
Epoch 7 Batch 0 Loss 0.4500 acc 0.8794
Epoch 7, Loss: 0.5178211331367493, Accuracy: 88.83881378173828, Test Loss:
0.5891263484954834, Test Accuracy: 91.95465087890625
====== Start Epoch 8 ========
Train loss
Epoch 8 Batch 0 Loss 0.4536 acc 0.8797
Epoch 8 Batch 40 Loss 0.3972 acc 0.8803
Epoch 8 Batch 80 Loss 0.5136 acc 0.8813
Validation loss
Epoch 8 Batch 0 Loss 0.6434 acc 0.8813
Epoch 8, Loss: 0.497597336769104, Accuracy: 89.0295639038086, Test Loss:
0.5786417722702026, Test Accuracy: 92.14285278320312
====== Start Epoch 9 ========
Train loss
Epoch 9 Batch 0 Loss 0.3983 acc 0.8816
Epoch 9 Batch 40 Loss 0.2863 acc 0.8822
Epoch 9 Batch 80 Loss 0.3296 acc 0.8829
Validation loss
Epoch 9 Batch 0 Loss 0.6489 acc 0.8830
Epoch 9, Loss: 0.48032885789871216, Accuracy: 89.2224349975586, Test Loss:
0.5813894271850586, Test Accuracy: 92.3270034790039

******count++ Increased******

====== Start Epoch 10 ========
Train loss
Epoch 10 Batch 0 Loss 0.7457 acc 0.8832
Epoch 10 Batch 40 Loss 0.5470 acc 0.8838
Epoch 10 Batch 80 Loss 0.4051 acc 0.8846
Validation loss
Epoch 10 Batch 0 Loss 0.4565 acc 0.8846
Epoch 10, Loss: 0.4641895890235901, Accuracy: 89.38082122802734, Test Loss:
0.5596616268157959, Test Accuracy: 92.49658966064453
====== Start Epoch 11 ========
Train loss
Epoch 11 Batch 0 Loss 0.4028 acc 0.8848
Epoch 11 Batch 40 Loss 0.2393 acc 0.8853
Epoch 11 Batch 80 Loss 0.4847 acc 0.8857
Validation loss
Epoch 11 Batch 0 Loss 0.6242 acc 0.8858
```

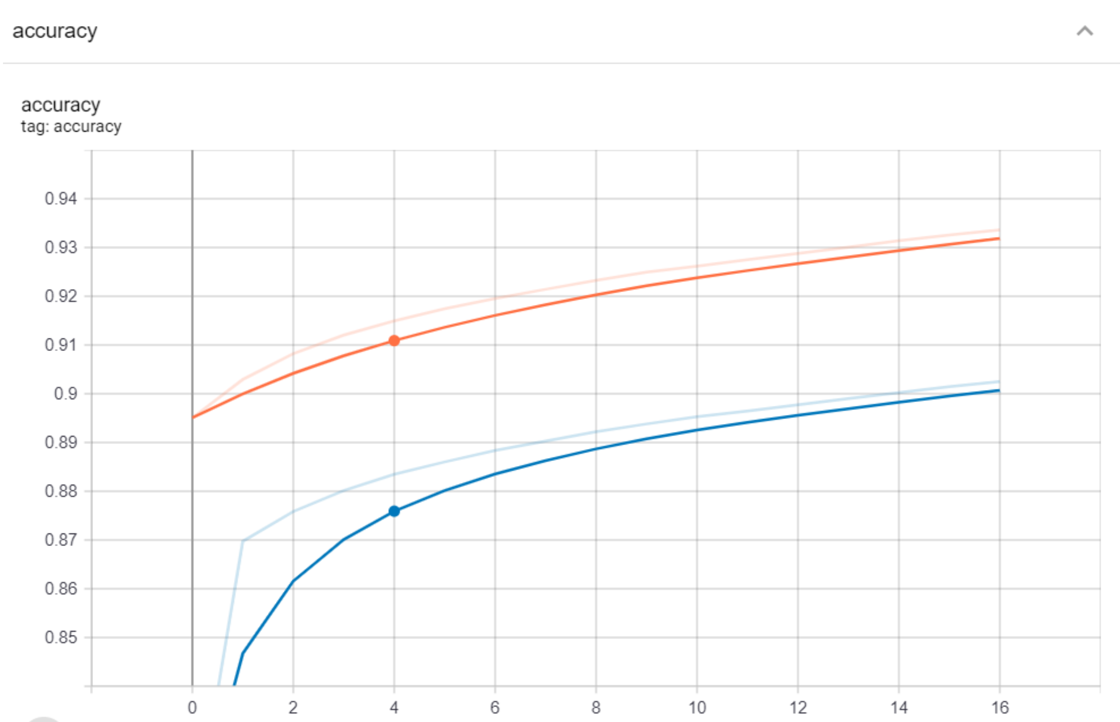
```
Epoch 11, Loss: 0.46988070011138916, Accuracy: 89.52869415283203, Test Loss:
0.5668795704841614, Test Accuracy: 92.61798858642578

******count++ Increased******

====== Start Epoch 12 ========
Train loss
Epoch 12 Batch 0 Loss 0.5953 acc 0.8860
Epoch 12 Batch 40 Loss 0.4195 acc 0.8863
Epoch 12 Batch 80 Loss 0.3853 acc 0.8870
Validation loss
Epoch 12 Batch 0 Loss 0.6681 acc 0.8871
Epoch 12, Loss: 0.4407421052455902, Accuracy: 89.64275360107422, Test Loss:
0.5485363006591797, Test Accuracy: 92.74858093261719
====== Start Epoch 13 ========
Train loss
Epoch 13 Batch 0 Loss 0.3659 acc 0.8873
Epoch 13 Batch 40 Loss 0.3675 acc 0.8876
Epoch 13 Batch 80 Loss 0.5513 acc 0.8883
Validation loss
Epoch 13 Batch 0 Loss 0.5401 acc 0.8884
Epoch 13, Loss: 0.42082515358924866, Accuracy: 89.77299499511719, Test Loss:
0.536683976650238, Test Accuracy: 92.87966918945312
====== Start Epoch 14 ========
Train loss
Epoch 14 Batch 0 Loss 0.3908 acc 0.8885
Epoch 14 Batch 40 Loss 0.4796 acc 0.8889
Epoch 14 Batch 80 Loss 0.4479 acc 0.8894
Validation loss
Epoch 14 Batch 0 Loss 0.8448 acc 0.8896
Epoch 14, Loss: 0.4039926528930664, Accuracy: 89.90110778808594, Test Loss:
0.5258252620697021, Test Accuracy: 93.00775909423828
====== Start Epoch 15 ========
Train loss
Epoch 15 Batch 0 Loss 0.3331 acc 0.8898
Epoch 15 Batch 40 Loss 0.4393 acc 0.8901
Epoch 15 Batch 80 Loss 0.3984 acc 0.8907
Validation loss
Epoch 15 Batch 0 Loss 0.8852 acc 0.8908
Epoch 15, Loss: 0.3891105353832245, Accuracy: 90.02294158935547, Test Loss:
0.5271832942962646, Test Accuracy: 93.13751220703125

******count++ Increased******

====== Start Epoch 16 ========
Train loss
Epoch 16 Batch 0 Loss 0.6256 acc 0.8909
Epoch 16 Batch 40 Loss 0.3845 acc 0.8914
```

```
Epoch 16 Batch 80 Loss 0.3670 acc 0.8918
Validation loss
Epoch 16 Batch 0 Loss 0.2573 acc 0.8919
Epoch 16, Loss: 0.3789263069629669, Accuracy: 90.14373779296875, Test Loss:
0.5365054607391357, Test Accuracy: 93.25570678710938

******count++ Increased******

====== Start Epoch 17 ========
Train loss
Epoch 17 Batch 0 Loss 0.5184 acc 0.8920
Epoch 17 Batch 40 Loss 0.3936 acc 0.8924
Epoch 17 Batch 80 Loss 0.2590 acc 0.8929
Validation loss
Epoch 17 Batch 0 Loss 0.2585 acc 0.8930
Epoch 17, Loss: 0.3705205023288727, Accuracy: 90.24910736083984, Test Loss:
0.5233340859413147, Test Accuracy: 93.36151123046875


