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

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

```python
import string
import numpy as np
import pandas as pd
from numpy import array
from pickle import load

from PIL import Image
import pickle
from collections import Counter
import matplotlib.pyplot as plt

import sys, time, os, warnings
warnings.filterwarnings("ignore")
import re

import keras
import tensorflow as tf
from tqdm import tqdm
from nltk.translate.bleu_score import sentence_bleu

# from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense, BatchNormalization
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import Dropout
# from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint
# from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.text import Tokenizer
from keras.applications.vgg16 import VGG16, preprocess_input

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
```

Data Loading and Preprocessing

```python
image_path = 
"/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset"
dir_Flickr_text = 
"/content/drive/MyDrive/Books/Chapter03/Chapter03/Flickr8k_text/Flickr
8k.token.txt"
```

```
jpgs = os.listdir(image_path)
print("Total Images in Dataset = {}".format(len(jpgs)))
```

```
Total Images in Dataset = 8091
```

We create a dataframe to store the image id and captions for ease of use

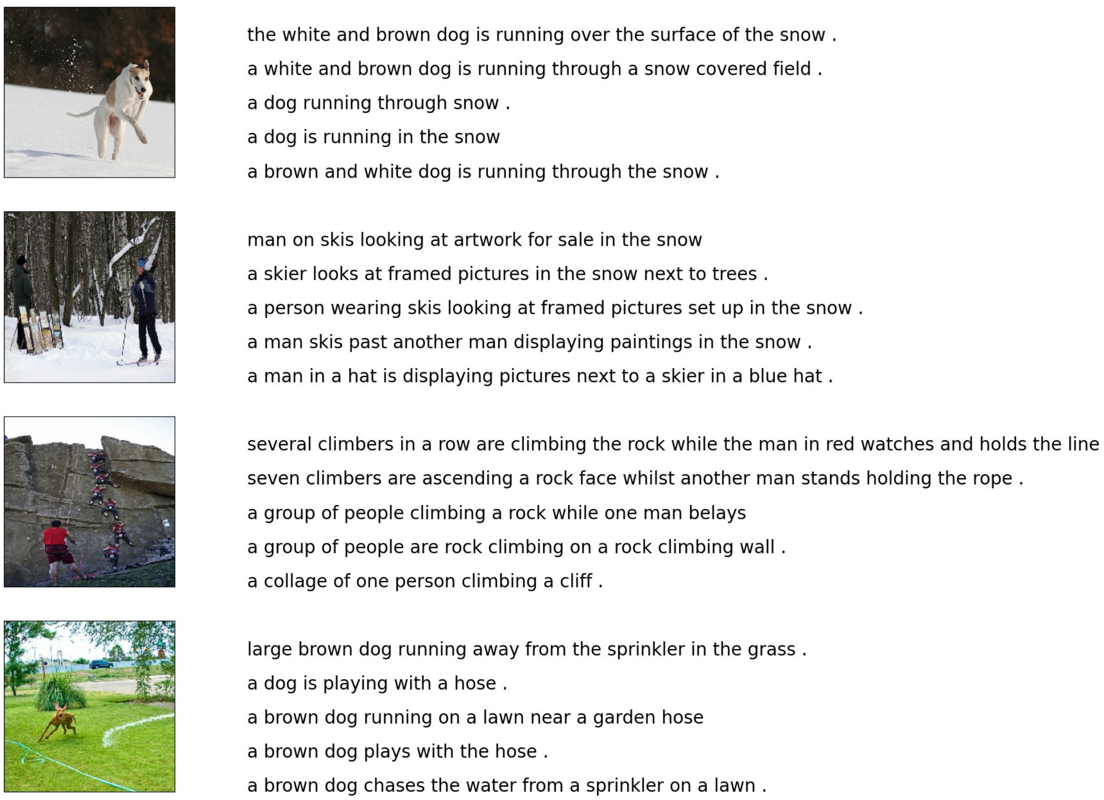
```
file = open(dir_Flickr_text,'r')
text = file.read()
file.close()

datatxt = []
for line in text.split('\n'):
   col = line.split('\t')
   if len(col) == 1:
       continue
   w = col[0].split("#")
   datatxt.append(w + [col[1].lower()])

data = pd.DataFrame(datatxt,columns=["filename","index","caption"])
data = data.reindex(columns =['index','filename','caption'])
data = data[data.filename != '2258277193_586949ec62.jpg.1']
uni_filenames = np.unique(data.filename.values)

data.head()
```

```
  index                   filename  \
0     0  1000268201_693b08cb0e.jpg
1     1  1000268201_693b08cb0e.jpg
2     2  1000268201_693b08cb0e.jpg
3     3  1000268201_693b08cb0e.jpg
4     4  1000268201_693b08cb0e.jpg

                                             caption
0  a child in a pink dress is climbing up a set o...
1              a girl going into a wooden building .
2   a little girl climbing into a wooden playhouse .
3  a little girl climbing the stairs to her playh...
4  a little girl in a pink dress going into a woo...
```

Visualizing few images with there corresponding captions

```
npic = 5
npix = 224
target_size = (npix,npix,3)
count = 1

fig = plt.figure(figsize=(10,20))
for jpgfnm in uni_filenames[10:14]:
   filename = image_path + '/' + jpgfnm
   captions =
list(data["caption"].loc[data["filename"]==jpgfnm].values)
```

```
    image_load = tf.keras.utils.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)
    count += 1
plt.show()
```




```
## Vocabulary Size
vocabulary = []
for txt in data.caption.values:
    vocabulary.extend(txt.split())
print('Vocabulary Size: %d' % len(set(vocabulary)))
```

```
Vocabulary Size: 8918
```

