An Implementation of Image Captioning using Show Attend and Tell paper

- The model architecture is inspired by the "Show, Attend and Tell" (https://arxiv.org/pdf/1502.03044.pdf) paper.

In [ ]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
..........
Mounted at /content/drive
```

In [2]:

```
# https://www.kaggle.com/hsankesara/flickr-image-dataset

! wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.105 Safari/537.36" --header="Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9" --header="Accept-Language: en-US,en;q=0.9" --header="Referer: https://www.kaggle.com/" "https://storage.googleapis.com/kaggle-data-sets/31296%2F39911%2Fbundle%2Farchive.zip?GoogleAccessId=gcp-kaggle-com@kaggle-161607.iam.gserviceaccount.com&Expires=1596439301&Signature=oPk4WVYzNAVsWsPVYfozX5HxQYpFC1HUcnbqS%2Flm6L3oC7SyGT3Qz67fsIHRDqvMS821E%2F1v1J2xBh3dyt8yCC8zC6cT6QFQ4wJ2a%2Fwt5ivgYRHL7LNI%2FMdxfwpqYQKPNTqRF%2BrKCeSvAB7yLQOosuJtNfIYAFoJa1GywJ9xdM5VIDP%2FyMo9HvO%2FbLSIbv2ZTV%2B50dHUu6cB2LectaT%2Bjw869z2z7aRXmMm2tIPUwD1oX6hV9MwQJzoqxyRS%2Fi68SeDMGvVLSeGtMhPxkPjZ34QwE8IvTfnYYpTKXCTNs5pqzK3KlU6QgxG%2FIdeXnl8YmHzKnNaLOwh4xYEfADWrtw%3D%3D" -c -O '31296_39911_bundle_archive.zip'
```

```
--2020-07-31 07:26:32--  https://storage.googleapis.com/kaggle-data-sets/31296%2F39911%2Fbundle%2Farchive.zip?GoogleAccessId=gcp-kaggle-com@kaggle-161607.iam.gserviceaccount.com&Expires=1596439301&Signature=oPk4WVYzNAVsWsPVYfozX5HxQYpFC1HUcnbqS%2Flm6L3oC7SyGT3Qz67fsIHRDqvMS821E%2F1v1J2xBh3dyt8yCC8zC6cT6QFQ4wJ2a%2Fwt5ivgYRHL7LNI%2FMdxfwpqYQKPNTqRF%2BrKCeSvAB7yLQOosuJtNfIYAFoJa1GywJ9xdM5VIDP%2FyMo9HvO%2FbLSIbv2ZTV%2B50dHUu6cB2LectaT%2Bjw869z2z7aRXmMm2tIPUwD1oX6hV9MwQJzoqxyRS%2Fi68SeDMGvVLSeGtMhPxkPjZ34QwE8IvTfnYYpTKXCTNs5pqzK3KlU6QgxG%2FIdeXnl8YmHzKnNaLOwh4xYEfADWrtw%3D%3D
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.204.128, 172.217.203.128, 173.194.216.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.204.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8765396518 (8.2G) [application/zip]
Saving to: '31296_39911_bundle_archive.zip'

31296_39911_bundle_ 100%[===================>]   8.16G   133MB/s    in 3m 2s

2020-07-31 07:29:35 (45.9 MB/s) - '31296_39911_bundle_archive.zip' saved [8765396518/8765396518]
```

In [3]:

```
import datetime
import time

start= time.time()

import zipfile
with zipfile.ZipFile("/content/31296_39911_bundle_archive.zip","r") as zip_ref:
  zip_ref.extractall()

print("Time Taken is: " + str(time.time() - start))
```

```
Time Taken is: 220.08534002304077
```

In [4]:

```
!pip3 install contractions
```

```
Collecting contractions
  Downloading https://files.pythonhosted.org/packages/00/92/a05b76a692ac08d470ae5c23873cf1c9a041532f1ee065e74b374f218306/contractions-0.0.25-py2.py3-none-any.whl
```

```
Collecting textsearch
  Downloading https://files.pythonhosted.org/packages/42/a8/03407021f9555043de5492a2bd7a35c56cc03c2510092b5ec018cae1bbf1/textsearch-0.0.17-py2.py3-none-any.whl
Collecting pyahocorasick
  Downloading https://files.pythonhosted.org/packages/f4/9f/f0d8e8850e12829eea2e778f1c90e3c53a9a799b7f412082a5d21cd19ae1/pyahocorasick-1.4.0.tar.gz (312kB)
     |████████████████████████████████| 317kB 3.7MB/s
Collecting Unidecode
  Downloading https://files.pythonhosted.org/packages/d0/42/d9edfed04228bacea2d824904cae367ee9efd05e6cce7ceaaedd0b0ad964/Unidecode-1.1.1-py2.py3-none-any.whl (238kB)
     |████████████████████████████████| 245kB 16.1MB/s
Building wheels for collected packages: pyahocorasick
  Building wheel for pyahocorasick (setup.py) ... done
  Created wheel for pyahocorasick: filename=pyahocorasick-1.4.0-cp36-cp36m-linux_x86_64.whl size=81701 sha256=d72dcfb3993949a84d7f61ebc5e74c89bd6286e81591375644c68fec65cb5f17
  Stored in directory: /root/.cache/pip/wheels/0a/90/61/87a55f5b459792fbb2b7ba6b31721b06ff5cf6bde541b40994
Successfully built pyahocorasick
Installing collected packages: pyahocorasick, Unidecode, textsearch, contractions
Successfully installed Unidecode-1.1.1 contractions-0.0.25 pyahocorasick-1.4.0 textsearch-0.0.17
```

In [5]:

```python
# Tensorboard
%load_ext tensorboard

# We import necessary Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import glob
import pickle
import cv2
from bs4 import BeautifulSoup
import contractions
import re
from collections import Counter
import sys, time, os, warnings
warnings.filterwarnings("ignore")
from tqdm import tqdm

from sklearn.model_selection import train_test_split

import tensorflow as tf
from keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector,\
                         Activation, Flatten, Reshape, concatenate, Dropout, \
                         BatchNormalization, Bidirectional, GRU
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from tensorflow.keras.applications.inception_v3 import InceptionV3 # import separately and use preprocess_input
from tensorflow.keras.applications.vgg16 import VGG16 # import separately and use preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras import Input, layers
from tensorflow.keras import optimizers
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
Using TensorFlow backend.
```

In [6]:

```python
# https://stackoverflow.com/a/18129082/10219869

df= pd.read_csv(r'/content/flickr30k_images/results.csv', error_bad_lines=False, sep='|')
print(df.shape)
df.head(10)
```

```
(158915, 3)
```

Out[6]:

	image_name	comment_number	comment
0	1000092795.jpg	0	Two young guys with shaggy hair look at their...
1	1000092795.jpg	1	Two young , White males are outside near many...
2	1000092795.jpg	2	Two men in green shirts are standing in a yard .
3	1000092795.jpg	3	A man in a blue shirt standing in a garden .
4	1000092795.jpg	4	Two friends enjoy time spent together .
5	10002456.jpg	0	Several men in hard hats are operating a gian...
6	10002456.jpg	1	Workers look down from up above on a piece of...
7	10002456.jpg	2	Two men working on a machine wearing hard hats .
8	10002456.jpg	3	Four men on top of a tall structure .
9	10002456.jpg	4	Three men on a large rig .

In [7]:

```
# To check the names of columns
df.columns
```

Out[7]:

```
Index(['image_name', ' comment_number', ' comment'], dtype='object')
```

In [8]:

```
# For every consecutive 5 images, these are different captions
df[' comment'][:5]
```

Out[8]:

```
0    Two young guys with shaggy hair look at their...
1    Two young , White males are outside near many...
2    Two men in green shirts are standing in a yard .
3         A man in a blue shirt standing in a garden .
4              Two friends enjoy time spent together .
Name:  comment, dtype: object
```

In [9]:

```
# for every five consecutive indexes, there is same image hence same name.
df['image_name'][:5]
```

Out[9]:

```
0    1000092795.jpg
1    1000092795.jpg
2    1000092795.jpg
3    1000092795.jpg
4    1000092795.jpg
Name: image_name, dtype: object
```

In [10]:

```
# Check for any nan values

print('Image_name:\n', df[df['image_name'].isnull()])
print('Comment_number:\n', df[df[' comment_number'].isnull()])
print('Comment:\n', df[df[' comment'].isnull()])
```

```
Image_name:
 Empty DataFrame
Columns: [image_name,  comment_number,  comment]
Index: []
Comment_number:
 Empty DataFrame
Columns: [image_name,  comment_number,  comment]
Index: []
Comment:
           image_name                      comment_number  comment
19999  2199200615.jpg   4   A dog runs across the grass .      NaN
```

In [11]:

```
# As we see there is a problem with index number 19999 and we shall solve it by keeping in place these values
# https://stackoverflow.com/a/37725243/10219869

df.loc[19999, ' comment_number'] = 4
df.loc[19999, ' comment'] = 'A dog runs across the grass .'
```

In [12]:

```
# Now it is good
df[19999:20000]
```

Out[12]:

	image_name	comment_number	comment
19999	2199200615.jpg	4	A dog runs across the grass .

In [13]:

```
# Considering 30,000 captions => 6000 Images

new_df= df[:30000]
new_df.tail(10)
```

Out[13]:

	image_name	comment_number	comment
29990	244829722.jpg	0	A woman in a white t-shirt is in a swing that...
29991	244829722.jpg	1	A girl going on a ride in a circus that spins...
29992	244829722.jpg	2	A young woman rides by herself on a swinging ...
29993	244829722.jpg	3	Girl in a swing ride at an amusement park or ...
29994	244829722.jpg	4	A woman is on a carnival swing ride .
29995	2448393373.jpg	0	A young boy wearing blue is holding a blue ba...
29996	2448393373.jpg	1	A boy with a plastic bat , looking skyward , ...
29997	2448393373.jpg	2	A small boy playing in the grass with a blue ...
29998	2448393373.jpg	3	A little boy plays baseball with himself .
29999	2448393373.jpg	4	A boy plays baseball .

In [14]:

```
#Function to plot the images and its description
# https://fairyonice.github.io/Develop_an_image_captioning_deep_learning_model_using_Flickr_8K_data.html

path= '/content/flickr30k_images/flickr30k_images/'   # the path is provided

def image_desc_plotter(data):
  npic = 5
  npix = 224
  target_size = (npix,npix,3)

  count = 1
  fig = plt.figure(figsize=(10,20))
  for jpgfnm in new_df['image_name'].unique()[20:25]:
    filename = path + jpgfnm
    captions = list(data[" comment"].loc[data["image_name"] == jpgfnm].values)
    image_load = image.load_img(filename, target_size = target_size)

    ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
    ax.imshow(image_load)
    count += 1

    ax = fig.add_subplot(npic,2,count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
    ax.set_ylim(0,len(captions))
    for i, caption in enumerate(captions):
      ax.text(0,i,caption,fontsize=20) #color='orange'
```

```
        ax.text(0,i,caption,fontsize=20,) #color='orange'
        count += 1
    plt.show()

image_desc_plotter(data = df)
```

A man in a jacket and jeans standing on a bridge .
A man stands on wooden supports and surveys damage .
A man in a gray coat is standing on a washed out bridge .
A large structure has broken and is laying in a roadway .
A person in gray stands alone on a structure outdoors in the dark .

Crowd standing outside a metro area .
A crowd is portrayed near a metro station .
A group of people are walking through a city street .
A large crowd of people stand outside in front of the entrance to a Metro station .
A man in a white t-shirt looks toward the camera surrounded by a crowd near a metro station .

A man getting a tattoo on his back .
A man is putting tattoo on his back .
A man with a black shirt giving another man a tattoo .
A man is putting a tattoo on a another 's man upper back .
A man with a goatee in a black shirt and white latex gloves is using a tattoo gun to place a tattoo on someone 's back .

Two kids sit on a seesaw .
2 kids playing on a seesaw
Two children sitting on a teeter totter .
Two children sit on a small seesaw in the sand .
two children , a girl and a boy are practicing their writing .

A man wearing a reflective vest and a hard hat holds a flag in the road
A construction worker is standing in the street and holding a red flag .
A man in bright vest and hard hat holds a flag on a street corner covered in spray paint
A man wearing a hard hat and a caution vest is standing in the street waving an orange flag .
A man in a blue hard hat and orange safety vest stands in an intersection while holding a flag .

In [15]:

```
# We remove the jpg extension from name of image and then remove the 5 duplicate names to 1 name
image= [i.replace('.jpg','') for i in new_df['image_name']]
```

In [16]:

```
# We need the words for corpus and we need to clean the sentences based on the below conditions

def clean_sentence(text):

    text = BeautifulSoup(text, 'lxml').get_text()        # removes html tags such as <br />
    text = ''.join([i for i in text if not i.isdigit()]) # removes numbers
    text = text.lower()                                  # converts text to lower case
    text = contractions.fix(text)                        # converts (don't) to (do not)
    # https://stackoverflow.com/a/23853882/10219869       remove special characters except these 3 -\'
    text = re.sub(r'[]|?|$|!|@|#|%|^|&|*|(|)|_|+|=|{|}|:|;|<|>|,|~|/|`|.|[]',r'', text)
    # https://stackoverflow.com/a/32706078/10219869 removes single letters except 'a' and 'i'
    text = re.sub('(\\b[bcdefghjklmnoprstuvwxyz] \\b|\\b [bcdefghjklmnoprstuvwxyz]\\b)', '', text)
    return text

# Clean the individual sentences and then obtain a set of word corpus
sentences= [clean_sentence(i) for i in new_df[' comment']]
print('Length of sentences:', len(sentences))
sentences[:10]
```

```
Length of sentences: 30000
```

Out[16]:

```
['two young guys with shaggy hair look at their hands while hanging out in the yard ',
 'two young  white males are outside near many bushes ',
 'two men in green shirts are standing in a yard ',
```

```
'two men in green shirts are standing in a yard ',
'a man in a blue shirt standing in a garden ',
'two friends enjoy time spent together ',
'several men in hard hats are operating a giant pulley system ',
'workers look down from up above on a piece of equipment ',
'two men working on a machine wearing hard hats ',
'four men on top of a tall structure ',
'three men on a large rig ']
```

In [17]:

```
d = pd.DataFrame(sentences)
d.to_csv('./sentences.csv')
```

Thanks to Grammarly software and with its help, I was able to clean the sentences gramatically

In [18]:

```
# https://stackoverflow.com/a/46000253 encoding worked out but in a different way though
new_sen = pd.read_csv('/content/drive/My Drive/new_sen1.csv', header= None, error_bad_lines= False, encoding='cp1252' )
```

In [19]:

```
# found unnecessary columns of NaN upto 37 and hence discarding all andalso last row and keeping only first column.
new_sen = new_sen[:30000]
new_sentences = new_sen[0]
print('The length of new sentences: ',len(new_sentences))
```

```
The length of new sentences:  30000
```

In [20]:

```
new_sentences.head()
```

Out[20]:

```
0    two young guys with shaggy hair look at their ...
1    two young white males are outside near many bu...
2      two men in green shirts are standing in a yard
3          a man in a blue shirt standing in a garden
4               two friends enjoy time spent together
Name: 0, dtype: object
```

In [21]:

```
# https://stackoverflow.com/a/39673666
from sklearn.utils import shuffle
img = new_df['image_name']
new_dff = pd.DataFrame()
new_dff['img'] = img
new_dff['sentences'] = new_sentences
new_dff.head()
```

Out[21]:

	img	sentences
0	1000092795.jpg	two young guys with shaggy hair look at their ...
1	1000092795.jpg	two young white males are outside near many bu...
2	1000092795.jpg	two men in green shirts are standing in a yard
3	1000092795.jpg	a man in a blue shirt standing in a garden
4	1000092795.jpg	two friends enjoy time spent together

In [22]:

```
df = shuffle(new_dff, random_state= 18).reset_index(drop=True)
print(len(df))
df.head()
```

```
30000
```

Out[22]:

	img	sentences
0	2316097768.jpg	a black guy smoking a cigarette
1	204886976.jpg	a silver statue of men on bikes
2	2334983049.jpg	a group is playing music on stage in front of ...
3	1523800748.jpg	boy and girl running along the beach
4	2215875786.jpg	this woman is sweeping the sidewalk outside of...

In [23]:

```
# would split this into 95:05 basis inorder to avoid data likeage
train_sentences = df['sentences'][:28500]
train_images = df['img'][:28500]
print('Length of train sentences:', len(train_sentences))
print('Length of train images:', len(train_images))
test_sentences = df['sentences'][28500:]
test_images = df['img'][28500:]
print('Length of test sentences:', len(test_sentences))
print('Length of test images:', len(test_images))
```

```
Length of train sentences: 28500
Length of train images: 28500
Length of test sentences: 1500
Length of test images: 1500
```

Train data processing

In [24]:

```
# To find the max and min length  of a description so that we can easily pad sequences

l= []
for i in train_sentences:
  i= i.split()
  l.append(len(i))

print('Max length of captions:', max(l))
print('Min length of captions:', min(l))
```

```
Max length of captions: 78
Min length of captions: 2
```

In [25]:

```
# We check the percentiles and decide the length of padding

indices= [i for i in range(0, 101, 10)]
per = np.percentile(a= l, q= range(1, 101, 10), )
for i,j in enumerate(list(per)):
  print(indices[i],'th percentile is:',j)
```

```
0 th percentile is: 5.0
10 th percentile is: 7.0
20 th percentile is: 8.0
30 th percentile is: 9.0
40 th percentile is: 10.0
50 th percentile is: 11.0
60 th percentile is: 12.0
70 th percentile is: 14.0
80 th percentile is: 16.0
90 th percentile is: 19.0
```

In [26]:

```
# We check the 99 percentile and decide the length of padding

indices= [i for i in range(90, 101)]
per = np.percentile(a= l, q= range(90, 101), )
for i,j in enumerate(list(per)):
```

```
  print(indices[i],'th percentile is:',j)
```

```
90 th percentile is: 18.0
91 th percentile is: 19.0
92 th percentile is: 19.0
93 th percentile is: 20.0
94 th percentile is: 20.0
95 th percentile is: 21.0
96 th percentile is: 22.0
97 th percentile is: 23.0
98 th percentile is: 25.0
99 th percentile is: 28.0
100 th percentile is: 78.0
```

In [27]:

```
# hence we consider maximum length based on 99th percentile
max_length= 28
```

In [28]:

```
# We append the ‘startseq’ and ‘endseq’ for each caption in order for data generator
# ‘startseq’ -> This is a start sequence token which will be added at the start of every caption.
# ‘endseq’ -> This is an end sequence token which will be added at the end of every caption.

def trunc(data):

  data = data.split()
  # if data length less than max length, then do padding
  if len(data) > max_length:
    del data[max_length:]
  data = ' '.join(data)

  return  data

new_values = [trunc(i) for i in train_sentences]

values= []
for i in new_values:
  a= '<startseq> ' + i + ' <endseq>'
  values.append(a)

values[:10]
```

Out[28]:

```
['<startseq> a black guy smoking a cigarette <endseq>',
 '<startseq> a silver statue of men on bikes <endseq>',
 '<startseq> a group is playing music on stage in front of a crowd of people <endseq>',
 '<startseq> boy and girl running along the beach <endseq>',
 '<startseq> this woman is sweeping the sidewalk outside of her boutique <endseq>',
 '<startseq> a black dog plays with another animal <endseq>',
 '<startseq> the boy wearing a black shirt and blue jeans is holding a red baseball bat <endseq>',
 '<startseq> a girl in a floral bathing suit jumping on the beach in front of the waves <endseq>',
 '<startseq> a man is interacting with two dogs holding one of them <endseq>',
 '<startseq> a construction worker is building scaffolding for the job <endseq>']
```

In [29]:

```
# To find the max and min length  of a description so that we can easily pad sequences
m= []
for i in values:
  i= i.split()
  m.append(len(i))

print('Max length of captions:', max(m))
print('Min length of captions:', min(m))
```

```
Max length of captions: 30
Min length of captions: 4
```

In [30]:

```
# all words (new_desc is train data) into this list
words_corpus=[]

for i in values:
```

```
  i = i.split(' ')
  for j in i:
    words_corpus.append(j)

print('Total words :', len(words_corpus))
word_corpus= set(words_corpus)
print('Total unique words:',len(word_corpus))
```

Total words : 400390
Total unique words: 9211

In [31]:

```
# We do tokenization and padding is considered as 0 number
# oov_token: if given, it will be added to word_index and used to replace out-of-vocabulary
# words during text_to_sequence calls
# unk means unknown (, oov_token= '<unk>')
# num_words = None becoz doing all train vocab of 9180
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words= None, filters='!#$%&()*+.,/:;=?@[\]^_`{|}~ ',
                                                  oov_token= '<unk>')

# this takes top 5000 words based on frequency from captions
tokenizer.fit_on_texts(texts= values)

# map pad to 0 and vice versa
tokenizer.word_index['PAD'] = 0
tokenizer.index_word[0] = 'PAD'

# Create the tokenized vectors of numbers generated from tokenizer above
train_seqs = tokenizer.texts_to_sequences(values)
```

In [32]:

```
print(values[0],'\n')
print(train_seqs[0],'\n')
print('OOV_token:',tokenizer.oov_token)
```

<startseq> a black guy smoking a cigarette <endseq>

[3, 2, 25, 189, 457, 2, 414, 4]

OOV_token: <unk>

In [33]:

```
for i in list(tokenizer.index_word.items())[:10]:
  print(i)
```

(1, '<unk>')
(2, 'a')
(3, '<startseq>')
(4, '<endseq>')
(5, 'in')
(6, 'the')
(7, 'on')
(8, 'is')
(9, 'man')
(10, 'and')

In [34]:

```
for i in list(tokenizer.word_index.items())[:10]:
  print(i)
```

('<unk>', 1)
('a', 2)
('<startseq>', 3)
('<endseq>', 4)
('in', 5)
('the', 6)
('on', 7)
('is', 8)
('man', 9)
('and', 10)

In [35]:

```
# We do padding
def padding_sequences(sequences, max_length):
  pad_vector = tf.keras.preprocessing.sequence.pad_sequences(sequences,  maxlen= max_length,
                                                             padding= 'post', truncating= 'post')
  return pad_vector

# max_length is 28 and we add +2 to include start and end tokens hence 30.
pad_captions = padding_sequences(sequences = train_seqs, max_length = max_length + 2)
print('Shape of pad_captions:', pad_captions.shape)
pad_captions[:3]
```

```
Shape of pad_captions: (28500, 30)
```

Out[35]:

```
array([[  3,   2,  25, 189, 457,   2, 414,   4,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0],
       [  3,   2, 639, 479,  11,  35,   7, 469,   4,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0],
       [  3,   2,  37,   8,  34, 339,   7, 177,   5,  36,  11,   2,  97,
         11,  18,   4,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0]], dtype=int32)
```

In [36]:

```
image_names = []

for i in train_images:
  image_names.append(path+i)

# because we find consecutive 5 images are same ones
unique_names = list(set(image_names[:10]))
unique_names
```

Out[36]:

```
['/content/flickr30k_images/flickr30k_images/2334983049.jpg',
 '/content/flickr30k_images/flickr30k_images/2082663150.jpg',
 '/content/flickr30k_images/flickr30k_images/204886976.jpg',
 '/content/flickr30k_images/flickr30k_images/1523800748.jpg',
 '/content/flickr30k_images/flickr30k_images/2316097768.jpg',
 '/content/flickr30k_images/flickr30k_images/2215875786.jpg',
 '/content/flickr30k_images/flickr30k_images/1511807116.jpg',
 '/content/flickr30k_images/flickr30k_images/10957138.jpg',
 '/content/flickr30k_images/flickr30k_images/2447284966.jpg',
 '/content/flickr30k_images/flickr30k_images/1148889628.jpg']
```

In [37]:

```
print('Length of unique image_names:',len(set(image_names)))
```

```
Length of unique image_names: 6000
```

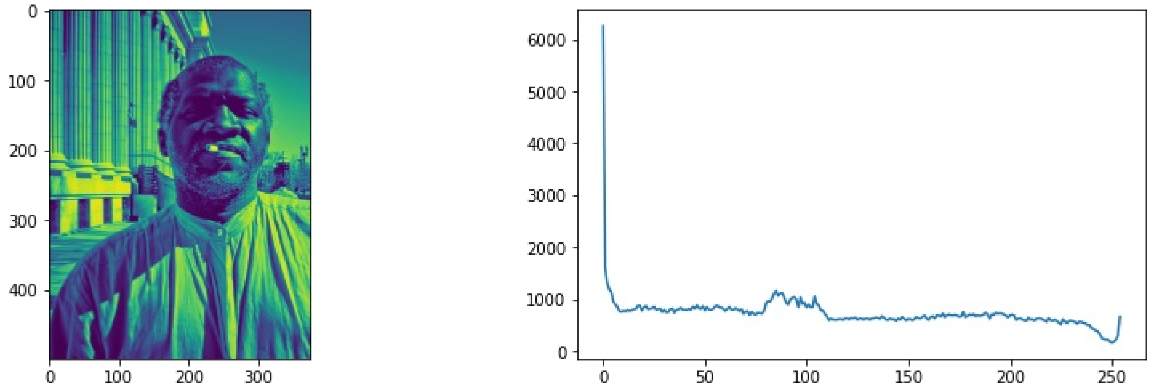
Image Histograms

- Histogram is considered as a graph or plot which is related to frequency of pixels in an Gray Scale Image with pixel values (ranging from 0 to 255). Grayscale image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information where pixel value varies from 0 to 255.

In [38]:

```
def img_hist(img):
  im = cv2.imread(img, 0)
  # find frequency of pixels in range 0-255
  hist = cv2.calcHist([im], [0],  None,  [255],  [0, 256])

  # show the plotting graph of an image
  fig = plt.figure(figsize=(14, 4))
  plt.subplot(1,2,1)
  plt.imshow(im)
  plt.subplot(1,2,2)
```

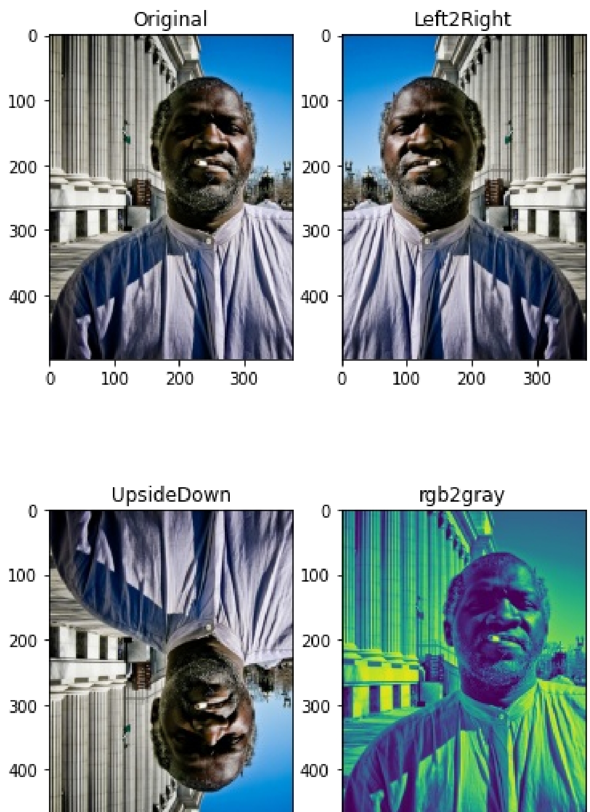
```
    plt.plot(hist)
    plt.show()

img_hist(image_names[0])
```



Trying image augmentation

In [39]:

```
import imageio

def flip_image(img):
    im1 = imageio.imread(img)
    # based on tensorflow.image
    im2 = tf.image.flip_left_right(image = im1)
    im3 = tf.image.flip_up_down(image = im1)
    im4 = tf.image.rgb_to_grayscale(images = im1) # similar to cv2.imread(x, 0)
    # Removes dimensions of size 1 from the shape of a tensor. or else error occurs
    im4 = tf.squeeze(im4)

    images = [im1, im2, im3, im4]
    titles = ['Original', 'Left2Right', 'UpsideDown', 'rgb2gray']

    plt.figure(figsize= (6, 10))
    for i in range(4):
        plt.subplot(2, 2, i+1)
        plt.imshow(images[i])
        plt.title(titles[i])

    plt.show()


flip_image(image_names[0])
```



Test data processing

In [40]:

```
ts_values = [trunc(i) for i in test_sentences]

test_values= []
for i in ts_values:
  a= '<startseq> ' + i + ' <endseq>'
  test_values.append(a)

test_values[:10]
```

Out[40]:

```
['<startseq> a man with a headscarf is gesturing with his eyes closed while holding some kind of stick <endseq>',
 '<startseq> two people on a grassy plain are gathering a parachute that one of the people just used <endseq>',
 '<startseq> a man is crossing the street and in the distance you can see a building under construction <endseq>',
 '<startseq> a man wearing khaki pants and a red jacket is lying on the ground beside a small tree <endseq>',
 '<startseq> a man in a black jacket with glasses is reaching into a blue bucket <endseq>',
 '<startseq> a dog sitting in ice and snow <endseq>',
 '<startseq> dirty men working on a car <endseq>',
 '<startseq> a man dressed in martial arts clothing is breaking a piece of wood with his foot as other students watch <endseq>',
 '<startseq> a man with brown hair is wearing gray pants and a black belt and is sitting at a table with four other people <endseq>',
 '<startseq> there is a man in black standing near vehicles and a camper setting up video equipment <endseq>']
```

In [41]:

```
test_seqs = tokenizer.texts_to_sequences(test_values)
```

In [42]:

```
test_pad_captions = padding_sequences(sequences = test_seqs, max_length = max_length + 2)
print('Shape of pad_captions:', test_pad_captions.shape)
test_pad_captions[:3]
```

```
Shape of pad_captions: (1500, 30)
```

Out[42]:

```
array([[   3,    2,    9,   12,    2, 1385,    8, 5228,   12,   30,  430,
         816,   27,   40,   78, 1044,   11,  268,    4,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0],
       [   3,   13,   18,    7,    2,  250, 1998,   16,  646,    2, 1481,
         109,   45,   11,    6,   18,  627, 2751,    4,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0],
       [   3,    2,    9,    8,  543,    6,   43,   10,    5,    6,  499,
        1616,  677,  960,    2,   86,  179,  157,    4,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0]], dtype=int32)
```

In [43]:

```
test_image_names = []

for i in test_images:
  test_image_names.append(path+i)
```

In [44]:

```
base_model = InceptionV3(include_top= False, weights = 'imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 1s 0us/step
```

In [45]:

```
model = Model(base_model.input, base_model.layers[-1].output)

model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, None, None,  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, None, None, 3 864         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, None, None, 3 96          conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, None, None, 3 0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, None, None, 3 9216        activation[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, None, None, 3 96          conv2d_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, None, None, 3 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, None, None, 6 18432       activation_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, None, None, 6 192         conv2d_2[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, None, None, 6 0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, None, None, 6 0           activation_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, None, None, 8 5120        max_pooling2d[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, None, None, 8 240         conv2d_3[0][0]
__________________________________________________________________________________________________
activation_3 (Activation)       (None, None, None, 8 0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, None, None, 1 138240      activation_3[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, None, None, 1 576         conv2d_4[0][0]
__________________________________________________________________________________________________
activation_4 (Activation)       (None, None, None, 1 0           batch_normalization_4[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, None, None, 1 0           activation_4[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, None, None, 6 12288       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, None, None, 6 192         conv2d_8[0][0]
__________________________________________________________________________________________________
activation_8 (Activation)       (None, None, None, 6 0           batch_normalization_8[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, None, None, 4 9216        max_pooling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, None, None, 9 55296       activation_8[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, None, None, 4 144         conv2d_6[0][0]
__________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, None, None, 9 288         conv2d_9[0][0]
__________________________________________________________________________________________________
activation_6 (Activation)       (None, None, None, 4 0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
activation_9 (Activation)       (None, None, None, 9 0           batch_normalization_9[0][0]
__________________________________________________________________________________________________
average_pooling2d (AveragePooli (None, None, None, 1 0           max_pooling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, None, None, 6 12288       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, None, None, 6 76800       activation_6[0][0]
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, None, None, 9 82944       activation_9[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, None, None, 3 6144        average_pooling2d[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, None, None, 6 192         conv2d_5[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, None, None, 6 192         conv2d_7[0][0]
__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, None, None, 9 288         conv2d_10[0][0]
__________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, None, None, 3 96          conv2d_11[0][0]
__________________________________________________________________________________________________
activation_5 (Activation)       (None, None, None, 6 0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
activation_7 (Activation)       (None, None, None, 6 0           batch_normalization_7[0][0]
__________________________________________________________________________________________________
activation_10 (Activation)      (None, None, None, 9 0           batch_normalization_10[0][0]
```

```
activation_10 (Activation)      (None, None, None, 9 0           batch_normalization_10[0][0]
__________________________________________________________________________________________________
activation_11 (Activation)      (None, None, None, 3 0           batch_normalization_11[0][0]
__________________________________________________________________________________________________
mixed0 (Concatenate)            (None, None, None, 2 0           activation_5[0][0]
                                                                 activation_7[0][0]
                                                                 activation_10[0][0]
                                                                 activation_11[0][0]
__________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, None, None, 6 16384       mixed0[0][0]
__________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, None, None, 6 192         conv2d_15[0][0]
__________________________________________________________________________________________________
activation_15 (Activation)      (None, None, None, 6 0           batch_normalization_15[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, None, None, 4 12288       mixed0[0][0]
__________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, None, None, 9 55296       activation_15[0][0]
__________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, None, None, 4 144         conv2d_13[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, None, None, 9 288         conv2d_16[0][0]
__________________________________________________________________________________________________
activation_13 (Activation)      (None, None, None, 4 0           batch_normalization_13[0][0]
__________________________________________________________________________________________________
activation_16 (Activation)      (None, None, None, 9 0           batch_normalization_16[0][0]
__________________________________________________________________________________________________
average_pooling2d_1 (AveragePoo (None, None, None, 2 0           mixed0[0][0]
__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, None, None, 6 16384       mixed0[0][0]
__________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, None, None, 6 76800       activation_13[0][0]
__________________________________________________________________________________________________
conv2d_17 (Conv2D)              (None, None, None, 9 82944       activation_16[0][0]
__________________________________________________________________________________________________
conv2d_18 (Conv2D)              (None, None, None, 6 16384       average_pooling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, None, None, 6 192         conv2d_12[0][0]
__________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, None, None, 6 192         conv2d_14[0][0]
__________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, None, None, 9 288         conv2d_17[0][0]
__________________________________________________________________________________________________
batch_normalization_18 (BatchNo (None, None, None, 6 192         conv2d_18[0][0]
__________________________________________________________________________________________________
activation_12 (Activation)      (None, None, None, 6 0           batch_normalization_12[0][0]
__________________________________________________________________________________________________
activation_14 (Activation)      (None, None, None, 6 0           batch_normalization_14[0][0]
__________________________________________________________________________________________________
activation_17 (Activation)      (None, None, None, 9 0           batch_normalization_17[0][0]
__________________________________________________________________________________________________
activation_18 (Activation)      (None, None, None, 6 0           batch_normalization_18[0][0]
__________________________________________________________________________________________________
mixed1 (Concatenate)            (None, None, None, 2 0           activation_12[0][0]
                                                                 activation_14[0][0]
                                                                 activation_17[0][0]
                                                                 activation_18[0][0]
__________________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, None, None, 6 18432       mixed1[0][0]
__________________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, None, None, 6 192         conv2d_22[0][0]
__________________________________________________________________________________________________
activation_22 (Activation)      (None, None, None, 6 0           batch_normalization_22[0][0]
__________________________________________________________________________________________________
conv2d_20 (Conv2D)              (None, None, None, 4 13824       mixed1[0][0]
__________________________________________________________________________________________________
conv2d_23 (Conv2D)              (None, None, None, 9 55296       activation_22[0][0]
__________________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, None, None, 4 144         conv2d_20[0][0]
__________________________________________________________________________________________________
batch_normalization_23 (BatchNo (None, None, None, 9 288         conv2d_23[0][0]
__________________________________________________________________________________________________
activation_20 (Activation)      (None, None, None, 4 0           batch_normalization_20[0][0]
__________________________________________________________________________________________________
activation_23 (Activation)      (None, None, None, 9 0           batch_normalization_23[0][0]
__________________________________________________________________________________________________
average_pooling2d_2 (AveragePoo (None, None, None, 2 0           mixed1[0][0]
__________________________________________________________________________________________________
conv2d_19 (Conv2D)              (None, None, None, 6 18432       mixed1[0][0]
__________________________________________________________________________________________________
conv2d_21 (Conv2D)              (None, None, None, 6 76800       activation_20[0][0]
__________________________________________________________________________________________________
conv2d_24 (Conv2D)              (None, None, None, 9 82944       activation_23[0][0]
```

```
____________________________________________________________________________________________________
conv2d_25 (Conv2D)              (None, None, None, 6 18432       average_pooling2d_2[0][0]
____________________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, None, None, 6 192         conv2d_19[0][0]
____________________________________________________________________________________________________
batch_normalization_21 (BatchNo (None, None, None, 6 192         conv2d_21[0][0]
____________________________________________________________________________________________________
batch_normalization_24 (BatchNo (None, None, None, 9 288         conv2d_24[0][0]
____________________________________________________________________________________________________
batch_normalization_25 (BatchNo (None, None, None, 6 192         conv2d_25[0][0]
____________________________________________________________________________________________________
activation_19 (Activation)      (None, None, None, 6 0           batch_normalization_19[0][0]
____________________________________________________________________________________________________
activation_21 (Activation)      (None, None, None, 6 0           batch_normalization_21[0][0]
____________________________________________________________________________________________________
activation_24 (Activation)      (None, None, None, 9 0           batch_normalization_24[0][0]
____________________________________________________________________________________________________
activation_25 (Activation)      (None, None, None, 6 0           batch_normalization_25[0][0]
____________________________________________________________________________________________________
mixed2 (Concatenate)            (None, None, None, 2 0           activation_19[0][0]
                                                                 activation_21[0][0]
                                                                 activation_24[0][0]
                                                                 activation_25[0][0]
____________________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, None, None, 6 18432       mixed2[0][0]
____________________________________________________________________________________________________
batch_normalization_27 (BatchNo (None, None, None, 6 192         conv2d_27[0][0]
____________________________________________________________________________________________________
activation_27 (Activation)      (None, None, None, 6 0           batch_normalization_27[0][0]
____________________________________________________________________________________________________
conv2d_28 (Conv2D)              (None, None, None, 9 55296       activation_27[0][0]
____________________________________________________________________________________________________
batch_normalization_28 (BatchNo (None, None, None, 9 288         conv2d_28[0][0]
____________________________________________________________________________________________________
activation_28 (Activation)      (None, None, None, 9 0           batch_normalization_28[0][0]
____________________________________________________________________________________________________
conv2d_26 (Conv2D)              (None, None, None, 3 995328      mixed2[0][0]
____________________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, None, None, 9 82944       activation_28[0][0]
____________________________________________________________________________________________________
batch_normalization_26 (BatchNo (None, None, None, 3 1152        conv2d_26[0][0]
____________________________________________________________________________________________________
batch_normalization_29 (BatchNo (None, None, None, 9 288         conv2d_29[0][0]
____________________________________________________________________________________________________
activation_26 (Activation)      (None, None, None, 3 0           batch_normalization_26[0][0]
____________________________________________________________________________________________________
activation_29 (Activation)      (None, None, None, 9 0           batch_normalization_29[0][0]
____________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, None, None, 2 0           mixed2[0][0]
____________________________________________________________________________________________________
mixed3 (Concatenate)            (None, None, None, 7 0           activation_26[0][0]
                                                                 activation_29[0][0]
                                                                 max_pooling2d_2[0][0]
____________________________________________________________________________________________________
conv2d_34 (Conv2D)              (None, None, None, 1 98304       mixed3[0][0]
____________________________________________________________________________________________________
batch_normalization_34 (BatchNo (None, None, None, 1 384         conv2d_34[0][0]
____________________________________________________________________________________________________
activation_34 (Activation)      (None, None, None, 1 0           batch_normalization_34[0][0]
____________________________________________________________________________________________________
conv2d_35 (Conv2D)              (None, None, None, 1 114688      activation_34[0][0]
____________________________________________________________________________________________________
batch_normalization_35 (BatchNo (None, None, None, 1 384         conv2d_35[0][0]
____________________________________________________________________________________________________
activation_35 (Activation)      (None, None, None, 1 0           batch_normalization_35[0][0]
____________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, None, None, 1 98304       mixed3[0][0]
____________________________________________________________________________________________________
conv2d_36 (Conv2D)              (None, None, None, 1 114688      activation_35[0][0]
____________________________________________________________________________________________________
batch_normalization_31 (BatchNo (None, None, None, 1 384         conv2d_31[0][0]
____________________________________________________________________________________________________
batch_normalization_36 (BatchNo (None, None, None, 1 384         conv2d_36[0][0]
____________________________________________________________________________________________________
activation_31 (Activation)      (None, None, None, 1 0           batch_normalization_31[0][0]
____________________________________________________________________________________________________
activation_36 (Activation)      (None, None, None, 1 0           batch_normalization_36[0][0]
____________________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, None, None, 1 114688      activation_31[0][0]
____________________________________________________________________________________________________
conv2d_37 (Conv2D)              (None, None, None, 1 114688      activation_36[0][0]
____________________________________________________________________________________________________
batch_normalization_32 (BatchNo (None, None, None, 1 384         conv2d_32[0][0]
```

```
batch_normalization_32 (BatchNo (None, None, None, 1 384        conv2d_32[0][0]
________________________________________________________________________________
batch_normalization_37 (BatchNo (None, None, None, 1 384        conv2d_37[0][0]
________________________________________________________________________________
activation_32 (Activation)      (None, None, None, 1 0          batch_normalization_32[0][0]
________________________________________________________________________________
activation_37 (Activation)      (None, None, None, 1 0          batch_normalization_37[0][0]
________________________________________________________________________________
average_pooling2d_3 (AveragePoo (None, None, None, 7 0          mixed3[0][0]
________________________________________________________________________________
conv2d_30 (Conv2D)              (None, None, None, 1 147456     mixed3[0][0]
________________________________________________________________________________
conv2d_33 (Conv2D)              (None, None, None, 1 172032     activation_32[0][0]
________________________________________________________________________________
conv2d_38 (Conv2D)              (None, None, None, 1 172032     activation_37[0][0]
________________________________________________________________________________
conv2d_39 (Conv2D)              (None, None, None, 1 147456     average_pooling2d_3[0][0]
________________________________________________________________________________
batch_normalization_30 (BatchNo (None, None, None, 1 576        conv2d_30[0][0]
________________________________________________________________________________
batch_normalization_33 (BatchNo (None, None, None, 1 576        conv2d_33[0][0]
________________________________________________________________________________
batch_normalization_38 (BatchNo (None, None, None, 1 576        conv2d_38[0][0]
________________________________________________________________________________
batch_normalization_39 (BatchNo (None, None, None, 1 576        conv2d_39[0][0]
________________________________________________________________________________
activation_30 (Activation)      (None, None, None, 1 0          batch_normalization_30[0][0]
________________________________________________________________________________
activation_33 (Activation)      (None, None, None, 1 0          batch_normalization_33[0][0]
________________________________________________________________________________
activation_38 (Activation)      (None, None, None, 1 0          batch_normalization_38[0][0]
________________________________________________________________________________
activation_39 (Activation)      (None, None, None, 1 0          batch_normalization_39[0][0]
________________________________________________________________________________
mixed4 (Concatenate)            (None, None, None, 7 0          activation_30[0][0]
                                                                 activation_33[0][0]
                                                                 activation_38[0][0]
                                                                 activation_39[0][0]
________________________________________________________________________________
conv2d_44 (Conv2D)              (None, None, None, 1 122880     mixed4[0][0]
________________________________________________________________________________
batch_normalization_44 (BatchNo (None, None, None, 1 480        conv2d_44[0][0]
________________________________________________________________________________
activation_44 (Activation)      (None, None, None, 1 0          batch_normalization_44[0][0]
________________________________________________________________________________
conv2d_45 (Conv2D)              (None, None, None, 1 179200     activation_44[0][0]
________________________________________________________________________________
batch_normalization_45 (BatchNo (None, None, None, 1 480        conv2d_45[0][0]
________________________________________________________________________________
activation_45 (Activation)      (None, None, None, 1 0          batch_normalization_45[0][0]
________________________________________________________________________________
conv2d_41 (Conv2D)              (None, None, None, 1 122880     mixed4[0][0]
________________________________________________________________________________
conv2d_46 (Conv2D)              (None, None, None, 1 179200     activation_45[0][0]
________________________________________________________________________________
batch_normalization_41 (BatchNo (None, None, None, 1 480        conv2d_41[0][0]
________________________________________________________________________________
batch_normalization_46 (BatchNo (None, None, None, 1 480        conv2d_46[0][0]
________________________________________________________________________________
activation_41 (Activation)      (None, None, None, 1 0          batch_normalization_41[0][0]
________________________________________________________________________________
activation_46 (Activation)      (None, None, None, 1 0          batch_normalization_46[0][0]
________________________________________________________________________________
conv2d_42 (Conv2D)              (None, None, None, 1 179200     activation_41[0][0]
________________________________________________________________________________
conv2d_47 (Conv2D)              (None, None, None, 1 179200     activation_46[0][0]
________________________________________________________________________________
batch_normalization_42 (BatchNo (None, None, None, 1 480        conv2d_42[0][0]
________________________________________________________________________________
batch_normalization_47 (BatchNo (None, None, None, 1 480        conv2d_47[0][0]
________________________________________________________________________________
activation_42 (Activation)      (None, None, None, 1 0          batch_normalization_42[0][0]
________________________________________________________________________________
activation_47 (Activation)      (None, None, None, 1 0          batch_normalization_47[0][0]
________________________________________________________________________________
average_pooling2d_4 (AveragePoo (None, None, None, 7 0          mixed4[0][0]
________________________________________________________________________________
conv2d_40 (Conv2D)              (None, None, None, 1 147456     mixed4[0][0]
________________________________________________________________________________
conv2d_43 (Conv2D)              (None, None, None, 1 215040     activation_42[0][0]
________________________________________________________________________________
conv2d_48 (Conv2D)              (None, None, None, 1 215040     activation_47[0][0]
________________________________________________________________________________
conv2d_49 (Conv2D)              (None, None, None, 1 147456     average_pooling2d_4[0][0]
```

conv2d_49 (Conv2D)	(None, None, None, 1	147456	average_pooling2d_4[0][0]
batch_normalization_40 (BatchNo	(None, None, None, 1	576	conv2d_40[0][0]
batch_normalization_43 (BatchNo	(None, None, None, 1	576	conv2d_43[0][0]
batch_normalization_48 (BatchNo	(None, None, None, 1	576	conv2d_48[0][0]
batch_normalization_49 (BatchNo	(None, None, None, 1	576	conv2d_49[0][0]
activation_40 (Activation)	(None, None, None, 1	0	batch_normalization_40[0][0]
activation_43 (Activation)	(None, None, None, 1	0	batch_normalization_43[0][0]
activation_48 (Activation)	(None, None, None, 1	0	batch_normalization_48[0][0]
activation_49 (Activation)	(None, None, None, 1	0	batch_normalization_49[0][0]
mixed5 (Concatenate)	(None, None, None, 7	0	activation_40[0][0] activation_43[0][0] activation_48[0][0] activation_49[0][0]
conv2d_54 (Conv2D)	(None, None, None, 1	122880	mixed5[0][0]
batch_normalization_54 (BatchNo	(None, None, None, 1	480	conv2d_54[0][0]
activation_54 (Activation)	(None, None, None, 1	0	batch_normalization_54[0][0]
conv2d_55 (Conv2D)	(None, None, None, 1	179200	activation_54[0][0]
batch_normalization_55 (BatchNo	(None, None, None, 1	480	conv2d_55[0][0]
activation_55 (Activation)	(None, None, None, 1	0	batch_normalization_55[0][0]
conv2d_51 (Conv2D)	(None, None, None, 1	122880	mixed5[0][0]
conv2d_56 (Conv2D)	(None, None, None, 1	179200	activation_55[0][0]
batch_normalization_51 (BatchNo	(None, None, None, 1	480	conv2d_51[0][0]
batch_normalization_56 (BatchNo	(None, None, None, 1	480	conv2d_56[0][0]
activation_51 (Activation)	(None, None, None, 1	0	batch_normalization_51[0][0]
activation_56 (Activation)	(None, None, None, 1	0	batch_normalization_56[0][0]
conv2d_52 (Conv2D)	(None, None, None, 1	179200	activation_51[0][0]
conv2d_57 (Conv2D)	(None, None, None, 1	179200	activation_56[0][0]
batch_normalization_52 (BatchNo	(None, None, None, 1	480	conv2d_52[0][0]
batch_normalization_57 (BatchNo	(None, None, None, 1	480	conv2d_57[0][0]
activation_52 (Activation)	(None, None, None, 1	0	batch_normalization_52[0][0]
activation_57 (Activation)	(None, None, None, 1	0	batch_normalization_57[0][0]
average_pooling2d_5 (AveragePoo	(None, None, None, 7	0	mixed5[0][0]
conv2d_50 (Conv2D)	(None, None, None, 1	147456	mixed5[0][0]
conv2d_53 (Conv2D)	(None, None, None, 1	215040	activation_52[0][0]
conv2d_58 (Conv2D)	(None, None, None, 1	215040	activation_57[0][0]
conv2d_59 (Conv2D)	(None, None, None, 1	147456	average_pooling2d_5[0][0]
batch_normalization_50 (BatchNo	(None, None, None, 1	576	conv2d_50[0][0]
batch_normalization_53 (BatchNo	(None, None, None, 1	576	conv2d_53[0][0]
batch_normalization_58 (BatchNo	(None, None, None, 1	576	conv2d_58[0][0]
batch_normalization_59 (BatchNo	(None, None, None, 1	576	conv2d_59[0][0]
activation_50 (Activation)	(None, None, None, 1	0	batch_normalization_50[0][0]
activation_53 (Activation)	(None, None, None, 1	0	batch_normalization_53[0][0]
activation_58 (Activation)	(None, None, None, 1	0	batch_normalization_58[0][0]

activation_59 (Activation)	(None, None, None, 1	0	batch_normalization_59[0][0]
mixed6 (Concatenate)	(None, None, None, 7	0	activation_50[0][0] activation_53[0][0] activation_58[0][0] activation_59[0][0]
conv2d_64 (Conv2D)	(None, None, None, 1	147456	mixed6[0][0]
batch_normalization_64 (BatchNo	(None, None, None, 1	576	conv2d_64[0][0]
activation_64 (Activation)	(None, None, None, 1	0	batch_normalization_64[0][0]
conv2d_65 (Conv2D)	(None, None, None, 1	258048	activation_64[0][0]
batch_normalization_65 (BatchNo	(None, None, None, 1	576	conv2d_65[0][0]
activation_65 (Activation)	(None, None, None, 1	0	batch_normalization_65[0][0]
conv2d_61 (Conv2D)	(None, None, None, 1	147456	mixed6[0][0]
conv2d_66 (Conv2D)	(None, None, None, 1	258048	activation_65[0][0]
batch_normalization_61 (BatchNo	(None, None, None, 1	576	conv2d_61[0][0]
batch_normalization_66 (BatchNo	(None, None, None, 1	576	conv2d_66[0][0]
activation_61 (Activation)	(None, None, None, 1	0	batch_normalization_61[0][0]
activation_66 (Activation)	(None, None, None, 1	0	batch_normalization_66[0][0]
conv2d_62 (Conv2D)	(None, None, None, 1	258048	activation_61[0][0]
conv2d_67 (Conv2D)	(None, None, None, 1	258048	activation_66[0][0]
batch_normalization_62 (BatchNo	(None, None, None, 1	576	conv2d_62[0][0]
batch_normalization_67 (BatchNo	(None, None, None, 1	576	conv2d_67[0][0]
activation_62 (Activation)	(None, None, None, 1	0	batch_normalization_62[0][0]
activation_67 (Activation)	(None, None, None, 1	0	batch_normalization_67[0][0]
average_pooling2d_6 (AveragePoo	(None, None, None, 7	0	mixed6[0][0]
conv2d_60 (Conv2D)	(None, None, None, 1	147456	mixed6[0][0]
conv2d_63 (Conv2D)	(None, None, None, 1	258048	activation_62[0][0]
conv2d_68 (Conv2D)	(None, None, None, 1	258048	activation_67[0][0]
conv2d_69 (Conv2D)	(None, None, None, 1	147456	average_pooling2d_6[0][0]
batch_normalization_60 (BatchNo	(None, None, None, 1	576	conv2d_60[0][0]
batch_normalization_63 (BatchNo	(None, None, None, 1	576	conv2d_63[0][0]
batch_normalization_68 (BatchNo	(None, None, None, 1	576	conv2d_68[0][0]
batch_normalization_69 (BatchNo	(None, None, None, 1	576	conv2d_69[0][0]
activation_60 (Activation)	(None, None, None, 1	0	batch_normalization_60[0][0]
activation_63 (Activation)	(None, None, None, 1	0	batch_normalization_63[0][0]
activation_68 (Activation)	(None, None, None, 1	0	batch_normalization_68[0][0]
activation_69 (Activation)	(None, None, None, 1	0	batch_normalization_69[0][0]
mixed7 (Concatenate)	(None, None, None, 7	0	activation_60[0][0] activation_63[0][0] activation_68[0][0] activation_69[0][0]
conv2d_72 (Conv2D)	(None, None, None, 1	147456	mixed7[0][0]
batch_normalization_72 (BatchNo	(None, None, None, 1	576	conv2d_72[0][0]
activation_72 (Activation)	(None, None, None, 1	0	batch_normalization_72[0][0]
conv2d_73 (Conv2D)	(None, None, None, 1	258048	activation_72[0][0]
batch_normalization_73 (BatchNo	(None, None, None, 1	576	conv2d_73[0][0]