EarlyStopping Invoked!!! Stopping the training
```

3 Accuracy and Loss Tensorboard Visualization

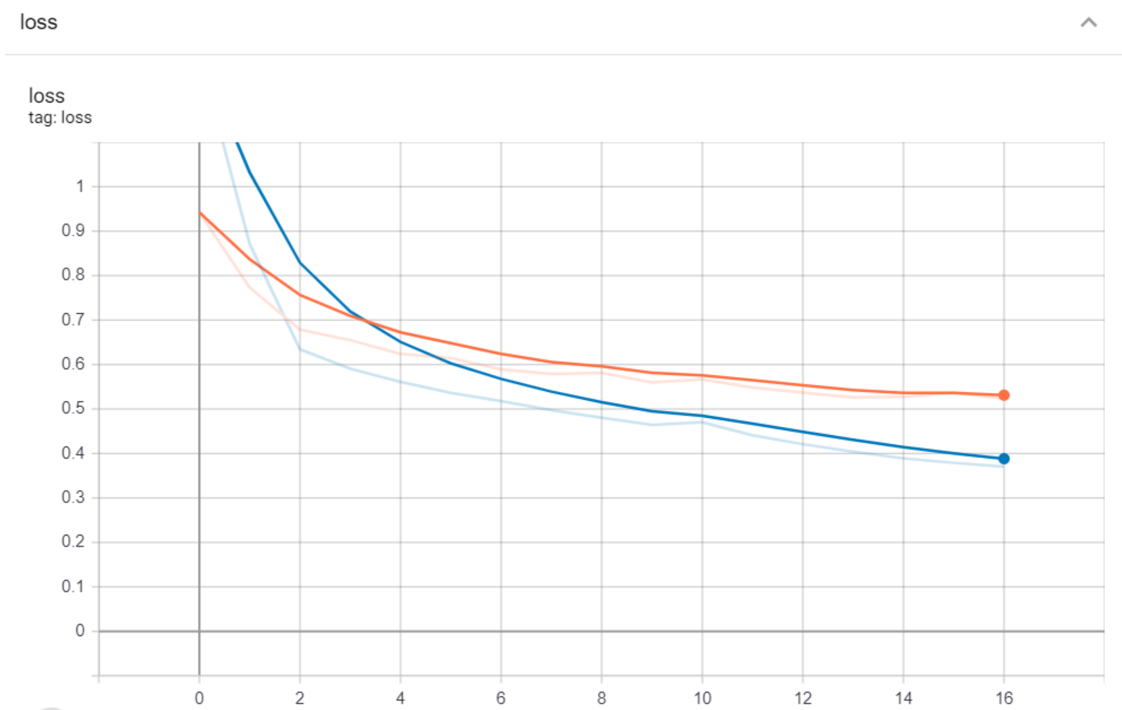
[ ]:
```
Image(filename='final_acc.png')
```

[ ]:



[ ]:
```
Image(filename='final_loss.png')
```

[ ]:



[ ]:
```
#%load_ext tensorboard
%reload_ext tensorboard
```

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

```
<IPython.core.display.Javascript object>
```

[19]:
```
def get_img_tensor(image_path, img_name, model_image):
    """Creates the image feature vector"""
    img = tf.io.read_file(image_path + str(img_name))
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = tf.keras.applications.xception.preprocess_input(img)
    img_features = model_image(tf.constant(img)[None, :])
    return img_features
```

[65]:
```
def evaluate(img_name):
    """Argmax search implementaion max prob score"""
    hidden =  tf.zeros((1, units))
```

```
    img_tensor = tf.convert_to_tensor([get_img_tensor(image_path,img_name[0],␣
 ↪image_features_model),
                                       get_img_tensor(image_path,img_name[1],␣
 ↪image_features_model)])
    img_features = tf.constant(img_tensor)[None, :]
    features_val = encoder(img_features)

    dec_input = tf.expand_dims([tokenizer.word_index['<start>']], 0)
    result = []
    text = ""
    for i in range(max_len_output):
        predictions, hidden = decoder([tf.cast(dec_input, tf.float32),␣
 ↪features_val, hidden])
        predicted_id = tf.argmax(predictions, axis=1)[0].numpy()
        if predicted_id ==0:
            word = ""
        else:
            word = tokenizer.index_word[predicted_id]
        result.append(word)
        text += " " + word
        if word == '<end>' or word == 'end':
            return result, text

        dec_input = tf.expand_dims([predicted_id], 0)
    return result, text
```

[52]:

```
#ref: https://yashk2810.github.io/
 ↪Image-Captioning-using-InceptionV3-and-Beam-Search/
#https://www.geeksforgeeks.org/sorted-function-python/

def calculate_score(x):
    """Calculates the cumulative score for the length of sentence"""
    return x[1]/len(x[0])

def beam_search(img_name, beam_index = 3):
    """Beam search implementaion takes images as input"""
    hidden =  tf.zeros((1, units))
    img_tensor = tf.convert_to_tensor([get_img_tensor("img/",img_name[0],␣
 ↪image_features_model),
                                       get_img_tensor("img/",img_name[1],␣
 ↪image_features_model)])
    img_features = tf.constant(img_tensor)[None, :]
    features_val = encoder(img_features)
    start = [tokenizer.word_index["<start>"]]
    dec_word = [[start, 0.0]]
    while len(dec_word[0][0]) < max_len_output:
```

```
        temp = []
        for s in dec_word:
            predictions, hidden = decoder([tf.cast(tf.expand_dims([s[0][-1]],␣
↪0), tf.float32), features_val, hidden])

            word_preds = np.argsort(predictions[0])[-beam_index:]
            # Getting the top <beam_index>(n) predictions and creating a
            # new list so as to put them via the model again
            for w in word_preds:
                next_cap, prob = s[0][:], s[1]
                next_cap.append(w)
                prob += predictions[0][w]
                temp.append([next_cap, prob.numpy()])
        dec_word = temp
        # Sorting according to the probabilities scores
        dec_word = sorted(dec_word, reverse=False, key=calculate_score)
        # Getting the top words
        dec_word = dec_word[-beam_index:]
    dec_word = dec_word[-1][0]
    impression = [tokenizer.index_word[i] for i in dec_word if i !=0]
    result = []

    for i in impression:
        if i != '<end>':
            result.append(i)
        else:
            break