Text preprocessing steps --

- Removing Punctuation

- Removing single characters
- Removing Numeric values

```
def remove_punctuation(text_original):
   text_no_punctuation = text_original.translate(string.punctuation)
   return(text_no_punctuation)

def remove_single_character(text):
   text_len_more_than1 = ""
   for word in text.split():
       if len(word) > 1:
           text_len_more_than1 += " " + word
   return(text_len_more_than1)

def remove_numeric(text):
   text_no_numeric = ""
   for word in text.split():
       isalpha = word.isalpha()
       if isalpha:
           text_no_numeric += " " + word
   return(text_no_numeric)

def text_clean(text_original):
   text = remove_punctuation(text_original)
   text = remove_single_character(text)
   text = remove_numeric(text)
   return(text)

for i, caption in enumerate(data.caption.values):
   newcaption = text_clean(caption)
   data["caption"].iloc[i] = newcaption

## Vocabulary Size after cleaning

clean_vocabulary = []
for txt in data.caption.values:
   clean_vocabulary.extend(txt.split())
print('Clean Vocabulary Size: %d' % len(set(clean_vocabulary)))

Clean Vocabulary Size: 8357
```

Adding and tags for every caption, so that model understands the start and end of each caption

```
PATH = 
"/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/"
all_captions = []
for caption  in data["caption"].astype(str):
   caption = '<start> ' + caption+ ' <end>'
   all_captions.append(caption)
```

```
all_captions[:10]
```

```
['<start>  child in pink dress is climbing up set of stairs in an
entry way <end>',
 '<start>  girl going into wooden building <end>',
 '<start>  little girl climbing into wooden playhouse <end>',
 '<start>  little girl climbing the stairs to her playhouse <end>',
 '<start>  little girl in pink dress going into wooden cabin <end>',
 '<start>  black dog and spotted dog are fighting <end>',
 '<start>  black dog and dog playing with each other on the road
<end>',
 '<start>  black dog and white dog with brown spots are staring at
each other in the street <end>',
 '<start>  two dogs of different breeds looking at each other on the
road <end>',
 '<start>  two dogs on pavement moving toward each other <end>']
```

```
all_img_name_vector = []
for annot in data["filename"]:
   full_image_path = PATH + annot
   all_img_name_vector.append(full_image_path)

all_img_name_vector[:10]
```

```
['/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1001773457_577c3a7d70.jpg',
 '/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1001773457_577c3a7d70.jpg',
 '/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1001773457_577c3a7d70.jpg',
 '/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1001773457_577c3a7d70.jpg',
 '/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1001773457_577c3a7d70.jpg']
```

```
print(f"len(all_img_name_vector) : {len(all_img_name_vector)}")
print(f"len(all_captions) : {len(all_captions)}")
```

```
len(all_img_name_vector) : 40455
len(all_captions) : 40455
```

Limiting our captions and images to 40000 ao that we can use batch_size = 64 and we have 625 batches in total

```
def data_limiter(num,total_captions,all_img_name_vector):
 train_captions, img_name_vector =
shuffle(total_captions,all_img_name_vector,random_state=1)
 train_captions = train_captions[:num]
 img_name_vector = img_name_vector[:num]
 return train_captions,img_name_vector

train_captions,img_name_vector =
data_limiter(40000,all_captions,all_img_name_vector)
print(f"len(train_captions) : {len(train_captions)}")
print(f"len(img_name_vector) : {len(img_name_vector)}")
```

```
len(train_captions) : 40000
len(img_name_vector) : 40000
```

Model Defintion --

We will be using VGG16 as image feature extraction model. We will be extracting an image vector for our images. Hence we will remove the softmax layer from the model

```
def load_image(image_path):
   img = tf.io.read_file(image_path)
   img = tf.image.decode_jpeg(img, channels=3)
   img = tf.image.resize(img, (224, 224))
   img = preprocess_input(img)
   return img, image_path

image_model = tf.keras.applications.VGG16(include_top=False,
weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output
image_features_extract_model = tf.keras.Model(new_input, hidden_layer)

image_features_extract_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 3s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792
```

```
 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856

 block2_conv2 (Conv2D)       (None, None, None, 128)   147584

 block2_pool (MaxPooling2D)  (None, None, None, 128)   0

 block3_conv1 (Conv2D)       (None, None, None, 256)   295168

 block3_conv2 (Conv2D)       (None, None, None, 256)   590080

 block3_conv3 (Conv2D)       (None, None, None, 256)   590080

 block3_pool (MaxPooling2D)  (None, None, None, 256)   0

 block4_conv1 (Conv2D)       (None, None, None, 512)   1180160

 block4_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block4_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block4_pool (MaxPooling2D)  (None, None, None, 512)   0

 block5_conv1 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)       (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)       (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)  (None, None, None, 512)   0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Mapping each image name to the function to load an image

```
encode_train = sorted(set(img_name_vector))
image_dataset = tf.data.Dataset.from_tensor_slices(encode_train)
image_dataset = image_dataset.map(load_image,
num_parallel_calls=tf.data.experimental.AUTOTUNE).batch(64)

%%time
for img, path in tqdm(image_dataset):
 batch_features = image_features_extract_model(img)
 batch_features = tf.reshape(batch_features,
```