```
batch_normalization_73 (BatchNo (None, None, None, 1 576        conv2d_73[0][0]
__________________________________________________________________________________________________
activation_73 (Activation)      (None, None, None, 1 0           batch_normalization_73[0][0]
__________________________________________________________________________________________________
conv2d_70 (Conv2D)              (None, None, None, 1 147456      mixed7[0][0]
__________________________________________________________________________________________________
conv2d_74 (Conv2D)              (None, None, None, 1 258048      activation_73[0][0]
__________________________________________________________________________________________________
batch_normalization_70 (BatchNo (None, None, None, 1 576         conv2d_70[0][0]
__________________________________________________________________________________________________
batch_normalization_74 (BatchNo (None, None, None, 1 576         conv2d_74[0][0]
__________________________________________________________________________________________________
activation_70 (Activation)      (None, None, None, 1 0           batch_normalization_70[0][0]
__________________________________________________________________________________________________
activation_74 (Activation)      (None, None, None, 1 0           batch_normalization_74[0][0]
__________________________________________________________________________________________________
conv2d_71 (Conv2D)              (None, None, None, 3 552960      activation_70[0][0]
__________________________________________________________________________________________________
conv2d_75 (Conv2D)              (None, None, None, 1 331776      activation_74[0][0]
__________________________________________________________________________________________________
batch_normalization_71 (BatchNo (None, None, None, 3 960         conv2d_71[0][0]
__________________________________________________________________________________________________
batch_normalization_75 (BatchNo (None, None, None, 1 576         conv2d_75[0][0]
__________________________________________________________________________________________________
activation_71 (Activation)      (None, None, None, 3 0           batch_normalization_71[0][0]
__________________________________________________________________________________________________
activation_75 (Activation)      (None, None, None, 1 0           batch_normalization_75[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, None, None, 7 0           mixed7[0][0]
__________________________________________________________________________________________________
mixed8 (Concatenate)            (None, None, None, 1 0           activation_71[0][0]
                                                                 activation_75[0][0]
                                                                 max_pooling2d_3[0][0]
__________________________________________________________________________________________________
conv2d_80 (Conv2D)              (None, None, None, 4 573440      mixed8[0][0]
__________________________________________________________________________________________________
batch_normalization_80 (BatchNo (None, None, None, 4 1344        conv2d_80[0][0]
__________________________________________________________________________________________________
activation_80 (Activation)      (None, None, None, 4 0           batch_normalization_80[0][0]
__________________________________________________________________________________________________
conv2d_77 (Conv2D)              (None, None, None, 3 491520      mixed8[0][0]
__________________________________________________________________________________________________
conv2d_81 (Conv2D)              (None, None, None, 3 1548288     activation_80[0][0]
__________________________________________________________________________________________________
batch_normalization_77 (BatchNo (None, None, None, 3 1152        conv2d_77[0][0]
__________________________________________________________________________________________________
batch_normalization_81 (BatchNo (None, None, None, 3 1152        conv2d_81[0][0]
__________________________________________________________________________________________________
activation_77 (Activation)      (None, None, None, 3 0           batch_normalization_77[0][0]
__________________________________________________________________________________________________
activation_81 (Activation)      (None, None, None, 3 0           batch_normalization_81[0][0]
__________________________________________________________________________________________________
conv2d_78 (Conv2D)              (None, None, None, 3 442368      activation_77[0][0]
__________________________________________________________________________________________________
conv2d_79 (Conv2D)              (None, None, None, 3 442368      activation_77[0][0]
__________________________________________________________________________________________________
conv2d_82 (Conv2D)              (None, None, None, 3 442368      activation_81[0][0]
__________________________________________________________________________________________________
conv2d_83 (Conv2D)              (None, None, None, 3 442368      activation_81[0][0]
__________________________________________________________________________________________________
average_pooling2d_7 (AveragePoo (None, None, None, 1 0           mixed8[0][0]
__________________________________________________________________________________________________
conv2d_76 (Conv2D)              (None, None, None, 3 409600      mixed8[0][0]
__________________________________________________________________________________________________
batch_normalization_78 (BatchNo (None, None, None, 3 1152        conv2d_78[0][0]
__________________________________________________________________________________________________
batch_normalization_79 (BatchNo (None, None, None, 3 1152        conv2d_79[0][0]
__________________________________________________________________________________________________
batch_normalization_82 (BatchNo (None, None, None, 3 1152        conv2d_82[0][0]
__________________________________________________________________________________________________
batch_normalization_83 (BatchNo (None, None, None, 3 1152        conv2d_83[0][0]
__________________________________________________________________________________________________
conv2d_84 (Conv2D)              (None, None, None, 1 245760      average_pooling2d_7[0][0]
__________________________________________________________________________________________________
batch_normalization_76 (BatchNo (None, None, None, 3 960         conv2d_76[0][0]
__________________________________________________________________________________________________
activation_78 (Activation)      (None, None, None, 3 0           batch_normalization_78[0][0]
__________________________________________________________________________________________________
activation_79 (Activation)      (None, None, None, 3 0           batch_normalization_79[0][0]
__________________________________________________________________________________________________
activation_82 (Activation)      (None, None, None, 3 0           batch_normalization_82[0][0]
__________________________________________________________________________________________________
activation_83 (Activation)      (None, None, None, 3 0           batch_normalization_83[0][0]
```

batch_normalization_84 (BatchNo	(None, None, None, 1	576	conv2d_84[0][0]
activation_76 (Activation)	(None, None, None, 3	0	batch_normalization_76[0][0]
mixed9_0 (Concatenate)	(None, None, None, 7	0	activation_78[0][0]
			activation_79[0][0]
concatenate (Concatenate)	(None, None, None, 7	0	activation_82[0][0]
			activation_83[0][0]
activation_84 (Activation)	(None, None, None, 1	0	batch_normalization_84[0][0]
mixed9 (Concatenate)	(None, None, None, 2	0	activation_76[0][0]
			mixed9_0[0][0]
			concatenate[0][0]
			activation_84[0][0]
conv2d_89 (Conv2D)	(None, None, None, 4	917504	mixed9[0][0]
batch_normalization_89 (BatchNo	(None, None, None, 4	1344	conv2d_89[0][0]
activation_89 (Activation)	(None, None, None, 4	0	batch_normalization_89[0][0]
conv2d_86 (Conv2D)	(None, None, None, 3	786432	mixed9[0][0]
conv2d_90 (Conv2D)	(None, None, None, 3	1548288	activation_89[0][0]
batch_normalization_86 (BatchNo	(None, None, None, 3	1152	conv2d_86[0][0]
batch_normalization_90 (BatchNo	(None, None, None, 3	1152	conv2d_90[0][0]
activation_86 (Activation)	(None, None, None, 3	0	batch_normalization_86[0][0]
activation_90 (Activation)	(None, None, None, 3	0	batch_normalization_90[0][0]
conv2d_87 (Conv2D)	(None, None, None, 3	442368	activation_86[0][0]
conv2d_88 (Conv2D)	(None, None, None, 3	442368	activation_86[0][0]
conv2d_91 (Conv2D)	(None, None, None, 3	442368	activation_90[0][0]
conv2d_92 (Conv2D)	(None, None, None, 3	442368	activation_90[0][0]
average_pooling2d_8 (AveragePoo	(None, None, None, 2	0	mixed9[0][0]
conv2d_85 (Conv2D)	(None, None, None, 3	655360	mixed9[0][0]
batch_normalization_87 (BatchNo	(None, None, None, 3	1152	conv2d_87[0][0]
batch_normalization_88 (BatchNo	(None, None, None, 3	1152	conv2d_88[0][0]
batch_normalization_91 (BatchNo	(None, None, None, 3	1152	conv2d_91[0][0]
batch_normalization_92 (BatchNo	(None, None, None, 3	1152	conv2d_92[0][0]
conv2d_93 (Conv2D)	(None, None, None, 1	393216	average_pooling2d_8[0][0]
batch_normalization_85 (BatchNo	(None, None, None, 3	960	conv2d_85[0][0]
activation_87 (Activation)	(None, None, None, 3	0	batch_normalization_87[0][0]
activation_88 (Activation)	(None, None, None, 3	0	batch_normalization_88[0][0]
activation_91 (Activation)	(None, None, None, 3	0	batch_normalization_91[0][0]
activation_92 (Activation)	(None, None, None, 3	0	batch_normalization_92[0][0]
batch_normalization_93 (BatchNo	(None, None, None, 1	576	conv2d_93[0][0]
activation_85 (Activation)	(None, None, None, 3	0	batch_normalization_85[0][0]
mixed9_1 (Concatenate)	(None, None, None, 7	0	activation_87[0][0]
			activation_88[0][0]
concatenate_1 (Concatenate)	(None, None, None, 7	0	activation_91[0][0]
			activation_92[0][0]
activation_93 (Activation)	(None, None, None, 1	0	batch_normalization_93[0][0]
mixed10 (Concatenate)	(None, None, None, 2	0	activation_85[0][0]
			mixed9_1[0][0]
			concatenate_1[0][0]

concatenate_1[0][0]
activation_93[0][0]
==
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
____

Caching the features extracted from InceptionV3

- You will pre-process each image with InceptionV3 and cache the output to disk. Caching the output in RAM would be faster but also memory intensive, requiring 8 *8* 2048 floats per image. At the time of writing, this exceeds the memory limitations of Colab (currently 12GB of memory).
- Performance could be improved with a more sophisticated caching strategy (for example, by sharding the images to reduce random access disk I/O), but that would require more code.
- The caching will take about 10 minutes to run in Colab with a GPU.

In [46]:

```
def preprocess_image(image_path):

  img = tf.io.read_file(image_path)
  img = tf.image.decode_jpeg(  img, channels=3)
  img = tf.image.resize(  img, (299, 299))

  # The preprocess_input function is meant to adequate your image to the format the model requires.
  # You don't need to worry about the internal details of preprocess_input.
  # But ideally, you should load images with the keras functions for that (so you guarantee that the images
  # you load are compatible with preprocess_input).
  img = tf.keras.applications.inception_v3.preprocess_input(img)

  return img, image_path
```

In [47]:

```
# Get unique images
encode_train = sorted(set(image_names))

# Feel free to change batch_size according to your system configuration
inception= tf.data.Dataset.from_tensor_slices(encode_train)
inception = inception.map( preprocess_image, num_parallel_calls= tf.data.experimental.AUTOTUNE).batch(64)

# <BatchDataset shapes: ((None, 299, 299, 3), (None,)), types: (tf.float32, tf.string)>
for img, path in tqdm(inception):
  # below shape shall be (1, 8, 8, 2048) => (31783, 8, 8, 2048) in total
  batch_features= model(img)
  # squashing the shape to (31783, 64, 2048)
  # below reshape shall be (1, 64, 2048) as (8*8*2048 floats per image as above described in text)
  batch_features = tf.reshape(tensor = batch_features, shape= (batch_features.shape[0], -1, batch_features.shape[3]))

  for b_f, p in zip(batch_features, path):

    path_of_feature = p.numpy().decode("utf-8")
    np.save(path_of_feature, b_f.numpy())
```

```
94it [00:54,  1.73it/s]
```

In [48]:

```
from pickle import dump
# save to file
dump(image_names, open('train_image_names.pkl', 'wb'))
dump(pad_captions, open('train_pad_captions.pkl', 'wb'))

dump(test_image_names, open('test_image_names.pkl', 'wb'))
dump(test_pad_captions, open('test_pad_captions.pkl', 'wb'))
```

In [49]:

```
from pickle import load
# load the files
train_image_names = load(open('train_image_names.pkl', 'rb'))
train_pad_captions = load(open('train_pad_captions.pkl', 'rb'))

test_image_names = load(open('test_image_names.pkl', 'rb'))
test_pad_captions = load(open('test_pad_captions.pkl', 'rb'))
```

In [50]:

```
print('Train images:', len(train_image_names))
print('Test images:', len(test_image_names))
print('Train captions:', len(train_pad_captions))
print('Test captions:', len(test_pad_captions))
```

```
Train images: 28500
Test images: 1500
Train captions: 28500
Test captions: 1500
```

Prefetching

- Prefetching overlaps the preprocessing and model execution of a training step. While the model is executing training step s, the input pipeline is reading the data for step s+1. Doing so reduces the step time to the maximum (as opposed to the sum) of the training and the time it takes to extract the data.
- The "tf.data" API provides the "tf.data.Dataset.prefetch" transformation. It can be used to decouple the time when data is produced from the time when data is consumed. In particular, the transformation uses a background thread and an internal buffer to prefetch elements from the input dataset ahead of the time they are requested. The number of elements to prefetch should be equal to (or possibly greater than) the number of batches consumed by a single training step. You could either manually tune this value, or set it to "tf.data.experimental.AUTOTUNE" which will prompt the "tf.data" runtime to tune the value dynamically at runtime.

In [51]:

```
# We need to convert images to tensor to create a dataset

batch_size= 64

# Buffer size to shuffle the dataset
# (TF data is designed to work with possibly infinite sequences,
# so it doesn't attempt to shuffle the entire sequence in memory. Instead,
# it maintains a buffer in which it shuffles elements).
buffer_size= 1000

def make_numpy(img, cap):
  # tensors: A dataset element, with each component having the same size in the first dimension.
  img_tensor = np.load(img.decode('utf-8')+'.npy')
  return img_tensor, cap

def create_dataset(img, cap):
  # The given tensors are sliced along their first dimension. This operation preserves the structure of the input tensors,
  # removing the first dimension of each tensor and using it as the dataset dimension.
  # All input tensors must have the same size in their first dimensions.
  dataset = tf.data.Dataset.from_tensor_slices((img, cap))

  # Use map to load the numpy files in parallel
  # inp: A list of `tf.Tensor` objects.
  # Tout: A list or tuple of tensorflow data types or a single tensorflow data
  # type if there is only one, indicating what `func` returns.
  dataset = dataset.map(lambda a, b: tf.numpy_function(func= make_numpy, inp= [a, b], Tout= [tf.float32, tf.int32]),
                        num_parallel_calls = tf.data.experimental.AUTOTUNE)

  # Shuffle the data
  dataset = dataset.shuffle(buffer_size= buffer_size)
  dataset = dataset.batch(batch_size= batch_size, drop_remainder= True)
  dataset = dataset.prefetch(buffer_size = tf.data.experimental.AUTOTUNE)

  return dataset
```

In [52]:

```
# https://github.com/tensorflow/tensorflow/issues/32912 (but we got shape seen below, phew)
# We create Train and Test datasets

train_dataset = create_dataset(img = train_image_names, cap= train_pad_captions)
test_dataset = create_dataset(img = test_image_names, cap = test_pad_captions)
```

Model

Fun fact: the decoder below is identical to the one in the example for https://www.tensorflow.org/tutorials/text/nmt_with_attention

Part last: the decoder below is identical to the one in the example for https://www.tensorflow.org/tutorials/text/nmt_with_attention.

- In this example, you extract the features from the lower convolutional layer of InceptionV3 giving us a vector of shape (8, 8, 2048).
- You squash that to a shape of (64, 2048).
- This vector is then passed through the CNN Encoder (which consists of a single Fully connected layer).
- The RNN (here GRU) attends over the image to predict the next word.

In [53]:

```
# Crosschecking the dataset and their shapes (batchsize, (8*8), 2048), (batchsize, len(single caption))

for i , j in train_dataset.take(count= 1):
  print('The shape of image converted to numpy array along with batch size:', i.shape)
  print('The shape of captions along with batch size:' , j.shape,'\n')

# Crosschecking the dataset and their shapes (batchsize, (8*8), 2048), (batchsize, len(single caption))

for i , j in test_dataset.take(count= 1):
  print('The shape of image converted to numpy array along with batch size:', i.shape)
  print('The shape of captions along with batch size:' , j.shape)
```

```
The shape of image converted to numpy array along with batch size: (64, 64, 2048)
The shape of captions along with batch size: (64, 30)

The shape of image converted to numpy array along with batch size: (64, 64, 2048)
The shape of captions along with batch size: (64, 30)
```

In [54]:

```
# Parameters

embedding_dim = 300
units = 1024

# https://stackoverflow.com/a/60294856/10219869 (if specify oov token, then use + 2)
# https://stackoverflow.com/a/58535412/10219869

vocab_size = len(word_corpus) + 1
num_steps = len(train_pad_captions) // batch_size # 468

# Shape of the vector extracted from InceptionV3 is (8*8* 2048) squashed to (64, 2048)
# These two variables represent that vector shape
features_shape = 2048                # these many filters
attention_features_shape = 64          # 8 x 8 is size of filter with all imp info

epochs= 50

print('Batch size:', batch_size)
print('Embedding dimentions:', embedding_dim)
print('Number of units to use in LSTM:', units)
print('Vocab size: ', vocab_size)
print('Steps per epoch: ', num_steps)
print('Image features: ', features_shape)
print('Image attention features: ', attention_features_shape)
print('Number of epochs: ', epochs)
```

```
Batch size: 64
Embedding dimentions: 300
Number of units to use in LSTM: 1024
Vocab size:  9212
Steps per epoch:  445
Image features:  2048
Image attention features:  64
Number of epochs:  50
```

Encoder CNN

In [55]:

```
class Encoder_CNN(tf.keras.Model):
  # Since you have already extracted the features and dumped it using pickle
  # This encoder passes those features through a Fully connected layer
  def __init__(self, unit):
    super(Encoder_CNN, self).__init__()
    # shape before fc is (64, 64, 2048)
    self.fc1 = Dense(units= units, activation= 'relu', name= 'Encoder')
    self.dropout = Dropout(rate= 0.5)
```

```
  self.fc2 = Dense(units= units, activation= 'relu')

  # shape after  fc is (64, 64, 1024)

 def call(self, x):
  x = self.fc1(x)
  #x = self.dropout(x)
  #x = self.fc2(x)

  return x
```

Global Bahdanau Attention

- Attention is placed only on few source positions (3, 4, 5)
- below 'ht' is hidden target, 'hs' is hidden source

the equations to know exactly what we need to do. Here is how we're gonna compute the alignment vector:

$$\begin{aligned} \boldsymbol{a}_t(s) &= \operatorname{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \\ &= \frac{\exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'} \exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \end{aligned}$$

Equation 1: Equation for alignment vector

Luong attention mechanism proposed three types of *score* function: *dot*, *general* and *concat*:

$$\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

Equation 2: Score functions

Since I'm not going to talk about Bahdanau-style attention, here's the key differences between the two:

- Bahdanau attention mechanism proposed only the *concat* score function
- Luong-style attention uses the current decoder output to compute the alignment vector, whereas Bahdanau's uses the output of the previous time step

In [56]:

```
# insert above image via https://stackoverflow.com/a/62337161/10219869
# image reference https://machinetalk.org/2019/03/29/neural-machine-translation-with-attention-mechanism/

class Bahdanau_Attention(tf.keras.Model):

 def __init__(self, units):
  super(Bahdanau_Attention, self).__init__()

  self.W1 = Dense(units= units) # 1024
  self.Ws = Dense(units= 1, activation= 'tanh')

 def call(self, features, hidden):
  # Features(CNN_encoder output) shape ==  (64, 64, 1024).
  # We consider hidden[0] which is hidden state == (64, 1024), and hidden[1] is cell state == (64, 1024) which we are not using.
  # Starting hidden shape (which is tf.zeros initialized hidden state from decoder) == (batch_size, hidden_size) (64, 1024).
  hidden = hidden[0]