    text = ' '.join(result[1:])
    return result, text
```

[99]:
```
import matplotlib.image as mpimg
from nltk.translate.bleu_score import sentence_bleu
def test_img_cap_beam(img_data, actual_text, beam_indexing):
    result, text = beam_search(img_data, beam_index = beam_indexing)
    """Displays images for given input array of image names"""
    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(image_path+img)
        imgplot = axs[count].imshow(img_, cmap = 'bone')
        count +=1
    plt.show()
    reference = [actual_text.split()[1:-1]]
    result = result[1:]
    print("Beam Search, index=", beam_indexing)
```

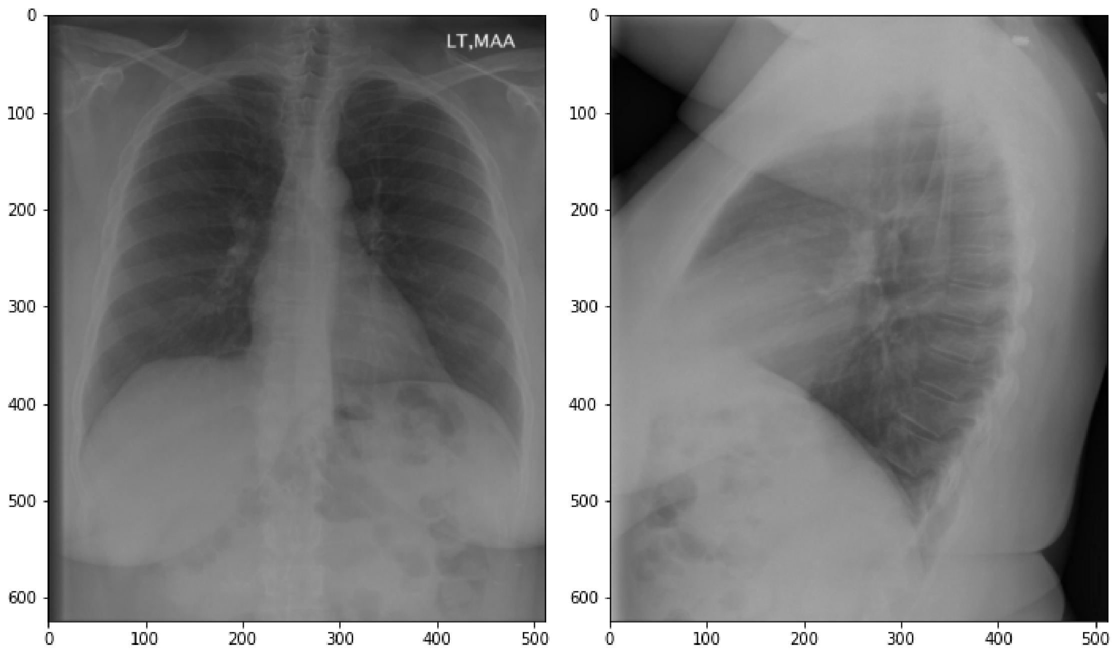
```
    print("="*50)
    print("Actual", actual_text)
    print("Predicted:",text)
    print("="*50)
    print('Individual 1-gram: {:.4f} Cumulative 1-gram: {:.4f}'.
↪format(sentence_bleu(reference, result, weights=(1, 0, 0, 0)),␣
↪sentence_bleu(reference, result, weights=(1, 0, 0, 0))))
    print('Individual 2-gram: {:.4f} Cumulative 2-gram: {:.4f}'.
↪format(sentence_bleu(reference, result, weights=(0, 1, 0, 0)),␣
↪sentence_bleu(reference, result, weights=(0.5, 0.5, 0, 0))))
    print('Individual 3-gram: {:.4f} Cumulative 3-gram: {:.4f}'.
↪format(sentence_bleu(reference, result, weights=(0, 0, 1, 0)),␣
↪sentence_bleu(reference, result, weights=(0.33, 0.33, 0.33, 0))))
    print('Individual 4-gram: {:.4f} Cumulative 4-gram: {:.4f}'.
↪format(sentence_bleu(reference, result, weights=(0, 0, 0, 1)),␣
↪sentence_bleu(reference, result, weights=(0.25, 0.25, 0.25, 0.25))))
```

[67]:
```
def test_img_cap(img_data, actual_text):
    result, text = evaluate(img_data)
    """Displays images for given input array of image names"""
    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(image_path+img)
        imgplot = axs[count].imshow(img_, cmap = 'bone')
        count +=1
    plt.show()
    reference = [actual_text.split()[1:-1]]
    result = result[:-1]
    print("="*50)
    print("Actual", actual_text)
    print("Predicted:",text)
    print("="*50)
    print('Individual 1-gram: {:.4f} Cumulative 1-gram: {:.4f}'.
↪format(sentence_bleu(reference, result, weights=(1, 0, 0, 0)),␣
↪sentence_bleu(reference, result, weights=(1, 0, 0, 0))))
    print('Individual 2-gram: {:.4f} Cumulative 2-gram: {:.4f}'.
↪format(sentence_bleu(reference, result, weights=(0, 1, 0, 0)),␣
↪sentence_bleu(reference, result, weights=(0.5, 0.5, 0, 0))))
    print('Individual 3-gram: {:.4f} Cumulative 3-gram: {:.4f}'.
↪format(sentence_bleu(reference, result, weights=(0, 0, 1, 0)),␣
↪sentence_bleu(reference, result, weights=(0.33, 0.33, 0.33, 0))))
    print('Individual 4-gram: {:.4f} Cumulative 4-gram: {:.4f}'.
↪format(sentence_bleu(reference, result, weights=(0, 0, 0, 1)),␣
↪sentence_bleu(reference, result, weights=(0.25, 0.25, 0.25, 0.25))))
```

4 Evaluation

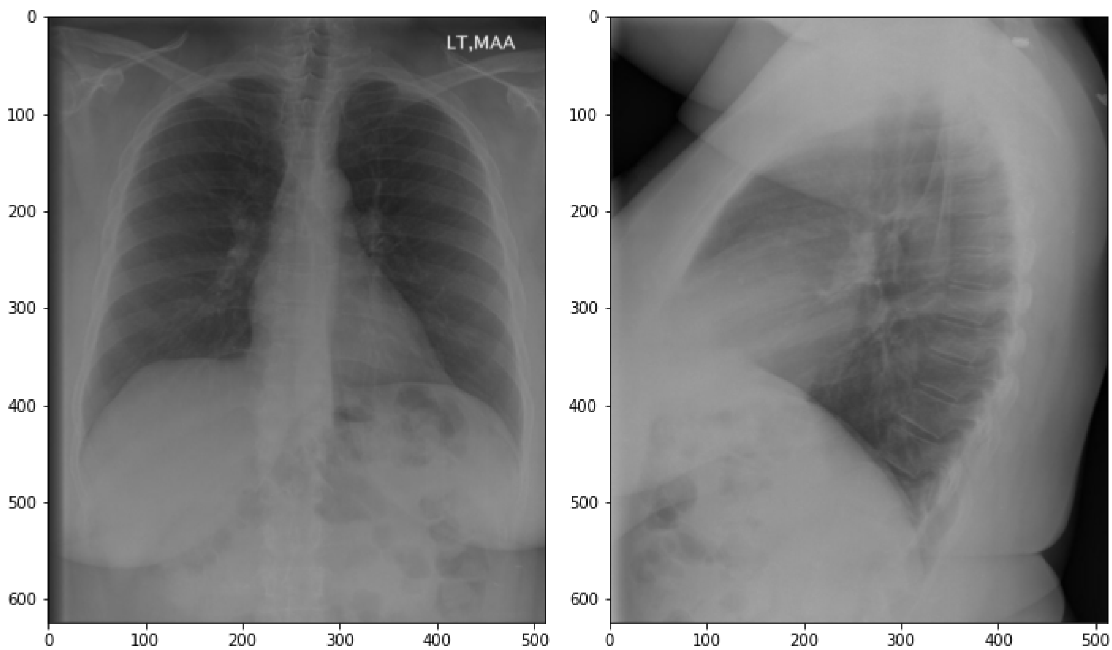
- below will perform the evaluation on both argmax search and beam search

[68]:
```
test_img_cap(in_test[3], out_test[3])
```



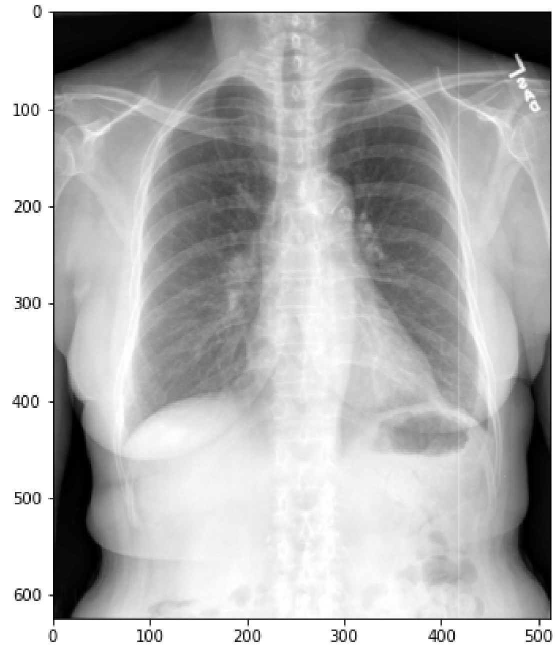

```
==================================================
Actual <start> no acute cardiopulmonary process <end>
Predicted:  no acute cardiopulmonary abnormality <end>
==================================================
Individual 1-gram: 0.7500 Cumulative 1-gram: 0.7500
Individual 2-gram: 0.6667 Cumulative 2-gram: 0.7071
Individual 3-gram: 0.5000 Cumulative 3-gram: 0.6329
Individual 4-gram: 1.0000 Cumulative 4-gram: 0.7071
```

[69]:
```
test_img_cap_beam(in_test[3], out_test[3], 3)
```



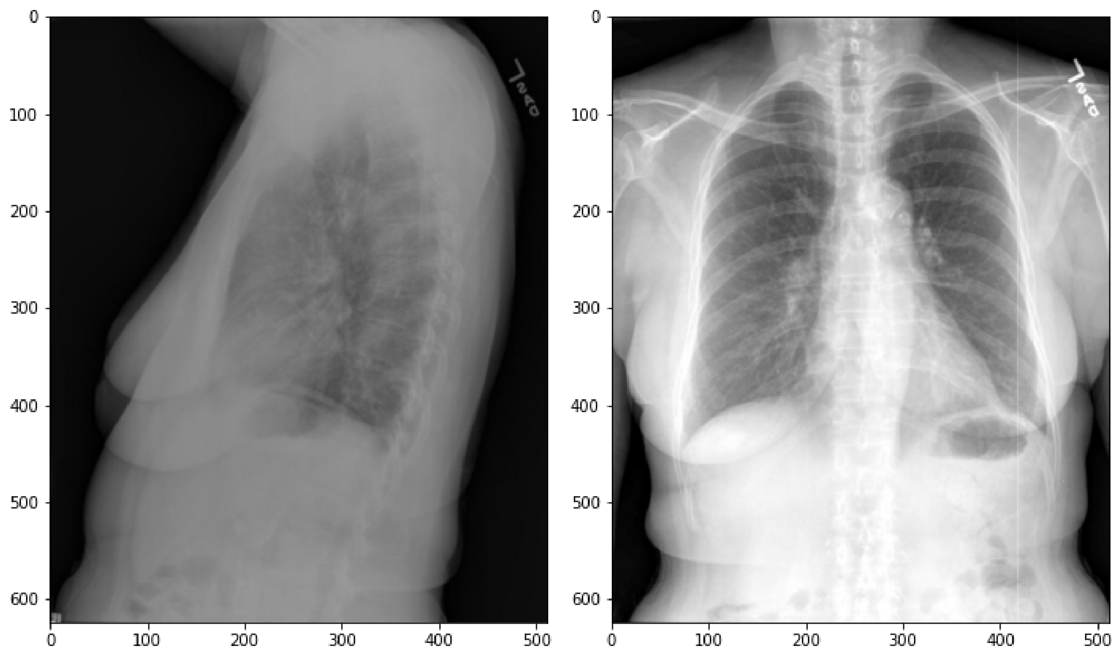

```
Beam Search, index= 3
====================================================
Actual <start> no acute cardiopulmonary process <end>
Predicted: no cardiopulmonary abnormalities
====================================================
Individual 1-gram: 0.4777 Cumulative 1-gram: 0.4777
Individual 2-gram: 0.7165 Cumulative 2-gram: 0.5850
Individual 3-gram: 0.7165 Cumulative 3-gram: 0.6268
Individual 4-gram: 0.7165 Cumulative 4-gram: 0.6475
```

[ ]:
```
test_img_cap(in_test[64], out_test[64])
```



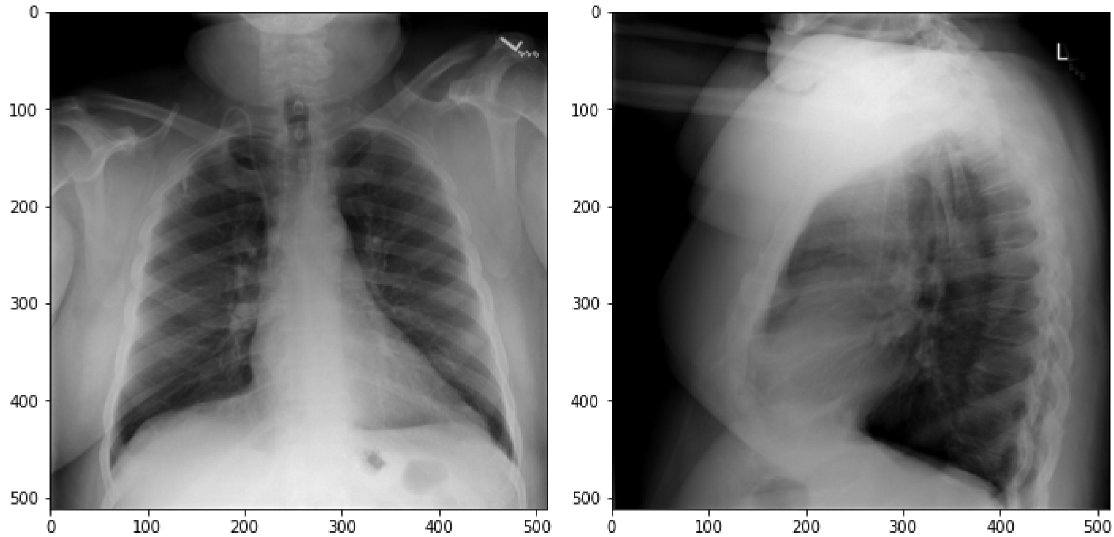

```
==================================================
Actual <start> no acute cardiopulmonary abnormalities <end>
Predicted:  no acute cardiopulmonary abnormality <end>
==================================================
Individual 1-gram: 0.7500 Cumulative 1-gram: 0.7500
Individual 2-gram: 0.6667 Cumulative 2-gram: 0.7071
Individual 3-gram: 0.5000 Cumulative 3-gram: 0.6329
Individual 4-gram: 1.0000 Cumulative 4-gram: 0.7071
```

[70]:
```
test_img_cap_beam(in_test[64], out_test[64], 3)
```



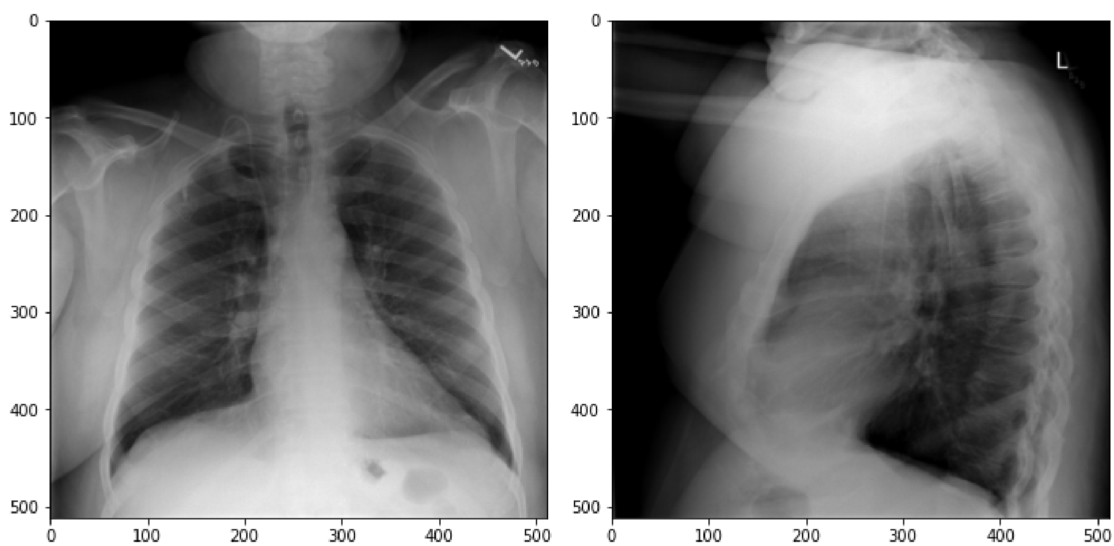

```
Beam Search, index= 3
====================================================
Actual <start> no acute cardiopulmonary abnormalities <end>
Predicted: no acute cardiopulmonary disease
====================================================
Individual 1-gram: 0.7500 Cumulative 1-gram: 0.7500
Individual 2-gram: 0.6667 Cumulative 2-gram: 0.7071
Individual 3-gram: 0.5000 Cumulative 3-gram: 0.6329
Individual 4-gram: 1.0000 Cumulative 4-gram: 0.7071
```

[ ]:
```
test_img_cap(in_test[29], out_test[29])
```



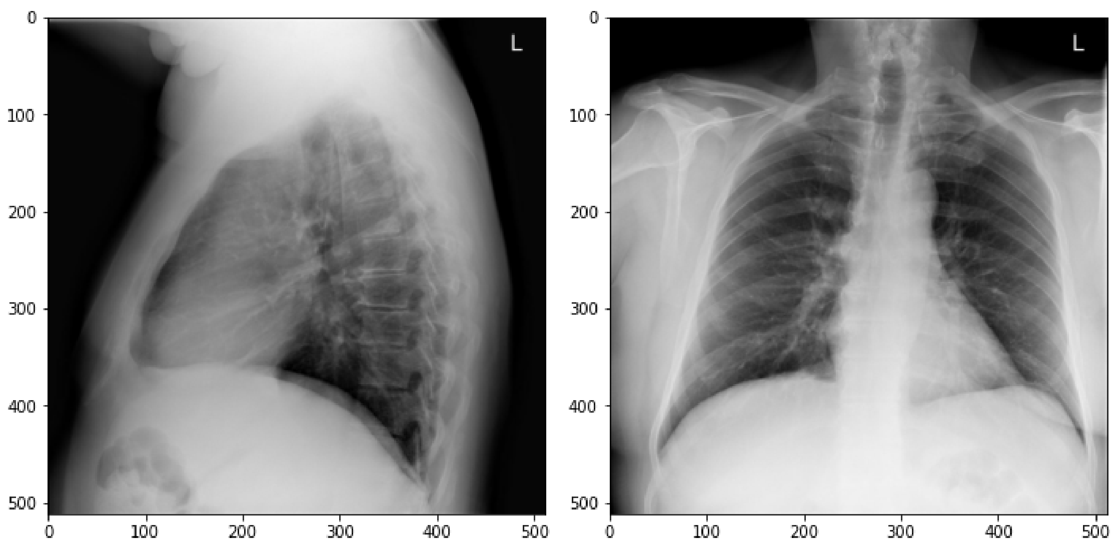
```
====================================================
Actual <start> rightsided chest in without demonstration of an acute
cardiopulmonary abnormality <end>
Predicted:  no evidence of acute cardiopulmonary disease <end>
====================================================
Individual 1-gram: 0.2567 Cumulative 1-gram: 0.2567
Individual 2-gram: 0.1027 Cumulative 2-gram: 0.1624
Individual 3-gram: 0.5134 Cumulative 3-gram: 0.2401
Individual 4-gram: 0.5134 Cumulative 4-gram: 0.2887
```

[81]:
```
test_img_cap_beam(in_test[29], out_test[29], 7)
```



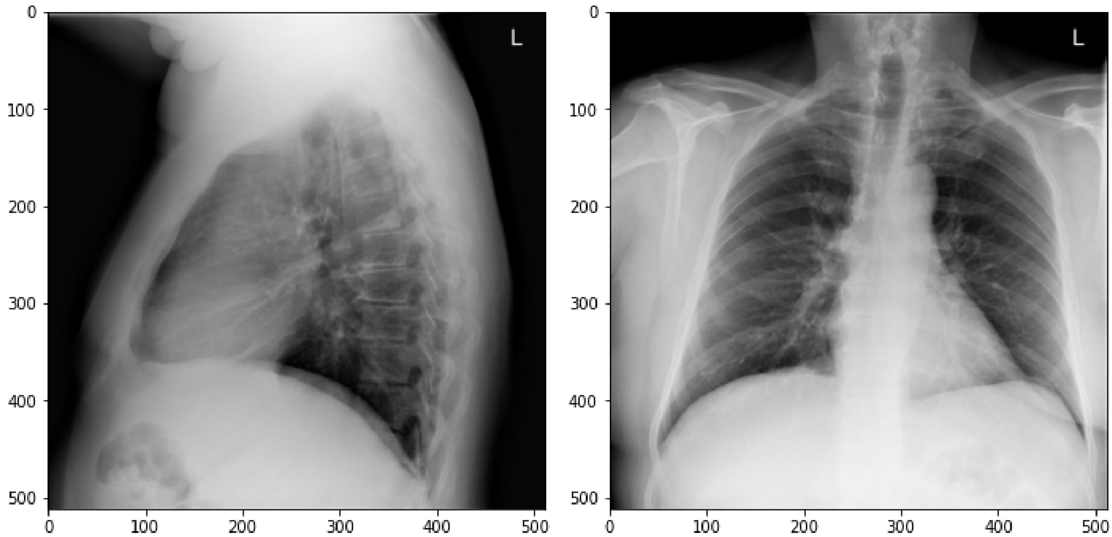
```
Beam Search, index= 7
==================================================
Actual <start> rightsided chest in without demonstration of an acute
cardiopulmonary abnormality <end>
Predicted: no evidence of the same
==================================================
Individual 1-gram: 0.0736 Cumulative 1-gram: 0.0736
Individual 2-gram: 0.3679 Cumulative 2-gram: 0.1645
Individual 3-gram: 0.3679 Cumulative 3-gram: 0.2163
Individual 4-gram: 0.3679 Cumulative 4-gram: 0.2460
```

[ ]:
```
test_img_cap(in_test[229], out_test[229])
```



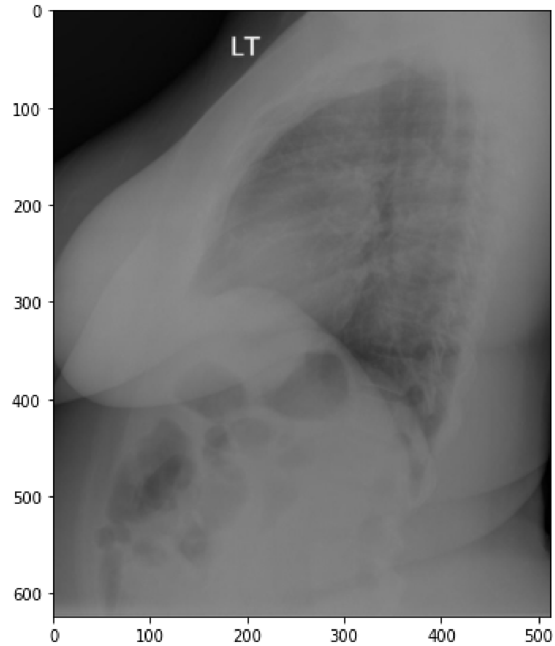
```
==================================================
Actual <start> heart size is normal lungs are clear no nodules or masses no
adenopathy or effusion stable slightly sclerotic posterior inferior of one of
the midthoracic vertebral bodies seen on the lateral radiograph only this most
represents overlying degenerative spurring than metastasis <end>
Predicted:  no acute cardio low lung volumes no pneumothoraces is normal heart
size and normal and clear lungs are grossly within normal limits no acute
cardiopulmonary abnormality identified <end>
==================================================
Individual 1-gram: 0.1985 Cumulative 1-gram: 0.1985
Individual 2-gram: 0.0687 Cumulative 2-gram: 0.1168
Individual 3-gram: 0.5954 Cumulative 3-gram: 0.2032
Individual 4-gram: 0.5954 Cumulative 4-gram: 0.2637
```

[86]:
```
test_img_cap_beam(in_test[229], out_test[229], 3)
```




```
Beam Search, index= 3
====================================================
Actual <start> heart size is normal lungs are clear no nodules or masses no
adenopathy or effusion stable slightly sclerotic posterior inferior of one of
the midthoracic vertebral bodies seen on the lateral radiograph only this most
represents overlying degenerative spurring than metastasis <end>
Predicted: no acute findings
====================================================
Individual 1-gram: 0.0000 Cumulative 1-gram: 0.0000
Individual 2-gram: 0.0000 Cumulative 2-gram: 0.0000
Individual 3-gram: 0.0000 Cumulative 3-gram: 0.0000
Individual 4-gram: 0.0000 Cumulative 4-gram: 0.0000
```

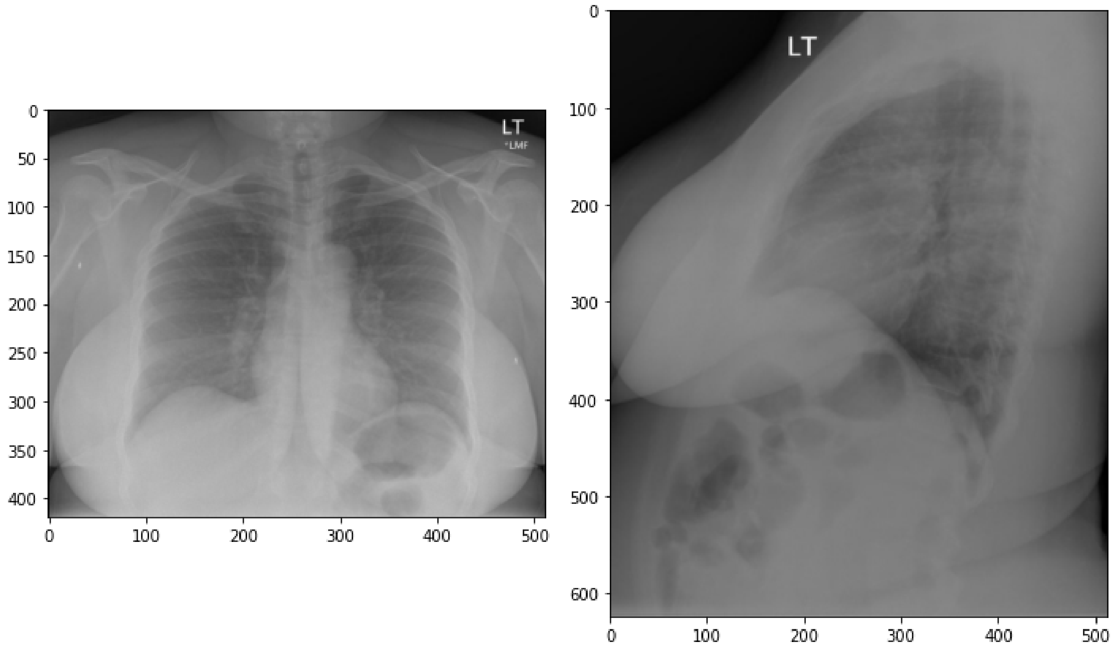
[ ]:
```
test_img_cap(in_test[365], out_test[365])
```

```
=================================================
Actual <start> no acute cardiopulmonary disease <end>
Predicted:  no acute cardiopulmonary disease <end>
=================================================
Individual 1-gram: 1.0000 Cumulative 1-gram: 1.0000
Individual 2-gram: 1.0000 Cumulative 2-gram: 1.0000
Individual 3-gram: 1.0000 Cumulative 3-gram: 1.0000
Individual 4-gram: 1.0000 Cumulative 4-gram: 1.0000
```

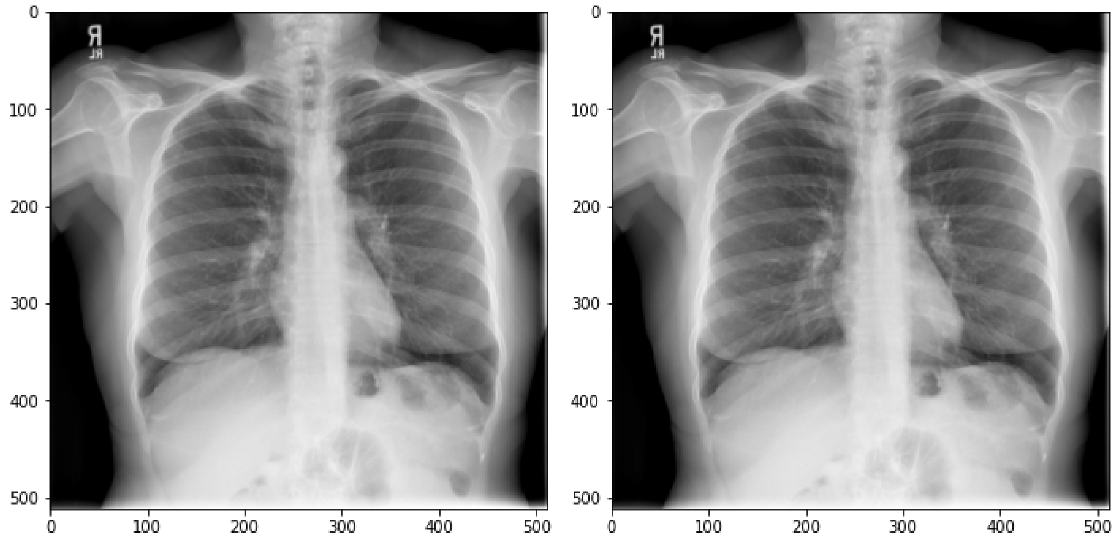
[87]:

```
test_img_cap_beam(in_test[365], out_test[365], 3)
```



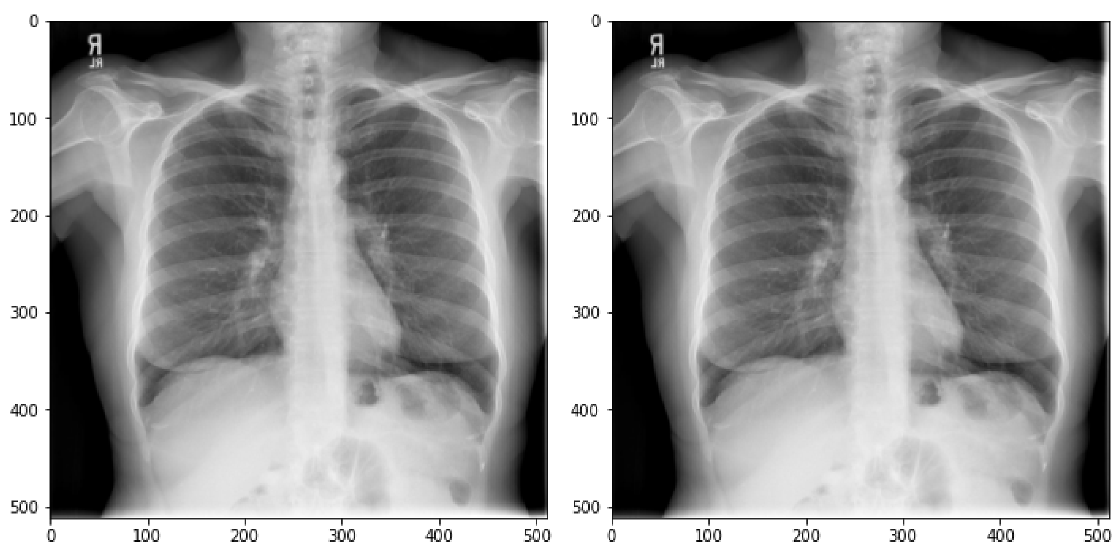