```
                             (batch_features.shape[0], -1,
batch_features.shape[3]))

 for bf, p in zip(batch_features, path):
   path_of_feature = p.numpy().decode("utf-8")
   np.save(path_of_feature, bf.numpy())
```

```
100%|██████████| 127/127 [09:08<00:00,  4.32s/it]

CPU times: user 1min 44s, sys: 5.73 s, total: 1min 50s
Wall time: 9min 8s
```

```
top_k = 5000
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=top_k,
                                                  oov_token="<unk>",
                                                  filters='!"#$
%&()*+.,-/:;=?@[\]^_`{|}~ ')

tokenizer.fit_on_texts(train_captions)
train_seqs = tokenizer.texts_to_sequences(train_captions)
tokenizer.word_index['<pad>'] = 0
tokenizer.index_word[0] = '<pad>'

train_seqs = tokenizer.texts_to_sequences(train_captions)
cap_vector = tf.keras.preprocessing.sequence.pad_sequences(train_seqs,
padding='post')
```

```
train_captions[:3]
```

```
['<start>  several children leaping into pile of leaves on the ground
<end>',
 '<start>  man hiking in the wilderness giving the camera thumbs up
<end>',
 '<start>  white dog is running through the water onto the shore
<end>']
```

```
train_seqs[:3]
```

```
[[2, 184, 62, 331, 64, 524, 12, 329, 6, 5, 167, 3],
 [2, 11, 588, 4, 5, 2384, 895, 5, 93, 1281, 53, 3],
 [2, 14, 9, 7, 32, 33, 5, 24, 238, 5, 280, 3]]
```

```
def calc_max_length(tensor):
   return max(len(t) for t in tensor)
max_length = calc_max_length(train_seqs)

def calc_min_length(tensor):
   return min(len(t) for t in tensor)
min_length = calc_min_length(train_seqs)
```

```
print('Max Length of any caption : Min Length of any caption = '+
str(max_length) +" : "+str(min_length))
```

```
Max Length of any caption : Min Length of any caption = 33 : 2
```

```
img_name_train, img_name_val, cap_train, cap_val =
train_test_split(img_name_vector,cap_vector, test_size=0.2,
random_state=0)
```

```
BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
vocab_size = len(tokenizer.word_index) + 1
num_steps = len(img_name_train) // BATCH_SIZE
features_shape = 512
attention_features_shape = 49


def map_func(img_name, cap):
 img_tensor = np.load(img_name.decode('utf-8')+'.npy')
 return img_tensor, cap
dataset = tf.data.Dataset.from_tensor_slices((img_name_train,
cap_train))

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

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

Using the model Architecture from "A Neural Image Caption Generator"