  # reshaping hidden state with timesteps.
  # hidden_with_time_axis shape == (batch_size, 1, hidden_size) (64, 1, 1024) [1 repeats 31 times using teacher forcing].
  hidden_with_time_axis = tf.reshape(tensor= hidden, shape= (hidden.shape[0], 1, hidden.shape[1]))

  # hidden source state --> Hs == (64, 64, 1024) to dense (64, 64, 1024).
  Hs = self.W1(features)
  # hidden target state --> Ht == (64, 1, 1024) to dense (64, 1, 1024).
  Ht = self.W1(hidden_with_time_axis)
```

```python
    Ht = self.W1(hidden_with_time_axis)

    # score shape from (64, 64, 1024) to (64, 64, 1) as 1024 (Hs + Ht) is passing through dense layer of 1.
    score = self.Ws(Hs + Ht)

    # attention_weights == (64, 64, 1).
    # you get 1 at the last axis because you are applying score to self.Ws for predictive distribution.
    # this gives importance when we multiply with features below to generate context vector.
    attention_weights = tf.nn.softmax(logits = score, axis= 1)

    # context_vector shape == (64, 64, 1024) broadcasting happens [(64, 64, 1) * (64, 64, 1024)]
    # This is global attention, hence multiplying with all timesteps (64) of features.
    context_vector = attention_weights * features

    # context_vector final shape == (64, 1024) col wise
    context_vector = tf.reduce_sum(input_tensor = context_vector, axis= 1)

    return context_vector, attention_weights
```

LSTM Decoder

In [57]:

```python
class Decoder_RNN(tf.keras.Model):

  def __init__(self, embedding_dim, units, vocab_size):
    super(Decoder_RNN, self).__init__()

    self.embedding = Embedding(input_dim= vocab_size, output_dim= embedding_dim )
    self.lstm1 = LSTM(units = units, return_sequences= True, return_state= True, recurrent_initializer='glorot_uniform',
                      kernel_initializer= 'he_normal')
    self.lstm2 = LSTM(units = units, return_sequences= True, return_state= True, recurrent_initializer='glorot_uniform',
                      kernel_initializer= 'he_normal')
    self.Lh = Dense(units= embedding_dim) # 300
    self.Lz = Dense(units= embedding_dim) # 300
    # Activation  = 'exponential' according to paper but not using any as it is giving NaN values
    self.Lo = Dense(units= vocab_size)#, activation= 'exponential') # 9212

    self.attention = Bahdanau_Attention(units)

  def call(self, x, features, hidden):
    # defining attention as a separate model
    context_vector, attention_weights = self.attention(features, hidden)

    # x shape before is dec_input which is (64, 1) coming from training_step function
    # x shape after passing through embedding == (batch_size, 1, embedding_dim) (64, 1, 300)
    E = self.embedding(x)

    # passing the concatenated vector to the LSTM
    output1, hidden_state1, cell_state1 = self.lstm1(E)        # o/p shape (64, 1, 1024), (64, 1024), (64, 1024)
    output2, hidden_state2, cell_state2 = self.lstm2(output1)  # o/p shape (64, 1, 1024), (64, 1024), (64, 1024)

    # E shape == (64, 300)
    E = tf.reshape(tensor= E, shape= (E.shape[0], E.shape[2]))
    # Lh shape == (64, 300) [(64, 1024) --> (64, 300)]
    Lh = self.Lh(hidden_state2)
    # Lz shape == (64, 300) [(64, 1024) --> (64, 300)]
    Lz = self.Lz(context_vector)


    # x shape == (64, 9212) [(64, 300) --> (64, 9212)] elementwise addition
    x = self.Lo(E + Lh + Lz)

    return x, (hidden_state2, cell_state2), attention_weights

  # tf.reduce_sum of img_tensor passed through MLPs, resembles output from encoder called features.
  def reset_state(self, features):
    # hidden state and cell state shape == (64, 1024) [(64, 64, 1024) --> (64, 1024)]
    return (tf.reduce_sum(input_tensor = features, axis= 1), tf.reduce_sum(input_tensor = features, axis= 1))
```

In [58]:

```python
encoder = Encoder_CNN(units)                               # (64, 64, 2048) to (64, 64, 1024)
decoder = Decoder_RNN(embedding_dim, units, vocab_size)      # (300, 1024, 9212)
```

Loss Function & Optimizer

In [59]:

```
optimizer = Adam(learning_rate= 0.002)

def loss_func(real, pred):
  # Exp: tf.math.logical_not(tf.constant([True, False]))
  # Ans: <tf.Tensor: shape=(2,), dtype=bool, numpy=array([False,  True])>

  # Exp: x = tf.constant([2, 4]), y = tf.constant(2) then tf.math.equal(x, y)
  # Ans: <tf.Tensor: shape=(2,), dtype=bool, numpy=array([ True, False])>
  # mask 'False' (0) for padding else 'True' (1) for easy calculation of loss when timesteps are lesser.
  mask = tf.math.logical_not(x = tf.math.equal(x= real, y= 0))

  # we must use below loss and not sparse_categorical_crossentropy because this is for "Computes the sparse crossentropy loss"
  # where as the below is for "Computes the crossentropy loss between the labels and predictions.""
  # https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy
  # tf.keras.losses.Reduction.NONE we get array if we use this or else we get SUM means single not array
  loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True, reduction = tf.keras.losses.Reduction.NONE)
  # real dtype = 'int32' and pred dtype = 'float32' and loss dtype = 'float32'
  loss = loss(y_true = real, y_pred = pred)

  # Casts a tensor to a new dtype.
  # Ex: x = tf.constant([1.8, 2.2], dtype=tf.float32) tf.dtypes.cast(x, tf.int32)
  mask = tf.cast(x= mask, dtype= loss.dtype)
  loss *= mask

  return tf.reduce_mean(input_tensor= loss)

"""
global_step = tf.Variable(0, trainable=False)

learning_rate = tf.compat.v1.train.exponential_decay(learning_rate = 0.01, global_step = global_step,
                                            decay_steps= 4000, decay_rate= 0.96, staircase=True)
# Passing global_step to minimize() will increment it at each step.
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=  learning_rate )
"""
```

Out[59]:

'\nglobal_step = tf.Variable(0, trainable=False)\n\nlearning_rate = tf.compat.v1.train.exponential_decay(learning_rate = 0.01, global_step = global_ste p,\n decay_steps= 4000, decay_rate= 0.96, staircase=True)\n# Passing global_step to minimize() will increment it a t each step.\noptimizer = tf.compat.v1.train.AdamOptimizer(learning_rate= learning_rate)\n'

Checkpoint

In [60]:

```
path = '/content/drive/My Drive/ckpt'

ckpt = tf.train.Checkpoint(encoder = encoder, decoder= decoder, optimizer= optimizer)
ckpt_manager = tf.train.CheckpointManager(checkpoint= ckpt, directory= path, max_to_keep= 5)
```

In [61]:

```
start_epoch= 0
if ckpt_manager.latest_checkpoint:
  start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
```

In [62]:

```
import datetime

# Clear any logs from previous runs
!rm -rf ./logs/

#tf.reset_default_graph()
log_dir = '/content/tensorboard' + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
summary_writer = tf.summary.create_file_writer(log_dir)
```

Training

- You extract the features stored in the respective .npy files and then pass those features through the encoder.
- The encoder output, hidden state(initialized to 0) and the decoder input (which is the start token) is passed to the decoder.
- The decoder returns the predictions and the decoder hidden state.
- The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.

- Use teacher forcing to decide the next input to the decoder.
- Teacher forcing is the technique where the target word is passed as the next input to the decoder.
- The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

In [63]:

```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

In [64]:

```
@tf.function
def train_step(img_tensor, target):
  loss = 0

  # initializing the hidden state for each batch
  # because the captions are not related from image to image
  # # hidden state shape == (64, 1024) [(64, 64, 2048) --> (64, 1024)] and cell state too (tuple[0] & [1])
  hidden = decoder.reset_state(encoder(img_tensor))

  # dec_input = (64, 1)
  # target shape == (64, 32)
  dec_input = tf.reshape(tensor= [tokenizer.word_index['<startseq>']] * target.shape[0], shape= (target.shape[0], 1))

  with tf.GradientTape() as tape:
    # img_tensor shape == (64, 64, 2048)
    # features shape == (64, 64, 1024) [(64, 64, 2048) --> (64, 64, 1024)]
    features = encoder(img_tensor)

    for i in range(1, target.shape[1]):

      # passing the features through the decoder
      # predictions == (64, 9212), hidden = (64, 1024)
      predictions, hidden_state, alignment_vector = decoder(dec_input, features, hidden)

      # loss calculated across each time step (total 32 losses gets summated)
      loss += loss_func(real = target[:, i], pred = predictions) # target[:, 1] = 64 first words per batchwise and
      # must match with the corresponding predictions and hence loss does gets reduced until 32

      # using teacher forcing (64, 1) total 31 times bcoz "<startseq>" is skipped
      dec_input = tf.reshape(tensor = target[:, i], shape=  (target.shape[0], 1))

  # the summated loss above across 32 timesteps are averaged and we arrive at total loss.
  total_loss = loss / int(target.shape[1]) # 32

  # Keras models and layers offer the convenient 'variables' and 'trainable_variables' properties,
  # which recursively gather up all dependent variables. This makes it easy to manage variables
  # locally to where they are being used.
  trainable_variables = encoder.trainable_variables + decoder.trainable_variables

  gradients = tape.gradient(loss, trainable_variables)

  optimizer.apply_gradients(grads_and_vars= zip(gradients, trainable_variables))#, global_step= global_step)

  return loss, total_loss
```

In [65]:

```
# set random seed
tf.random.set_seed(seed= 9)

start_time= time.time()

for i in range(start_epoch, epochs):
  start = time.time()
  total_loss = 0

  for (batch, (img_tensor, target)) in enumerate(train_dataset): # (batch, (64, 64, 2048), (64, 32))
    batch_loss, t_loss = train_step(img_tensor, target)
    total_loss += t_loss

    if batch % 100 == 0:
      print ('Epoch {} Batch {} Loss {:.4f}'.format(i + 1, batch, batch_loss.numpy() / int(target.shape[1])))

    # Tensorboard
  with summary_writer.as_default():
    tf.summary.scalar('LossPlot', (total_loss/ num_steps), step= i)
```

```
    # storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    if i % 5 == 0:
      ckpt_manager.save()

    print('Epoch {} Loss {:.6f}'.format(i + 1, total_loss / num_steps))
    print('Time taken for 1 epochs {} sec\n'.format(time.time() - start))

print("Time Taken is: " + str(time.time() - start_time))
```

```
Epoch 14 Batch 0 Loss 3.9276
Epoch 14 Batch 100 Loss 2.0861
Epoch 14 Batch 200 Loss 1.6643
Epoch 14 Batch 300 Loss 1.6096
Epoch 14 Batch 400 Loss 1.7355
Epoch 14 Loss 1.870403
Time taken for 1 epochs 190.62551283836365 sec

Epoch 15 Batch 0 Loss 1.6234
Epoch 15 Batch 100 Loss 1.5043
Epoch 15 Batch 200 Loss 1.5179
Epoch 15 Batch 300 Loss 1.5268
Epoch 15 Batch 400 Loss 1.4776
Epoch 15 Loss 1.562673
Time taken for 1 epochs 129.6772632598877 sec

Epoch 16 Batch 0 Loss 1.5258
Epoch 16 Batch 100 Loss 1.5104
Epoch 16 Batch 200 Loss 1.5009
Epoch 16 Batch 300 Loss 1.3952
Epoch 16 Batch 400 Loss 1.2975
Epoch 16 Loss 1.437533
Time taken for 1 epochs 130.55550360679626 sec

Epoch 17 Batch 0 Loss 1.4252
Epoch 17 Batch 100 Loss 1.4058
Epoch 17 Batch 200 Loss 1.4538
Epoch 17 Batch 300 Loss 1.3748
Epoch 17 Batch 400 Loss 1.4376
Epoch 17 Loss 1.347381
Time taken for 1 epochs 129.82747888565063 sec

Epoch 18 Batch 0 Loss 1.2599
Epoch 18 Batch 100 Loss 1.2341
Epoch 18 Batch 200 Loss 1.4081
Epoch 18 Batch 300 Loss 1.2903
Epoch 18 Batch 400 Loss 1.2282
Epoch 18 Loss 1.283865
Time taken for 1 epochs 129.71327233314514 sec

Epoch 19 Batch 0 Loss 1.2677
Epoch 19 Batch 100 Loss 1.1127
Epoch 19 Batch 200 Loss 1.1854
Epoch 19 Batch 300 Loss 1.1785
Epoch 19 Batch 400 Loss 1.2989
Epoch 19 Loss 1.234446
Time taken for 1 epochs 129.92643785476685 sec

Epoch 20 Batch 0 Loss 1.2907
Epoch 20 Batch 100 Loss 1.1452
Epoch 20 Batch 200 Loss 1.0833
Epoch 20 Batch 300 Loss 1.2523
Epoch 20 Batch 400 Loss 1.1176
Epoch 20 Loss 1.193697
Time taken for 1 epochs 129.86042070388794 sec

Epoch 21 Batch 0 Loss 1.1650
Epoch 21 Batch 100 Loss 1.2605
Epoch 21 Batch 200 Loss 1.1218
Epoch 21 Batch 300 Loss 1.1896
Epoch 21 Batch 400 Loss 1.1099
Epoch 21 Loss 1.159167
Time taken for 1 epochs 131.69571328163147 sec

Epoch 22 Batch 0 Loss 1.2350
Epoch 22 Batch 100 Loss 1.1905
Epoch 22 Batch 200 Loss 1.0555
Epoch 22 Batch 300 Loss 1.0671
Epoch 22 Batch 400 Loss 1.2192
Epoch 22 Loss 1.130094
Time taken for 1 epochs 129.92939162254333 sec
```

```
Epoch 23 Batch 0 Loss 1.0335
Epoch 23 Batch 100 Loss 0.9694
Epoch 23 Batch 200 Loss 1.1469
Epoch 23 Batch 300 Loss 1.1294
Epoch 23 Batch 400 Loss 1.1912
Epoch 23 Loss 1.104305
Time taken for 1 epochs 129.81146121025085 sec

Epoch 24 Batch 0 Loss 1.0825
Epoch 24 Batch 100 Loss 1.1578
Epoch 24 Batch 200 Loss 1.0673
Epoch 24 Batch 300 Loss 1.0044
Epoch 24 Batch 400 Loss 1.0855
Epoch 24 Loss 1.079203
Time taken for 1 epochs 129.87606811523438 sec

Epoch 25 Batch 0 Loss 1.0808
Epoch 25 Batch 100 Loss 1.0170
Epoch 25 Batch 200 Loss 0.9243
Epoch 25 Batch 300 Loss 1.1043
Epoch 25 Batch 400 Loss 0.9928
Epoch 25 Loss 1.056981
Time taken for 1 epochs 129.8359932899475 sec

Epoch 26 Batch 0 Loss 1.0912
Epoch 26 Batch 100 Loss 1.0399
Epoch 26 Batch 200 Loss 1.0470
Epoch 26 Batch 300 Loss 1.0998
Epoch 26 Batch 400 Loss 0.9859
Epoch 26 Loss 1.037957
Time taken for 1 epochs 130.68443870544434 sec

Epoch 27 Batch 0 Loss 1.0579
Epoch 27 Batch 100 Loss 0.9650
Epoch 27 Batch 200 Loss 1.1211
Epoch 27 Batch 300 Loss 1.0275
Epoch 27 Batch 400 Loss 0.9665
Epoch 27 Loss 1.018855
Time taken for 1 epochs 129.86678791046143 sec

Epoch 28 Batch 0 Loss 1.0035
Epoch 28 Batch 100 Loss 1.0334
Epoch 28 Batch 200 Loss 0.9267
Epoch 28 Batch 300 Loss 0.9014
Epoch 28 Batch 400 Loss 0.9522
Epoch 28 Loss 1.001846
Time taken for 1 epochs 129.8619465827942 sec

Epoch 29 Batch 0 Loss 0.8607
Epoch 29 Batch 100 Loss 1.0379
Epoch 29 Batch 200 Loss 0.8646
Epoch 29 Batch 300 Loss 1.0968
Epoch 29 Batch 400 Loss 1.0107
Epoch 29 Loss 0.984785
Time taken for 1 epochs 129.9681990146637 sec

Epoch 30 Batch 0 Loss 1.0024
Epoch 30 Batch 100 Loss 0.9406
Epoch 30 Batch 200 Loss 0.9956
Epoch 30 Batch 300 Loss 0.9130
Epoch 30 Batch 400 Loss 0.9564
Epoch 30 Loss 0.970843
Time taken for 1 epochs 129.7864921092987 sec

Epoch 31 Batch 0 Loss 0.9186
Epoch 31 Batch 100 Loss 1.0079
Epoch 31 Batch 200 Loss 0.9739
Epoch 31 Batch 300 Loss 0.8800
Epoch 31 Batch 400 Loss 0.9503
Epoch 31 Loss 0.956301
Time taken for 1 epochs 130.77502274513245 sec

Epoch 32 Batch 0 Loss 0.8509
Epoch 32 Batch 100 Loss 1.0347
Epoch 32 Batch 200 Loss 0.9490
Epoch 32 Batch 300 Loss 0.8731
Epoch 32 Batch 400 Loss 0.9622
Epoch 32 Loss 0.942684
Time taken for 1 epochs 129.96516728401184 sec

Epoch 33 Batch 0 Loss 0.9253
Epoch 33 Batch 100 Loss 0.9803
```

Epoch 33 Batch 100 Loss 0.9803
Epoch 33 Batch 200 Loss 0.9004
Epoch 33 Batch 300 Loss 0.9532
Epoch 33 Batch 400 Loss 0.9698
Epoch 33 Loss 0.931024
Time taken for 1 epochs 129.844952583313 sec

Epoch 34 Batch 0 Loss 1.0012
Epoch 34 Batch 100 Loss 0.9006
Epoch 34 Batch 200 Loss 0.9688
Epoch 34 Batch 300 Loss 0.9643
Epoch 34 Batch 400 Loss 0.9314
Epoch 34 Loss 0.918737
Time taken for 1 epochs 129.99864315986633 sec

Epoch 35 Batch 0 Loss 0.9874
Epoch 35 Batch 100 Loss 0.9967
Epoch 35 Batch 200 Loss 1.0568
Epoch 35 Batch 300 Loss 0.9470
Epoch 35 Batch 400 Loss 0.9251
Epoch 35 Loss 0.907165
Time taken for 1 epochs 129.9254026412964 sec

Epoch 36 Batch 0 Loss 0.9138
Epoch 36 Batch 100 Loss 0.8844
Epoch 36 Batch 200 Loss 0.8720
Epoch 36 Batch 300 Loss 0.9025
Epoch 36 Batch 400 Loss 0.8744
Epoch 36 Loss 0.896657
Time taken for 1 epochs 131.6855788230896 sec

Epoch 37 Batch 0 Loss 0.8523
Epoch 37 Batch 100 Loss 0.9178
Epoch 37 Batch 200 Loss 0.8354
Epoch 37 Batch 300 Loss 1.0417
Epoch 37 Batch 400 Loss 0.8623
Epoch 37 Loss 0.887949
Time taken for 1 epochs 129.76962637901306 sec

Epoch 38 Batch 0 Loss 0.9633
Epoch 38 Batch 100 Loss 0.8887
Epoch 38 Batch 200 Loss 0.8700
Epoch 38 Batch 300 Loss 0.8958
Epoch 38 Batch 400 Loss 0.9476
Epoch 38 Loss 0.878731
Time taken for 1 epochs 129.87667679786682 sec

Epoch 39 Batch 0 Loss 0.8357
Epoch 39 Batch 100 Loss 0.8703
Epoch 39 Batch 200 Loss 0.8369
Epoch 39 Batch 300 Loss 0.7687
Epoch 39 Batch 400 Loss 0.8639
Epoch 39 Loss 0.869280
Time taken for 1 epochs 129.7986650466919 sec

Epoch 40 Batch 0 Loss 0.9186
Epoch 40 Batch 100 Loss 0.7839
Epoch 40 Batch 200 Loss 0.8932
Epoch 40 Batch 300 Loss 0.8762
Epoch 40 Batch 400 Loss 0.8331
Epoch 40 Loss 0.858868
Time taken for 1 epochs 129.8954050540924 sec

Epoch 41 Batch 0 Loss 0.8054
Epoch 41 Batch 100 Loss 0.9047
Epoch 41 Batch 200 Loss 0.9323
Epoch 41 Batch 300 Loss 0.8736
Epoch 41 Batch 400 Loss 0.7513
Epoch 41 Loss 0.851196
Time taken for 1 epochs 130.5920009613037 sec

Epoch 42 Batch 0 Loss 0.7961
Epoch 42 Batch 100 Loss 0.7479
Epoch 42 Batch 200 Loss 0.7774
Epoch 42 Batch 300 Loss 0.9171
Epoch 42 Batch 400 Loss 0.8477
Epoch 42 Loss 0.842411
Time taken for 1 epochs 129.76124238967896 sec

Epoch 43 Batch 0 Loss 0.8656
Epoch 43 Batch 100 Loss 0.7919
Epoch 43 Batch 200 Loss 0.8586
Epoch 43 Batch 300 Loss 0.8072

Epoch 43 Batch 400 Loss 0.8541
Epoch 43 Loss 0.834280
Time taken for 1 epochs 129.9461531639099 sec

Epoch 44 Batch 0 Loss 0.8627
Epoch 44 Batch 100 Loss 0.7536
Epoch 44 Batch 200 Loss 0.8775
Epoch 44 Batch 300 Loss 0.8031
Epoch 44 Batch 400 Loss 0.7007
Epoch 44 Loss 0.826913
Time taken for 1 epochs 129.84298253059387 sec

Epoch 45 Batch 0 Loss 0.8221
Epoch 45 Batch 100 Loss 0.7951
Epoch 45 Batch 200 Loss 0.8113
Epoch 45 Batch 300 Loss 0.7493
Epoch 45 Batch 400 Loss 0.8130
Epoch 45 Loss 0.820849
Time taken for 1 epochs 129.97634077072144 sec

Epoch 46 Batch 0 Loss 0.8825
Epoch 46 Batch 100 Loss 0.7643
Epoch 46 Batch 200 Loss 0.8753
Epoch 46 Batch 300 Loss 0.7461
Epoch 46 Batch 400 Loss 0.8493
Epoch 46 Loss 0.813670
Time taken for 1 epochs 130.677321434021 sec

Epoch 47 Batch 0 Loss 0.9140
Epoch 47 Batch 100 Loss 0.8913
Epoch 47 Batch 200 Loss 0.7953
Epoch 47 Batch 300 Loss 0.8665
Epoch 47 Batch 400 Loss 0.7394
Epoch 47 Loss 0.807013
Time taken for 1 epochs 129.95418548583984 sec

Epoch 48 Batch 0 Loss 1.0102
Epoch 48 Batch 100 Loss 0.7001
Epoch 48 Batch 200 Loss 0.7942
Epoch 48 Batch 300 Loss 0.7904
Epoch 48 Batch 400 Loss 0.7467
Epoch 48 Loss 0.800409
Time taken for 1 epochs 129.82566571235657 sec

Epoch 49 Batch 0 Loss 0.9117
Epoch 49 Batch 100 Loss 0.7988
Epoch 49 Batch 200 Loss 0.8713
Epoch 49 Batch 300 Loss 0.7945
Epoch 49 Batch 400 Loss 0.7750
Epoch 49 Loss 0.793553
Time taken for 1 epochs 129.88287353515625 sec

Epoch 50 Batch 0 Loss 0.8715
Epoch 50 Batch 100 Loss 0.8244
Epoch 50 Batch 200 Loss 0.8798
Epoch 50 Batch 300 Loss 0.8209
Epoch 50 Batch 400 Loss 0.7786
Epoch 50 Loss 0.790296
Time taken for 1 epochs 129.93987202644348 sec

Time Taken is: 4873.441141605377

In [71]:

```
ckpt_manager.latest_checkpoint
```

Out[71]:

'/content/drive/My Drive/ckpt/ckpt-7'