```
Beam Search, index= 3
=====================================================
Actual <start> no acute cardiopulmonary disease <end>
Predicted: no acute abnormalities
=====================================================
Individual 1-gram: 0.4777 Cumulative 1-gram: 0.4777
Individual 2-gram: 0.3583 Cumulative 2-gram: 0.4137
Individual 3-gram: 0.7165 Cumulative 3-gram: 0.4986
Individual 4-gram: 0.7165 Cumulative 4-gram: 0.5444
```

[ ]:
```
test_img_cap(in_test[363], out_test[363])
```



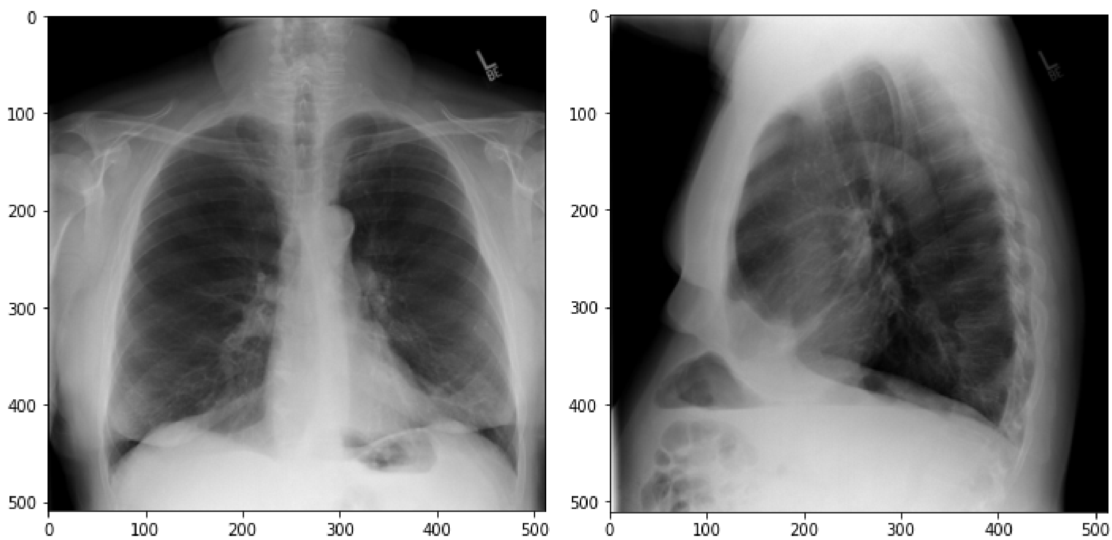
```
======================================================
Actual <start> comparison no suspicious appearing lung nodules identified
wellexpanded and clear lungs mediastinal contour within normal limits no acute
cardiopulmonary abnormality identified <end>
Predicted:  no impression nodules of size normal and posterior inferior
degenerative changes without superimposed pleural based suspected <end>
======================================================
Individual 1-gram: 0.1829 Cumulative 1-gram: 0.1829
Individual 2-gram: 0.7316 Cumulative 2-gram: 0.3658
Individual 3-gram: 0.7316 Cumulative 3-gram: 0.4630
Individual 4-gram: 0.7316 Cumulative 4-gram: 0.5173
```

[89]:
```
test_img_cap_beam(in_test[363], out_test[363], 3)
```



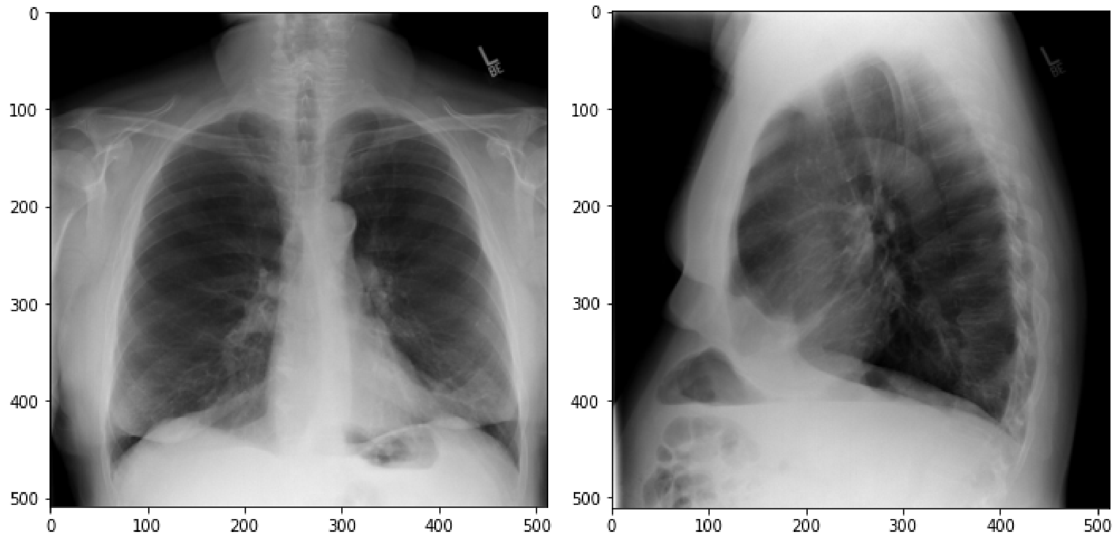
```
Beam Search, index= 3
==================================================
Actual <start> comparison no suspicious appearing lung nodules identified
wellexpanded and clear lungs mediastinal contour within normal limits no acute
cardiopulmonary abnormality identified <end>
Predicted: no evidence for disease
==================================================
Individual 1-gram: 0.0036 Cumulative 1-gram: 0.0036
Individual 2-gram: 0.0143 Cumulative 2-gram: 0.0071
Individual 3-gram: 0.0143 Cumulative 3-gram: 0.0090
Individual 4-gram: 0.0143 Cumulative 4-gram: 0.0101
```

[ ]:
```
test_img_cap(in_test[103], out_test[103])
```



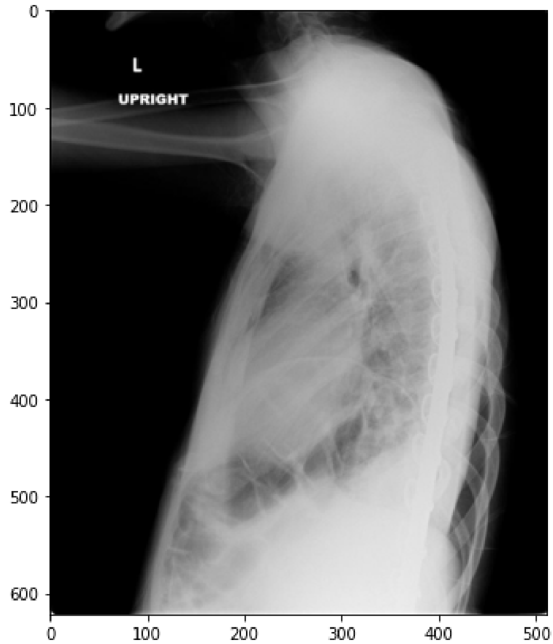
```
==================================================
Actual <start> no focal airspace consolidation hyperexpanded lungs suggestive of
emphysema lingular subsegmental atelectasis or scarring <end>
Predicted:  no acute abnormality noted stable previous and appearance of
atelectasis <end>
==================================================
Individual 1-gram: 0.2011 Cumulative 1-gram: 0.2011
Individual 2-gram: 0.6703 Cumulative 2-gram: 0.3671
Individual 3-gram: 0.6703 Cumulative 3-gram: 0.4505
Individual 4-gram: 0.6703 Cumulative 4-gram: 0.4961
```

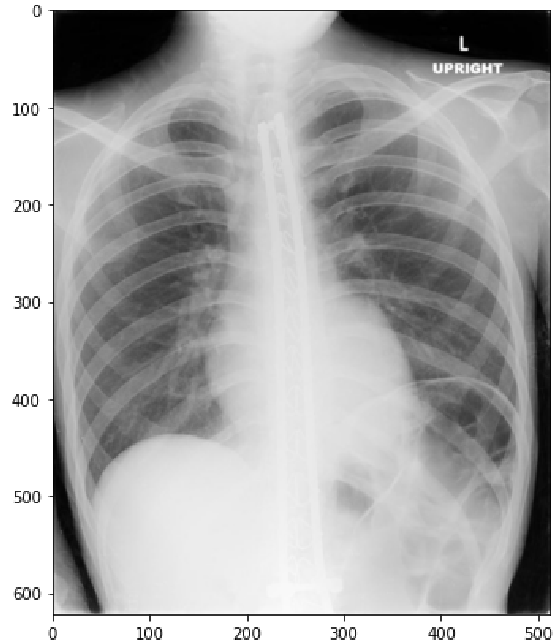
[94]:
```
test_img_cap_beam(in_test[103], out_test[103], 3)
```




```
Beam Search, index= 3
==================================================
Actual <start> no focal airspace consolidation hyperexpanded lungs suggestive of
emphysema lingular subsegmental atelectasis or scarring <end>
Predicted: low lung features negative
==================================================
Individual 1-gram: 0.0000 Cumulative 1-gram: 0.0000
Individual 2-gram: 0.0000 Cumulative 2-gram: 0.0000
Individual 3-gram: 0.0000 Cumulative 3-gram: 0.0000
Individual 4-gram: 0.0000 Cumulative 4-gram: 0.0000
```

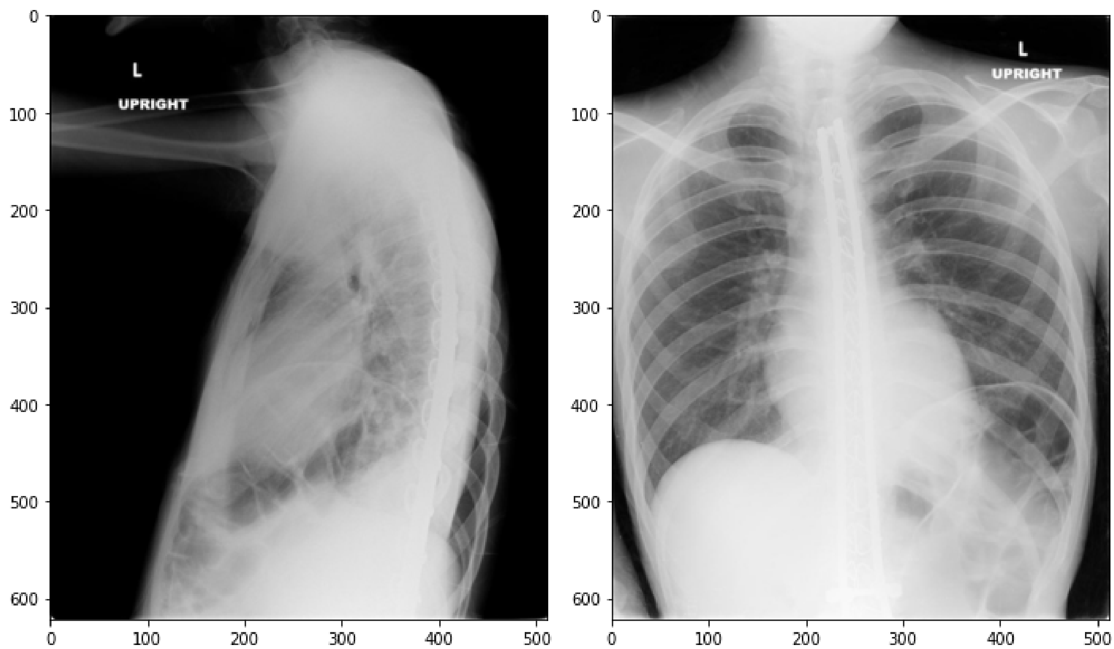
[ ]:
```
test_img_cap(in_test[65], out_test[65])
```





```
==================================================
Actual <start> no active cardiopulmonary disease left humeral head is positioned
anterior and inferior to the glenoid concerning for anterior shoulder
subluxation this is related to the muscular dystrophy and decreased shoulder
muscles support postoperative changes from the spinal placement <end>
Predicted:  no active disease no evidence for contour no degenerative spurring
of prominent head of symptoms from no acute tuberculosis since mediastinal
contour no acute abnormalities since patients symptoms of right volumes and l
this may be artifact of previous exam is recommended no typical findings of
pulmonary edema <end>
==================================================
Individual 1-gram: 0.1875 Cumulative 1-gram: 0.1875
Individual 2-gram: 0.0213 Cumulative 2-gram: 0.0632
Individual 3-gram: 1.0000 Cumulative 3-gram: 0.1615
Individual 4-gram: 1.0000 Cumulative 4-gram: 0.2513
```

[100]:
```
test_img_cap_beam(in_test[65], out_test[65], 3)
```



Beam Search, index= 3
==
Actual <start> no active cardiopulmonary disease left humeral head is positioned anterior and inferior to the glenoid concerning for anterior shoulder subluxation this is related to the muscular dystrophy and decreased shoulder muscles support postoperative changes from the spinal placement <end>
Predicted: no acute cardiopulmonary disease
==
Individual 1-gram: 0.0002 Cumulative 1-gram: 0.0002
Individual 2-gram: 0.0001 Cumulative 2-gram: 0.0001
Individual 3-gram: 0.0002 Cumulative 3-gram: 0.0001
Individual 4-gram: 0.0002 Cumulative 4-gram: 0.0001

5 Conclusion

- The model build on Bidirectional LSTM with attention seems better model than than basic model.
- Initially we had some data errors then with some idea managed to construct the data without any leakage. check construct data paint section for further reference.
- Model Architecture source Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification
- Loss is converged to 0.3 with accuracy of 89% train and 92% validation from the result we can see there is similarity between each predicted and actual output.
- There are some major impression identified in the predicted output if there

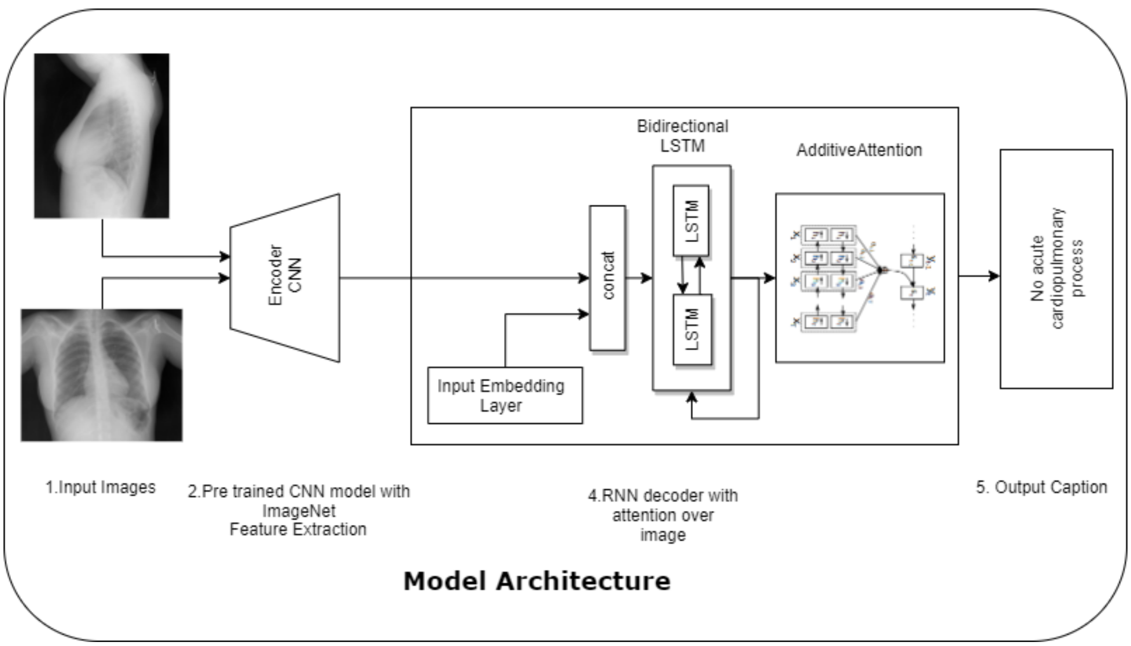
are any major actual impression.
- I have used Inception model due to its size and tensor output when compared with other imageNet trained model.
- Model feature vector is from the pretrained Inception model. saved weights are set while model creation.
- This feature extracted vector size(1,2048) then input to CNN encoder layer. this encoded vector input to BiLSTM layer. Attention layer gives the weights for (1,Unit_size) each word by teacher forcing the output vector is input to decoder layer (BiLSTM layer) the predicted value and the hidden state is calculated and fed back to the decoder cell. Below is the architecture of our model.

5.1 Inferences Analysis:

- As from the result the beam search method is not performing well at longer sentances.
- longer sentences argmax search is performing well. beam sear is giving fault results some time even if we increase the beam width/index.

[ ]:
```
Image(filename='cs.png')
```

[ ]:



6 Future work

- We can also modify the decoder layer with state of the art BERT Transformer instead of attention layer.
- We can further increase the Encoder CNN layer to deep layer for improvements.

- Deep CNN encoder with Decoder Bert transformer will give better result than this.