Encoder - Decoder architecture with Attention

```
class VGG16_Encoder(tf.keras.Model):
   # This encoder passes the features through a Fully connected layer
   def __init__(self, embedding_dim):
       super(VGG16_Encoder, self).__init__()
       # shape after fc == (batch_size, 49, embedding_dim)
       self.fc = tf.keras.layers.Dense(embedding_dim)
       self.dropout = tf.keras.layers.Dropout(0.5, noise_shape=None,
seed=None)

   def call(self, x):
       #x= self.dropout(x)
       x = self.fc(x)
```

```
        x = tf.nn.relu(x)
        return x

def rnn_type(units):
   if tf.test.is_gpu_available():
       return tf.compat.v1.keras.layers.CuDNNLSTM(units,
                                                return_sequences=True,
                                                return_state=True,

recurrent_initializer='glorot_uniform')
   else:
       return tf.keras.layers.GRU(units,
                                 return_sequences=True,
                                 return_state=True,
                                 recurrent_activation='sigmoid',

recurrent_initializer='glorot_uniform')

'''The encoder output(i.e. 'features'), hidden state(initialized to 0)
(i.e. 'hidden') and
the decoder input (which is the start token)(i.e. 'x') is passed to
the decoder.'''

class Rnn_Local_Decoder(tf.keras.Model):
 def __init__(self, embedding_dim, units, vocab_size):
   super(Rnn_Local_Decoder, self).__init__()
   self.units = units
   self.embedding = tf.keras.layers.Embedding(vocab_size,
embedding_dim)
   self.gru = tf.keras.layers.GRU(self.units,
                                 return_sequences=True,
                                 return_state=True,

recurrent_initializer='glorot_uniform')

   self.fc1 = tf.keras.layers.Dense(self.units)

   self.dropout = tf.keras.layers.Dropout(0.5, noise_shape=None,
seed=None)
   self.batchnormalization = tf.keras.layers.BatchNormalization(axis=-
1, momentum=0.99, epsilon=0.001, center=True, scale=True,
beta_initializer='zeros', gamma_initializer='ones',
moving_mean_initializer='zeros', moving_variance_initializer='ones',
beta_regularizer=None, gamma_regularizer=None, beta_constraint=None,
gamma_constraint=None)

   self.fc2 = tf.keras.layers.Dense(vocab_size)

   # Implementing Attention Mechanism
   self.Uattn = tf.keras.layers.Dense(units)
```

```
    self.Wattn = tf.keras.layers.Dense(units)
    self.Vattn = tf.keras.layers.Dense(1)

  def call(self, x, features, hidden):
    # features shape ==> (64,49,256) ==> Output from ENCODER
    # hidden shape == (batch_size, hidden_size) ==>(64,512)
    # hidden_with_time_axis shape == (batch_size, 1, hidden_size) ==>
(64,1,512)

    hidden_with_time_axis = tf.expand_dims(hidden, 1)

    # score shape == (64, 49, 1)
    # Attention Function
    '''e(ij) = f(s(t-1),h(j))'''
    ''' e(ij) = Vattn(T)*tanh(Uattn * h(j) + Wattn * s(t))'''

    score = self.Vattn(tf.nn.tanh(self.Uattn(features) +
self.Wattn(hidden_with_time_axis)))

    # self.Uattn(features) : (64,49,512)
    # self.Wattn(hidden_with_time_axis) : (64,1,512)
    # tf.nn.tanh(self.Uattn(features) +
self.Wattn(hidden_with_time_axis)) : (64,49,512)
    # self.Vattn(tf.nn.tanh(self.Uattn(features) +
self.Wattn(hidden_with_time_axis))) : (64,49,1) ==> score

    # you get 1 at the last axis because you are applying score to
self.Vattn
    # Then find Probability using Softmax
    '''attention_weights(alpha(ij)) = softmax(e(ij))'''

    attention_weights = tf.nn.softmax(score, axis=1)

    # attention_weights shape == (64, 49, 1)
    # Give weights to the different pixels in the image
    ''' C(t) = Summation(j=1 to T) (attention_weights * VGG-16
features) '''

    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)

    # Context Vector(64,256) = AttentionWeights(64,49,1) *
features(64,49,256)
    # context_vector shape after sum == (64, 256)
    # x shape after passing through embedding == (64, 1, 256)

    x = self.embedding(x)
    # x shape after concatenation == (64, 1,  512)
```

```
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
    # passing the concatenated vector to the GRU

    output, state = self.gru(x)
    # shape == (batch_size, max_length, hidden_size)

    x = self.fc1(output)
    # x shape == (batch_size * max_length, hidden_size)

    x = tf.reshape(x, (-1, x.shape[2]))

    # Adding Dropout and BatchNorm Layers
    x= self.dropout(x)
    x= self.batchnormalization(x)

    # output shape == (64 * 512)
    x = self.fc2(x)

    # shape : (64 * 8329(vocab))
    return x, state, attention_weights

  def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.units))


encoder = VGG16_Encoder(embedding_dim)
decoder = Rnn_Local_Decoder(embedding_dim, units, vocab_size)

optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)
  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_mean(loss_)

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 = decoder.reset_state(batch_size=target.shape[0])
```

```
 dec_input = tf.expand_dims([tokenizer.word_index['<start>']] *
BATCH_SIZE, 1)

 with tf.GradientTape() as tape:
     features = encoder(img_tensor)
     for i in range(1, target.shape[1]):
         # passing the features through the decoder
         predictions, hidden, _ = decoder(dec_input, features, hidden)
         loss += loss_function(target[:, i], predictions)

         # using teacher forcing
         dec_input = tf.expand_dims(target[:, i], 1)

 total_loss = (loss / int(target.shape[1]))
 trainable_variables = encoder.trainable_variables +
decoder.trainable_variables
 gradients = tape.gradient(loss, trainable_variables)
 optimizer.apply_gradients(zip(gradients, trainable_variables))

 return loss, total_loss

start_epoch = 0
EPOCHS = 20
for epoch in range(start_epoch, EPOCHS):
   start = time.time()
   total_loss = 0

   for (batch, (img_tensor, target)) in enumerate(dataset):
       batch_loss, t_loss = train_step(img_tensor, target)
       total_loss += t_loss

       if batch % 100 == 0:
           print ('Epoch {} Batch {} Loss {:.4f}'.format(
             epoch + 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)

   print ('Epoch {} Loss {:.6f}'.format(epoch + 1,
                                        total_loss/num_steps))

   print ('Time taken for 1 epoch {} sec\n'.format(time.time() -
start))
```

```
Epoch 1 Batch 0 Loss 2.6517
Epoch 1 Batch 100 Loss 1.7509
Epoch 1 Batch 200 Loss 1.4415
Epoch 1 Batch 300 Loss 1.4333
Epoch 1 Batch 400 Loss 1.2631
Epoch 1 Loss 1.472358
```