In [72]:

```
if ckpt_manager.latest_checkpoint:
  start_epoch = 50
```

In [73]:

```
loss_plot= []
```

```
@tf.function
def train_step(img_tensor, target):
  loss = 0

  # initializing the hidden state for each batch
  # because the captions are not related from image to image
  # hidden = (64, 512)
  hidden = decoder.reset_state(encoder(img_tensor))

  # dec_input = (64, 1)
  dec_input = tf.reshape(tensor= [tokenizer.word_index['<startseq>']] * target.shape[0], shape= (target.shape[0], 1))

  with tf.GradientTape() as tape:
    # img_tensor = (64, 64, 2048), target = (64, 32)
    features = encoder(img_tensor) # (64, 64, 2048) to (64, 64, 300)

    for i in range(1, target.shape[1]):

      # passing the features through the decoder
      # predictions = (64, 5001), hidden = (64, 512)
      predictions, hidden_state, alignment_vector = decoder(dec_input, features, hidden)

      loss += loss_func(real = target[:, i], pred = predictions) # target[:, 1] = 64 first words per datapoint and
      # must match with the corresponding predictions and hence loss does gets reduced until 32

      # using teacher forcing (64, 1) total 31 times bcoz "<startseq>" is skipped
      dec_input = tf.reshape(tensor = target[:, i], shape=  (target.shape[0], 1))

  total_loss = loss / int(target.shape[1])

  # Keras models and layers offer the convenient 'variables' and 'trainable_variables' properties,
  # which recursively gather up all dependent variables. This makes it easy to manage variables
  # locally to where they are being used.
  trainable_variables = encoder.trainable_variables + decoder.trainable_variables

  gradients = tape.gradient(loss, trainable_variables)

  optimizer.apply_gradients(grads_and_vars= zip(gradients, trainable_variables))#, global_step= global_step)

  return loss, total_loss
```

In [74]:

```
# set random seed
tf.random.set_seed(seed= 9)

start_time= time.time()

for i in range(start_epoch, epochs+50):
  start = time.time()
  total_loss = 0

  for (batch, (img_tensor, target)) in enumerate(train_dataset): # (batch, (64, 64, 2048), (64, 32))
    batch_loss, t_loss = train_step(img_tensor, target)
    total_loss += t_loss

    if batch % 100 == 0:
      print ('Epoch {} Batch {} Loss {:.4f}'.format(i + 1, batch, batch_loss.numpy() / int(target.shape[1])))

  # storing the epoch end loss value to plot later
  loss_plot.append(total_loss / num_steps)

  # Tensorboard
  with summary_writer.as_default():
    tf.summary.scalar('LossPlot', (total_loss/ num_steps), step= i)

  if i % 5 == 0:
    ckpt_manager.save()

  print('Epoch {} Loss {:.6f}'.format(i + 1, total_loss / num_steps))
  print('Time taken for 1 epochs {} sec\n'.format(time.time() - start))

print("Time Taken is: " + str(time.time() - start_time))
```

```
Epoch 51 Batch 0 Loss 0.7734
Epoch 51 Batch 100 Loss 0.7446
Epoch 51 Batch 200 Loss 0.7886
Epoch 51 Batch 300 Loss 0.7237
Epoch 51 Batch 400 Loss 0.7457
Epoch 51 Loss 0.782212
Time taken for 1 epochs 167.8932592868805 sec
```

```
Epoch 52 Batch 0 Loss 0.8646
Epoch 52 Batch 100 Loss 0.7426
Epoch 52 Batch 200 Loss 0.8675
Epoch 52 Batch 300 Loss 0.7984
Epoch 52 Batch 400 Loss 0.7554
Epoch 52 Loss 0.776998
Time taken for 1 epochs 129.9073257446289 sec

Epoch 53 Batch 0 Loss 0.8829
Epoch 53 Batch 100 Loss 0.8011
Epoch 53 Batch 200 Loss 0.7779
Epoch 53 Batch 300 Loss 0.7180
Epoch 53 Batch 400 Loss 0.8518
Epoch 53 Loss 0.771681
Time taken for 1 epochs 129.79376554489136 sec

Epoch 54 Batch 0 Loss 0.9003
Epoch 54 Batch 100 Loss 0.7254
Epoch 54 Batch 200 Loss 0.8120
Epoch 54 Batch 300 Loss 0.7897
Epoch 54 Batch 400 Loss 0.6847
Epoch 54 Loss 0.766530
Time taken for 1 epochs 129.90160036087036 sec

Epoch 55 Batch 0 Loss 0.7810
Epoch 55 Batch 100 Loss 0.6877
Epoch 55 Batch 200 Loss 0.7894
Epoch 55 Batch 300 Loss 0.7560
Epoch 55 Batch 400 Loss 0.8125
Epoch 55 Loss 0.761109
Time taken for 1 epochs 129.9458873271942 sec

Epoch 56 Batch 0 Loss 0.7621
Epoch 56 Batch 100 Loss 0.6790
Epoch 56 Batch 200 Loss 0.7086
Epoch 56 Batch 300 Loss 0.6668
Epoch 56 Batch 400 Loss 0.8282
Epoch 56 Loss 0.758625
Time taken for 1 epochs 130.71980047225952 sec

Epoch 57 Batch 0 Loss 0.7839
Epoch 57 Batch 100 Loss 0.7969
Epoch 57 Batch 200 Loss 0.7669
Epoch 57 Batch 300 Loss 0.7506
Epoch 57 Batch 400 Loss 0.6572
Epoch 57 Loss 0.752061
Time taken for 1 epochs 129.8746838569641 sec

Epoch 58 Batch 0 Loss 0.7851
Epoch 58 Batch 100 Loss 0.7818
Epoch 58 Batch 200 Loss 0.7734
Epoch 58 Batch 300 Loss 0.7478
Epoch 58 Batch 400 Loss 0.7197
Epoch 58 Loss 0.747599
Time taken for 1 epochs 129.84001898765564 sec

Epoch 59 Batch 0 Loss 0.8477
Epoch 59 Batch 100 Loss 0.7425
Epoch 59 Batch 200 Loss 0.7849
Epoch 59 Batch 300 Loss 0.8011
Epoch 59 Batch 400 Loss 0.6149
Epoch 59 Loss 0.742722
Time taken for 1 epochs 129.88213777542114 sec

Epoch 60 Batch 0 Loss 0.7204
Epoch 60 Batch 100 Loss 0.7359
Epoch 60 Batch 200 Loss 0.6745
Epoch 60 Batch 300 Loss 0.7559
Epoch 60 Batch 400 Loss 0.7121
Epoch 60 Loss 0.739939
Time taken for 1 epochs 129.86262774467468 sec

Epoch 61 Batch 0 Loss 0.8276
Epoch 61 Batch 100 Loss 0.7152
Epoch 61 Batch 200 Loss 0.7055
Epoch 61 Batch 300 Loss 0.6679
Epoch 61 Batch 400 Loss 0.7219
Epoch 61 Loss 0.734065
Time taken for 1 epochs 131.5027871131897 sec

Epoch 62 Batch 0 Loss 0.6489
Epoch 62 Batch 100 Loss 0.7659
```

Epoch 62 Batch 200 Loss 0.7460
Epoch 62 Batch 300 Loss 0.6918
Epoch 62 Batch 400 Loss 0.6733
Epoch 62 Loss 0.730987
Time taken for 1 epochs 129.85659909248352 sec

Epoch 63 Batch 0 Loss 0.8391
Epoch 63 Batch 100 Loss 0.7792
Epoch 63 Batch 200 Loss 0.6552
Epoch 63 Batch 300 Loss 0.6451
Epoch 63 Batch 400 Loss 0.5978
Epoch 63 Loss 0.725875
Time taken for 1 epochs 129.93333268165588 sec

Epoch 64 Batch 0 Loss 0.7695
Epoch 64 Batch 100 Loss 0.6800
Epoch 64 Batch 200 Loss 0.7515
Epoch 64 Batch 300 Loss 0.6185
Epoch 64 Batch 400 Loss 0.6678
Epoch 64 Loss 0.723425
Time taken for 1 epochs 129.892409324646 sec

Epoch 65 Batch 0 Loss 0.8467
Epoch 65 Batch 100 Loss 0.6630
Epoch 65 Batch 200 Loss 0.6132
Epoch 65 Batch 300 Loss 0.7190
Epoch 65 Batch 400 Loss 0.6571
Epoch 65 Loss 0.722124
Time taken for 1 epochs 129.8640797138214 sec

Epoch 66 Batch 0 Loss 0.7998
Epoch 66 Batch 100 Loss 0.7295
Epoch 66 Batch 200 Loss 0.6477
Epoch 66 Batch 300 Loss 0.6818
Epoch 66 Batch 400 Loss 0.7211
Epoch 66 Loss 0.718116
Time taken for 1 epochs 131.3415036201477 sec

Epoch 67 Batch 0 Loss 0.7414
Epoch 67 Batch 100 Loss 0.6801
Epoch 67 Batch 200 Loss 0.7511
Epoch 67 Batch 300 Loss 0.6473
Epoch 67 Batch 400 Loss 0.6200
Epoch 67 Loss 0.713034
Time taken for 1 epochs 129.98901271820068 sec

Epoch 68 Batch 0 Loss 0.6701
Epoch 68 Batch 100 Loss 0.7568
Epoch 68 Batch 200 Loss 0.6893
Epoch 68 Batch 300 Loss 0.7044
Epoch 68 Batch 400 Loss 0.7503
Epoch 68 Loss 0.710837
Time taken for 1 epochs 129.8066167831421 sec

Epoch 69 Batch 0 Loss 0.7404
Epoch 69 Batch 100 Loss 0.7265
Epoch 69 Batch 200 Loss 0.7239
Epoch 69 Batch 300 Loss 0.6769
Epoch 69 Batch 400 Loss 0.8190
Epoch 69 Loss 0.706327
Time taken for 1 epochs 129.81652903556824 sec

Epoch 70 Batch 0 Loss 0.7163
Epoch 70 Batch 100 Loss 0.6799
Epoch 70 Batch 200 Loss 0.6881
Epoch 70 Batch 300 Loss 0.6809
Epoch 70 Batch 400 Loss 0.6359
Epoch 70 Loss 0.703854
Time taken for 1 epochs 129.74514055252075 sec

Epoch 71 Batch 0 Loss 0.7082
Epoch 71 Batch 100 Loss 0.7273
Epoch 71 Batch 200 Loss 0.6760
Epoch 71 Batch 300 Loss 0.7417
Epoch 71 Batch 400 Loss 0.6775
Epoch 71 Loss 0.700053
Time taken for 1 epochs 130.60804724693298 sec

Epoch 72 Batch 0 Loss 0.7203
Epoch 72 Batch 100 Loss 0.6949
Epoch 72 Batch 200 Loss 0.6941
Epoch 72 Batch 300 Loss 0.6217

Epoch 72 Batch 400 Loss 0.7297
Epoch 72 Loss 0.697481
Time taken for 1 epochs 129.8870747089386 sec

Epoch 73 Batch 0 Loss 0.6603
Epoch 73 Batch 100 Loss 0.6415
Epoch 73 Batch 200 Loss 0.6818
Epoch 73 Batch 300 Loss 0.7044
Epoch 73 Batch 400 Loss 0.7248
Epoch 73 Loss 0.695149
Time taken for 1 epochs 129.83783769607544 sec

Epoch 74 Batch 0 Loss 0.8347
Epoch 74 Batch 100 Loss 0.6422
Epoch 74 Batch 200 Loss 0.6525
Epoch 74 Batch 300 Loss 0.6946
Epoch 74 Batch 400 Loss 0.7226
Epoch 74 Loss 0.690922
Time taken for 1 epochs 129.82494616508484 sec

Epoch 75 Batch 0 Loss 0.6562
Epoch 75 Batch 100 Loss 0.7753
Epoch 75 Batch 200 Loss 0.6869
Epoch 75 Batch 300 Loss 0.6739
Epoch 75 Batch 400 Loss 0.7234
Epoch 75 Loss 0.689717
Time taken for 1 epochs 129.92838525772095 sec

Epoch 76 Batch 0 Loss 0.7192
Epoch 76 Batch 100 Loss 0.6831
Epoch 76 Batch 200 Loss 0.6835
Epoch 76 Batch 300 Loss 0.7078
Epoch 76 Batch 400 Loss 0.7141
Epoch 76 Loss 0.688722
Time taken for 1 epochs 131.4518325328827 sec

Epoch 77 Batch 0 Loss 0.6847
Epoch 77 Batch 100 Loss 0.7708
Epoch 77 Batch 200 Loss 0.7178
Epoch 77 Batch 300 Loss 0.6496
Epoch 77 Batch 400 Loss 0.6932
Epoch 77 Loss 0.684700
Time taken for 1 epochs 129.8712465763092 sec

Epoch 78 Batch 0 Loss 0.7471
Epoch 78 Batch 100 Loss 0.7215
Epoch 78 Batch 200 Loss 0.6567
Epoch 78 Batch 300 Loss 0.7183
Epoch 78 Batch 400 Loss 0.7270
Epoch 78 Loss 0.682667
Time taken for 1 epochs 129.83231687545776 sec

Epoch 79 Batch 0 Loss 0.8025
Epoch 79 Batch 100 Loss 0.6847
Epoch 79 Batch 200 Loss 0.7355
Epoch 79 Batch 300 Loss 0.6598
Epoch 79 Batch 400 Loss 0.6897
Epoch 79 Loss 0.680446
Time taken for 1 epochs 129.86632823944092 sec

Epoch 80 Batch 0 Loss 0.7958
Epoch 80 Batch 100 Loss 0.6421
Epoch 80 Batch 200 Loss 0.7006
Epoch 80 Batch 300 Loss 0.5838
Epoch 80 Batch 400 Loss 0.6373
Epoch 80 Loss 0.678561
Time taken for 1 epochs 129.73432636260986 sec

Epoch 81 Batch 0 Loss 0.6950
Epoch 81 Batch 100 Loss 0.7187
Epoch 81 Batch 200 Loss 0.6501
Epoch 81 Batch 300 Loss 0.6156
Epoch 81 Batch 400 Loss 0.6743
Epoch 81 Loss 0.675943
Time taken for 1 epochs 130.5440948009491 sec

Epoch 82 Batch 0 Loss 0.8717
Epoch 82 Batch 100 Loss 0.6238
Epoch 82 Batch 200 Loss 0.6708
Epoch 82 Batch 300 Loss 0.5977
Epoch 82 Batch 400 Loss 0.7614
Epoch 82 Loss 0.673295
Time taken for 1 epochs 129.89604330062866 sec

Epoch 83 Batch 0 Loss 0.6367
Epoch 83 Batch 100 Loss 0.7094
Epoch 83 Batch 200 Loss 0.6781
Epoch 83 Batch 300 Loss 0.6980
Epoch 83 Batch 400 Loss 0.7057
Epoch 83 Loss 0.671538
Time taken for 1 epochs 129.89586329460144 sec

Epoch 84 Batch 0 Loss 0.8405
Epoch 84 Batch 100 Loss 0.6429
Epoch 84 Batch 200 Loss 0.6989
Epoch 84 Batch 300 Loss 0.6796
Epoch 84 Batch 400 Loss 0.6249
Epoch 84 Loss 0.670760
Time taken for 1 epochs 129.85379266738892 sec

Epoch 85 Batch 0 Loss 0.7449
Epoch 85 Batch 100 Loss 0.6747
Epoch 85 Batch 200 Loss 0.7739
Epoch 85 Batch 300 Loss 0.6808
Epoch 85 Batch 400 Loss 0.6619
Epoch 85 Loss 0.666551
Time taken for 1 epochs 129.82766461372375 sec

Epoch 86 Batch 0 Loss 0.7533
Epoch 86 Batch 100 Loss 0.6465
Epoch 86 Batch 200 Loss 0.6737
Epoch 86 Batch 300 Loss 0.6789
Epoch 86 Batch 400 Loss 0.6567
Epoch 86 Loss 0.664196
Time taken for 1 epochs 130.6171269416809 sec

Epoch 87 Batch 0 Loss 0.7195
Epoch 87 Batch 100 Loss 0.6340
Epoch 87 Batch 200 Loss 0.6719
Epoch 87 Batch 300 Loss 0.6329
Epoch 87 Batch 400 Loss 0.5886
Epoch 87 Loss 0.662244
Time taken for 1 epochs 129.85042428970337 sec

Epoch 88 Batch 0 Loss 0.7461
Epoch 88 Batch 100 Loss 0.6374
Epoch 88 Batch 200 Loss 0.6815
Epoch 88 Batch 300 Loss 0.6293
Epoch 88 Batch 400 Loss 0.6598
Epoch 88 Loss 0.661449
Time taken for 1 epochs 129.76882362365723 sec

Epoch 89 Batch 0 Loss 0.6834
Epoch 89 Batch 100 Loss 0.7062
Epoch 89 Batch 200 Loss 0.6464
Epoch 89 Batch 300 Loss 0.5959
Epoch 89 Batch 400 Loss 0.6735
Epoch 89 Loss 0.660172
Time taken for 1 epochs 129.90768790245056 sec

Epoch 90 Batch 0 Loss 0.7491
Epoch 90 Batch 100 Loss 0.6516
Epoch 90 Batch 200 Loss 0.7076
Epoch 90 Batch 300 Loss 0.6442
Epoch 90 Batch 400 Loss 0.6565
Epoch 90 Loss 0.657546
Time taken for 1 epochs 129.65483856201172 sec

Epoch 91 Batch 0 Loss 0.7108
Epoch 91 Batch 100 Loss 0.6341
Epoch 91 Batch 200 Loss 0.6719
Epoch 91 Batch 300 Loss 0.7469
Epoch 91 Batch 400 Loss 0.7423
Epoch 91 Loss 0.656642
Time taken for 1 epochs 131.05210494995117 sec

Epoch 92 Batch 0 Loss 0.7650
Epoch 92 Batch 100 Loss 0.6647
Epoch 92 Batch 200 Loss 0.7144
Epoch 92 Batch 300 Loss 0.6907
Epoch 92 Batch 400 Loss 0.7171
Epoch 92 Loss 0.654223
Time taken for 1 epochs 129.03716230392456 sec

Epoch 93 Batch 0 Loss 0.7146

Epoch 93 Batch 100 Loss 0.7025
Epoch 93 Batch 200 Loss 0.6296
Epoch 93 Batch 300 Loss 0.5636
Epoch 93 Batch 400 Loss 0.7251
Epoch 93 Loss 0.653931
Time taken for 1 epochs 128.93463945388794 sec

Epoch 94 Batch 0 Loss 0.6730
Epoch 94 Batch 100 Loss 0.5802
Epoch 94 Batch 200 Loss 0.7278
Epoch 94 Batch 300 Loss 0.6343
Epoch 94 Batch 400 Loss 0.6068
Epoch 94 Loss 0.652549
Time taken for 1 epochs 128.94554901123047 sec

Epoch 95 Batch 0 Loss 0.7397
Epoch 95 Batch 100 Loss 0.6281
Epoch 95 Batch 200 Loss 0.7492
Epoch 95 Batch 300 Loss 0.6390
Epoch 95 Batch 400 Loss 0.7078
Epoch 95 Loss 0.649928
Time taken for 1 epochs 128.9322555065155 sec

Epoch 96 Batch 0 Loss 0.6739
Epoch 96 Batch 100 Loss 0.6519
Epoch 96 Batch 200 Loss 0.6781
Epoch 96 Batch 300 Loss 0.6006
Epoch 96 Batch 400 Loss 0.8060
Epoch 96 Loss 0.648538
Time taken for 1 epochs 129.58567452430725 sec

Epoch 97 Batch 0 Loss 0.6656
Epoch 97 Batch 100 Loss 0.5719
Epoch 97 Batch 200 Loss 0.5762
Epoch 97 Batch 300 Loss 0.6733
Epoch 97 Batch 400 Loss 0.6779
Epoch 97 Loss 0.648269
Time taken for 1 epochs 128.99445462226868 sec

Epoch 98 Batch 0 Loss 0.6259
Epoch 98 Batch 100 Loss 0.5715
Epoch 98 Batch 200 Loss 0.5858
Epoch 98 Batch 300 Loss 0.5700
Epoch 98 Batch 400 Loss 0.6339
Epoch 98 Loss 0.644387
Time taken for 1 epochs 128.95287823677063 sec

Epoch 99 Batch 0 Loss 0.6565
Epoch 99 Batch 100 Loss 0.5496
Epoch 99 Batch 200 Loss 0.6419
Epoch 99 Batch 300 Loss 0.6186
Epoch 99 Batch 400 Loss 0.6678
Epoch 99 Loss 0.644378
Time taken for 1 epochs 128.91300177574158 sec

Epoch 100 Batch 0 Loss 0.7724
Epoch 100 Batch 100 Loss 0.6321
Epoch 100 Batch 200 Loss 0.7208
Epoch 100 Batch 300 Loss 0.5955
Epoch 100 Batch 400 Loss 0.6856
Epoch 100 Loss 0.644025
Time taken for 1 epochs 128.81597685813904 sec

Time Taken is: 6532.1954872608185

In [79]:

```
ckpt_manager.latest_checkpoint
```

Out[79]:

'/content/drive/My Drive/ckpt/ckpt-22'

In [80]:

```
if ckpt_manager.latest_checkpoint:
  start_epoch = 113
```

In [81]:

```
loss_plot= []

@tf.function
def train_step(img_tensor, target):
  loss = 0

  # initializing the hidden state for each batch
  # because the captions are not related from image to image
  # hidden = (64, 512)
  hidden = decoder.reset_state(encoder(img_tensor))

  # dec_input = (64, 1)
  dec_input = tf.reshape(tensor= [tokenizer.word_index['<startseq>']] * target.shape[0], shape= (target.shape[0], 1))

  with tf.GradientTape() as tape:
    # img_tensor = (64, 64, 2048), target = (64, 32)
    features = encoder(img_tensor) # (64, 64, 2048) to (64, 64, 300)

    for i in range(1, target.shape[1]):

      # passing the features through the decoder
      # predictions = (64, 5001), hidden = (64, 512)
      predictions, hidden_state, alignment_vector = decoder(dec_input, features, hidden)

      loss += loss_func(real = target[:, i], pred = predictions) # target[:, 1] = 64 first words per datapoint and
      # must match with the corresponding predictions and hence loss does gets reduced until 32

      # using teacher forcing (64, 1) total 31 times bcoz "<startseq>" is skipped
      dec_input = tf.reshape(tensor = target[:, i], shape=  (target.shape[0], 1))

  total_loss = loss / int(target.shape[1])

  # Keras models and layers offer the convenient 'variables' and 'trainable_variables' properties,
  # which recursively gather up all dependent variables. This makes it easy to manage variables
  # locally to where they are being used.
  trainable_variables = encoder.trainable_variables + decoder.trainable_variables

  gradients = tape.gradient(loss, trainable_variables)

  optimizer.apply_gradients(grads_and_vars= zip(gradients, trainable_variables))#, global_step= global_step)

  return loss, total_loss
```

In [82]:

```
# set random seed
tf.random.set_seed(seed= 9)

start_time= time.time()

for i in range(start_epoch, epochs+100):
  start = time.time()
  total_loss = 0

  for (batch, (img_tensor, target)) in enumerate(train_dataset): # (batch, (64, 64, 2048), (64, 32))
    batch_loss, t_loss = train_step(img_tensor, target)
    total_loss += t_loss

    if batch % 100 == 0:
      print ('Epoch {} Batch {} Loss {:.4f}'.format(i + 1, batch, batch_loss.numpy() / int(target.shape[1])))

  # storing the epoch end loss value to plot later
  loss_plot.append(total_loss / num_steps)

  # Tensorboard
  with summary_writer.as_default():
    tf.summary.scalar('LossPlot', (total_loss/ num_steps), step= i)

  if i % 5 == 0:
    ckpt_manager.save()

  print('Epoch {} Loss {:.6f}'.format(i + 1, total_loss / num_steps))
  print('Time taken for 1 epochs {} sec\n'.format(time.time() - start))

print("Time Taken is: " + str(time.time() - start_time))
```

```
Epoch 114 Batch 0 Loss 0.6484
Epoch 114 Batch 100 Loss 0.6719
Epoch 114 Batch 200 Loss 0.6365
Epoch 114 Batch 300 Loss 0.6132
Epoch 114 Batch 400 Loss 0.5846
```

Epoch 114 Batch 400 Loss 0.5846
Epoch 114 Loss 0.611588
Time taken for 1 epochs 163.72888374328613 sec

