Import the important modules

In [6]:

```
import pandas as pd # for Dataframe
import numpy as np # for array formation.
import datetime # for datetime.
import string
import random
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.
from tensorflow.keras.layers import Dense, Dropout, Input, Conv2D
```

Let's import the data files

In [ ]:

```
# data_frame=pd.read_csv('/content/data_frame.csv')# preprocessed data after extraction from csv f
iles.
# 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 hav
ing only one image in xml files.
```

oversampling of data_lateral_frontal

In [ ]:

```
# temp=pd.DataFrame(df_lateral_frontal['impression'].value_counts()).reset_index()
# temp=temp.rename(columns={'index':'impression','impression':'counts'})
# upsample_dict={}
# for i,caption in tqdm(enumerate(temp['impression'])):
#   if i == 0:
#     max_count=temp["counts"][i]
#     print(max_count)
#   else:
#     index_of_interest=[]
#
index_of_interest=list(df_lateral_frontal[df_lateral_frontal['impression']==caption].index.values)
#     # print(len(index_of_interest))
#     range_val=max_count-len(index_of_interest)
#     upsampled_list=[]

#     for val in  [random.randint(0,(len(index_of_interest)-1)) for x in range(range_val)]:
#       # print(val)
#       upsampled_list.append(index_of_interest[val])


#     upsample_dict[caption]=upsampled_list


# main_upsample_df=pd.DataFrame([])

# for k in tqdm(upsample_dict):
#   sub_upsample_list=[]
#   for v in upsample_dict[k]:
#     sub_upsample_list.append(df_lateral_frontal.loc[v])
#   sub_upsample_df=pd.DataFrame(sub_upsample_list)
#   main_upsample_df=pd.concat([main_upsample_df,sub_upsample_df])

# df_lateral_frontal_oversample=pd.concat([df_lateral_frontal,main_upsample_df])
# df_lateral_frontal_oversample.to_csv('dd_lateral_frontal_oversample_final.csv')
```

oversampling for duplicate data

In [ ]:

```
# temp=pd.DataFrame(df_duplicate_images['impression'].value_counts()).reset_index()
# temp=temp.rename(columns={'index':'impression','impression':'counts'})
# upsample_dict={}
# for i,caption in tqdm(enumerate(temp['impression'])):
#   if i == 0:
#     max_count=temp["counts"][i]
#     print(max_count)
#   else:
#     index_of_interest=[]
#
index_of_interest=list(df_duplicate_images[df_duplicate_images['impression']==caption].index.values

#     # print(len(index_of_interest))
#     range_val=max_count-len(index_of_interest)
#     upsampled_list=[]

#     for val in  [random.randint(0,(len(index_of_interest)-1)) for x in range(range_val)]:
#       # print(val)
#       upsampled_list.append(index_of_interest[val])


#     upsample_dict[caption]=upsampled_list


# main_upsample_df=pd.DataFrame([])

# for k in tqdm(upsample_dict):
#   sub_upsample_list=[]
#   for v in upsample_dict[k]:
#     sub_upsample_list.append(df_duplicate_images.loc[v])
#   sub_upsample_df=pd.DataFrame(sub_upsample_list)
#   main_upsample_df=pd.concat([main_upsample_df,sub_upsample_df])

# df_dupicate_oversample=pd.concat([df_duplicate_images,main_upsample_df])
# df_dupicate_oversample.to_csv('df_duplicate_oversample_final.csv')
```

In [7]:

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

In [8]:

```
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: (145495, 5)
The shape of df_duplicate_images is: (3544, 5)
```

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

In [9]:

```
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 [10]:

```
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 imag
e.
```

In [11]:

```
df_lateral_frontal['impression']
```

Out[11]:

```
0                              <start> normal chest eam <end>
1                <start> no acute abnormality identified <end>
2              <start> no acute cardiopulmonary findings <end>
3            <start> no acute abnormality no evidence pulmo...
4            <start> no acute cardiopulmonary findings eten...
                                ...
145490       <start> borderline enlargement the cardiac sil...
145491       <start> borderline enlargement the cardiac sil...
145492       <start> borderline enlargement the cardiac sil...
145493       <start> borderline enlargement the cardiac sil...
145494       <start> borderline enlargement the cardiac sil...
Name: impression, Length: 145495, dtype: object
```

In [12]:

```
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 [13]:

```
df_lateral_frontal_imp=df_lateral_frontal[['image_1','image_2','impression']] # just to have the i
mpression 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 [14]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145495 entries, 0 to 145494
Data columns (total 3 columns):
 #   Column      Non-Null Count   Dtype 
---  ------      --------------   ----- 
 0   image_1     145495 non-null  object
 1   image_2     145495 non-null  object
 2   impression  145495 non-null  object
dtypes: object(3)
memory usage: 3.3+ MB
```

In [15]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3544 entries, 0 to 3543
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_1     3544 non-null   object
 1   image_2     3544 non-null   object
 2   impression  3544 non-null   object
dtypes: object(3)
memory usage: 83.2+ KB
```

In [16]:

```
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, 682221.01it/s]
```

Extract the features from image.

In [17]:

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [18]:

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

In [19]:

```
from tensorflow.keras.applications import densenet
chex = densenet.DenseNet121(include_top=False, weights = None, input_shape=(299,299,3))
X = chex.output
X = Dense(14, activation="sigmoid", name="predictions")(X)
model = Model(inputs=chex.input, outputs=X)
model.load_weights('/content/drive/MyDrive/brucechou1983_CheXNet_Keras_0.3.0_weights.h5')
model = Model(inputs = model.input, outputs = model.layers[-2].output)
avg_pooling=tf.keras.layers.GlobalAveragePooling2D(data_format=None)(model.output)
model_for_image_features=Model(inputs=model.input,outputs=avg_pooling)
```

In [20]:

```
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)# pre-process the image data.
  features_of_image = model_for_image_features(tf.constant(i)[None,:]) # apply the VGG16 model.
  # features_of_image = tf.reshape(features_of_image, (-1, features_of_image.shape[1]))# reshape t
he features of image.
  tensor_img.append(features_of_image)# here we append the tensor of image.
```

```
100%|██████████| 7470/7470 [43:40<00:00,  2.85it/s]
```

Split the data for data_frame having lateral and frontal images

In [21]:

```
# fixing numpy RS
```

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

Split the data

In [22]:

```
# 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 [23]:

```
# 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 [24]:

```
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: (104756, 2)
shape of output train is: (104756,)
shape of input validation is: (26189, 2)
shape of output validation is: (26189,)
================================================================================
shape of input test is: (14550, 2)
shape of output test is: (14550,)
```

split the data having duplicate images

In [25]:

```
# 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 [26]:

```
# 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 [27]:

```
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 validatio
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: (2551, 2)
shape of output train is: (2551,)
shape of output train is: ([illegible],)
shape of input validation is: (638, 2)
shape of output validation is: (638,)
================================================================================
shape of input test is: (355, 2)
shape of output test is: (355,)
```

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

In [28]:

```
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 (107307, 2) and train output data is (107307,)
shape of validation input data is (26827, 2) and validation output data is (26827,)
shape of test_input data is (14905, 2) and test_output data is (14905,)
```

for removing biasness we do shuffling in data

In [29]:

```
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.

MAKE THE WORD EMBEDDINGS USING FATSTEXT PRE-TRAINED MODELS

In [30]:

```
from gensim.models import KeyedVectors
model_ft = KeyedVectors.load_word2vec_format('/content/drive/MyDrive/wiki-news-300d-1M.vec',
binary=False)
```

In [31]:

```
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 [32]:

```
# import pickle
# filehandler = open(b"tokenizer.pkl","wb")
# pickle.dump(token,filehandler)
# filehandler.close()
```

In [33]:

```
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: 1445
```

In [34]:

```
MAX_NUM_WORDS=size_of_vocabulary
```

In [35]:

```
EMBEDDING_DIM = 300
print('Preparing embedding matrix.fast text')

# prepare embedding matrix
num_words = min(MAX_NUM_WORDS, len(dictionary) + 1)
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))
for word, i in dictionary.items():
    if i >= MAX_NUM_WORDS:
        continue
    if word in model_ft.vocab:
      embedding_vector = model_ft[word]
      embedding_vector = np.array(embedding_vector)
      if embedding_vector is not None:
          # words not found in embedding index will be all-zeros.
          embedding_matrix[i] = embedding_vector
print(embedding_matrix.shape)
```

```
Preparing embedding matrix.fast text
(1445, 300)
```

In [36]:

```
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[36]:

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

Padding the text sequences

In [37]:

```
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.
```

In [38]:

```
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 [39]:

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

```
 The shape of train_text_output after padding is: (107307, 60)
```

Convert multiple image to tensor and corresponding to their impression

In [40]:

```
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 [41]:

```
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 [42]:

```
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 [43]:

```
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 [44]:

```
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 [45]:

```
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(
[[[6.8125380e-05 1.9806186e-03 1.2980084e-03 ... 5.3322703e-01
   9.0216279e-01 4.3330270e-01]]

 [[0.0000000e+00 1.9169146e-03 1.7418395e-03 ... 4.9886906e-01
   8.4669465e-01 4.0609512e-01]]], shape=(2, 1, 1024), dtype=float32) tf.Tensor(
[  2   4  37  28  45 106  49  52  44  22  91  70 317   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], shape=(60,), dtype=int32)
```

In [46]:

```
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(
[[[5.1232191e-06 2.1424771e-03 3.3893003e-03 ... 4.9685919e-01
   8.6349893e-01 4.0606925e-01]]

 [[2.0604946e-06 1.6301807e-03 1.8653016e-03 ... 4.7350183e-01
   8.7504834e-01 3.8882539e-01]]], shape=(2, 1, 1024), dtype=float32) tf.Tensor(
[  2  20   5   7  16  60 228   7 132   4   6  13  27   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], shape=(60,), dtype=int32)
```

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

In [47]:

```
SIZE_OF_BATCH=32 #Batch size
SIZE_OF_BUFFER= 500  #Batch Buffer
DIMENSION_OF_EMBEDDING= 300 #Embedding
UNITS= 300  #units
```

Shuffle and set data according to batchsize

Train dataset

In [48]:

```
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 [49]:

```
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)
```

Lets make the Encoder and Decoder model

Let's make the class of encoder

In [50]:

```
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, Reshape, Concatenate
class encoder(tf.keras.Model):
  """ encoder for image features extracted by pre-trained model"""
  def __init__(self,DIMENSION_OF_EMBEDDING):
    super(encoder,self).__init__()
    self.flat=tf.keras.layers.Flatten()
    self.dense = tf.keras.layers.Dense(DIMENSION_OF_EMBEDDING, kernel_initializer=tf.keras.initiali
zers.glorot_uniform(seed=45),name='output_layer_of_encoder')# dense layer.

  def call(self, a):
    concatination_enc= Concatenate()([a[:,0], a[:,1]])# concatenate
    a=self.flat(concatination_enc)
    a =self.dense(a)
    return a
```

Decoder for textual attention

In [51]:

```
from keras.initializers import Constant
```

In [52]:

```
class Decoder(tf.keras.Model):
  """ RNN decoder with attention over image features."""
  def __init__(self,  DIMENSION_OF_EMBEDDING, UNITS, size_of_vocabulary):
        super(Decoder, self).__init__()
        self.units = UNITS
        self.concat = tf.keras.layers.Concatenate()
        self.embedding = tf.keras.layers.Embedding(size_of_vocabulary, DIMENSION_OF_EMBEDDING,weigh
ts=[embedding_matrix], input_length=maximum_length_output_sentences, trainable=False)
        self.lstm = tf.keras.layers.LSTM(self.units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer=tf.keras.initializers.glorot_uniform(s
ed=45))
        self.dense = tf.keras.layers.Dense(size_of_vocabulary, kernel_initializer=tf.keras.initiali
zers.glorot_uniform(seed=45))
        self.attention = tf.keras.layers.AdditiveAttention(self.units)
        self.flatten = tf.keras.layers.Flatten()
  def call(self, a):
        attention_outputs = self.attention([a[1],a[2]]) # attention have the encoder features and t
he hidden states
        embedding = self.embedding(a[0])# here we make the embedding of text
        concat_output = self.concat([embedding,tf.expand_dims(attention_outputs,1)])# here we conca
t the embedding and attention output
        output, prev_state_vector,_ = self.lstm(concat_output)# here we pass the concat output
        a= self.flatten(output)
        a= self.dense(a)
        return a,prev_state_vector
```

Loss function

In [53]:

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

In [54]:

```
loss_function = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction='auto')
```

In [55]:

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

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

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

    #applying the mask to loss
    loss_ = loss_*mask

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

Accuracy Function

In [56]:

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

Perplexity function

In [57]:

```
from keras import backend as K
def perplexity(y_true, y_pred):
  """ function calculate the perplexity"""
  return K.pow(2.0, K.mean(K.sparse_categorical_crossentropy(y_true, y_pred)))
```

Directory and log time

In [58]:

```
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 [59]:

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

In [60]:

```
!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 [61]:

```
@tf.function
def training(tensor, target):
  """ here we perform our training"""
  train_loss= 0 # Intialize the loss.
  train_acc= 0 # Initialize the accuracy.
  perplexity_k= 0
  input_dec = tf.expand_dims([token.word_index['<start>']] * target.shape[0], 1)
  hidden_1 =  tf.zeros((target.shape[0], UNITS))
  # hidden_1 = tf.zeros()
  actual,predicted= list(), list()
  with tf.GradientTape() as tape:
    feat= Encoder(tensor) #features
    for i in range(1, target.shape[1]):
      # pred,hidden_1 = Decoder([input_dec,feat, hidden_1])
      pred,hidden_1 =  Decoder([input_dec,feat, hidden_1])
      train_loss+=  function_loss(target[:, i], pred) # validation loss
      train_acc+= Function_Accuracy(target[:,i], pred) # validation accuracy
      perplexity_k += perplexity(target[:,i], pred)
      # here we do the teacher forcing
      input_dec = tf.expand_dims(target[:,i], 1) #input decoder


  train_loss_tot = (train_loss / int(target.shape[1])) # validation loss total
  train_acc_tot = (train_acc / int(target.shape[1]))  # validation accuracy total
  peprlexity_tot =(perplexity_k/ int(target.shape[1]))
  trainable_variables= Encoder.trainable_variables + Decoder.trainable_variables

  gradients = tape.gradient(train_loss, trainable_variables) #gradients
  OPTIMIZER.apply_gradients(zip(gradients, trainable_variables))

  return train_loss, train_loss_tot, train_acc_tot,peprlexity_tot, gradients
```

validation_function

In [62]:

```
@tf.function
def validation(tensor, target):
  """ here we perform our validation"""
  validation_loss= 0 # Intialize the loss.
  validation_acc= 0 # Initialize the accuracy.
  perplexity_k_val=0
  input_dec = tf.expand_dims([token.word_index['<start>']] * target.shape[0], 1)
  hidden_validation =  tf.zeros((target.shape[0], UNITS))
  actual,predicted= list(), list()
  with tf.GradientTape() as tape:
    feat= Encoder(tensor) #features
    for i in range(1, target.shape[1]):
      pred_validation, hidden_validation = Decoder([input_dec,feat, hidden_validation])
      validation_loss+=  function_loss(target[:, i], pred_validation) # validation loss
      validation_acc+= Function_Accuracy(target[:,i], pred_validation) # validation accuracy
      perplexity_k_val +=perplexity(target[:,i], pred_validation)


      input_dec = tf.expand_dims(target[:, i], 1)
  validation_loss_tot = (validation_loss / int(target.shape[1])) # validation loss total
  validation_acc_tot = (validation_acc / int(target.shape[1]))  # validation accuracy total
  validation_perplexity_tot= (perplexity_k_val/int(target.shape[1]))
  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,validation_perplexity_tot, gradie
nts_validation
```

Let's do the training of encoder decoder model

In [63]:

```
tf.keras.backend.clear_session()
epochs_used= 22 # number of epochs
train_loss_p=[] # train loss plot
validation_loss_p= [] # validation loss plot
cnt=0
# 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
  train_tot_perplexity=0
  validation_tot_loss= 0 # validation total loss
  validation_tot_acc= 0 # validation total accuracy
  validation_tot_perplexity=0
  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,peprlexity_tot, gradients = training(tensor_jpeg, target) # do th
e training on the tensor_jpeg, target
    train_tot_loss += tot_loss # total loss
    train_tot_acc += tot_acc # total accuracy
    train_tot_perplexity += peprlexity_tot # tot perplexity

    if batch % 1000 == 0:
      print('The Epoch is {} batch is{} loss is {:.4f} , the accuracy is {:.4f} and the perplexity
is {:.4f}'.format(ep+1, batch, batch_loss / int(target.shape[1]), tot_acc,peprlexity_tot))
  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) #summar
y of train loss
    tf.summary.scalar('accuracy', train_tot_acc/int(len(train_input)// SIZE_OF_BATCH), step=ep) #su
mmary of train accuracy.
    tf.summary.scalar('perplexity', train_tot_perplexity/int(len(train_input)// SIZE_OF_BATCH), ste
p=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,validation_peprlexity_tot,gradients_validation = validation(tensor_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
    validation_tot_perplexity += validation_peprlexity_tot # validation total bleu

    if batch % 1000 == 0:
      print('The Epoch is {} batch is{} loss is {:.4f} and the accuracy is {:.4f} and the
perplexity is {:.4f}'.format(ep+1, batch, validation_batch_loss / int(target.shape[1]),
validation_t_acc,validation_peprlexity_tot))
  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.
      tf.summary.scalar('perplexity', validation_tot_perplexity/int(len(validation_input)// SIZE_OF
_BATCH), step=ep)

  print(' The epoch is {}, loss is {:.4f}, Accuracy is: {:.4f}, perplexity is {:.4f}, test loss: {:
.4f} and test accuracy: {:.4f} and test perplexity: {:.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,
               train_tot_perplexity/int(len(train_input)//SIZE_OF_BATCH),
               validation_tot_loss/ int(len(validation_input)// SIZE_OF_BATCH),
               (validation_tot_acc/ int(len(validation_input)//SIZE_OF_BATCH))*100,
               validation_tot_perplexity/int(len(validation_input)//SIZE_OF_BATCH)))

  if ep >6:
    if cnt >=4:
      print('Early stopping|| stop the training')
      break
    else:
      if validation_loss_p[ep-1]<validation_loss_p[ep]:
        print(' count increased')
        cnt+=1
```

```
Intialize the epoch1
The training loss batchwise
The Epoch is 1 batch is0 loss is 2.0992 , the accuracy is 0.0012 and the perplexity is 3632.7617
The Epoch is 1 batch is1000 loss is 0.5485 , the accuracy is 0.7717 and the perplexity is 32.7393
The Epoch is 1 batch is2000 loss is 0.5191 , the accuracy is 0.7853 and the perplexity is 32.3165
The Epoch is 1 batch is3000 loss is 0.4737 , the accuracy is 0.7924 and the perplexity is 34.2910
The validation loss batchwise
The Epoch is 1 batch is0 loss is 0.5634 and the accuracy is 0.7945 and the perplexity is 34.4560
 The epoch is 1, loss is 0.5850, Accuracy is: 77.4571, perplexity is 36.8645, test loss: 0.4501 an
d test accuracy: 79.7214 and test perplexity: 33.5977
Intialize the epoch2
The training loss batchwise
The Epoch is 2 batch is0 loss is 0.5253 , the accuracy is 0.7978 and the perplexity is 32.8107
The Epoch is 2 batch is1000 loss is 0.3886 , the accuracy is 0.8013 and the perplexity is 31.5130
The Epoch is 2 batch is2000 loss is 0.3577 , the accuracy is 0.8046 and the perplexity is 31.2775
The Epoch is 2 batch is3000 loss is 0.3839 , the accuracy is 0.8076 and the perplexity is 31.4733
The validation loss batchwise
The Epoch is 2 batch is0 loss is 0.3211 and the accuracy is 0.8086 and the perplexity is 29.7276
 The epoch is 2, loss is 0.4006, Accuracy is: 80.3657, perplexity is 31.8365, test loss: 0.3722 an
d test accuracy: 81.0573 and test perplexity: 30.4027
Intialize the epoch3
The training loss batchwise
The Epoch is 3 batch is0 loss is 0.3809 , the accuracy is 0.8105 and the perplexity is 30.7582
The Epoch is 3 batch is1000 loss is 0.3560 , the accuracy is 0.8128 and the perplexity is 29.2971
The Epoch is 3 batch is2000 loss is 0.3702 , the accuracy is 0.8150 and the perplexity is 29.7205
The Epoch is 3 batch is3000 loss is 0.3291 , the accuracy is 0.8172 and the perplexity is 28.6661
The validation loss batchwise
The Epoch is 3 batch is0 loss is 0.3502 and the accuracy is 0.8180 and the perplexity is 28.1952
 The epoch is 3, loss is 0.3489, Accuracy is: 81.4507, perplexity is 29.1983, test loss: 0.3377 an
d test accuracy: 81.9824 and test perplexity: 28.3550
Intialize the epoch4
The training loss batchwise
The Epoch is 4 batch is0 loss is 0.2948 , the accuracy is 0.8196 and the perplexity is 27.9708
The Epoch is 4 batch is1000 loss is 0.3165 , the accuracy is 0.8215 and the perplexity is 27.2521
The Epoch is 4 batch is2000 loss is 0.2923 , the accuracy is 0.8233 and the perplexity is 25.8386
The Epoch is 4 batch is3000 loss is 0.3509 , the accuracy is 0.8249 and the perplexity is 25.9906
The validation loss batchwise
The Epoch is 4 batch is0 loss is 0.3350 and the accuracy is 0.8255 and the perplexity is 25.7989
 The epoch is 4, loss is 0.3255, Accuracy is: 82.2911, perplexity is 26.7496, test loss: 0.3225 an
d test accuracy: 82.7100 and test perplexity: 25.7720
Intialize the epoch5
The training loss batchwise
The Epoch is 5 batch is0 loss is 0.2645 , the accuracy is 0.8267 and the perplexity is 25.2798
The Epoch is 5 batch is1000 loss is 0.3104 , the accuracy is 0.8281 and the perplexity is 25.7031
The Epoch is 5 batch is2000 loss is 0.3477 , the accuracy is 0.8294 and the perplexity is 25.5814
The Epoch is 5 batch is3000 loss is 0.2843 , the accuracy is 0.8306 and the perplexity is 24.7429
The validation loss batchwise
The Epoch is 5 batch is0 loss is 0.2824 and the accuracy is 0.8311 and the perplexity is 24.2386
 The epoch is 5, loss is 0.3146, Accuracy is: 82.9182, perplexity is 25.2269, test loss: 0.3136 an
d test accuracy: 83.2533 and test perplexity: 24.4523
Intialize the epoch6
The training loss batchwise
The Epoch is 6 batch is0 loss is 0.3557 , the accuracy is 0.8320 and the perplexity is 24.5781
The Epoch is 6 batch is1000 loss is 0.3362 , the accuracy is 0.8332 and the perplexity is 24.6125
The Epoch is 6 batch is2000 loss is 0.3276 , the accuracy is 0.8344 and the perplexity is 24.3273
The Epoch is 6 batch is3000 loss is 0.3377 , the accuracy is 0.8356 and the perplexity is 24.5271
The validation loss batchwise
The Epoch is 6 batch is0 loss is 0.2972 and the accuracy is 0.8360 and the perplexity is 23.8877
 The epoch is 6, loss is 0.3083, Accuracy is: 83.4273, perplexity is 24.3101, test loss: 0.3097 an
d test accuracy: 83.7382 and test perplexity: 23.9563
Intialize the epoch7
The training loss batchwise
The Epoch is 7 batch is0 loss is 0.2707 , the accuracy is 0.8368 and the perplexity is 23.7544
The Epoch is 7 batch is1000 loss is 0.3265 , the accuracy is 0.8378 and the perplexity is 23.4805
The Epoch is 7 batch is2000 loss is 0.3588 , the accuracy is 0.8387 and the perplexity is 23.7356
The Epoch is 7 batch is3000 loss is 0.2821 , the accuracy is 0.8396 and the perplexity is 23.3686
The validation loss batchwise
The Epoch is 7 batch is0 loss is 0.3184 and the accuracy is 0.8399 and the perplexity is 23.1220
 The epoch is 7, loss is 0.3044, Accuracy is: 83.8616, perplexity is 23.6334, test loss: 0.3059 an
d test accuracy: 84.1178 and test perplexity: 23.1910
Intialize the epoch8
The training loss batchwise
The Epoch is 8 batch is0 loss is 0.2858 , the accuracy is 0.8405 and the perplexity is 23.0375
The Epoch is 8 batch is1000 loss is 0.2575 , the accuracy is 0.8412 and the perplexity is 22.9144
The Epoch is 8 batch is2000 loss is 0.3183 , the accuracy is 0.8420 and the perplexity is 22.9442
The Epoch is 8 batch is3000 loss is 0.3129 , the accuracy is 0.8427 and the perplexity is 22.9437
The validation loss batchwise
The Epoch is 8 batch is0 loss is 0.2815 and the accuracy is 0.8429 and the perplexity is 22.6969
 The epoch is 8, loss is 0.3013, Accuracy is: 84.1977, perplexity is 23.0683, test loss: 0.3032 an
d test accuracy: 84.4182 and test perplexity: 22.7419
Intialize the epoch9
The training loss batchwise
The Epoch is 9 batch is0 loss is 0.3201 , the accuracy is 0.8434 and the perplexity is 22.6587
The Epoch is 9 batch is1000 loss is 0.3003 , the accuracy is 0.8440 and the perplexity is 22.5721
The Epoch is 9 batch is2000 loss is 0.3393 , the accuracy is 0.8446 and the perplexity is 22.9322
The Epoch is 9 batch is3000 loss is 0.2814 , the accuracy is 0.8452 and the perplexity is 22.3923
The validation loss batchwise
The Epoch is 9 batch is0 loss is 0.2905 and the accuracy is 0.8454 and the perplexity is 22.2227
 The epoch is 9, loss is 0.2989, Accuracy is: 84.4692, perplexity is 22.6324, test loss: 0.3012 an
d test accuracy: 84.6657 and test perplexity: 22.3742
Intialize the epoch10
The training loss batchwise
The Epoch is 10 batch is0 loss is 0.3037 , the accuracy is 0.8459 and the perplexity is 22.0443
The Epoch is 10 batch is1000 loss is 0.3012 , the accuracy is 0.8464 and the perplexity is 22.3000
The Epoch is 10 batch is2000 loss is 0.3201 , the accuracy is 0.8469 and the perplexity is 22.2500
The Epoch is 10 batch is3000 loss is 0.2729 , the accuracy is 0.8474 and the perplexity is 22.3626
The validation loss batchwise
The Epoch is 10 batch is0 loss is 0.3021 and the accuracy is 0.8476 and the perplexity is 22.1599
 The epoch is 10, loss is 0.2969, Accuracy is: 84.6973, perplexity is 22.2474, test loss: 0.2993 a
nd test accuracy: 84.8792 and test perplexity: 22.1055
Intialize the epoch11
The training loss batchwise
The Epoch is 11 batch is0 loss is 0.2983 , the accuracy is 0.8480 and the perplexity is 22.4357
The Epoch is 11 batch is1000 loss is 0.2802 , the accuracy is 0.8486 and the perplexity is 21.7320
The Epoch is 11 batch is2000 loss is 0.2607 , the accuracy is 0.8492 and the perplexity is 21.6615
The Epoch is 11 batch is3000 loss is 0.3439 , the accuracy is 0.8497 and the perplexity is 22.0694
The validation loss batchwise
The Epoch is 11 batch is0 loss is 0.2491 and the accuracy is 0.8499 and the perplexity is 21.5087
 The epoch is 11, loss is 0.2949, Accuracy is: 84.9238, perplexity is 21.9046, test loss: 0.2972 a
nd test accuracy: 85.1149 and test perplexity: 21.9892
Intialize the epoch12
The training loss batchwise
The Epoch is 12 batch is0 loss is 0.2652 , the accuracy is 0.8503 and the perplexity is 21.6986
The Epoch is 12 batch is1000 loss is 0.2665 , the accuracy is 0.8509 and the perplexity is 21.3579
The Epoch is 12 batch is2000 loss is 0.3287 , the accuracy is 0.8514 and the perplexity is 21.7793
The Epoch is 12 batch is3000 loss is 0.2945 , the accuracy is 0.8518 and the perplexity is 22.0002
The validation loss batchwise
The Epoch is 12 batch is0 loss is 0.2708 and the accuracy is 0.8520 and the perplexity is 21.5287
 The epoch is 12, loss is 0.2933, Accuracy is: 85.1443, perplexity is 21.6013, test loss: 0.2953 a
nd test accuracy: 85.3206 and test perplexity: 21.6660
Intialize the epoch13
The training loss batchwise
The Epoch is 13 batch is0 loss is 0.2920 , the accuracy is 0.8524 and the perplexity is 21.7621
The Epoch is 13 batch is1000 loss is 0.2916 , the accuracy is 0.8528 and the perplexity is 20.9774
The Epoch is 13 batch is2000 loss is 0.2854 , the accuracy is 0.8533 and the perplexity is 21.2689
The Epoch is 13 batch is3000 loss is 0.3208 , the accuracy is 0.8537 and the perplexity is 21.3500
The validation loss batchwise
The Epoch is 13 batch is0 loss is 0.3027 and the accuracy is 0.8538 and the perplexity is 21.2302
 The epoch is 13, loss is 0.2917, Accuracy is: 85.3363, perplexity is 21.3751, test loss: 0.2940 a
nd test accuracy: 85.5008 and test perplexity: 21.2201
Intialize the epoch14
The training loss batchwise
The Epoch is 14 batch is0 loss is 0.3154 , the accuracy is 0.8542 and the perplexity is 21.2067
The Epoch is 14 batch is1000 loss is 0.2862 , the accuracy is 0.8546 and the perplexity is 21.2682
The Epoch is 14 batch is2000 loss is 0.2866 , the accuracy is 0.8550 and the perplexity is 21.4608
The Epoch is 14 batch is3000 loss is 0.2850 , the accuracy is 0.8553 and the perplexity is 20.8611
The validation loss batchwise
The Epoch is 14 batch is0 loss is 0.3287 and the accuracy is 0.8555 and the perplexity is 20.9835
 The epoch is 14, loss is 0.2901, Accuracy is: 85.5080, perplexity is 21.1786, test loss: 0.2920 a
nd test accuracy: 85.6651 and test perplexity: 21.0696
Intialize the epoch15
The training loss batchwise
The Epoch is 15 batch is0 loss is 0.2497 , the accuracy is 0.8558 and the perplexity is 20.7272
The Epoch is 15 batch is1000 loss is 0.2629 , the accuracy is 0.8561 and the perplexity is 20.5994
The Epoch is 15 batch is2000 loss is 0.2687 , the accuracy is 0.8565 and the perplexity is 21.0932
The Epoch is 15 batch is3000 loss is 0.2828 , the accuracy is 0.8568 and the perplexity is 20.8638
The validation loss batchwise
The Epoch is 15 batch is0 loss is 0.2730 and the accuracy is 0.8570 and the perplexity is 20.7677
 The epoch is 15, loss is 0.2886, Accuracy is: 85.6630, perplexity is 20.9169, test loss: 0.2906 a
nd test accuracy: 85.8120 and test perplexity: 20.8781
Intialize the epoch16
The training loss batchwise
The Epoch is 16 batch is0 loss is 0.2642 , the accuracy is 0.8572 and the perplexity is 20.6047
The Epoch is 16 batch is1000 loss is 0.3269 , the accuracy is 0.8576 and the perplexity is 21.0831
The Epoch is 16 batch is2000 loss is 0.2632 , the accuracy is 0.8579 and the perplexity is 20.4535
The Epoch is 16 batch is3000 loss is 0.3026 , the accuracy is 0.8582 and the perplexity is 20.7287
The validation loss batchwise
The Epoch is 16 batch is0 loss is 0.3189 and the accuracy is 0.8583 and the perplexity is 20.8495
 The epoch is 16, loss is 0.2871, Accuracy is: 85.8035, perplexity is 20.7305, test loss: 0.2894 a
nd test accuracy: 85.9468 and test perplexity: 20.6588
Intialize the epoch17
The training loss batchwise
The Epoch is 17 batch is0 loss is 0.2687 , the accuracy is 0.8586 and the perplexity is 20.9584
The Epoch is 17 batch is1000 loss is 0.2753 , the accuracy is 0.8589 and the perplexity is 20.3874
The Epoch is 17 batch is2000 loss is 0.2938 , the accuracy is 0.8592 and the perplexity is 21.0728
The Epoch is 17 batch is3000 loss is 0.2555 , the accuracy is 0.8595 and the perplexity is 20.3098
The validation loss batchwise
The Epoch is 17 batch is0 loss is 0.2828 and the accuracy is 0.8596 and the perplexity is 20.5501
 The epoch is 17, loss is 0.2857, Accuracy is: 85.9335, perplexity is 20.5261, test loss: 0.2876 a
nd test accuracy: 86.0716 and test perplexity: 20.4068
Intialize the epoch18
The training loss batchwise
The Epoch is 18 batch is0 loss is 0.3059 , the accuracy is 0.8598 and the perplexity is 20.4815
The Epoch is 18 batch is1000 loss is 0.3294 , the accuracy is 0.8601 and the perplexity is 20.3566
The Epoch is 18 batch is2000 loss is 0.2835 , the accuracy is 0.8604 and the perplexity is 20.1704
The Epoch is 18 batch is3000 loss is 0.2934 , the accuracy is 0.8606 and the perplexity is 20.3995
The validation loss batchwise
The Epoch is 18 batch is0 loss is 0.2525 and the accuracy is 0.8607 and the perplexity is 19.9212
 The epoch is 18, loss is 0.2843, Accuracy is: 86.0531, perplexity is 20.3400, test loss: 0.2866 a
nd test accuracy: 86.1870 and test perplexity: 20.2075
Intialize the epoch19
The training loss batchwise
The Epoch is 19 batch is0 loss is 0.2630 , the accuracy is 0.8609 and the perplexity is 20.1326
The Epoch is 19 batch is1000 loss is 0.2373 , the accuracy is 0.8612 and the perplexity is 20.1715
The Epoch is 19 batch is2000 loss is 0.2688 , the accuracy is 0.8615 and the perplexity is 20.3789
The Epoch is 19 batch is3000 loss is 0.2882 , the accuracy is 0.8617 and the perplexity is 20.2264
The validation loss batchwise
The Epoch is 19 batch is0 loss is 0.3178 and the accuracy is 0.8618 and the perplexity is 20.1494
 The epoch is 19, loss is 0.2831, Accuracy is: 86.1642, perplexity is 20.2178, test loss: 0.2856 a
nd test accuracy: 86.2952 and test perplexity: 20.0149
Intialize the epoch20
The training loss batchwise
The Epoch is 20 batch is0 loss is 0.3658 , the accuracy is 0.8620 and the perplexity is 20.2680
The Epoch is 20 batch is1000 loss is 0.2535 , the accuracy is 0.8623 and the perplexity is 19.9050
The Epoch is 20 batch is2000 loss is 0.2940 , the accuracy is 0.8626 and the perplexity is 20.2890
The Epoch is 20 batch is3000 loss is 0.2445 , the accuracy is 0.8629 and the perplexity is 19.7482
The validation loss batchwise
The Epoch is 20 batch is0 loss is 0.3357 and the accuracy is 0.8630 and the perplexity is 20.1726
 The epoch is 20, loss is 0.2820, Accuracy is: 86.2735, perplexity is 20.0411, test loss: 0.2840 a
nd test accuracy: 86.4144 and test perplexity: 20.0261
Intialize the epoch21
The training loss batchwise
The Epoch is 21 batch is0 loss is 0.2569 , the accuracy is 0.8632 and the perplexity is 19.7089
The Epoch is 21 batch is1000 loss is 0.2575 , the accuracy is 0.8636 and the perplexity is 20.0163
The Epoch is 21 batch is2000 loss is 0.2404 , the accuracy is 0.8639 and the perplexity is 19.7817
The Epoch is 21 batch is3000 loss is 0.2875 , the accuracy is 0.8642 and the perplexity is 20.0393
The validation loss batchwise
The Epoch is 21 batch is0 loss is 0.2691 and the accuracy is 0.8643 and the perplexity is 19.8708
 The epoch is 21, loss is 0.2807, Accuracy is: 86.4012, perplexity is 19.9707, test loss: 0.2833 a
nd test accuracy: 86.5409 and test perplexity: 19.8965
Intialize the epoch22
The training loss batchwise
The Epoch is 22 batch is0 loss is 0.2582 , the accuracy is 0.8645 and the perplexity is 19.6539
The Epoch is 22 batch is1000 loss is 0.2260 , the accuracy is 0.8648 and the perplexity is 19.8079
The Epoch is 22 batch is2000 loss is 0.3055 , the accuracy is 0.8651 and the perplexity is 19.7222
The Epoch is 22 batch is3000 loss is 0.2334 , the accuracy is 0.8653 and the perplexity is 19.3351
The validation loss batchwise
The Epoch is 22 batch is0 loss is 0.2860 and the accuracy is 0.8654 and the perplexity is 19.9280
 The epoch is 22, loss is 0.2796, Accuracy is: 86.5228, perplexity is 19.8131, test loss: 0.2821 a
nd test accuracy: 86.6579 and test perplexity: 19.8330
```

In [ ]:

```
Decoder.save_weights('abhi_krishna_oversample_weights_decoder_final_final.h5')
```

In [ ]:

```
Encoder.save_weights('abhi_krishna_oversample_weights_encoder_final_final.h5')
```

In [64]:

```
%reload_ext tensorboard
```

Tensorboard

In [65]:

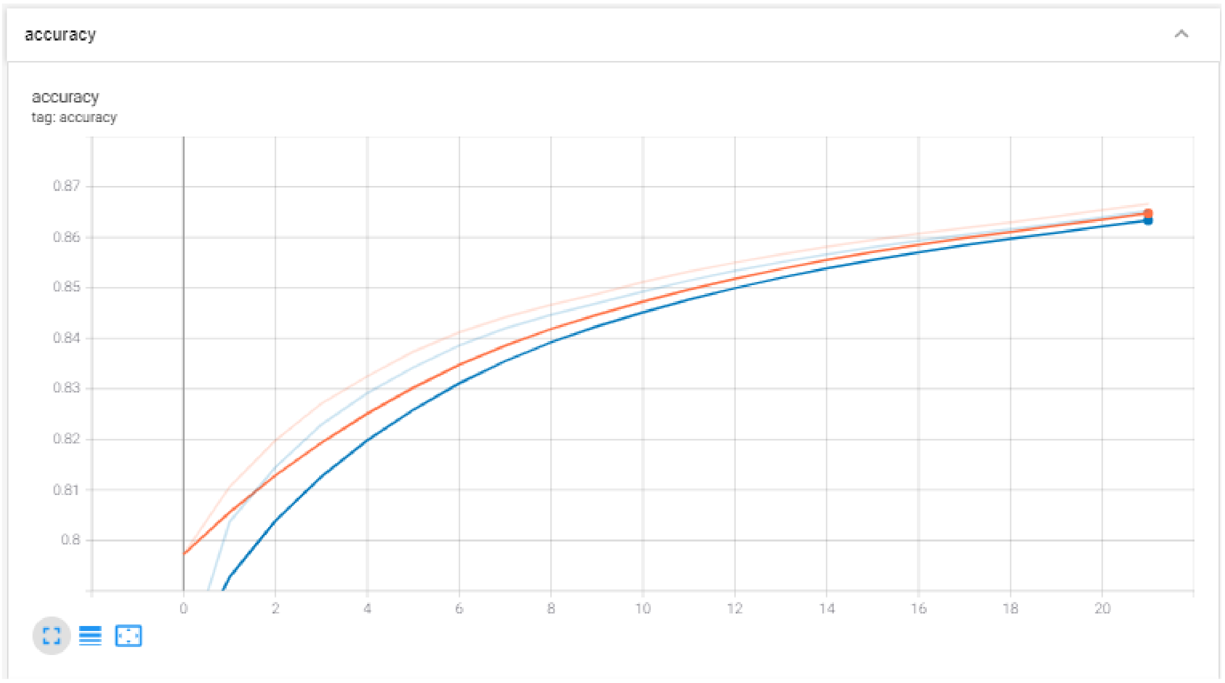
```
tensorboard --logdir=/content/content/gradient_tape/20201230-140659
```

Accuracy plot

In [66]:

```
from IPython.display import Image
Image(filename='/content/accuracy.png')
```

Out[66]:

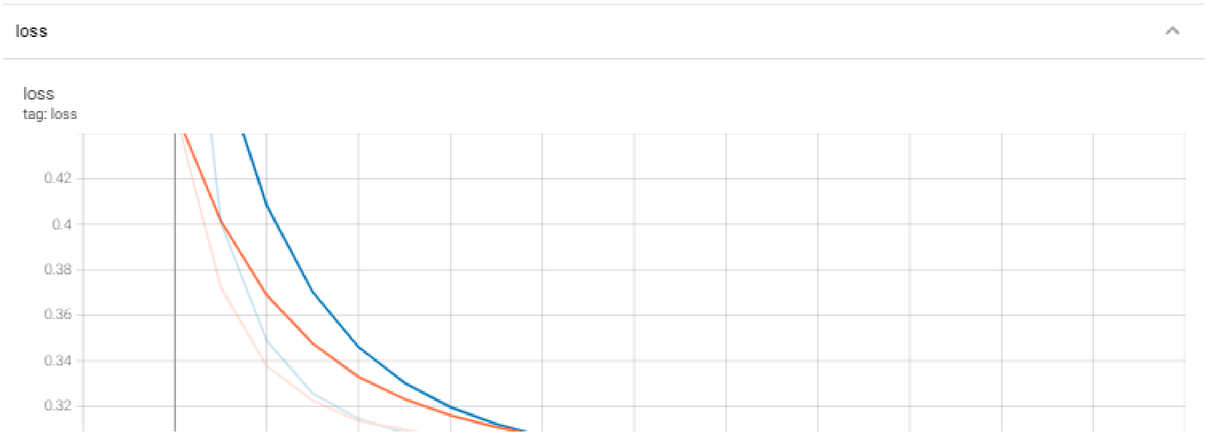


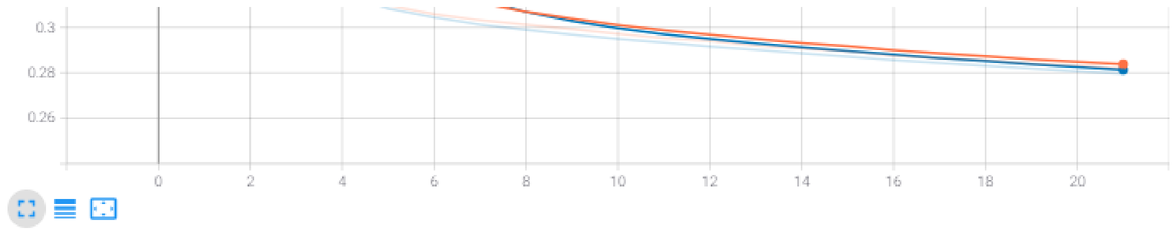
Loss plot

In [67]:

```
from IPython.display import Image
Image(filename='/content/loss.png')
```

Out[67]:



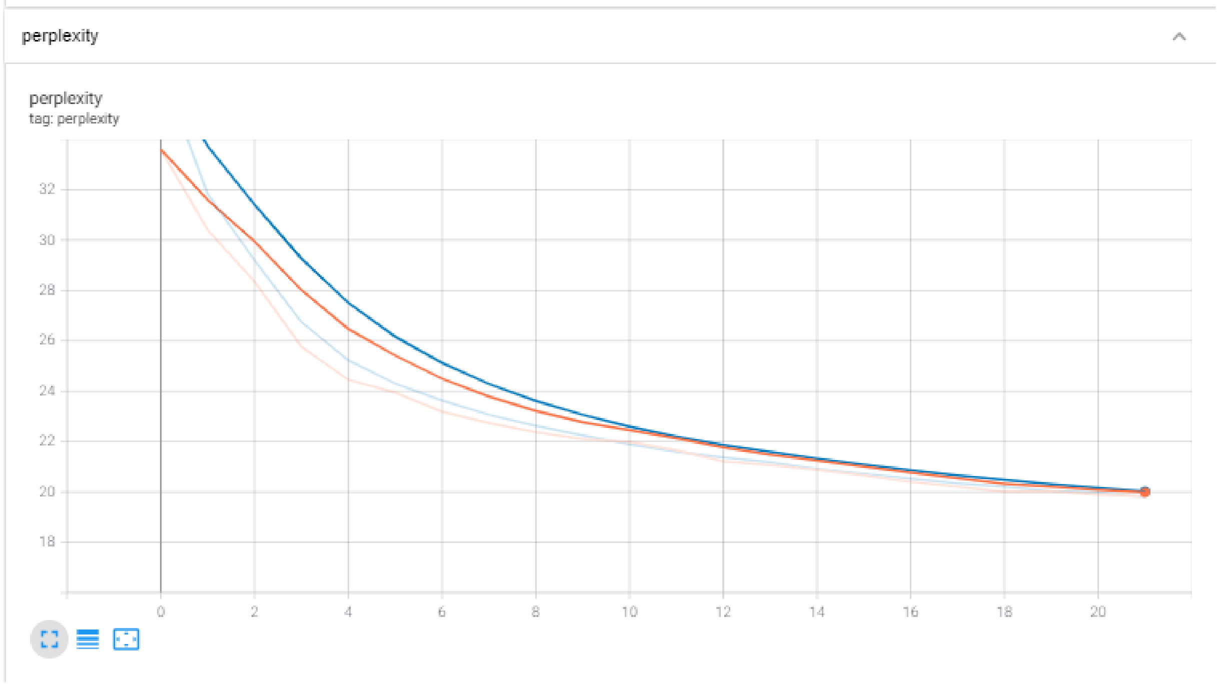


perplexity plot

In [68]:

```
from IPython.display import Image
Image(filename='/content/perplexity.png')
```

Out[68]:

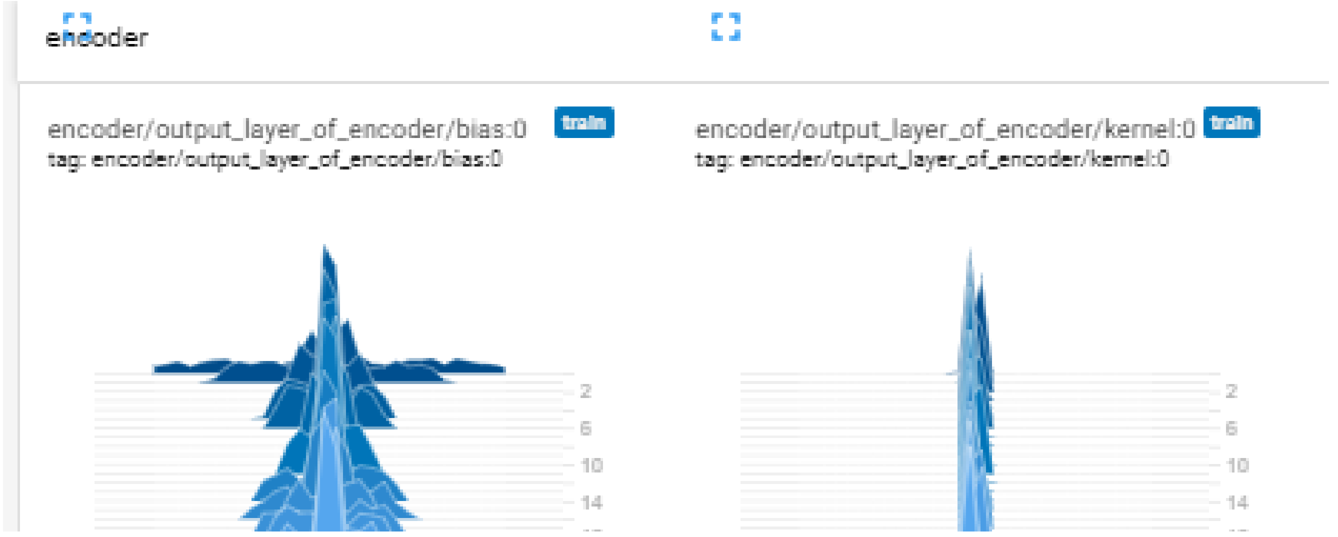


Gradients histograms for encoder

In [ ]:

```
from IPython.display import Image
Image(filename='/content/encoder.png',width=1200)
```

Out[ ]:

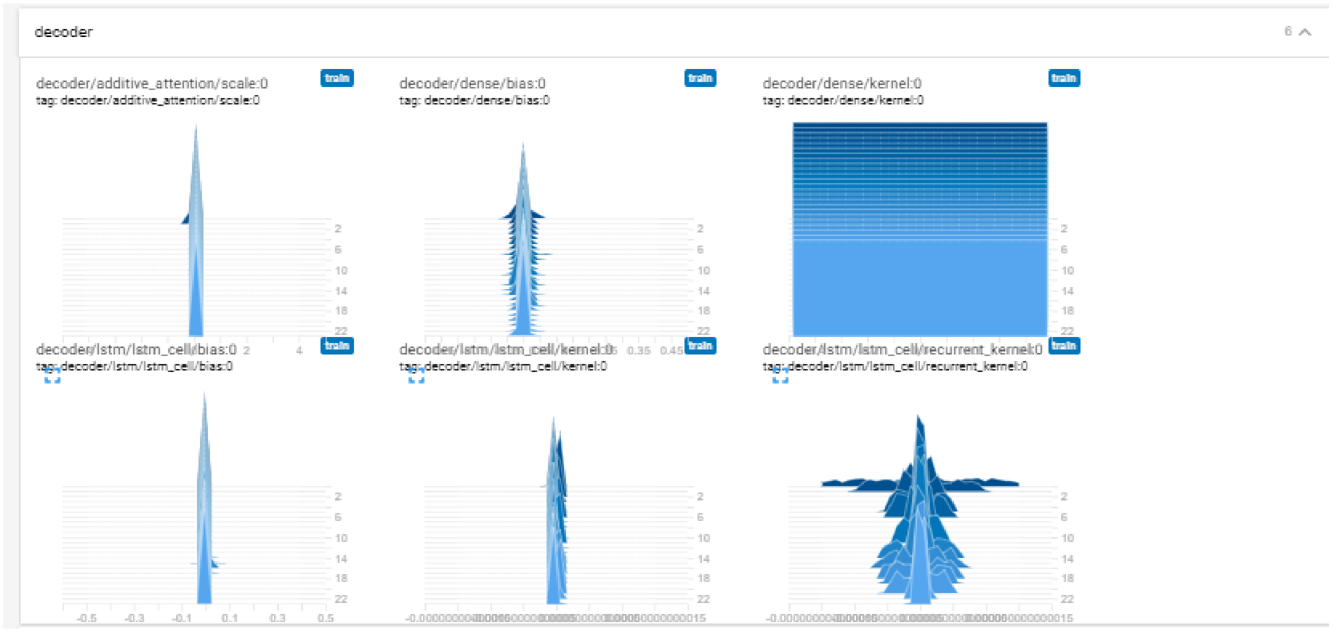


Gradients histogram for decoder

In [ ]:

```
from IPython.display import Image
Image(filename='/content/decoder.png',width=1200)
```

Out[ ]:



Let's evaluate the model

In [ ]:

```
def tensor_of_image(path_of_image, name_of_image, model):
  """ Here we extract the features of the image"""
  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 [ ]:

```
def evaluate(name_of_image):
    """ here we found the max prob score for words prediction."""
    hidden_1 =  tf.zeros((1, UNITS)) # Intialize the hidden state
    tensor_of_im = tf.convert_to_tensor([tensor_of_image(path_of_image, path[0],
model_for_image_features),# img[0], img[1]
                                        tensor_of_image(path_of_image, path[1],
model_for_image_features)])
    features_of_image = tf.constant(tensor_of_im)[None, :]# get the features of image
    values_of_features = Encoder(features_of_image) # get the encoder output of features.

    input_dec = tf.expand_dims([token.word_index['<start>']], 0) # get the words
    result = [] # store the results
    text = ""  # text
    for i in range(maximum_length_output_sentences):
        predict,hidden_1 = Decoder([input_dec, values_of_features, hidden_1])# get the output and
state from the decoder
        predict_id =  tf.argmax(predict,axis=1)[0].numpy() # get the argmax of the prediction getti
ng from decoder.
```

```
        if predict_id ==0:
            word = ""
        else:
            word = token.index_word[predict_id]
        result.append(word)
        text += " " + word
        if word == '<end>' or word == 'end':
            return result, text
        # here we do the teacher forcing
        input_dec = tf.expand_dims([predict_id], 0)
    return result, text
```

In [ ]:

```
def score(x):
    """cumulative score of the sentences"""
    return x[1]/len(x[0])

def beam_search(name_of_image, beam_index = 3):
    """take image as input in beam search"""
    hidden_1 =  tf.zeros((1, UNITS))# Initialize the hidden state
    tensor_of_im = tf.convert_to_tensor([tensor_of_image(path_of_image,path[0],
model_for_image_features),# img[0],img[1]
                                         tensor_of_image(path_of_image,path[1],
model_for_image_features)])
    features_of_image = tf.constant(tensor_of_im)[None, :]# get the features of the image
    values_of_features = Encoder(features_of_image)# get the encoder output
    start = [token.word_index["<start>"]] # here we get the start index
    word_decoder = [[start, 0.0]]
    while len(word_decoder[0][0]) < maximum_length_output_sentences:
        temp = []
        for s in word_decoder:
            predict,hidden_1 = Decoder([tf.cast(tf.expand_dims([s[0][-1]], 0), tf.float32),
values_of_features, hidden_1]) # het the output from the decoder
            word_preds = np.argsort(predict[0])[-beam_index:]# here we return the indices of the pr
edictions
            # 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]# here we get the next impresssiona and probability s
core

                next_cap.append(w)
                prob += predict[0][w]
                temp.append([next_cap, prob.numpy()])
        word_decoder = temp
        # Sorting according to the probabilities scores
        word_decoder = sorted(word_decoder, reverse=False, key=score)
        # Getting the top words
        word_decoder = word_decoder[-beam_index:]
    word_decoder = word_decoder[-1][0]
    impression = [token.index_word[i] for i in word_decoder if i !=0]
    result = []

    for i in impression:
        if i != '<end>':
            result.append(i)
        else:
            break
    text = ' '.join(result[1:])
    return result, text
```

In [ ]:

```
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"""
    """ it will display the images for the given array of images 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(path_of_image+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 is :", beam_indexing)
    print("Actual impression is:", actual_text)
    print("Predicted impression is :",text)
    print('*'*50)
    print('One-gram: {:.4f}  || Cumulative  one gram: {:.4f}'.format(sentence_bleu(reference, resul
t, weights=(1, 0, 0, 0)), sentence_bleu(reference, result, weights=(1, 0, 0, 0))))
    print('Two-gram: {:.4f}  || Cumulative  two gram: {:.4f}'.format(sentence_bleu(reference, resul
t, weights=(0, 1, 0, 0)), sentence_bleu(reference, result, weights=(0.5, 0.5, 0, 0))))
    print('Three-gram: {:.4f}|| Cumulative  three gram: {:.4f}'.format(sentence_bleu(reference, res
ult, weights=(0, 0, 1, 0)), sentence_bleu(reference, result, weights=(0.33, 0.33, 0.33, 0))))
    print('Four-gram: {:.4f} || Cumulative four gram: {:.4f}'.format(sentence_bleu(reference, resul
t, weights=(0, 0, 0, 1)), sentence_bleu(reference, result, weights=(0.25, 0.25, 0.25, 0.25))))
```

In [ ]:

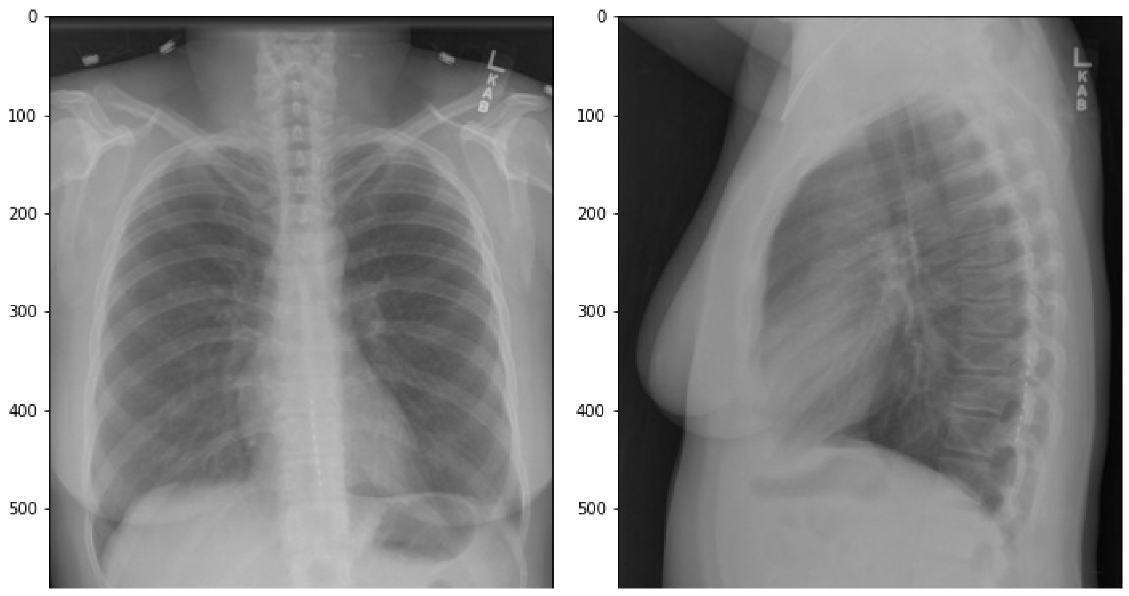
```
def test_img_cap(img_data, actual_text):
    result, text = evaluate(img_data)
    """Displays images for given input array of image names"""
    fig, axs = plt.subplots(1, len(img_data), figsize = (10,10), tight_layout=True)
    count = 0
    for img, subplot in zip(img_data, axs.flatten()):
        img_=mpimg.imread(path_of_image+img)
        imgplot = axs[count].imshow(img_, cmap = 'bone')
        count +=1
    plt.show()
    reference = [actual_text.split()[1:-1]]
    result = result[:-1]

    print("Actual impression is:", actual_text)
    print("Predicted impression is :",text)
    print('*'*50)
    print('One-gram: {:.4f}  || Cumulative  one gram: {:.4f}'.format(sentence_bleu(reference, resul
t, weights=(1, 0, 0, 0)), sentence_bleu(reference, result, weights=(1, 0, 0, 0))))
    print('Two-gram: {:.4f}  || Cumulative  two gram: {:.4f}'.format(sentence_bleu(reference, resul
t, weights=(0, 1, 0, 0)), sentence_bleu(reference, result, weights=(0.5, 0.5, 0, 0))))
    print('Three-gram: {:.4f}|| Cumulative  three gram: {:.4f}'.format(sentence_bleu(reference, res
ult, weights=(0, 0, 1, 0)), sentence_bleu(reference, result, weights=(0.33, 0.33, 0.33, 0))))
    print('Four-gram: {:.4f} || Cumulative four gram: {:.4f}'.format(sentence_bleu(reference, resul
t, weights=(0, 0, 0, 1)), sentence_bleu(reference, result, weights=(0.25, 0.25, 0.25, 0.25))))
```

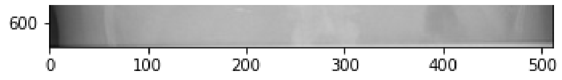
Let's test the image caption

Here we see the bleu score for multiple grams based on argmax and beam search

In [ ]:

```
test_img_cap(test_input[3], test_output[3])
```

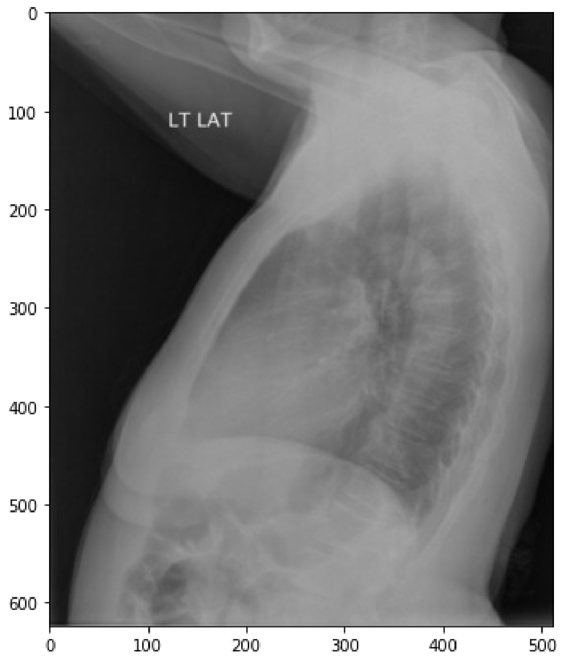


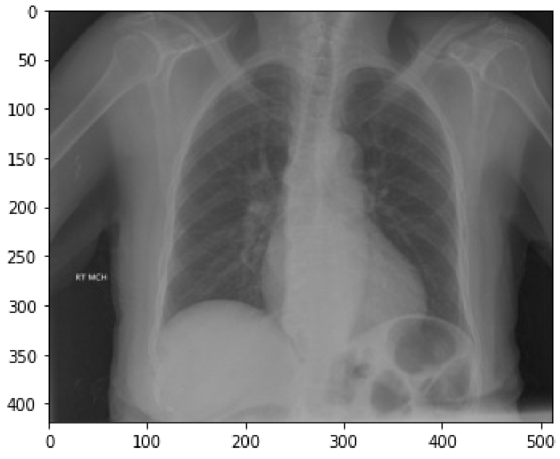


```
Actual impression is: <start> clear lungs <end>
Predicted impression is :  no acute cardiopulmonary abnormality <end>
**************************************************
One-gram: 0.0000  || Cumulative  one gram: 0.0000
Two-gram: 0.0000  || Cumulative  two gram: 0.0000
Three-gram: 0.0000|| Cumulative  three gram: 0.0000
Four-gram: 0.0000 || Cumulative four gram: 0.0000
```

In [ ]:

```
test_img_cap_beam(test_input[2], test_output[2], 2)
```

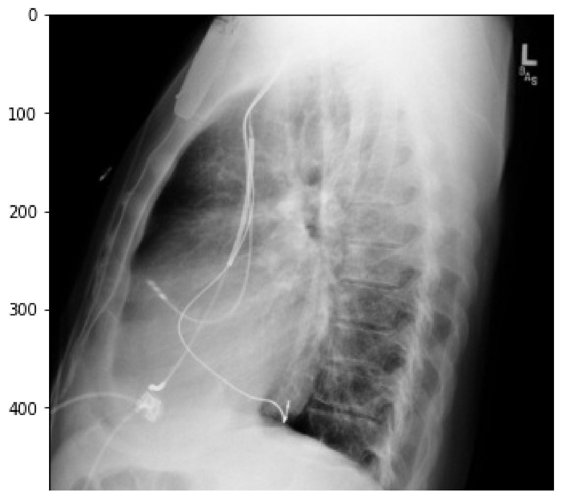
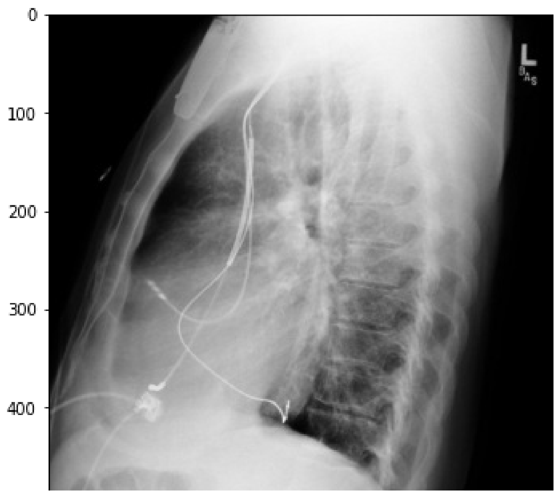


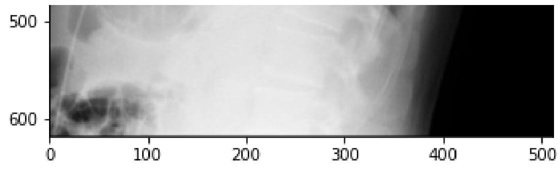
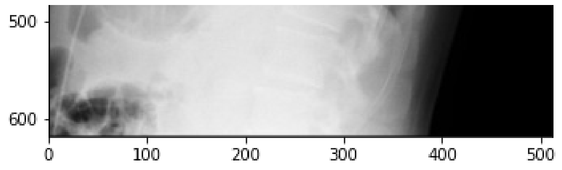


```
Beam Search index is : 2
Actual impression is: <start> no acute cardiopulmonary disease <end>
Predicted impression is : no acute cardiopulmonary process
**************************************************
One-gram: 0.7500  || Cumulative  one gram: 0.7500
Two-gram: 0.6667  || Cumulative  two gram: 0.7071
Three-gram: 0.5000|| Cumulative  three gram: 0.6329
Four-gram: 1.0000 || Cumulative four gram: 0.7071
```

In [ ]:

```
test_img_cap(test_input[45], test_output[45])
```

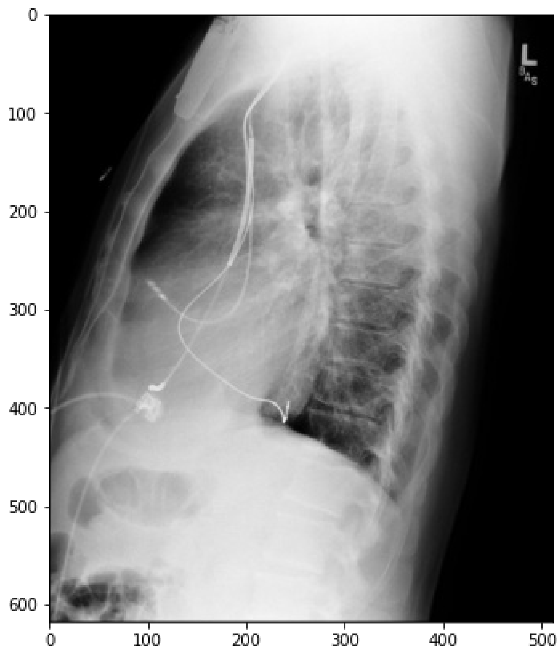



```
Actual impression is: <start> no acute pulmonary abnormality moderate cardiomegaly without
pulmonary edema <end>
Predicted impression is :  sternum increased pacemaker communicated lucency subsegmental unless st
aff interface shift therapy cavitary paralysis eacerbation disruption questionable study
obscuration junction one subpleural pancreatitis enhanced neoplastic focal accompanied quadrant hy
poinflation assessment lungs degraded side hyperinflation indolent node based appropriate stent di
slocation aspiration noncontrasted glenohumeral stomach radiodense concern origin presumed
atherosclerosis wedge soft pneumothoraces surgical developed asbestos was retrohilar edema eposure
demonstration disc
**************************************************
One-gram: 0.0169  || Cumulative  one gram: 0.0169
Two-gram: 1.0000  || Cumulative  two gram: 0.1302
Three-gram: 1.0000|| Cumulative  three gram: 0.2604
Four-gram: 1.0000 || Cumulative four gram: 0.3608
```

In [ ]:

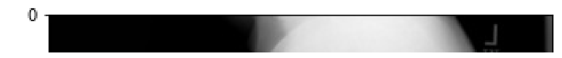
```
test_img_cap_beam(test_input[45], test_output[45], 9)
```

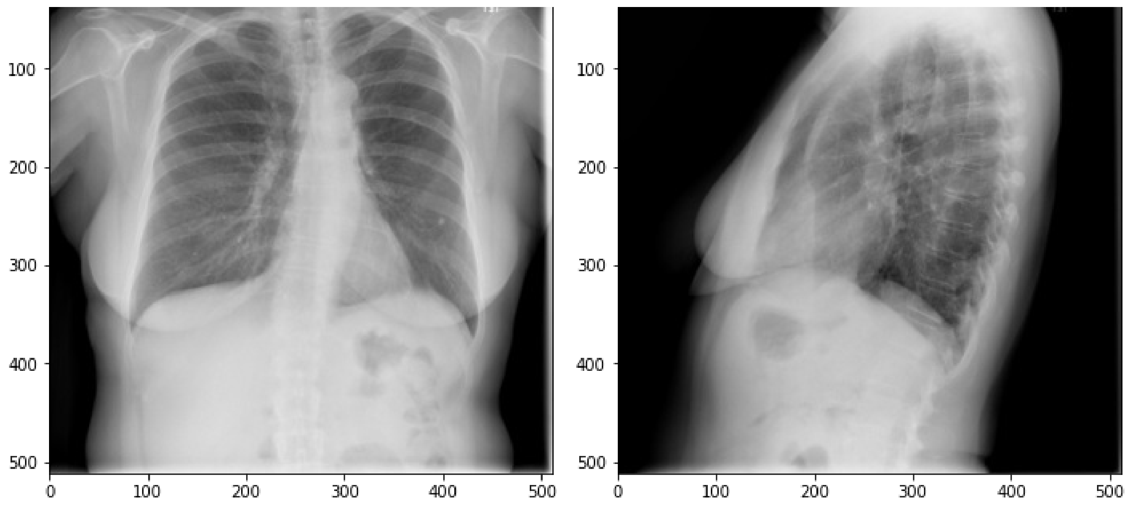


```
Beam Search index is : 9
Actual impression is: <start> no acute pulmonary abnormality moderate cardiomegaly without
pulmonary edema <end>
Predicted impression is : shadows duct mid tumor demineralization sulcus thickening abnormality fo
ot airspace lymphadenopathy periphery element intra inspiratory duct mid ossification tumor
demineralization sulcus thickening abnormality aeration pacemaker scapular post demineralization s
ulcus thickening abnormality intra inspiratory duct mid tumor demineralization sulcus thickening a
bnormality foot airspace lymphadenopathy periphery foot localize atherosclerosis widening aorta mi
d tumor demineralization sulcus thickening abnormality foot localize atherosclerosis widening
**************************************************
One-gram: 0.0169  || Cumulative  one gram: 0.0169
Two-gram: 1.0000  || Cumulative  two gram: 0.1302
Three-gram: 1.0000|| Cumulative  three gram: 0.2604
Four-gram: 1.0000 || Cumulative four gram: 0.3608
```

In [ ]:

```
test_img_cap(test_input[98], test_output[98])
```

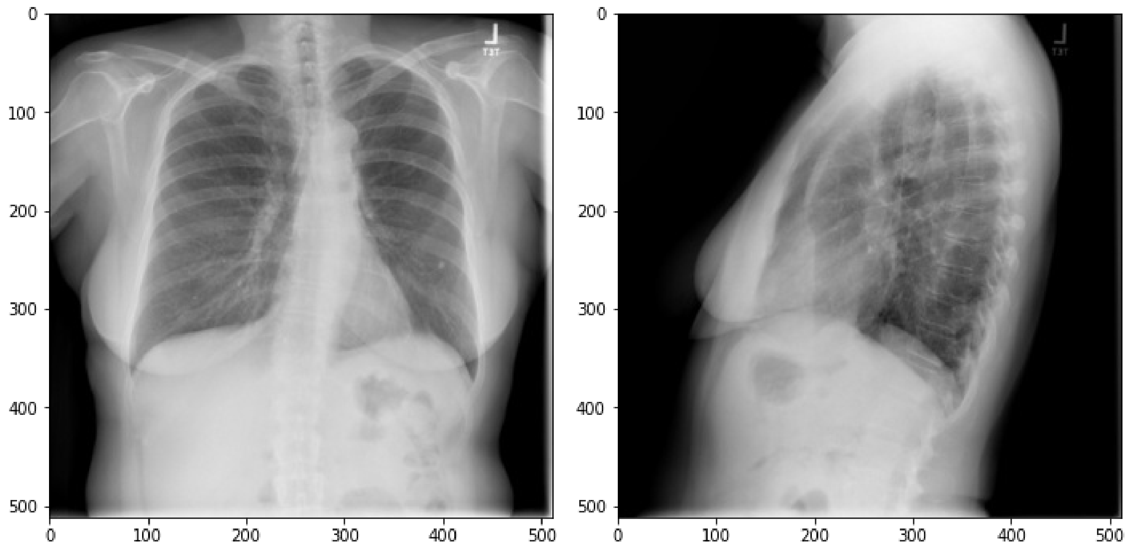




```
Actual impression is: <start> heart size normal lungs are clear calcified granuloma the left
midlung status post resection left upper lobe no adenopathy nodules masses no effusion <end>
Predicted impression is :  worse notify airway being incompletely long having aorta rotator core c
haracterization diagnostic overepanded relatively masses especially consider recent diameter indic
ate blunting verify conspicuous sternotomy developed discordant have due message loss assess order
ing graft overlapping cavity superimposing neck cardiopulmonary end
**************************************************
One-gram: 0.0263  || Cumulative  one gram: 0.0263
Two-gram: 1.0000  || Cumulative  two gram: 0.1622
Three-gram: 1.0000|| Cumulative  three gram: 0.3011
Four-gram: 1.0000 || Cumulative four gram: 0.4028
```

In [ ]:

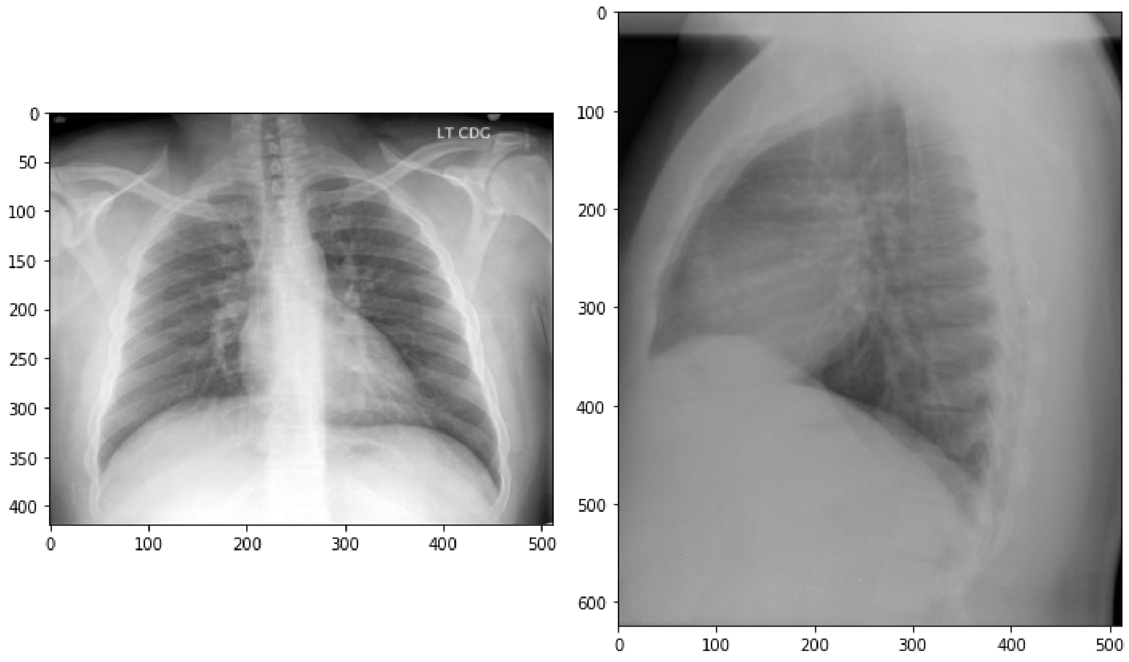
```
test_img_cap_beam(test_input[98], test_output[98], 9)
```



```
Beam Search index is : 9
Actual impression is: <start> heart size normal lungs are clear calcified granuloma the left
midlung status post resection left upper lobe no adenopathy nodules masses no effusion <end>
Predicted impression is : shadows duct mid tumor demineralization sulcus thickening abnormality fo
ot airspace lymphadenopathy periphery element intra inspiratory duct mid ossification tumor
demineralization sulcus thickening abnormality aeration pacemaker scapular post demineralization s
ulcus thickening abnormality intra inspiratory duct mid tumor demineralization sulcus thickening a
bnormality foot airspace lymphadenopathy periphery foot localize atherosclerosis widening aorta mi
d tumor demineralization sulcus thickening abnormality foot localize atherosclerosis widening
**************************************************
One-gram: 0.0169  || Cumulative  one gram: 0.0169
Two-gram: 1.0000  || Cumulative  two gram: 0.1302
Three-gram: 1.0000|| Cumulative  three gram: 0.2604
Four-gram: 1.0000 || Cumulative four gram: 0.3608
```

In [ ]:

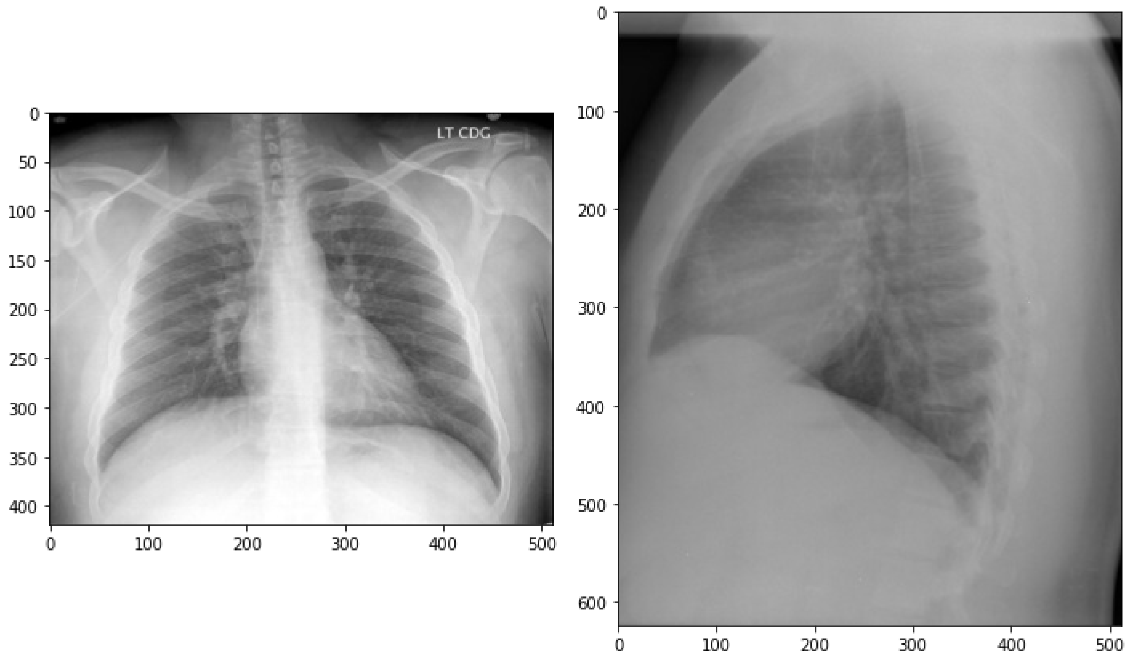
```
test_img_cap(test_input[150], test_output[150])
```



```
Actual impression is: <start> picc catheter tip mid svc heart size normal lungs clear <end>
Predicted impression is :  laparoscopic clavicles focus overload jugular arteries pancreatic
confirm concomitant paratracheal reticulonodular interstitial now increased projected suggestion h
ave central malpositioned radiation differences aortic superimposing increased mid treatment
largest parapneumonic patients possibility vague define subacute advanced completely stripe hypoin
flation another pneumothora brachiocephalic diameter paraspinal apparently echange catheter
suggestions relatively enlarging abnormality disruption hyperdensity minimally placed rotator
largest oriented cardiac bibasilar syndesmophytes based
**************************************************
One-gram: 0.0339  || Cumulative  one gram: 0.0339
Two-gram: 1.0000  || Cumulative  two gram: 0.1841
Three-gram: 1.0000|| Cumulative  three gram: 0.3273
Four-gram: 1.0000 || Cumulative four gram: 0.4291
```

In [ ]:

```
test_img_cap_beam(test_input[150], test_output[150], 9)
```



```
Beam Search index is : 9
Actual impression is: <start> picc catheter tip mid svc heart size normal lungs clear <end>
```

```
Predicted impression is : shadows duct mid tumor demineralization sulcus thickening abnormality fo
ot airspace lymphadenopathy periphery element intra inspiratory duct mid ossification tumor
demineralization sulcus thickening abnormality aeration pacemaker scapular post demineralization s
ulcus thickening abnormality intra inspiratory duct mid tumor demineralization sulcus thickening a
bnormality foot airspace lymphadenopathy periphery foot localize atherosclerosis widening aorta mi
d tumor demineralization sulcus thickening abnormality foot localize atherosclerosis widening
*****************************************************
One-gram: 0.0169  || Cumulative  one gram: 0.0169
Two-gram: 1.0000  || Cumulative  two gram: 0.1302
Three-gram: 1.0000|| Cumulative  three gram: 0.2604
Four-gram: 1.0000 || Cumulative four gram: 0.3608
```

INFERENCES ANALYSIS:

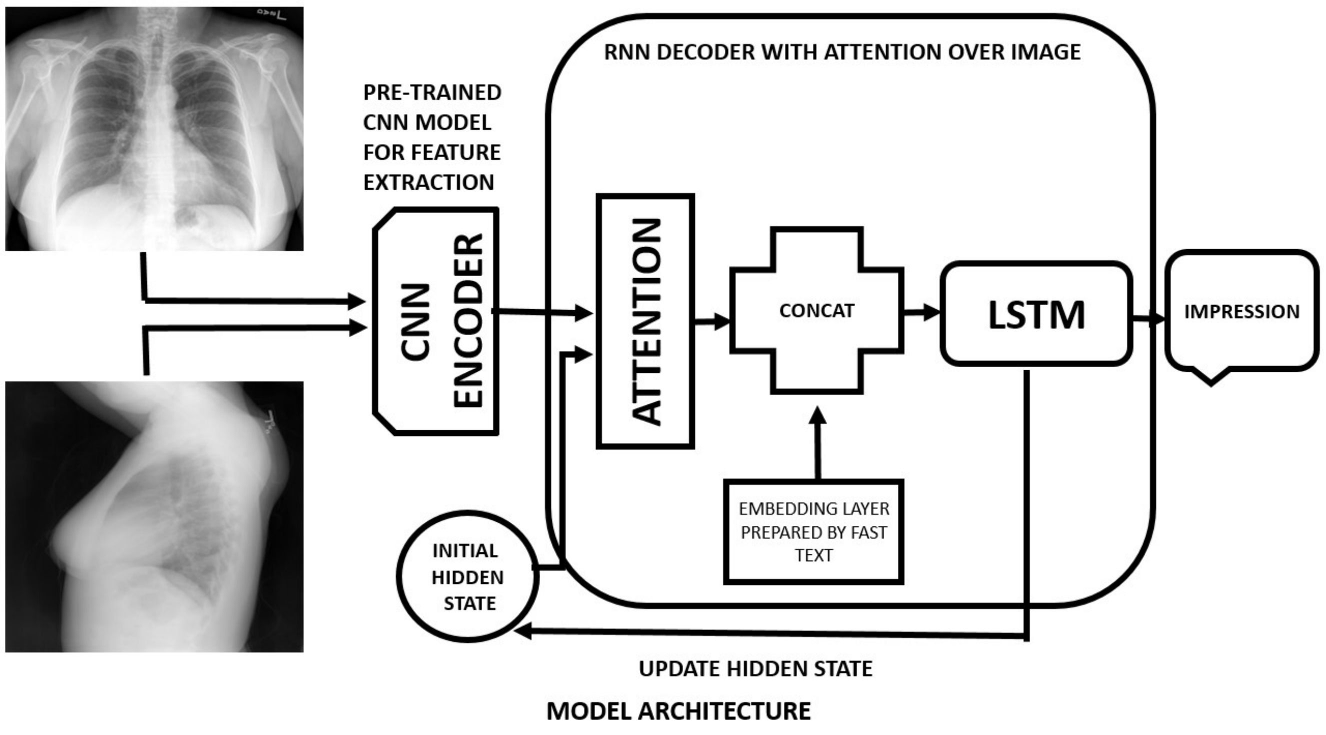
- AS from the results the beam search method is not performing well at longer sentences.
- for longer sentences argmax search is performing well. Beam search is giving fault results some time even if we increase the beam width/index.

Model Architecture

In [ ]:

```python
from IPython.display import Image
Image(filename='/content/Slide2.jpg')
```

Out[ ]:


RNN DECODER WITH ATTENTION OVER IMAGE
PRE-TRAINED CNN MODEL FOR FEATURE EXTRACTION
CNN ENCODER
ATTENTION
CONCAT
LSTM
IMPRESSION
EMBEDDING LAYER PREPARED BY FAST TEXT
INITIAL HIDDEN STATE
UPDATE HIDDEN STATE
MODEL ARCHITECTURE

Conclusion:

- The model is based on LSTM with attention seems better model than the basic model.
- Loss is converged to 0.27 with accuracy of 86 percent train and 82 percent validation from the result we can see there is similarity between each predicted and actual output.
- There are many major impression identified in the predicted output if there are any major actual impression.

- I used the chexnet to extract the featurs of the images so that i can make the tensor of the images.
- The features ectracted is done by them converting into tensor form and then it act as the input to the CNN encoder layer and then this encoded vector input to attention Layer and prev_state_vector is also given to attention.Then we concat the attention output and the embedding layer output which is given to the lstm . The predicted value and the hidden state is calculated and fedback to the decoder cell.