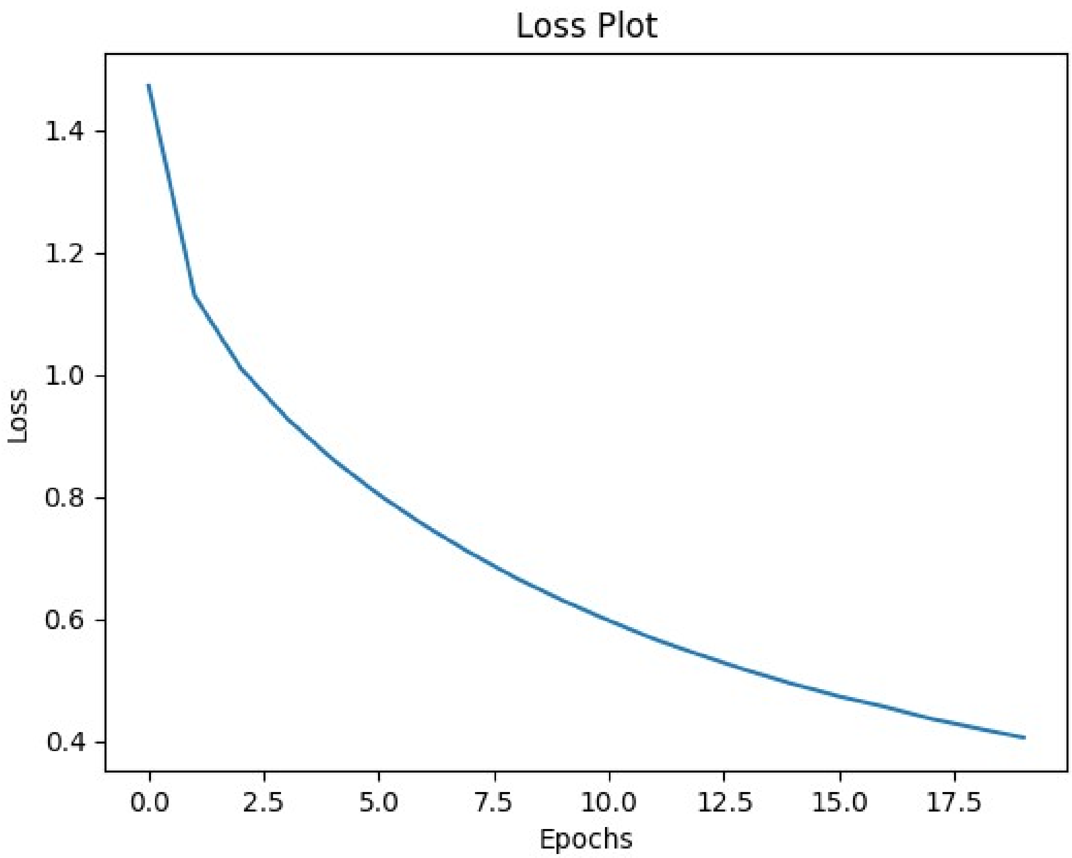
```
Time taken for 1 epoch 135.4782428741455 sec

Epoch 2 Batch 0 Loss 1.2459
Epoch 2 Batch 100 Loss 1.0828
Epoch 2 Batch 200 Loss 1.2530
Epoch 2 Batch 300 Loss 1.1329
Epoch 2 Batch 400 Loss 1.1330
Epoch 2 Loss 1.128897
Time taken for 1 epoch 80.45122170448303 sec

Epoch 3 Batch 0 Loss 1.0524
Epoch 3 Batch 100 Loss 1.0775
Epoch 3 Batch 200 Loss 1.1177
Epoch 3 Batch 300 Loss 0.9564
Epoch 3 Batch 400 Loss 0.9722
Epoch 3 Loss 1.010716
Time taken for 1 epoch 81.90771460533142 sec

Epoch 4 Batch 0 Loss 0.9866
Epoch 4 Batch 100 Loss 0.9027
Epoch 4 Batch 200 Loss 0.9322
Epoch 4 Batch 300 Loss 0.8906
Epoch 4 Batch 400 Loss 0.9899
Epoch 4 Loss 0.928541
Time taken for 1 epoch 76.51114130020142 sec

Epoch 5 Batch 0 Loss 0.9089
Epoch 5 Batch 100 Loss 0.9130
Epoch 5 Batch 200 Loss 0.8803
Epoch 5 Batch 300 Loss 0.9650
Epoch 5 Batch 400 Loss 0.8170
Epoch 5 Loss 0.861021
Time taken for 1 epoch 74.95623564720154 sec

Epoch 6 Batch 0 Loss 0.7617
Epoch 6 Batch 100 Loss 0.8217
Epoch 6 Batch 200 Loss 0.7510
Epoch 6 Batch 300 Loss 0.8455
Epoch 6 Batch 400 Loss 0.7426
Epoch 6 Loss 0.803324
Time taken for 1 epoch 74.30901718139648 sec

Epoch 7 Batch 0 Loss 0.7387
Epoch 7 Batch 100 Loss 0.7192
Epoch 7 Batch 200 Loss 0.7309
Epoch 7 Batch 300 Loss 0.7866
Epoch 7 Batch 400 Loss 0.7319
Epoch 7 Loss 0.752589
Time taken for 1 epoch 74.56031823158264 sec
```

```
Epoch 8 Batch 0 Loss 0.7518
Epoch 8 Batch 100 Loss 0.6564
Epoch 8 Batch 200 Loss 0.8358
Epoch 8 Batch 300 Loss 0.6750
Epoch 8 Batch 400 Loss 0.6522
Epoch 8 Loss 0.707357
Time taken for 1 epoch 73.89059281349182 sec

Epoch 9 Batch 0 Loss 0.6498
Epoch 9 Batch 100 Loss 0.6591
Epoch 9 Batch 200 Loss 0.5916
Epoch 9 Batch 300 Loss 0.7880
Epoch 9 Batch 400 Loss 0.6365
Epoch 9 Loss 0.666132
Time taken for 1 epoch 73.77722930908203 sec

Epoch 10 Batch 0 Loss 0.6274
Epoch 10 Batch 100 Loss 0.6020
Epoch 10 Batch 200 Loss 0.6983
Epoch 10 Batch 300 Loss 0.6305
Epoch 10 Batch 400 Loss 0.6190
Epoch 10 Loss 0.629869
Time taken for 1 epoch 73.04188752174377 sec

Epoch 11 Batch 0 Loss 0.6250
Epoch 11 Batch 100 Loss 0.5613
Epoch 11 Batch 200 Loss 0.5972
Epoch 11 Batch 300 Loss 0.5343
Epoch 11 Batch 400 Loss 0.5197
Epoch 11 Loss 0.597114
Time taken for 1 epoch 72.80356454849243 sec

Epoch 12 Batch 0 Loss 0.5930
Epoch 12 Batch 100 Loss 0.6338
Epoch 12 Batch 200 Loss 0.5587
Epoch 12 Batch 300 Loss 0.5422
Epoch 12 Batch 400 Loss 0.4771
Epoch 12 Loss 0.566411
Time taken for 1 epoch 72.67205381393433 sec

Epoch 13 Batch 0 Loss 0.5923
Epoch 13 Batch 100 Loss 0.5831
Epoch 13 Batch 200 Loss 0.5269
Epoch 13 Batch 300 Loss 0.5595
Epoch 13 Batch 400 Loss 0.5981
Epoch 13 Loss 0.540405
Time taken for 1 epoch 73.17373156547546 sec
```

```
Epoch 14 Batch 0 Loss 0.5614
Epoch 14 Batch 100 Loss 0.5265
Epoch 14 Batch 200 Loss 0.5558
Epoch 14 Batch 300 Loss 0.5089
Epoch 14 Batch 400 Loss 0.4719
Epoch 14 Loss 0.515674
Time taken for 1 epoch 73.01869130134583 sec

Epoch 15 Batch 0 Loss 0.4882
Epoch 15 Batch 100 Loss 0.5132
Epoch 15 Batch 200 Loss 0.5191
Epoch 15 Batch 300 Loss 0.5013
Epoch 15 Batch 400 Loss 0.4183
Epoch 15 Loss 0.493401
Time taken for 1 epoch 72.72160053253174 sec

Epoch 16 Batch 0 Loss 0.4639
Epoch 16 Batch 100 Loss 0.4776
Epoch 16 Batch 200 Loss 0.5020
Epoch 16 Batch 300 Loss 0.4943
Epoch 16 Batch 400 Loss 0.4573
Epoch 16 Loss 0.473175
Time taken for 1 epoch 72.37288761138916 sec

Epoch 17 Batch 0 Loss 0.4887
Epoch 17 Batch 100 Loss 0.4695
Epoch 17 Batch 200 Loss 0.4721
Epoch 17 Batch 300 Loss 0.4520
Epoch 17 Batch 400 Loss 0.4992
Epoch 17 Loss 0.455807
Time taken for 1 epoch 73.47014212608337 sec

Epoch 18 Batch 0 Loss 0.4917
Epoch 18 Batch 100 Loss 0.4628
Epoch 18 Batch 200 Loss 0.4204
Epoch 18 Batch 300 Loss 0.4431
Epoch 18 Batch 400 Loss 0.3945
Epoch 18 Loss 0.436327
Time taken for 1 epoch 72.20186638832092 sec

Epoch 19 Batch 0 Loss 0.4113
Epoch 19 Batch 100 Loss 0.4419
Epoch 19 Batch 200 Loss 0.4115
Epoch 19 Batch 300 Loss 0.4188
Epoch 19 Batch 400 Loss 0.3708
Epoch 19 Loss 0.420880
Time taken for 1 epoch 75.37601017951965 sec

Epoch 20 Batch 0 Loss 0.4210
Epoch 20 Batch 100 Loss 0.4890
```

```
Epoch 20 Batch 200 Loss 0.3833
Epoch 20 Batch 300 Loss 0.4825
Epoch 20 Batch 400 Loss 0.3961
Epoch 20 Loss 0.405734
Time taken for 1 epoch 73.49871945381165 sec
```

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

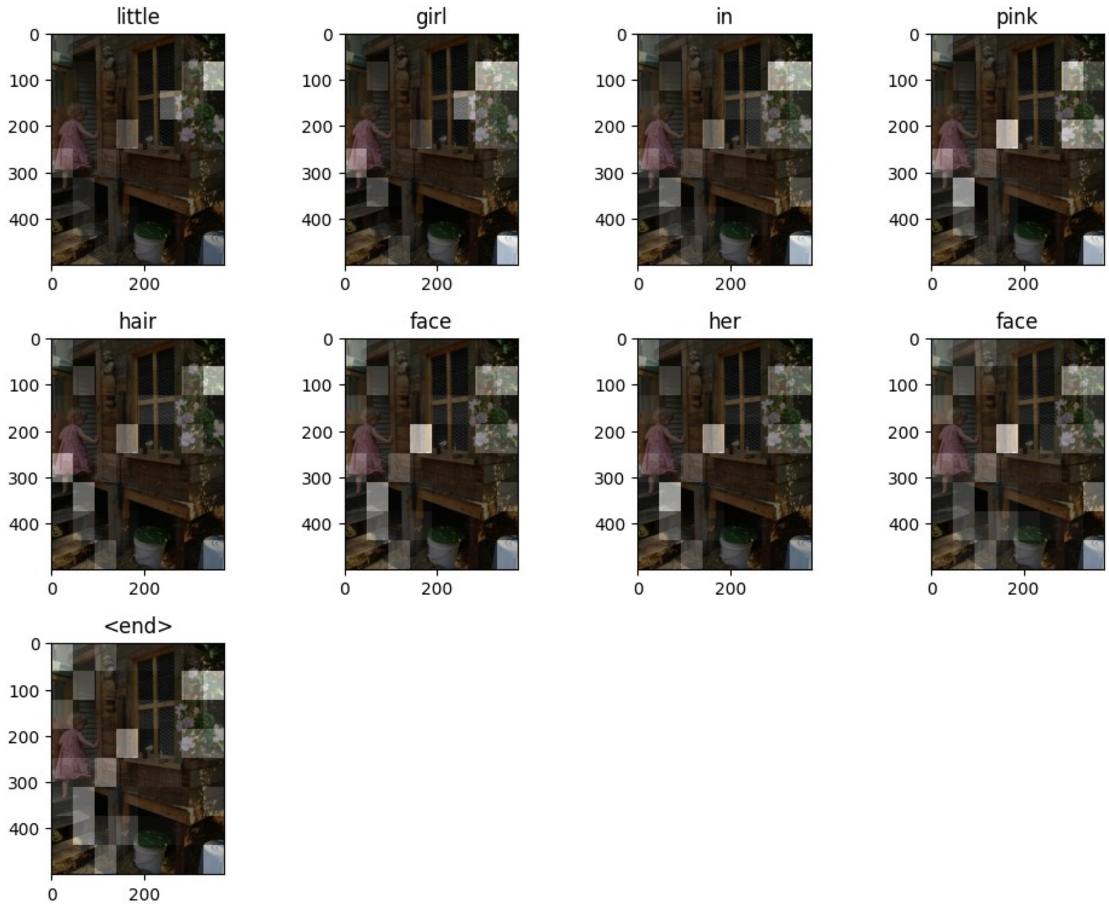



```python
def evaluate(image):
   attention_plot = np.zeros((max_length, attention_features_shape))

   hidden = decoder.reset_state(batch_size=1)
   temp_input = tf.expand_dims(load_image(image)[0], 0)
   img_tensor_val = image_features_extract_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)
   dec_input = tf.expand_dims([tokenizer.word_index['<start>']], 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()
        result.append(tokenizer.index_word[predicted_id])

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

        dec_input = tf.expand_dims([predicted_id], 0)
    attention_plot = attention_plot[:len(result), :]

    return result, attention_plot

def plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))
    fig = plt.figure(figsize=(10, 10))
    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()

# captions on the validation set
rid = np.random.randint(0, len(img_name_val))
image =
'/content/drive/MyDrive/Books/Chapter03/Chapter03/Flicker8k_Dataset/
1000268201_693b08cb0e.jpg'

# real_caption = ' '.join([tokenizer.index_word[i] for i in
cap_val[rid] if i not in [0]])
result, attention_plot = evaluate(image)

# remove <start> and <end> from the real_caption
real_caption = 'Two white dogs are playing in the snow'
first = real_caption.split(' ', 1)[1]


#remove "<unk>" in result
for i in result:
```

```
    if i=="<unk>":
        result.remove(i)

for i in real_caption:
    if i=="<unk>":
        real_caption.remove(i)

#remove <end> from result
result_join = ' '.join(result)
result_final = result_join.rsplit(' ', 1)[0]

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

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

print ('Real Caption:', real_caption)
print ('Prediction Caption:', result_final)
plot_attention(image, result, attention_plot)
```

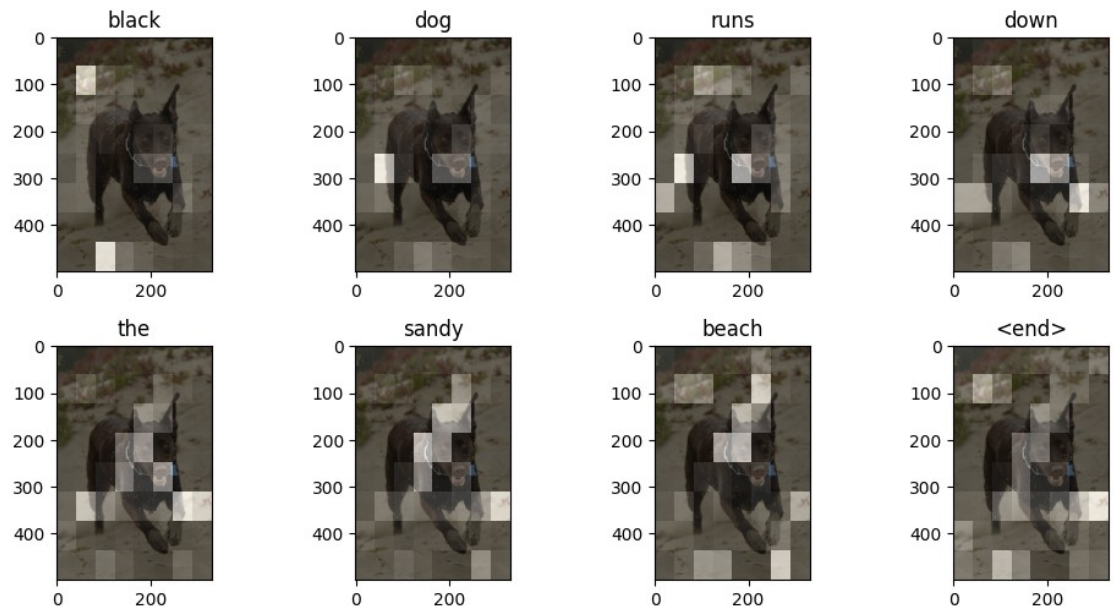
```
BELU score: 1.0518351895246306e-229
Real Caption: Two white dogs are playing in the snow
Prediction Caption: little girl in pink hair face her face
```




```
rid = np.random.randint(0, len(img_name_val))
image = img_name_val[rid]
start = time.time()
real_caption = ' '.join([tokenizer.index_word[i] for i in cap_val[rid]
if i not in [0]])
result, attention_plot = evaluate(image)

first = real_caption.split(' ', 1)[1]
real_caption = first.rsplit(' ', 1)[0]

#remove "<unk>" in result
for i in result:
   if i=="<unk>":
       result.remove(i)

#remove <end> from result
result_join = ' '.join(result)
result_final = result_join.rsplit(' ', 1)[0]

real_appn = []
real_appn.append(real_caption.split())
reference = real_appn
```

```python
candidate = result_final

print ('Real Caption:', real_caption)
print ('Prediction Caption:', result_final)

plot_attention(image, result, attention_plot)
print(f"time took to Predict: {round(time.time()-start)} sec")

Image.open(img_name_val[rid])
```

```
Real Caption: brown dog runs in the sand
Prediction Caption: black dog runs down the sandy beach
```

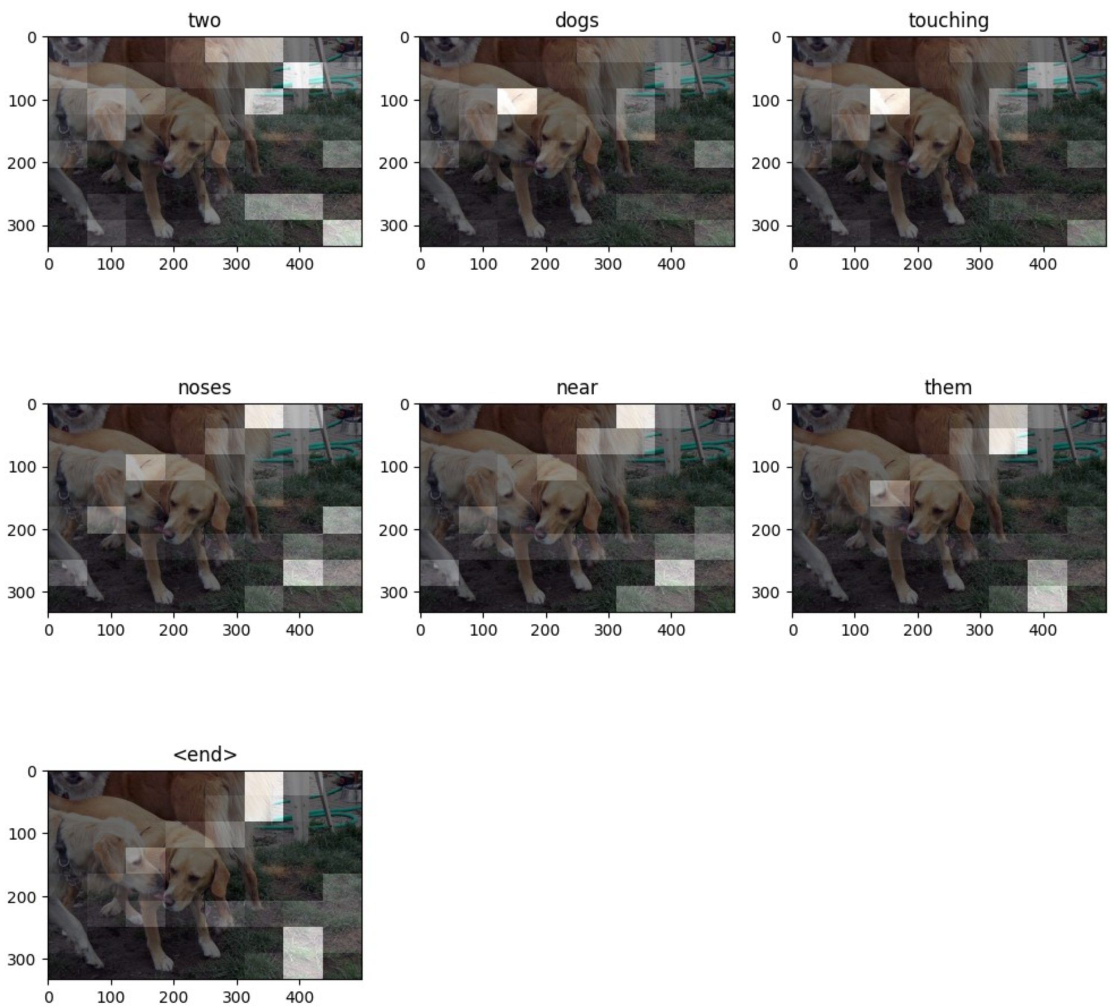



```
time took to Predict: 1 sec
```

```
rid = np.random.randint(0, len(img_name_val))
image = img_name_val[rid]

real_caption = ' '.join([tokenizer.index_word[i] for i in cap_val[rid]
if i not in [0]])
result, attention_plot = evaluate(image)

# remove <start> and <end> from the real_caption
first = real_caption.split(' ', 1)[1]
real_caption = first.rsplit(' ', 1)[0]
```

```
#remove "<unk>" in result
for i in result:
   if i=="<unk>":
       result.remove(i)

for i in real_caption:
   if i=="<unk>":
       real_caption.remove(i)

#remove <end> from result
result_join = ' '.join(result)
result_final = result_join.rsplit(' ', 1)[0]

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

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

print ('Real Caption:', real_caption)
print ('Prediction Caption:', result_final)

plot_attention(image, result, attention_plot)
```

```
BELU score: 1.1200407237786664e-229
Real Caption: three brown dogs on the patchy grass
Prediction Caption: two dogs touching noses near them
```


two
dogs
touching
noses
near
them
<end>
0
100
200
300
400