Epoch 115 Batch 0 Loss 0.7049
Epoch 115 Batch 100 Loss 0.6194
Epoch 115 Batch 200 Loss 0.6606
Epoch 115 Batch 300 Loss 0.5470
Epoch 115 Batch 400 Loss 0.6234
Epoch 115 Loss 0.611014
Time taken for 1 epochs 129.36210942268372 sec

Epoch 116 Batch 0 Loss 0.6384
Epoch 116 Batch 100 Loss 0.5671
Epoch 116 Batch 200 Loss 0.5676
Epoch 116 Batch 300 Loss 0.5374
Epoch 116 Batch 400 Loss 0.6345
Epoch 116 Loss 0.610961
Time taken for 1 epochs 130.1203851699829 sec

Epoch 117 Batch 0 Loss 0.7899
Epoch 117 Batch 100 Loss 0.6648
Epoch 117 Batch 200 Loss 0.6096
Epoch 117 Batch 300 Loss 0.6613
Epoch 117 Batch 400 Loss 0.6942
Epoch 117 Loss 0.609951
Time taken for 1 epochs 129.32949376106262 sec

Epoch 118 Batch 0 Loss 0.5894
Epoch 118 Batch 100 Loss 0.5605
Epoch 118 Batch 200 Loss 0.6184
Epoch 118 Batch 300 Loss 0.5336
Epoch 118 Batch 400 Loss 0.5607
Epoch 118 Loss 0.609458
Time taken for 1 epochs 129.37587094306946 sec

Epoch 119 Batch 0 Loss 0.6971
Epoch 119 Batch 100 Loss 0.6140
Epoch 119 Batch 200 Loss 0.5257
Epoch 119 Batch 300 Loss 0.5776
Epoch 119 Batch 400 Loss 0.5495
Epoch 119 Loss 0.607742
Time taken for 1 epochs 129.29529690742493 sec

Epoch 120 Batch 0 Loss 0.7037
Epoch 120 Batch 100 Loss 0.6111
Epoch 120 Batch 200 Loss 0.5725
Epoch 120 Batch 300 Loss 0.5729
Epoch 120 Batch 400 Loss 0.5158
Epoch 120 Loss 0.607004
Time taken for 1 epochs 129.4304530620575 sec

Epoch 121 Batch 0 Loss 0.6566
Epoch 121 Batch 100 Loss 0.5899
Epoch 121 Batch 200 Loss 0.6217
Epoch 121 Batch 300 Loss 0.5635
Epoch 121 Batch 400 Loss 0.6128
Epoch 121 Loss 0.605062
Time taken for 1 epochs 130.04561257362366 sec

Epoch 122 Batch 0 Loss 0.6506
Epoch 122 Batch 100 Loss 0.5379
Epoch 122 Batch 200 Loss 0.5584
Epoch 122 Batch 300 Loss 0.6678
Epoch 122 Batch 400 Loss 0.6154
Epoch 122 Loss 0.605525
Time taken for 1 epochs 129.3939745426178 sec

Epoch 123 Batch 0 Loss 0.6691
Epoch 123 Batch 100 Loss 0.6265
Epoch 123 Batch 200 Loss 0.6304
Epoch 123 Batch 300 Loss 0.6511
Epoch 123 Batch 400 Loss 0.6496
Epoch 123 Loss 0.605613
Time taken for 1 epochs 129.43552899360657 sec

Epoch 124 Batch 0 Loss 0.6875
Epoch 124 Batch 100 Loss 0.6278
Epoch 124 Batch 200 Loss 0.5618
Epoch 124 Batch 300 Loss 0.5434
Epoch 124 Batch 400 Loss 0.5683
Epoch 124 Loss 0.605701
Time taken for 1 epochs 129.32611513137817 sec

```
Epoch 125 Batch 0 Loss 0.6935
Epoch 125 Batch 100 Loss 0.5620
Epoch 125 Batch 200 Loss 0.5200
Epoch 125 Batch 300 Loss 0.6276
Epoch 125 Batch 400 Loss 0.6184
Epoch 125 Loss 0.603331
Time taken for 1 epochs 129.3937759399414 sec

Epoch 126 Batch 0 Loss 0.6230
Epoch 126 Batch 100 Loss 0.5416
Epoch 126 Batch 200 Loss 0.6534
Epoch 126 Batch 300 Loss 0.6251
Epoch 126 Batch 400 Loss 0.5970
Epoch 126 Loss 0.602354
Time taken for 1 epochs 131.12012648582458 sec

Epoch 127 Batch 0 Loss 0.6684
Epoch 127 Batch 100 Loss 0.6622
Epoch 127 Batch 200 Loss 0.5587
Epoch 127 Batch 300 Loss 0.5710
Epoch 127 Batch 400 Loss 0.5851
Epoch 127 Loss 0.603500
Time taken for 1 epochs 129.4444420337677 sec

Epoch 128 Batch 0 Loss 0.6376
Epoch 128 Batch 100 Loss 0.6189
Epoch 128 Batch 200 Loss 0.5928
Epoch 128 Batch 300 Loss 0.5835
Epoch 128 Batch 400 Loss 0.6003
Epoch 128 Loss 0.602058
Time taken for 1 epochs 129.34798645973206 sec

Epoch 129 Batch 0 Loss 0.6464
Epoch 129 Batch 100 Loss 0.6084
Epoch 129 Batch 200 Loss 0.5744
Epoch 129 Batch 300 Loss 0.6236
Epoch 129 Batch 400 Loss 0.5688
Epoch 129 Loss 0.602518
Time taken for 1 epochs 129.35055541992188 sec

Epoch 130 Batch 0 Loss 0.6027
Epoch 130 Batch 100 Loss 0.5642
Epoch 130 Batch 200 Loss 0.5810
Epoch 130 Batch 300 Loss 0.6482
Epoch 130 Batch 400 Loss 0.5855
Epoch 130 Loss 0.600567
Time taken for 1 epochs 129.33909106254578 sec

Epoch 131 Batch 0 Loss 0.6587
Epoch 131 Batch 100 Loss 0.6423
Epoch 131 Batch 200 Loss 0.6565
Epoch 131 Batch 300 Loss 0.5787
Epoch 131 Batch 400 Loss 0.5774
Epoch 131 Loss 0.600254
Time taken for 1 epochs 131.12042498588562 sec

Epoch 132 Batch 0 Loss 0.5930
Epoch 132 Batch 100 Loss 0.6380
Epoch 132 Batch 200 Loss 0.5974
Epoch 132 Batch 300 Loss 0.5880
Epoch 132 Batch 400 Loss 0.5592
Epoch 132 Loss 0.599059
Time taken for 1 epochs 129.49289441108704 sec

Epoch 133 Batch 0 Loss 0.7391
Epoch 133 Batch 100 Loss 0.6040
Epoch 133 Batch 200 Loss 0.5745
Epoch 133 Batch 300 Loss 0.5956
Epoch 133 Batch 400 Loss 0.5727
Epoch 133 Loss 0.597989
Time taken for 1 epochs 129.47803115844727 sec

Epoch 134 Batch 0 Loss 0.6376
Epoch 134 Batch 100 Loss 0.6078
Epoch 134 Batch 200 Loss 0.5888
Epoch 134 Batch 300 Loss 0.5800
Epoch 134 Batch 400 Loss 0.5477
Epoch 134 Loss 0.597688
Time taken for 1 epochs 129.29650402069092 sec

Epoch 135 Batch 0 Loss 0.6696
Epoch 135 Batch 100 Loss 0.6672
```

Epoch 135 Batch 100 Loss 0.6072
Epoch 135 Batch 200 Loss 0.6761
Epoch 135 Batch 300 Loss 0.6556
Epoch 135 Batch 400 Loss 0.5067
Epoch 135 Loss 0.597218
Time taken for 1 epochs 129.33413457870483 sec

Epoch 136 Batch 0 Loss 0.6879
Epoch 136 Batch 100 Loss 0.5331
Epoch 136 Batch 200 Loss 0.6294
Epoch 136 Batch 300 Loss 0.6091
Epoch 136 Batch 400 Loss 0.6073
Epoch 136 Loss 0.596846
Time taken for 1 epochs 130.1009931564331 sec

Epoch 137 Batch 0 Loss 0.6486
Epoch 137 Batch 100 Loss 0.5280
Epoch 137 Batch 200 Loss 0.5918
Epoch 137 Batch 300 Loss 0.6290
Epoch 137 Batch 400 Loss 0.6046
Epoch 137 Loss 0.597135
Time taken for 1 epochs 129.39744877815247 sec

Epoch 138 Batch 0 Loss 0.5934
Epoch 138 Batch 100 Loss 0.5837
Epoch 138 Batch 200 Loss 0.6371
Epoch 138 Batch 300 Loss 0.5912
Epoch 138 Batch 400 Loss 0.5677
Epoch 138 Loss 0.595903
Time taken for 1 epochs 129.4817910194397 sec

Epoch 139 Batch 0 Loss 0.6324
Epoch 139 Batch 100 Loss 0.6586
Epoch 139 Batch 200 Loss 0.6004
Epoch 139 Batch 300 Loss 0.5493
Epoch 139 Batch 400 Loss 0.5405
Epoch 139 Loss 0.594208
Time taken for 1 epochs 129.56749296188354 sec

Epoch 140 Batch 0 Loss 0.6177
Epoch 140 Batch 100 Loss 0.5585
Epoch 140 Batch 200 Loss 0.5829
Epoch 140 Batch 300 Loss 0.5483
Epoch 140 Batch 400 Loss 0.6103
Epoch 140 Loss 0.594541
Time taken for 1 epochs 129.54147505760193 sec

Epoch 141 Batch 0 Loss 0.7075
Epoch 141 Batch 100 Loss 0.5501
Epoch 141 Batch 200 Loss 0.6175
Epoch 141 Batch 300 Loss 0.5858
Epoch 141 Batch 400 Loss 0.5789
Epoch 141 Loss 0.595555
Time taken for 1 epochs 131.18875360488892 sec

Epoch 142 Batch 0 Loss 0.6421
Epoch 142 Batch 100 Loss 0.6187
Epoch 142 Batch 200 Loss 0.5749
Epoch 142 Batch 300 Loss 0.5027
Epoch 142 Batch 400 Loss 0.5855
Epoch 142 Loss 0.594528
Time taken for 1 epochs 129.29490327835083 sec

Epoch 143 Batch 0 Loss 0.6026
Epoch 143 Batch 100 Loss 0.5973
Epoch 143 Batch 200 Loss 0.6879
Epoch 143 Batch 300 Loss 0.5475
Epoch 143 Batch 400 Loss 0.5958
Epoch 143 Loss 0.592734
Time taken for 1 epochs 129.39008331298828 sec

Epoch 144 Batch 0 Loss 0.5937
Epoch 144 Batch 100 Loss 0.6224
Epoch 144 Batch 200 Loss 0.6287
Epoch 144 Batch 300 Loss 0.5805
Epoch 144 Batch 400 Loss 0.5896
Epoch 144 Loss 0.592077
Time taken for 1 epochs 129.40058588981628 sec

Epoch 145 Batch 0 Loss 0.6383
Epoch 145 Batch 100 Loss 0.6198
Epoch 145 Batch 200 Loss 0.5488
Epoch 145 Batch 300 Loss 0.6213

```
Epoch 145 Batch 400 Loss 0.5374
Epoch 145 Loss 0.593512
Time taken for 1 epochs 129.3157923221588 sec

Epoch 146 Batch 0 Loss 0.6373
Epoch 146 Batch 100 Loss 0.5262
Epoch 146 Batch 200 Loss 0.5230
Epoch 146 Batch 300 Loss 0.5582
Epoch 146 Batch 400 Loss 0.6395
Epoch 146 Loss 0.591869
Time taken for 1 epochs 131.06957364082336 sec

Epoch 147 Batch 0 Loss 0.6588
Epoch 147 Batch 100 Loss 0.5250
Epoch 147 Batch 200 Loss 0.5847
Epoch 147 Batch 300 Loss 0.5904
Epoch 147 Batch 400 Loss 0.5044
Epoch 147 Loss 0.591854
Time taken for 1 epochs 129.4037253856659 sec

Epoch 148 Batch 0 Loss 0.7168
Epoch 148 Batch 100 Loss 0.6245
Epoch 148 Batch 200 Loss 0.5753
Epoch 148 Batch 300 Loss 0.5633
Epoch 148 Batch 400 Loss 0.6007
Epoch 148 Loss 0.590867
Time taken for 1 epochs 129.37575817108154 sec

Epoch 149 Batch 0 Loss 0.7271
Epoch 149 Batch 100 Loss 0.5628
Epoch 149 Batch 200 Loss 0.6323
Epoch 149 Batch 300 Loss 0.5343
Epoch 149 Batch 400 Loss 0.6041
Epoch 149 Loss 0.589197
Time taken for 1 epochs 129.49817514419556 sec

Epoch 150 Batch 0 Loss 0.7378
Epoch 150 Batch 100 Loss 0.6176
Epoch 150 Batch 200 Loss 0.6247
Epoch 150 Batch 300 Loss 0.5464
Epoch 150 Batch 400 Loss 0.5668
Epoch 150 Loss 0.590142
Time taken for 1 epochs 129.27155303955078 sec

Time Taken is: 4830.862156152725
```

In [83]:

```
ckpt_manager.latest_checkpoint
```

Out[83]:

```
'/content/drive/My Drive/ckpt/ckpt-29'
```

In [84]:

```
if ckpt_manager.latest_checkpoint:
  start_epoch = 150
```

In [85]:

```
loss_plot= []

@tf.function
def train_step(img_tensor, target):
  loss = 0

  # initializing the hidden state for each batch
  # because the captions are not related from image to image
  # hidden = (64, 512)
  hidden = decoder.reset_state(encoder(img_tensor))

  # dec_input = (64, 1)
  dec_input = tf.reshape(tensor= [tokenizer.word_index['<startseq>']] * target.shape[0], shape= (target.shape[0], 1))

  with tf.GradientTape() as tape:
    # img_tensor = (64, 64, 2048), target = (64, 32)
    features = encoder(img_tensor) # (64, 64, 2048) to (64, 64, 300)
```

```
  for i in range(1, target.shape[1]):

    # passing the features through the decoder
    # predictions = (64, 5001), hidden = (64, 512)
    predictions, hidden_state, alignment_vector = decoder(dec_input, features, hidden)

    loss += loss_func(real = target[:, i], pred = predictions) # target[:, 1] = 64 first words per datapoint and
    # must match with the corresponding predictions and hence loss does gets reduced until 32

    # using teacher forcing (64, 1) total 31 times bcoz "<startseq>" is skipped
    dec_input = tf.reshape(tensor = target[:, i], shape=  (target.shape[0], 1))

  total_loss = loss / int(target.shape[1])

  # Keras models and layers offer the convenient 'variables' and 'trainable_variables' properties,
  # which recursively gather up all dependent variables. This makes it easy to manage variables
  # locally to where they are being used.
  trainable_variables = encoder.trainable_variables + decoder.trainable_variables

  gradients = tape.gradient(loss, trainable_variables)

  optimizer.apply_gradients(grads_and_vars= zip(gradients, trainable_variables))#, global_step= global_step)

  return loss, total_loss
```

In [87]:

```
# set random seed
tf.random.set_seed(seed= 9)

start_time= time.time()

for i in range(start_epoch, epochs+150):
  start = time.time()
  total_loss = 0

  for (batch, (img_tensor, target)) in enumerate(train_dataset): # (batch, (64, 64, 2048), (64, 32))
    batch_loss, t_loss = train_step(img_tensor, target)
    total_loss += t_loss

    if batch % 100 == 0:
      print ('Epoch {} Batch {} Loss {:.4f}'.format(i + 1, batch, batch_loss.numpy() / int(target.shape[1])))

  # storing the epoch end loss value to plot later
  loss_plot.append(total_loss / num_steps)

  # Tensorboard
  with summary_writer.as_default():
    tf.summary.scalar('LossPlot', (total_loss/ num_steps), step= i)

  if i % 5 == 0:
    ckpt_manager.save()

  print('Epoch {} Loss {:.6f}'.format(i + 1, total_loss / num_steps))
  print('Time taken for 1 epochs {} sec\n'.format(time.time() - start))

print("Time Taken is: " + str(time.time() - start_time))
```

```
Epoch 151 Batch 0 Loss 0.6618
Epoch 151 Batch 100 Loss 0.6017
Epoch 151 Batch 200 Loss 0.5184
Epoch 151 Batch 300 Loss 0.5345
Epoch 151 Batch 400 Loss 0.4879
Epoch 151 Loss 0.590583
Time taken for 1 epochs 164.17692351341248 sec

Epoch 152 Batch 0 Loss 0.7274
Epoch 152 Batch 100 Loss 0.5800
Epoch 152 Batch 200 Loss 0.6120
Epoch 152 Batch 300 Loss 0.6074
Epoch 152 Batch 400 Loss 0.6094
Epoch 152 Loss 0.588629
Time taken for 1 epochs 129.44908213615417 sec

Epoch 153 Batch 0 Loss 0.6024
Epoch 153 Batch 100 Loss 0.5703
Epoch 153 Batch 200 Loss 0.6502
Epoch 153 Batch 300 Loss 0.5855
Epoch 153 Batch 400 Loss 0.5922
Epoch 153 Loss 0.589783
Time taken for 1 epochs 129.51028323173523 sec
```

```
Epoch 154 Batch 0 Loss 0.7248
Epoch 154 Batch 100 Loss 0.5897
Epoch 154 Batch 200 Loss 0.6263
Epoch 154 Batch 300 Loss 0.5363
Epoch 154 Batch 400 Loss 0.6466
Epoch 154 Loss 0.588054
Time taken for 1 epochs 129.542578458786 sec

Epoch 155 Batch 0 Loss 0.6219
Epoch 155 Batch 100 Loss 0.5247
Epoch 155 Batch 200 Loss 0.5729
Epoch 155 Batch 300 Loss 0.5909
Epoch 155 Batch 400 Loss 0.4892
Epoch 155 Loss 0.587249
Time taken for 1 epochs 129.51015615463257 sec

Epoch 156 Batch 0 Loss 0.6013
Epoch 156 Batch 100 Loss 0.5778
Epoch 156 Batch 200 Loss 0.5841
Epoch 156 Batch 300 Loss 0.5893
Epoch 156 Batch 400 Loss 0.5948
Epoch 156 Loss 0.588262
Time taken for 1 epochs 131.20460844039917 sec

Epoch 157 Batch 0 Loss 0.6378
Epoch 157 Batch 100 Loss 0.6057
Epoch 157 Batch 200 Loss 0.5616
Epoch 157 Batch 300 Loss 0.6080
Epoch 157 Batch 400 Loss 0.6536
Epoch 157 Loss 0.586416
Time taken for 1 epochs 129.492045879364 sec

Epoch 158 Batch 0 Loss 0.6962
Epoch 158 Batch 100 Loss 0.5751
Epoch 158 Batch 200 Loss 0.6130
Epoch 158 Batch 300 Loss 0.5376
Epoch 158 Batch 400 Loss 0.5308
Epoch 158 Loss 0.585742
Time taken for 1 epochs 129.4658124446869 sec

Epoch 159 Batch 0 Loss 0.7529
Epoch 159 Batch 100 Loss 0.5023
Epoch 159 Batch 200 Loss 0.6056
Epoch 159 Batch 300 Loss 0.5716
Epoch 159 Batch 400 Loss 0.5634
Epoch 159 Loss 0.584915
Time taken for 1 epochs 129.51373600959778 sec

Epoch 160 Batch 0 Loss 0.7068
Epoch 160 Batch 100 Loss 0.6033
Epoch 160 Batch 200 Loss 0.5904
Epoch 160 Batch 300 Loss 0.5753
Epoch 160 Batch 400 Loss 0.5919
Epoch 160 Loss 0.585613
Time taken for 1 epochs 129.6074559688568 sec

Epoch 161 Batch 0 Loss 0.6677
Epoch 161 Batch 100 Loss 0.5920
Epoch 161 Batch 200 Loss 0.5800
Epoch 161 Batch 300 Loss 0.5541
Epoch 161 Batch 400 Loss 0.5595
Epoch 161 Loss 0.586555
Time taken for 1 epochs 131.32144808769226 sec

Epoch 162 Batch 0 Loss 0.6366
Epoch 162 Batch 100 Loss 0.5822
Epoch 162 Batch 200 Loss 0.5465
Epoch 162 Batch 300 Loss 0.6492
Epoch 162 Batch 400 Loss 0.5939
Epoch 162 Loss 0.586337
Time taken for 1 epochs 129.57906007766724 sec

Epoch 163 Batch 0 Loss 0.6669
Epoch 163 Batch 100 Loss 0.6233
Epoch 163 Batch 200 Loss 0.4859
Epoch 163 Batch 300 Loss 0.5262
Epoch 163 Batch 400 Loss 0.6423
Epoch 163 Loss 0.584049
Time taken for 1 epochs 129.50962829589844 sec

Epoch 164 Batch 0 Loss 0.6726
Epoch 164 Batch 100 Loss 0.5926
```

Epoch 164 Batch 200 Loss 0.6003
Epoch 164 Batch 300 Loss 0.5754
Epoch 164 Batch 400 Loss 0.5563
Epoch 164 Loss 0.584837
Time taken for 1 epochs 129.513117313385 sec

Epoch 165 Batch 0 Loss 0.7030
Epoch 165 Batch 100 Loss 0.6316
Epoch 165 Batch 200 Loss 0.6086
Epoch 165 Batch 300 Loss 0.5588
Epoch 165 Batch 400 Loss 0.5843
Epoch 165 Loss 0.583817
Time taken for 1 epochs 129.56684923171997 sec

Epoch 166 Batch 0 Loss 0.5751
Epoch 166 Batch 100 Loss 0.6391
Epoch 166 Batch 200 Loss 0.5587
Epoch 166 Batch 300 Loss 0.5197
Epoch 166 Batch 400 Loss 0.6258
Epoch 166 Loss 0.584728
Time taken for 1 epochs 130.23449969291687 sec

Epoch 167 Batch 0 Loss 0.7419
Epoch 167 Batch 100 Loss 0.5806
Epoch 167 Batch 200 Loss 0.5869
Epoch 167 Batch 300 Loss 0.6172
Epoch 167 Batch 400 Loss 0.5037
Epoch 167 Loss 0.581478
Time taken for 1 epochs 129.5051510334015 sec

Epoch 168 Batch 0 Loss 0.6450
Epoch 168 Batch 100 Loss 0.6363
Epoch 168 Batch 200 Loss 0.5586
Epoch 168 Batch 300 Loss 0.5523
Epoch 168 Batch 400 Loss 0.5186
Epoch 168 Loss 0.582484
Time taken for 1 epochs 129.37760710716248 sec

Epoch 169 Batch 0 Loss 0.6771
Epoch 169 Batch 100 Loss 0.5372
Epoch 169 Batch 200 Loss 0.6114
Epoch 169 Batch 300 Loss 0.6002
Epoch 169 Batch 400 Loss 0.5945
Epoch 169 Loss 0.580894
Time taken for 1 epochs 129.50735330581665 sec

Epoch 170 Batch 0 Loss 0.6749
Epoch 170 Batch 100 Loss 0.6276
Epoch 170 Batch 200 Loss 0.5775
Epoch 170 Batch 300 Loss 0.5587
Epoch 170 Batch 400 Loss 0.5843
Epoch 170 Loss 0.582340
Time taken for 1 epochs 129.39967918395996 sec

Epoch 171 Batch 0 Loss 0.7140
Epoch 171 Batch 100 Loss 0.6007
Epoch 171 Batch 200 Loss 0.5798
Epoch 171 Batch 300 Loss 0.5438
Epoch 171 Batch 400 Loss 0.5803
Epoch 171 Loss 0.580343
Time taken for 1 epochs 131.16550636291504 sec

Epoch 172 Batch 0 Loss 0.6251
Epoch 172 Batch 100 Loss 0.6044
Epoch 172 Batch 200 Loss 0.5977
Epoch 172 Batch 300 Loss 0.5414
Epoch 172 Batch 400 Loss 0.6109
Epoch 172 Loss 0.579784
Time taken for 1 epochs 129.41586828231812 sec

Epoch 173 Batch 0 Loss 0.6424
Epoch 173 Batch 100 Loss 0.5183
Epoch 173 Batch 200 Loss 0.5555
Epoch 173 Batch 300 Loss 0.5667
Epoch 173 Batch 400 Loss 0.4780
Epoch 173 Loss 0.579930
Time taken for 1 epochs 129.45251369476318 sec

Epoch 174 Batch 0 Loss 0.6611
Epoch 174 Batch 100 Loss 0.5919
Epoch 174 Batch 200 Loss 0.5827
Epoch 174 Batch 300 Loss 0.5400

Epoch 174 Batch 400 Loss 0.5413
Epoch 174 Loss 0.580356
Time taken for 1 epochs 129.54307794570923 sec

Epoch 175 Batch 0 Loss 0.5421
Epoch 175 Batch 100 Loss 0.5714
Epoch 175 Batch 200 Loss 0.6329
Epoch 175 Batch 300 Loss 0.5999
Epoch 175 Batch 400 Loss 0.5729
Epoch 175 Loss 0.581215
Time taken for 1 epochs 129.47649025917053 sec

Epoch 176 Batch 0 Loss 0.5742
Epoch 176 Batch 100 Loss 0.6301
Epoch 176 Batch 200 Loss 0.6287
Epoch 176 Batch 300 Loss 0.5075
Epoch 176 Batch 400 Loss 0.5581
Epoch 176 Loss 0.578794
Time taken for 1 epochs 131.3936207294464 sec

Epoch 177 Batch 0 Loss 0.6668
Epoch 177 Batch 100 Loss 0.5979
Epoch 177 Batch 200 Loss 0.5345
Epoch 177 Batch 300 Loss 0.5639
Epoch 177 Batch 400 Loss 0.6196
Epoch 177 Loss 0.580729
Time taken for 1 epochs 129.5371606349945 sec

Epoch 178 Batch 0 Loss 0.6711
Epoch 178 Batch 100 Loss 0.5825
Epoch 178 Batch 200 Loss 0.5650
Epoch 178 Batch 300 Loss 0.5313
Epoch 178 Batch 400 Loss 0.5934
Epoch 178 Loss 0.577752
Time taken for 1 epochs 129.4258472919464 sec

Epoch 179 Batch 0 Loss 0.6234
Epoch 179 Batch 100 Loss 0.5438
Epoch 179 Batch 200 Loss 0.5649
Epoch 179 Batch 300 Loss 0.5396
Epoch 179 Batch 400 Loss 0.5903
Epoch 179 Loss 0.579631
Time taken for 1 epochs 129.5435438156128 sec

Epoch 180 Batch 0 Loss 0.6533
Epoch 180 Batch 100 Loss 0.5596
Epoch 180 Batch 200 Loss 0.5774
Epoch 180 Batch 300 Loss 0.5916
Epoch 180 Batch 400 Loss 0.5843
Epoch 180 Loss 0.578339
Time taken for 1 epochs 129.49614477157593 sec

Epoch 181 Batch 0 Loss 0.6225
Epoch 181 Batch 100 Loss 0.6129
Epoch 181 Batch 200 Loss 0.6670
Epoch 181 Batch 300 Loss 0.6326
Epoch 181 Batch 400 Loss 0.5860
Epoch 181 Loss 0.577626
Time taken for 1 epochs 131.23964309692383 sec

Epoch 182 Batch 0 Loss 0.6881
Epoch 182 Batch 100 Loss 0.5334
Epoch 182 Batch 200 Loss 0.5410
Epoch 182 Batch 300 Loss 0.6732
Epoch 182 Batch 400 Loss 0.5579
Epoch 182 Loss 0.577103
Time taken for 1 epochs 129.5015003681183 sec

Epoch 183 Batch 0 Loss 0.5938
Epoch 183 Batch 100 Loss 0.5999
Epoch 183 Batch 200 Loss 0.5663
Epoch 183 Batch 300 Loss 0.5838
Epoch 183 Batch 400 Loss 0.5491
Epoch 183 Loss 0.577157
Time taken for 1 epochs 129.4216742515564 sec

Epoch 184 Batch 0 Loss 0.6330
Epoch 184 Batch 100 Loss 0.6406
Epoch 184 Batch 200 Loss 0.5868
Epoch 184 Batch 300 Loss 0.5287
Epoch 184 Batch 400 Loss 0.5995
Epoch 184 Loss 0.576551
Time taken for 1 epochs 129.4736683368683 sec

Epoch 185 Batch 0 Loss 0.6250
Epoch 185 Batch 100 Loss 0.5292
Epoch 185 Batch 200 Loss 0.6238
Epoch 185 Batch 300 Loss 0.5424
Epoch 185 Batch 400 Loss 0.5612
Epoch 185 Loss 0.578152
Time taken for 1 epochs 129.42225074768066 sec

Epoch 186 Batch 0 Loss 0.6730
Epoch 186 Batch 100 Loss 0.6876
Epoch 186 Batch 200 Loss 0.5492
Epoch 186 Batch 300 Loss 0.5725
Epoch 186 Batch 400 Loss 0.6074
Epoch 186 Loss 0.576979
Time taken for 1 epochs 130.29354596138 sec

Epoch 187 Batch 0 Loss 0.7376
Epoch 187 Batch 100 Loss 0.4828
Epoch 187 Batch 200 Loss 0.6158
Epoch 187 Batch 300 Loss 0.6135
Epoch 187 Batch 400 Loss 0.6218
Epoch 187 Loss 0.575938
Time taken for 1 epochs 129.4975700378418 sec

Epoch 188 Batch 0 Loss 0.5690
Epoch 188 Batch 100 Loss 0.5707
Epoch 188 Batch 200 Loss 0.6421
Epoch 188 Batch 300 Loss 0.5387
Epoch 188 Batch 400 Loss 0.5538
Epoch 188 Loss 0.575637
Time taken for 1 epochs 129.5249412059784 sec

Epoch 189 Batch 0 Loss 0.5993
Epoch 189 Batch 100 Loss 0.6525
Epoch 189 Batch 200 Loss 0.6046
Epoch 189 Batch 300 Loss 0.5592
Epoch 189 Batch 400 Loss 0.5684
Epoch 189 Loss 0.576559
Time taken for 1 epochs 129.5307776927948 sec

Epoch 190 Batch 0 Loss 0.6559
Epoch 190 Batch 100 Loss 0.5307
Epoch 190 Batch 200 Loss 0.5412
Epoch 190 Batch 300 Loss 0.5756
Epoch 190 Batch 400 Loss 0.6321
Epoch 190 Loss 0.576822
Time taken for 1 epochs 129.61427450180054 sec

Epoch 191 Batch 0 Loss 0.6428
Epoch 191 Batch 100 Loss 0.5861
Epoch 191 Batch 200 Loss 0.5337
Epoch 191 Batch 300 Loss 0.5638
Epoch 191 Batch 400 Loss 0.6116
Epoch 191 Loss 0.574807
Time taken for 1 epochs 131.23274612426758 sec

Epoch 192 Batch 0 Loss 0.6160
Epoch 192 Batch 100 Loss 0.5475
Epoch 192 Batch 200 Loss 0.4749
Epoch 192 Batch 300 Loss 0.5246
Epoch 192 Batch 400 Loss 0.5922
Epoch 192 Loss 0.573668
Time taken for 1 epochs 129.44420552253723 sec

Epoch 193 Batch 0 Loss 0.5451
Epoch 193 Batch 100 Loss 0.6416
Epoch 193 Batch 200 Loss 0.5078
Epoch 193 Batch 300 Loss 0.4924
Epoch 193 Batch 400 Loss 0.5163
Epoch 193 Loss 0.574692
Time taken for 1 epochs 129.4692144393921 sec

Epoch 194 Batch 0 Loss 0.6691
Epoch 194 Batch 100 Loss 0.6297
Epoch 194 Batch 200 Loss 0.5578
Epoch 194 Batch 300 Loss 0.5738
Epoch 194 Batch 400 Loss 0.5879
Epoch 194 Loss 0.574040
Time taken for 1 epochs 129.53544211387634 sec

Epoch 195 Batch 0 Loss 0.6180

Epoch 195 Batch 100 Loss 0.5536
Epoch 195 Batch 200 Loss 0.6996
Epoch 195 Batch 300 Loss 0.5697
Epoch 195 Batch 400 Loss 0.5075
Epoch 195 Loss 0.573960
Time taken for 1 epochs 129.42593955993652 sec

Epoch 196 Batch 0 Loss 0.6910
Epoch 196 Batch 100 Loss 0.6108
Epoch 196 Batch 200 Loss 0.6263
Epoch 196 Batch 300 Loss 0.5625
Epoch 196 Batch 400 Loss 0.5626
Epoch 196 Loss 0.574013
Time taken for 1 epochs 131.12108707427979 sec

Epoch 197 Batch 0 Loss 0.6604
Epoch 197 Batch 100 Loss 0.5359
Epoch 197 Batch 200 Loss 0.5467
Epoch 197 Batch 300 Loss 0.5180
Epoch 197 Batch 400 Loss 0.5459
Epoch 197 Loss 0.572214
Time taken for 1 epochs 129.43514323234558 sec

Epoch 198 Batch 0 Loss 0.6895
Epoch 198 Batch 100 Loss 0.5717
Epoch 198 Batch 200 Loss 0.6012
Epoch 198 Batch 300 Loss 0.6733
Epoch 198 Batch 400 Loss 0.5943
Epoch 198 Loss 0.573973
Time taken for 1 epochs 129.6772780418396 sec

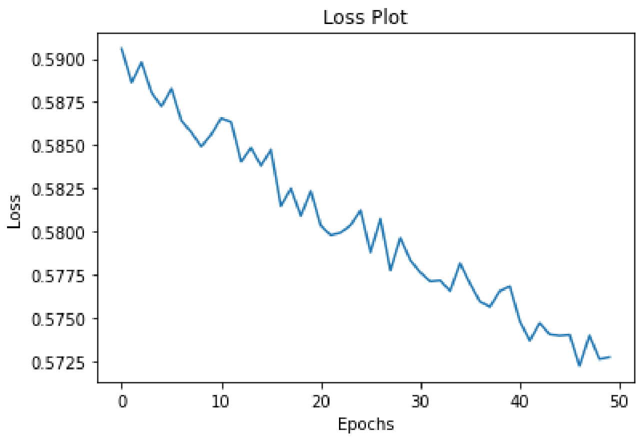
Epoch 199 Batch 0 Loss 0.6895
Epoch 199 Batch 100 Loss 0.5603
Epoch 199 Batch 200 Loss 0.5421
Epoch 199 Batch 300 Loss 0.5521
Epoch 199 Batch 400 Loss 0.5535
Epoch 199 Loss 0.572607
Time taken for 1 epochs 129.37486362457275 sec

Epoch 200 Batch 0 Loss 0.5899
Epoch 200 Batch 100 Loss 0.6621
Epoch 200 Batch 200 Loss 0.5040
Epoch 200 Batch 300 Loss 0.5561
Epoch 200 Batch 400 Loss 0.5882
Epoch 200 Loss 0.572718
Time taken for 1 epochs 129.49344444274902 sec

Time Taken is: 6523.168703794479

In [88]:

```
plt.plot(loss_plot)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



In [89]:

```
%tensorboard --logdir /content/tensorboard
```

Reusing TensorBoard on port 6006 (pid 878), started 6:44:40 ago. (Use '!kill 878' to kill it.)

Prediction Caption!

- The evaluate function is similar to the training loop, except you don't use teacher forcing here. The input to the decoder at each time step is its previous predictions along with the hidden state and the encoder output.
- Stop predicting when the model predicts the end token.
- And store the attention weights for every time step.

In [90]:

```
def plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))

    fig = plt.figure(figsize=(15, 18))

    len_result = len(result)
    for l in range(len_result):
        temp_att = np.resize(attention_plot[l], (8, 8))
        ax = fig.add_subplot(len_result//2, len_result//2, l+1)
        ax.set_title(result[l])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_extent())

    plt.tight_layout()
    plt.show()
```

Greedy Search

- This search is the model generates a 30-long vector(in the sample example while 9181-long vector in the original) which is a probability distribution across all the words in the vocabulary.
- This is called as Maximum Likelihood Estimation (MLE) i.e. we select that word which is most likely according to the model for the given input. And sometimes this method is also called as Greedy Search, as we greedily select the word with maximum probability, given the feature vector and partial caption.

In [93]:

```
def evaluate(image):
    attention_plot = np.zeros((max_length, attention_features_shape))

    temp_input = tf.expand_dims(input = preprocess_image(image)[0],axis = 0)
    img_tensor_val = model(temp_input)
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0], -1, img_tensor_val.shape[3]))

    features = encoder(img_tensor_val)

    hidden = decoder.reset_state(features)

    dec_input = tf.expand_dims([tokenizer.word_index['<startseq>']], 0)
    result = []

    for i in range(max_length):
        predictions, hidden, attention_weights = decoder(dec_input, features, hidden)

        attention_plot[i] = tf.reshape(attention_weights, (-1, )).numpy()

        predicted_id = tf.argmax(predictions[0]).numpy()#tf.random.categorical(predictions, 1)[0][0].numpy()
        result.append(tokenizer.index_word[predicted_id])

        if tokenizer.index_word[predicted_id] == '<endseq>':
            return result, attention_plot

        dec_input = tf.expand_dims([predicted_id], 0)

    attention_plot = attention_plot[:len(result), :]
    return result, attention_plot
```

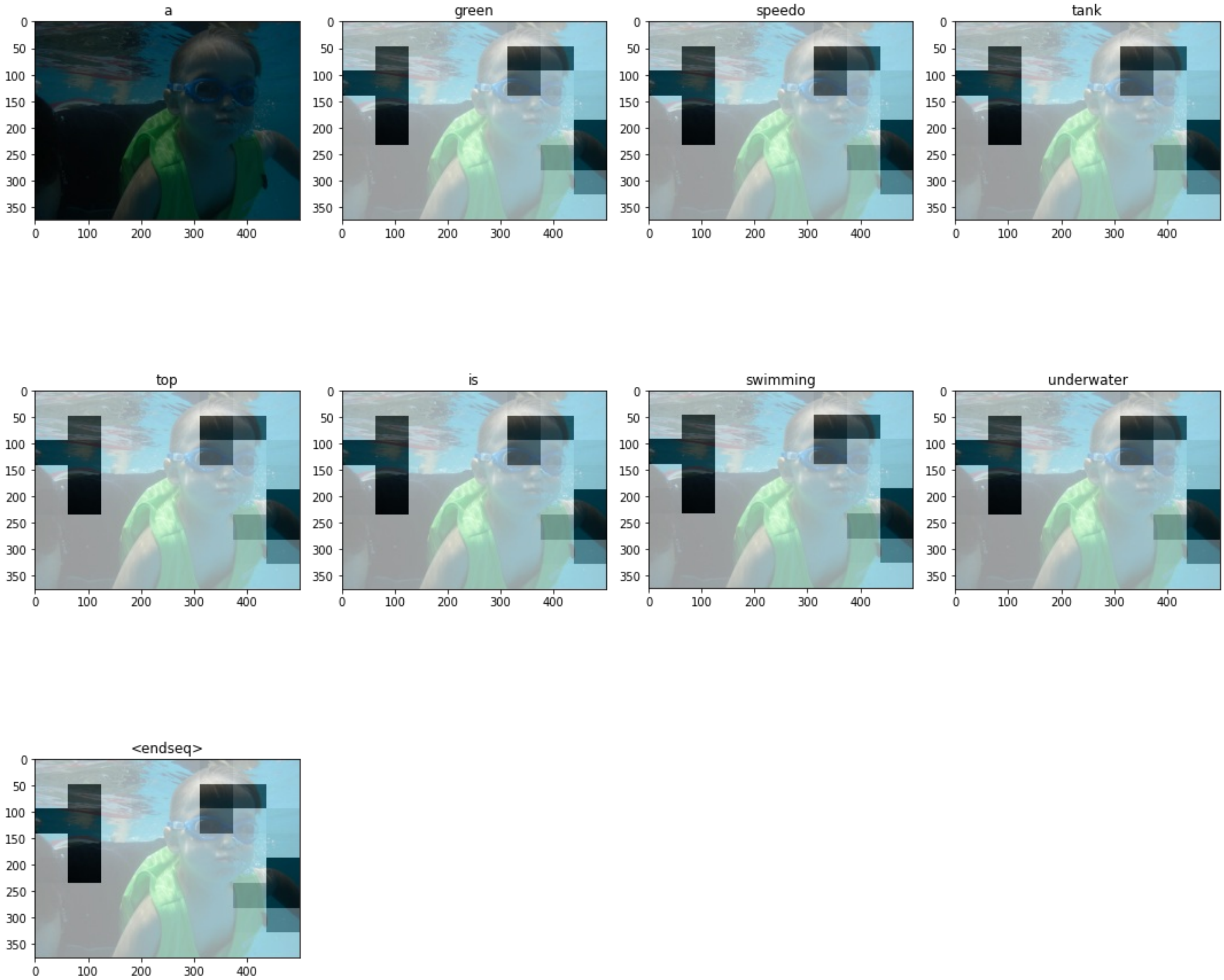
Relevant Predictions

In [102]:

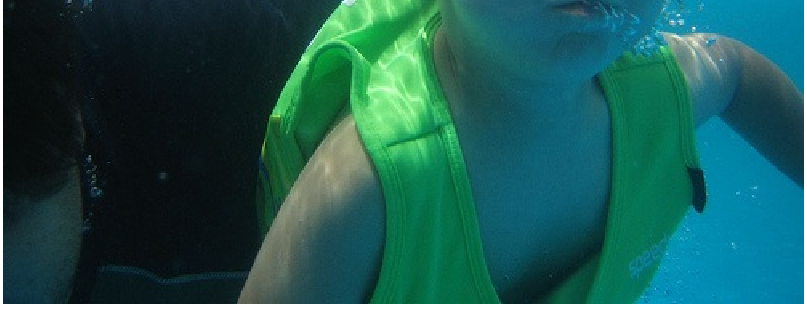
```
from nltk.translate.bleu_score import sentence_bleu
```

```python
# captions on the validation set
rid = np.random.randint(0, len(test_image_names))
image = test_image_names[rid]
real_caption = ' '.join([tokenizer.index_word[i] for i in test_pad_captions[rid] if i not in [0]])
result, attention_plot = evaluate(image)

print ('Real Caption:', real_caption)
print ('Prediction Caption:', ' '.join(result))

reference = []
reference.append(real_caption.split())
candidate = result

score = sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0))
print(f"BELU score: {score*100}")

plot_attention(image, result, attention_plot)
Image.open(image)
```

Real Caption: <startseq> a boy underwater with someone helping him to swim <endseq>
Prediction Caption: a green speedo tank top is swimming underwater <endseq>
BELU score: 46.230595512422084



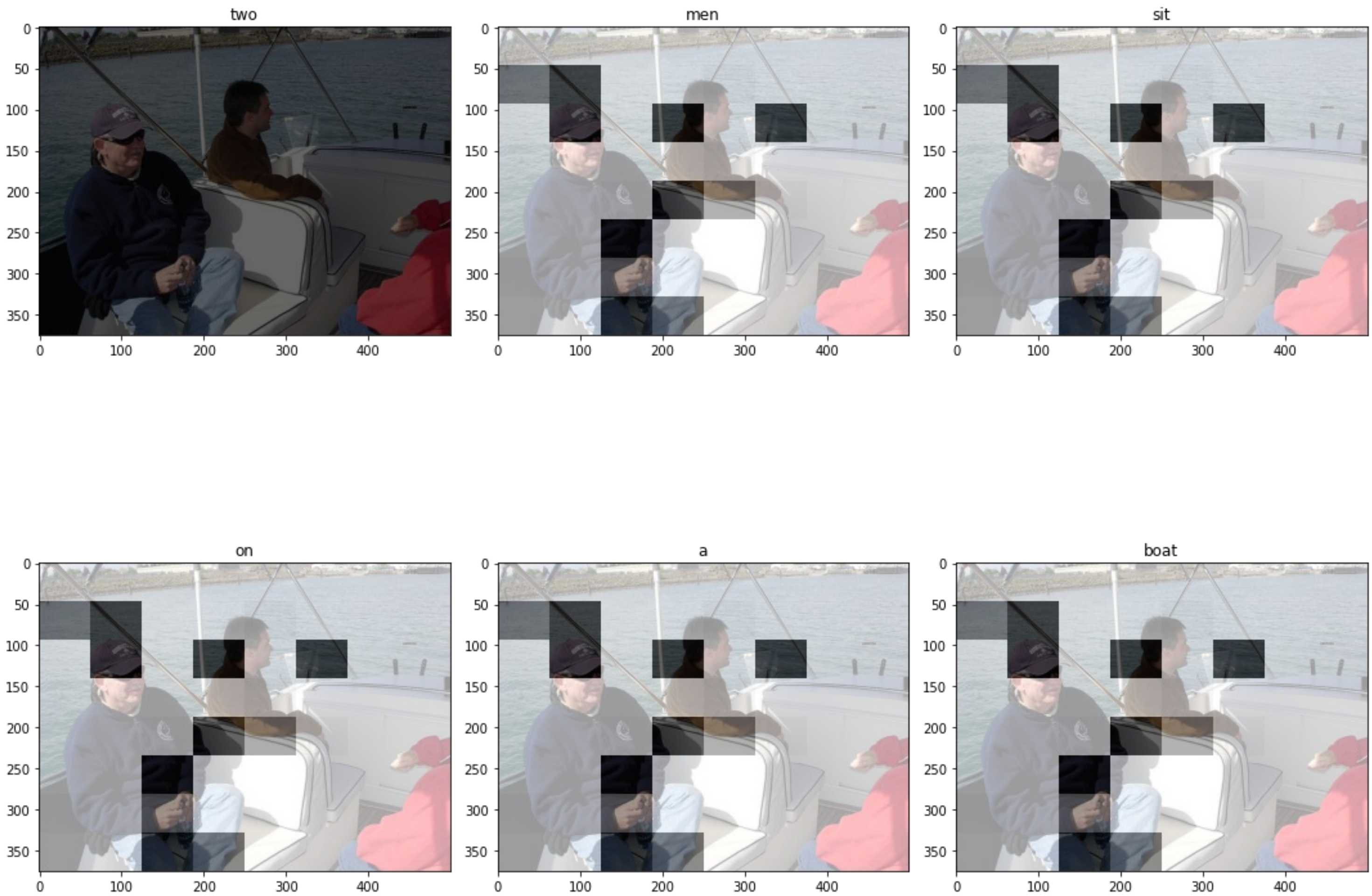
Out[102]:



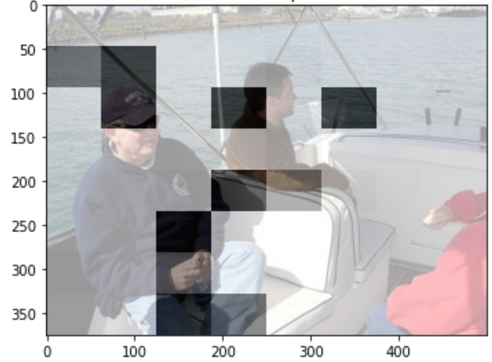
In [119]:

```
# captions on the validation set
rid = np.random.randint(0, len(test_image_names))
image = test_image_names[rid]
real_caption = ' '.join([tokenizer.index_word[i] for i in test_pad_captions[rid] if i not in [0]])
result, attention_plot = evaluate(image)

print ('Real Caption:', real_caption)
print ('Prediction Caption:', ' '.join(result))

reference = []
reference.append(real_caption.split())
candidate = result

score = sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0))
print(f"BELU score: {score*100}")

plot_attention(image, result, attention_plot)
Image.open(image)
```

Real Caption: <startseq> three people are having a conversation on a white boat <endseq>
Prediction Caption: two men sit on a boat <endseq>
BELU score: 21.365349626228948



<endseq>

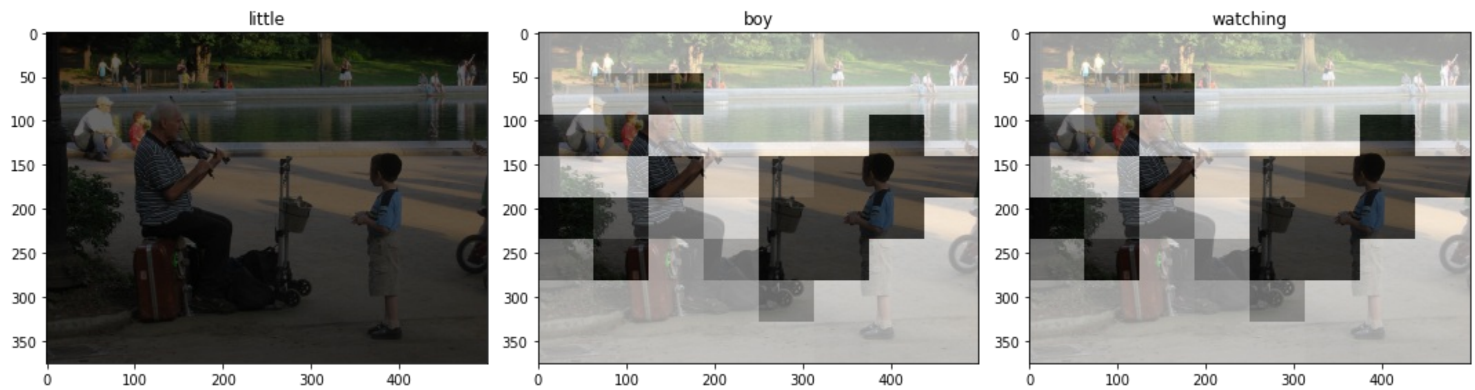


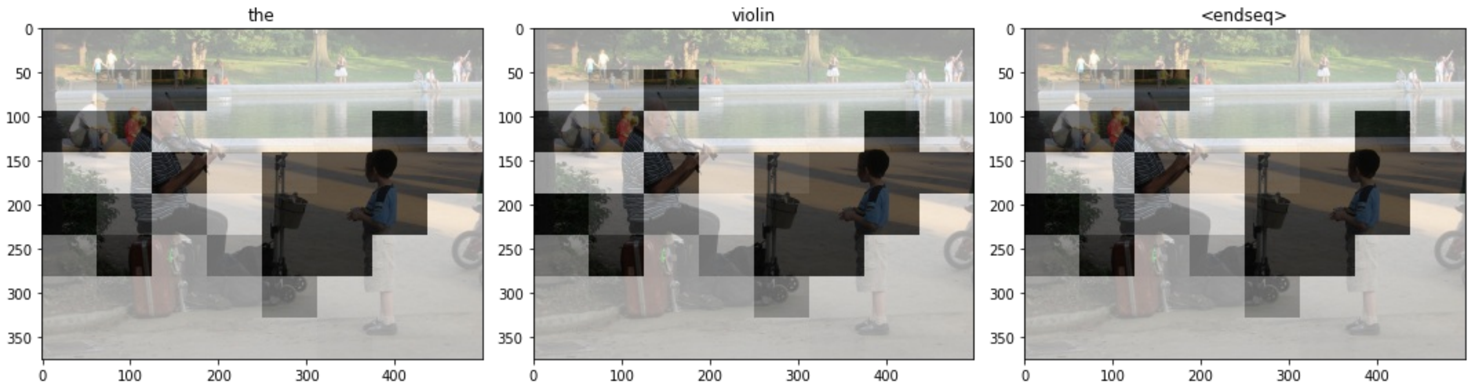
Out[119]:

In [109]:

```python
# captions on the validation set
rid = np.random.randint(0, len(test_image_names))
image = test_image_names[rid]
real_caption = ' '.join([tokenizer.index_word[i] for i in test_pad_captions[rid] if i not in [0]])
result, attention_plot = evaluate(image)

print ('Real Caption:', real_caption)
print ('Prediction Caption:', ' '.join(result))

reference = []
reference.append(real_caption.split())
candidate = result

score = sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0))
print(f"BELU score: {score*100}")

plot_attention(image, result, attention_plot)
Image.open(image)
```

Real Caption: <startseq> a little jewish boy is watching an old man playing the violin in a park <endseq>
Prediction Caption: little boy watching the violin <endseq>
BELU score: 7.150039609192022





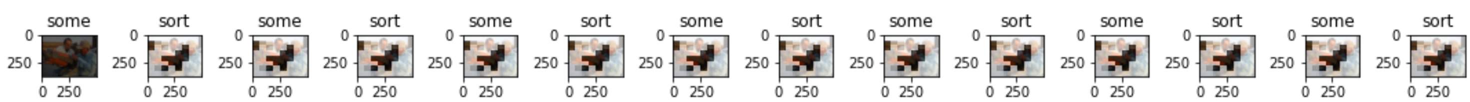
Out[109]:

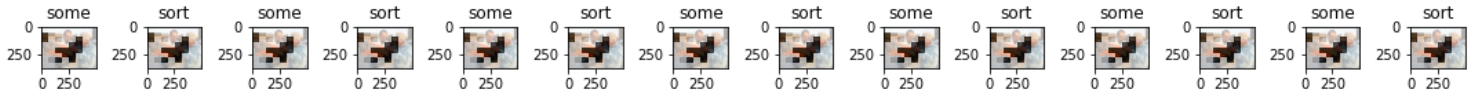
Irrelevant Captions

In [120]:

```python
# captions on the validation set
rid = np.random.randint(0, len(test_image_names))
image = test_image_names[rid]
real_caption = ' '.join([tokenizer.index_word[i] for i in test_pad_captions[rid] if i not in [0]])
result, attention_plot = evaluate(image)

print ('Real Caption:', real_caption)
print ('Prediction Caption:', ' '.join(result))

reference = []
reference.append(real_caption.split())
candidate = result

score = sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0))
print(f"BELU score: {score*100}")

plot_attention(image, result, attention_plot)
Image.open(image)
```

Real Caption: <startseq> a man fixes a telescope like a machine <endseq>
Prediction Caption: some sort some sort some sort some sort some sort some sort some sort some sort some sort some sort some sort some sort so me sort some sort
BELU score: 0



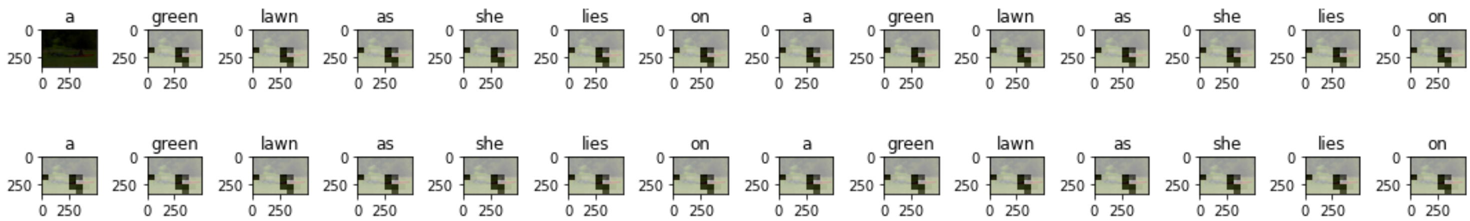


Out[120]:

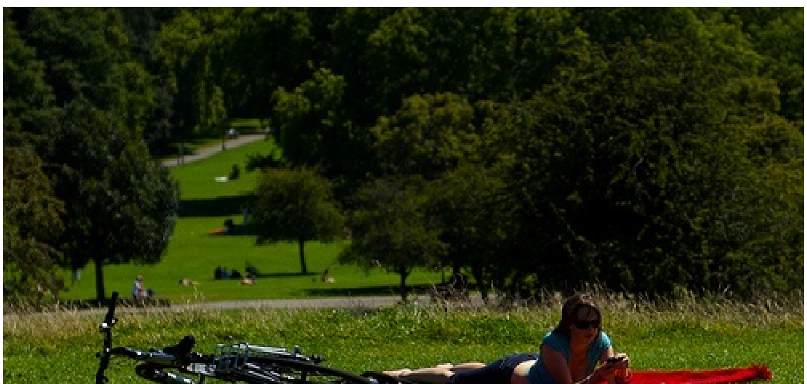
In [121]:

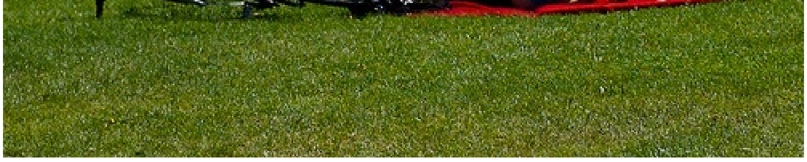
```python
# captions on the validation set
rid = np.random.randint(0, len(test_image_names))
image = test_image_names[rid]
real_caption = ' '.join([tokenizer.index_word[i] for i in test_pad_captions[rid] if i not in [0]])
result, attention_plot = evaluate(image)

print ('Real Caption:', real_caption)
print ('Prediction Caption:', ' '.join(result))

reference = []
reference.append(real_caption.split())
candidate = result

score = sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0))
print(f"BELU score: {score*100}")

plot_attention(image, result, attention_plot)
Image.open(image)
```

Real Caption: <startseq> a woman relaxes with her bike on a red blanket in a field of green grass and trees <endseq>
Prediction Caption: a green lawn as she lies on a green lawn as she lies on a green lawn as she lies on a green lawn as she lies on
BELU score: 8.132500607904442



Out[121]:





In [123]:

```
# captions on the validation set
rid = np.random.randint(0, len(test_image_names))
image = test_image_names[rid]
real_caption = ' '.join([tokenizer.index_word[i] for i in test_pad_captions[rid] if i not in [0]])
result, attention_plot = evaluate(image)

print ('Real Caption:', real_caption)
print ('Prediction Caption:', ' '.join(result))

reference = []
reference.append(real_caption.split())
candidate = result

score = sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0))
print(f"BELU score: {score*100}")

plot_attention(image, result, attention_plot)
Image.open(image)
```

Real Caption: <startseq> a man in a striped scarf walking down the sidewalk <endseq>
Prediction Caption: a black down the street <endseq>
BELU score: 13.433057891466246

Out[123]:

Beam Search

- This Search is where we take top k predictions, feed them again in the model and then sort them using the probabilities returned by the model. So, the list will always contain the top k predictions. In the end, we take the one with the highest probability and go through it till we encounter <seqend or reach the maximum caption length.

In [124]:

```
def plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))

    fig = plt.figure(figsize=(15, 18))

    for l in range(len(result.split())):
        temp_att = np.resize(attention_plot[l], (8, 8))
        ax = fig.add_subplot(len(result.split())//2, len(result.split())//2, l+1)
        ax.set_title(result.split()[l])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap= 'gray', alpha= 0.6, extent= img.get_extent())

    plt.tight_layout()
    plt.show()
```

In [131]:

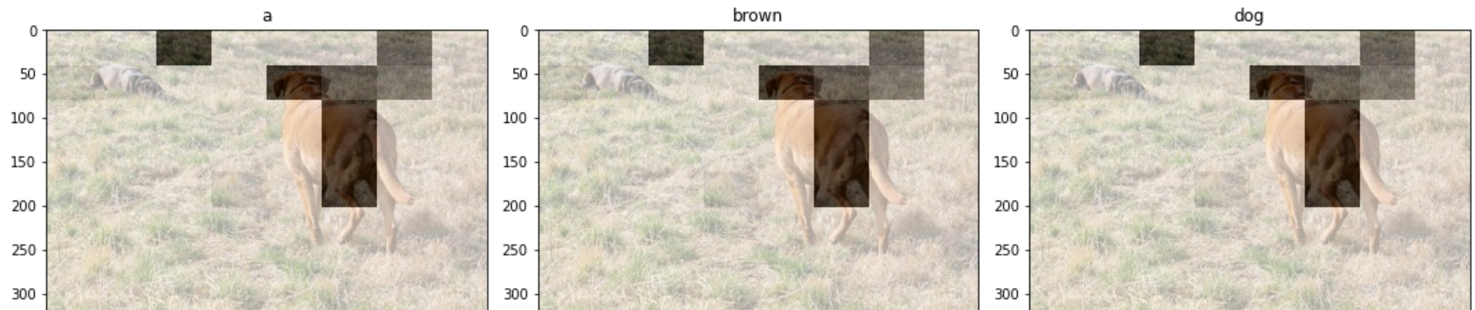
```
# https://yashk2810.github.io/Image-Captioning-using-InceptionV3-and-Beam-Search/

def beam_evaluate(image, beam_index):

  attention_plot = np.zeros((max_length, attention_features_shape))

  temp_input = tf.expand_dims(preprocess_image(image)[0], 0)
  img_tensor_val = model(temp_input)
  img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0], -1, img_tensor_val.shape[3]))

  features = encoder(img_tensor_val)

  hidden = decoder.reset_state(features)

  dec_input = tf.expand_dims([tokenizer.word_index['<startseq>']], 0)

  start = [tokenizer.word_index['<startseq>']]
  # result[0][0] = index of the starting word
  # result[0][1] = probability of the word predicted
  result = [[start, 0.0]]

  while len(result[0][0]) < max_length:
    temp = []
    for i, s in enumerate(result):

      predictions, hidden, attention_weights = decoder(dec_input, features, hidden)
      attention_plot[i] = tf.reshape(attention_weights, (-1, )).numpy()

      # Getting the top <beam_index>(n) predictions
      word_preds = np.argsort(predictions[0])[-beam_index:]
      # 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]
```

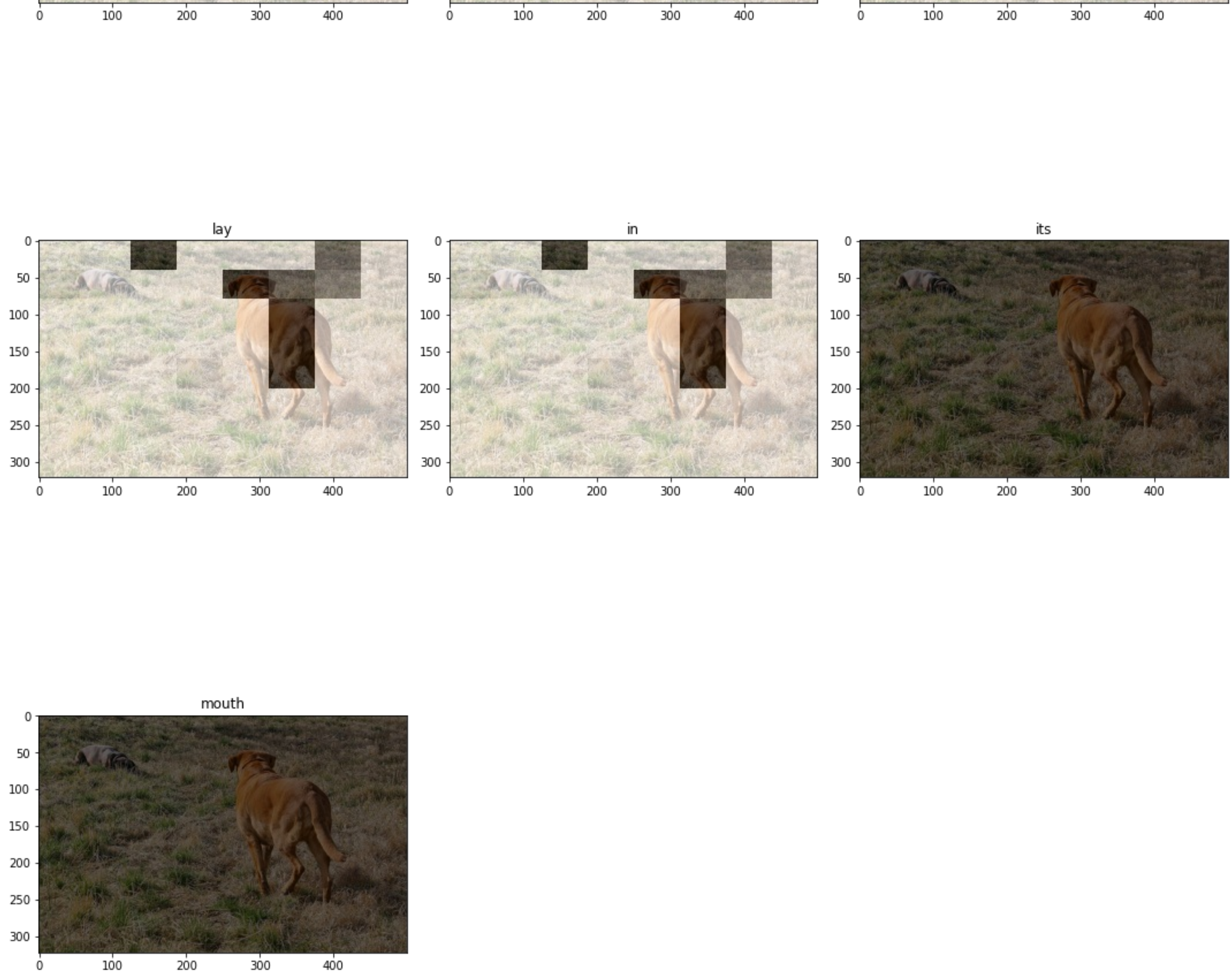
```python
        next_cap.append(w)
        prob += predictions[0][w]
        temp.append([next_cap, prob])

    result = temp
    # Sorting according to the probabilities
    result = sorted(result, reverse= False, key= lambda x: x[1])
    # Getting the top words
    result = result[-beam_index:]
    predicted_id = result[-1] # with Max Probability
    pred_list = predicted_id[0]
    prd_id = pred_list[-1]

    if (prd_id != 3):
      # Decoder input is the word predicted with highest probability among the top_k words predicted
      dec_input = tf.expand_dims(input= [prd_id],axis= 0)
    else:
      break

  result = result[-1][0]
  intermediate_caption = [tokenizer.index_word[i] for i in result]
  final_caption = []
  for i in intermediate_caption:
    if i != '<endseq>':
      final_caption.append(i)
    else:
      break

  attention_plot = attention_plot[:len(result), :]
  final_caption = ' '.join(final_caption[1:])
  return final_caption,attention_plot
```

In [140]:

```python
# Beam = 5

# captions on the Test set
from PIL import Image
from nltk.translate.bleu_score import sentence_bleu

# captions on the Test set
rid = np.random.randint(0, len(test_image_names))
image = test_image_names[rid]
print(image)

start = time.time()
real_caption = ' '.join([tokenizer.index_word[i] for i in test_pad_captions[rid] if i not in [0]])
result, attention_plot = beam_evaluate(image, 5)
print ('Real Caption:', real_caption)
print ('Prediction Caption using beam:', result)

reference = []
reference.append(real_caption.split())
candidate = result

score = sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0))
print(f"BELU score: {score*100}")

plot_attention(image, result, attention_plot)

print(f"time took to Predict: {round(time.time()-start)} sec")
# opening the image
Image.open(image)
```

```
/content/flickr30k_images/flickr30k_images/2314732154.jpg
Real Caption: <startseq> a dog walks in the grass towards a dog lying down <endseq>
Prediction Caption using beam: a brown dog lay in its mouth
BELU score: 26.726124191242434
```





time took to Predict: 4 sec

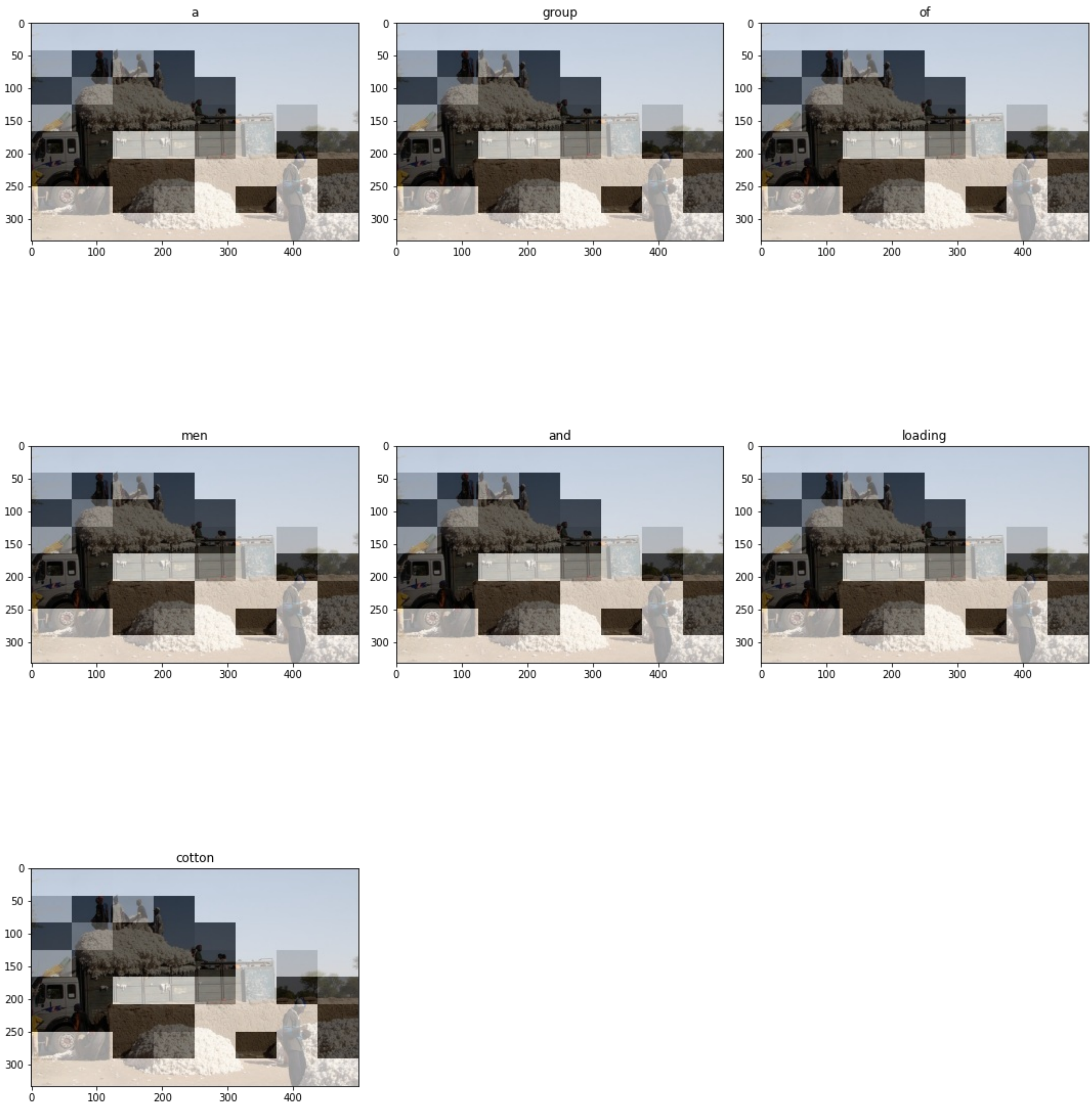
Out[140]:

In [149]:

```python
# Beam = 7

# captions on the Test set
from PIL import Image
from nltk.translate.bleu_score import sentence_bleu

# captions on the Test set
rid = np.random.randint(0, len(test_image_names))
image = test_image_names[rid]
print(image)
```

```
start = time.time()
real_caption = ' '.join([tokenizer.index_word[i] for i in test_pad_captions[rid] if i not in [0]])
result, attention_plot = beam_evaluate(image, 7)
print ('Real Caption:', real_caption)
print ('Prediction Caption using beam:', result)

reference = []
reference.append(real_caption.split())
candidate = result

score = sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0))
print(f"BELU score: {score*100}")

plot_attention(image, result, attention_plot)

print(f"time took to Predict: {round(time.time()-start)} sec")
# opening the image
Image.open(image)
```

```
/content/flickr30k_images/flickr30k_images/1018148011.jpg
Real Caption: <startseq> workers load <unk> wool onto a truck <endseq>
Prediction Caption using beam: a group of men and loading cotton
BELU score: 17.407765595569785
```




```
time took to Predict: 5 sec
```

time took to Predict: 5 sec

Out[149]:

Observations:

- The dataset used is Flicker30K and images used are 6000 and its associated 30k captions which is 5 captions per image.
- More than 30% captions are gramatically incorrect and to some extent I have cleaned them using Grammerly software.
- The CNN architecture used is Inception which has appx 24M parameters. I have tried VGG-16 which has appx 134M parameters, but the predictions were almost similar and hence I didnt used the VGG-16.
- The Decoder architecture is based on Show Attend and Tell paper but the loss is not converging less than 0.57 even after 200 epochs and hence need to stop.
- The results are not at par Bahdanau Attention Decoder.
- Played with batchsize of 32 and 64, but both giving appx same results and I have choosen 64 as it has less iterations per epoch.
- Have used Greedy search and Beam Search with 5 and 7 predictions.
- The dataset is having varieties of images and hence if more data is used for training, predictions could have been much better, but that need heavy GPU.
- Some predictions are repetitive and this behaviour is not understandable, tried to solve but coud'nt due to lack of expertise.
- Thanks to Tensorflow blog on this topic through which this woud'nt be possible.

In [ ]: