In [1]:
```
from google.colab import drive
drive.mount("/content/drive/")
```

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

In [2]:
```
!wget --header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.
0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
90.0.4430.212 Safari/537.36" --header="Accept: text/html,application/xhtml+xm
l,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.9" --header="Accept-Language: en-US,en;q=0.9" --hea
der="Referer: https://www.kaggle.com/" "https://storage.googleapis.com/kaggle-
data-sets/623289/1111676/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&
X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F202
10526%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20210526T170031Z&X-Goog-Exp
ires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=356683d4035c5ce3fb1aa29
ad63a9696248263723c3c5365249aa55c7f88a91841c92b9a571d67d97f437227d4b7747dc257d
ab9f42f6f1a8ac7c30933d839bf4d7dab62999a62549f529910e4e1963ff3eb24007268349e07c
584d6a44f8696c55a2f0f403ea01cbaa6c912b73648feee1880d4a5c13ee41d376070fd569c0dd
a2b99bd5115ef08e71e9854a0d8334f6d14769c267af22bc83e05fd5549e176e871d02ef6dbe9f
5c9a467c2fa305adbf17bc12a1697b33a7ae43dd50aa2173547c783c666be2a9d8db62fefee633
84413372590c86c99ba9ef5027394a18c559ca776b48a8fd9b2177ae4d47fd2db67450b69d2bc8
10857cbe71990820ff2f8" -c -O 'archive.zip'
```

```
--2021-05-28 17:41:48--  https://storage.googleapis.com/kaggle-data-sets/6232
89/1111676/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Creden
tial=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20210526%2Faut
o%2Fstorage%2Fgoog4_request&X-Goog-Date=20210526T170031Z&X-Goog-Expires=25919
9&X-Goog-SignedHeaders=host&X-Goog-Signature=356683d4035c5ce3fb1aa29ad63a9696
248263723c3c5365249aa55c7f88a91841c92b9a571d67d97f437227d4b7747dc257dab9f42f6
f1a8ac7c30933d839bf4d7dab62999a62549f529910e4e1963ff3eb24007268349e07c584d6a4
4f8696c55a2f0f403ea01cbaa6c912b73648feee1880d4a5c13ee41d376070fd569c0dda2b99b
d5115ef08e71e9854a0d8334f6d14769c267af22bc83e05fd5549e176e871d02ef6dbe9f5c9a4
67c2fa305adbf17bc12a1697b33a7ae43dd50aa2173547c783c666be2a9d8db62fefee6338441
3372590c86c99ba9ef5027394a18c559ca776b48a8fd9b2177ae4d47fd2db67450b69d2bc8108
57cbe71990820ff2f8
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.13.80, 1
72.217.13.240, 142.251.33.208, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.13.80|:
443... connected.
HTTP request sent, awaiting response... 416 Requested range not satisfiable

    The file is already fully retrieved; nothing to do.
```

In [3]:
```
#Extracting the
from zipfile import ZipFile
with ZipFile("archive.zip","r") as z:
  z.extractall()
  print("done")
```

```
done
```

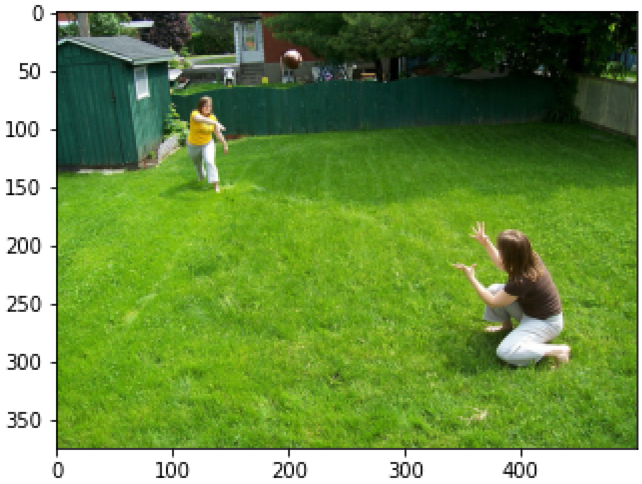
Processing The Image

In [4]:

```
import matplotlib.pyplot as plt
import cv2
import os
k=os.listdir("Images")
image_list=k[:2000]
image_name=k[:2000]
for i in range(len(image_list)):
  image_list[i]="Images/"+image_list[i]

img=cv2.imread("Images/534200447_b0f3ff02be.jpg")
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x7f8f175853d0>
```



In [ ]:

```
test_images=k[2000:2100]
print(test_images)
for i in range(len(test_images)):
  test_images[i]="Images/"+test_images[i]
```

In [43]:

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

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (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
_________________________________________________________________
```

In [ ]:

```
image_model.save("/content/drive/My Drive/CASE Studies/Image captioning/Attention/image_model.h5")
```

In [7]:

```
#for that I m writing a function which takes image and coverts it to vector

def img_2_vec(x):
  img=cv2.imread(x)
  img=cv2.resize(img, (224,224))
  img=img.reshape(1,224,224,3)
  img=image_model.predict(img)
  img=img.reshape(49,512)
  return img

img_vector=[]
for i in image_list:
  img_vector.append(img_2_vec(i))
```

In [8]:

```
len(img_vector)
```

Out[8]:

```
2000
```

In [9]:

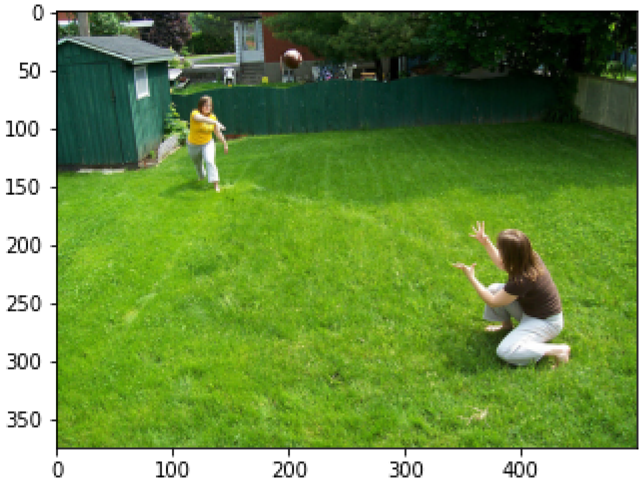
```
#saving image_name and its vector in dictionary
dict_vector={}
for i in range(len(image_list)):
    dict_vector[image_list[i].split("/")[1]]=img_vector[i]

len(dict_vector)
```

Out[9]:

```
2000
```

In [10]:
```
# image and its feature vector
img=cv2.imread("Images/534200447_b0f3ff02be.jpg")
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
print("vector=",dict_vector[image_list[5].split("/")[1]])
```

```
vector= [[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         3.6347606 0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 7.5758348 0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```



Processing Captions

In [11]:
```
# reading the Captions file
with open("captions.txt",'r') as file:
  text=file.readlines()
```

In [12]:

```
# creating dictionary  which contains image name and its captions
dict_captions={}
for i in text:
  name=i.split(",")[0]
  caption=i.split(",")[1]
  if name not in dict_captions:
    dict_captions[name]=[caption]
  else:
    dict_captions[name].append(caption)
dict_captions["534200447_b0f3ff02be.jpg"]

cap={}
for i in text:
  name=i.split(",")[0]
  caption=i.split(",")[1]
  if name not in cap:
    cap[name]=[caption]
  else:
    cap[name].append(caption)
```

In [13]:

```
dict_captions=dict([(key ,dict_captions[key]) for key in image_name])
print(len(dict_captions))
#cap=dict([(key ,cap[key]) for key in image_name])
```

```
2000
```

In [14]:

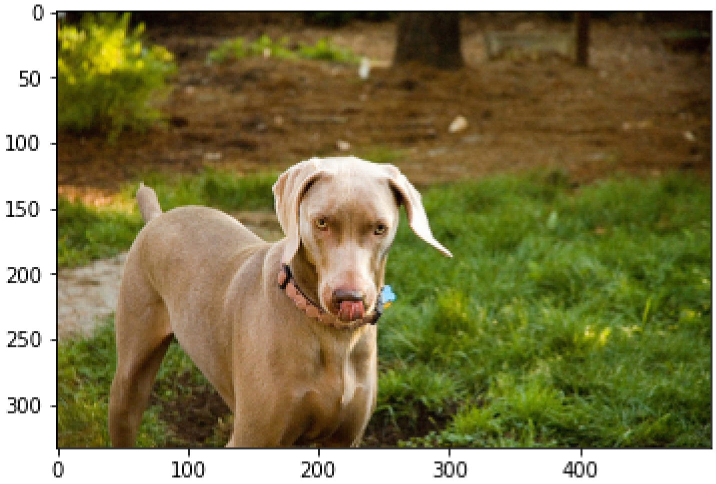
```
#Preprocessing the Captions
import re

def preprocess(d):
  for k,v in d.items():
    for i in range(len(v)):
      v[i]=re.sub('[^A-Za-z]+',' ',v[i])
      v[i]=v[i].lower()
      v[i]=v[i].strip()

      v[i]="start "+v[i]+" end"
  return d
dict_captions=preprocess(dict_captions)
cap=preprocess(cap)
dict_captions['2975018306_0e8da316f5.jpg']
```

Out[14]:

```
['start a brown dog is licking its nose end',
 'start a brown dog is on the green grass end',
 'start a brown dog licking his nose in the middle of some grass end',
 'start a brown dog licks his nose while standing in green grass by the woods
end',
 'start a brown dog with his tongue out looks at the camera end']
```

In [15]:
```
#Plotting the image and its captions
import cv2
import matplotlib.pyplot as plt
img=cv2.imread('Images/2975018306_0e8da316f5.jpg')
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
dict_captions['2975018306_0e8da316f5.jpg']
```

Out[15]:
```
['start a brown dog is licking its nose end',
 'start a brown dog is on the green grass end',
 'start a brown dog licking his nose in the middle of some grass end',
 'start a brown dog licks his nose while standing in green grass by the woods
end',
 'start a brown dog with his tongue out looks at the camera end']
```



In [16]:

```
#Creating Vocabulary for the words and giving  them the index/integer value
vocab=set()
for k,v in dict_captions.items():
  for i in range(len(v)):
    for j in v[i].split():
      vocab.add(j)
vocab=sorted(vocab)
vocabulary={}
for i,j in enumerate(vocab,1):
  vocabulary[j]=i

print("The Unique Number of words are ",len(vocab))


#trasforming these index to captions
for k,v in dict_captions.items():
  for i in v:
    t=[]
    for j in i.split():
      t.append(vocabulary[j])

    dict_captions[k][v.index(i)]=t

for i in dict_captions['2975018306_0e8da316f5.jpg']:
  print(i)
```

```
The Unique Number of words are  4329
[3595, 1, 461, 1061, 1887, 2090, 1895, 2404, 1208]
[3595, 1, 461, 1061, 1887, 2458, 3842, 1596, 1586, 1208]
[3595, 1, 461, 1061, 2090, 1764, 2404, 1847, 3842, 2253, 2440, 3471, 1586, 12
08]
[3595, 1, 461, 1061, 2091, 1764, 2404, 4229, 3585, 1847, 1596, 1586, 523, 384
2, 4283, 1208]
[3595, 1, 461, 1061, 4270, 1764, 3914, 2498, 2146, 145, 3842, 538, 1208]
```

In [17]:

```
len(vocab)
```

Out[17]:

```
4329
```

In [18]:

```
#finding the maxlenght of captions from train captions
max_len=[]
for k,v in dict_captions.items():
  for i in v:
    max_len.append(len(i))
max_length=max(max_len)
print("The maximum lenght of caption is",max(max_len))
```

```
The maximum lenght of caption is 35
```

In [19]:

```
#Creating the dataset
from keras.preprocessing.sequence import pad_sequences
import numpy as np
X = []
y_in = []
for k, v in dict_captions.items():
  for j in v:
    X.append(dict_vector[k])
    in_seq = pad_sequences([j], maxlen=max_length, padding='post', truncating=
'post')[0]
    y_in.append(in_seq)
X=np.array(X)
y_in=np.array(y_in)
print(X.shape,y_in.shape)
```

```
(10000, 49, 512) (10000, 35)
```

In [20]:

```
#splitting the according to images names
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,y_in,test_size=0.2)
```

In [21]:

```
def create_dataset(img_name_train,caption_train):
  dataset = tf.data.Dataset.from_tensor_slices((img_name_train, caption_train
))
  dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).prefetch(buffer_siz
e=tf.data.experimental.AUTOTUNE)
  return dataset
```

In [22]:

```
import tensorflow as tf
BATCH_SIZE = 64
BUFFER_SIZE = 1000
train_dataset = create_dataset(x_train,y_train)
test_dataset =  create_dataset(x_test,y_test)
```

Modelling

In [23]:

```
BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
vocab_size = len(vocabulary) + 1
num_steps = len(x_train) // BATCH_SIZE
features_shape = 512
attention_features_shape = 49
```

CNN_ENCODER

In [24]:

```
class VGG16_Encoder(tf.keras.Model):
  def __init__(self, embedding_dim):
    super(VGG16_Encoder, self).__init__()
    self.fc = tf.keras.layers.Dense(embedding_dim)
    self.dropout = tf.keras.layers.Dropout(0.5)
  def call(self, x):
    x= self.dropout(x)
    x = self.fc(x)
    x = tf.nn.relu(x)
    return x
```

BahdanauAttention

In [25]:

```
class BahdanauAttention(tf.keras.Model):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, features, hidden):
    # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)

    # hidden shape == (batch_size, hidden_size)
    # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
    hidden_with_time_axis = tf.expand_dims(hidden, 1)

    # attention_hidden_layer shape == (batch_size, 64, units)
    attention_hidden_layer = (tf.nn.tanh(self.W1(features) + self.W2(hidden_wi
th_time_axis)))

    # score shape == (batch_size, 64, 1)
    # This gives you an unnormalized score for each image feature.
    score = self.V(attention_hidden_layer)

    # attention_weights shape == (batch_size, 64, 1)
    attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size, hidden_size)
    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
```

Decoder

In [26]:

```
class RNN_Decoder(tf.keras.Model):
  def __init__(self, embedding_dim, units, vocab_size):
    super(RNN_Decoder, self).__init__()
    self.units = units
    self.vocab_size=vocab_size
    self.embedding_dim=embedding_dim

    self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.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.fc2 = tf.keras.layers.Dense(self.vocab_size)

    self.attention = BahdanauAttention(self.units)

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

    # x shape after passing through embedding == (64, 1, 256)

    x = self.embedding(x)
    # x shape after concatenation == (64, 1, 256 + 256)
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state = self.gru(x)
    x = self.fc1(output)
    x = tf.reshape(x, (-1, x.shape[2]))
    x = self.fc2(x)

    return x, state, attention_weights

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

In [27]:

```
encoder = VGG16_Encoder(embedding_dim)
#vocab_size=4462
decoder = RNN_Decoder(embedding_dim, units, vocab_size)
```

In [28]:

```
#custom loss function
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_)
```

In [29]:

```
checkpoint_path = "/content/drive/My Drive/CASE Studies/Image captioning/Atten
tion/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5
)

start_epoch = 0
if ckpt_manager.latest_checkpoint:
  start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
  # restoring the latest checkpoint in checkpoint_path
  ckpt.restore(ckpt_manager.latest_checkpoint)
```

In [30]:

```
# Training
loss_plot = []

@tf.function
def train_step(img_tensor, target):
  loss = 0
  hidden = decoder.reset_state(batch_size=target.shape[0])

  dec_input = tf.expand_dims([vocabulary['start']] * target.shape[0], 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
```

In [31]:

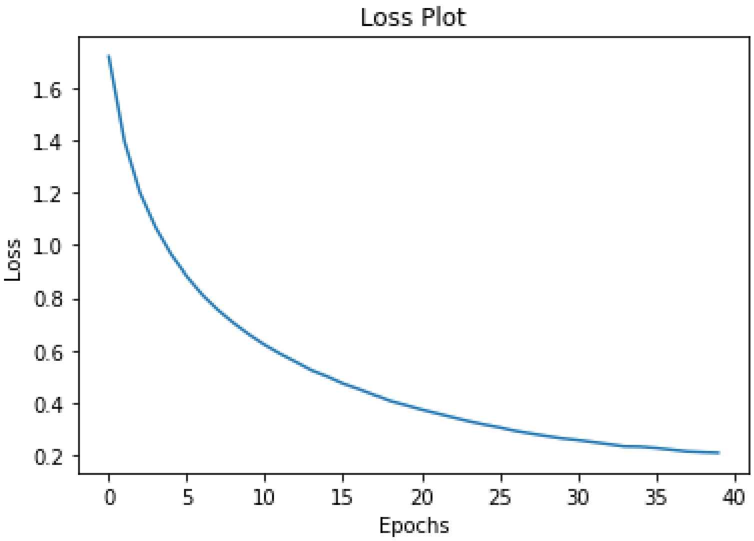
```
import time
EPOCHS = 40
start_epoch=0
for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

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

        if batch % 100 == 0:
            average_batch_loss = batch_loss.numpy()/int(target.shape[1])
            print(f'Epoch {epoch+1} Batch {batch} Loss {average_batch_loss:.4f
}')
    # storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)
    if epoch % 5 == 0:
      ckpt_manager.save()
    print(f'Epoch {epoch+1} Loss {total_loss/num_steps:.6f}')
    print(f'Time taken for 1 epoch {time.time()-start:.2f} sec\n')
```

```
Epoch 1 Batch 0 Loss 2.6503
Epoch 1 Batch 100 Loss 1.5088
Epoch 1 Loss 1.719162
Time taken for 1 epoch 52.17 sec

Epoch 2 Batch 0 Loss 1.5823
Epoch 2 Batch 100 Loss 1.2052
Epoch 2 Loss 1.395903
Time taken for 1 epoch 15.32 sec

Epoch 3 Batch 0 Loss 1.3097
Epoch 3 Batch 100 Loss 1.0028
Epoch 3 Loss 1.199016
Time taken for 1 epoch 14.80 sec

Epoch 4 Batch 0 Loss 1.0878
Epoch 4 Batch 100 Loss 1.0665
Epoch 4 Loss 1.066809
Time taken for 1 epoch 15.24 sec

Epoch 5 Batch 0 Loss 0.9904
Epoch 5 Batch 100 Loss 0.9153
Epoch 5 Loss 0.963886
Time taken for 1 epoch 15.16 sec

Epoch 6 Batch 0 Loss 0.9911
Epoch 6 Batch 100 Loss 0.7892
Epoch 6 Loss 0.881250
Time taken for 1 epoch 15.61 sec

Epoch 7 Batch 0 Loss 0.8170
Epoch 7 Batch 100 Loss 0.7873
Epoch 7 Loss 0.810918
Time taken for 1 epoch 15.35 sec

Epoch 8 Batch 0 Loss 0.7405
Epoch 8 Batch 100 Loss 0.7721
Epoch 8 Loss 0.753207
Time taken for 1 epoch 16.06 sec

Epoch 9 Batch 0 Loss 0.7681
Epoch 9 Batch 100 Loss 0.7041
Epoch 9 Loss 0.703781
Time taken for 1 epoch 15.88 sec

Epoch 10 Batch 0 Loss 0.7247
Epoch 10 Batch 100 Loss 0.5876
Epoch 10 Loss 0.659476
Time taken for 1 epoch 15.40 sec

Epoch 11 Batch 0 Loss 0.5910
Epoch 11 Batch 100 Loss 0.6382
Epoch 11 Loss 0.619931
Time taken for 1 epoch 15.58 sec

Epoch 12 Batch 0 Loss 0.6836
Epoch 12 Batch 100 Loss 0.5133
Epoch 12 Loss 0.585789
Time taken for 1 epoch 15.38 sec

Epoch 13 Batch 0 Loss 0.5868
Epoch 13 Batch 100 Loss 0.4773
Epoch 13 Loss 0.554913
Time taken for 1 epoch 15.14 sec

Epoch 14 Batch 0 Loss 0.5253
Epoch 14 Batch 100 Loss 0.5028
Epoch 14 Loss 0.523101
Time taken for 1 epoch 15.08 sec

Epoch 15 Batch 0 Loss 0.5910
Epoch 15 Batch 100 Loss 0.4918
Epoch 15 Loss 0.499763
Time taken for 1 epoch 15.19 sec

Epoch 16 Batch 0 Loss 0.4266
Epoch 16 Batch 100 Loss 0.5046
Epoch 16 Loss 0.472944
Time taken for 1 epoch 15.56 sec

Epoch 17 Batch 0 Loss 0.4615
Epoch 17 Batch 100 Loss 0.4043
Epoch 17 Loss 0.452070
Time taken for 1 epoch 15.28 sec

Epoch 18 Batch 0 Loss 0.4170
Epoch 18 Batch 100 Loss 0.4546
Epoch 18 Loss 0.429387
Time taken for 1 epoch 15.17 sec

Epoch 19 Batch 0 Loss 0.4680
Epoch 19 Batch 100 Loss 0.4194
Epoch 19 Loss 0.406293
Time taken for 1 epoch 15.20 sec

Epoch 20 Batch 0 Loss 0.3931
Epoch 20 Batch 100 Loss 0.3819
Epoch 20 Loss 0.390194
Time taken for 1 epoch 15.13 sec

Epoch 21 Batch 0 Loss 0.4456
Epoch 21 Batch 100 Loss 0.3311
Epoch 21 Loss 0.373407
Time taken for 1 epoch 15.48 sec

Epoch 22 Batch 0 Loss 0.4193
Epoch 22 Batch 100 Loss 0.3330
Epoch 22 Loss 0.358847
Time taken for 1 epoch 15.24 sec

Epoch 23 Batch 0 Loss 0.3829
Epoch 23 Batch 100 Loss 0.3026
Epoch 23 Loss 0.343873
Time taken for 1 epoch 15.23 sec
```

```
Epoch 24 Batch 0 Loss 0.3739
Epoch 24 Batch 100 Loss 0.2983
Epoch 24 Loss 0.329319
Time taken for 1 epoch 15.29 sec

Epoch 25 Batch 0 Loss 0.3714
Epoch 25 Batch 100 Loss 0.2705
Epoch 25 Loss 0.316592
Time taken for 1 epoch 15.12 sec

Epoch 26 Batch 0 Loss 0.3879
Epoch 26 Batch 100 Loss 0.2930
Epoch 26 Loss 0.305256
Time taken for 1 epoch 15.58 sec

Epoch 27 Batch 0 Loss 0.3268
Epoch 27 Batch 100 Loss 0.3047
Epoch 27 Loss 0.292208
Time taken for 1 epoch 15.31 sec

Epoch 28 Batch 0 Loss 0.3481
Epoch 28 Batch 100 Loss 0.2762
Epoch 28 Loss 0.282092
Time taken for 1 epoch 15.32 sec

Epoch 29 Batch 0 Loss 0.3637
Epoch 29 Batch 100 Loss 0.2506
Epoch 29 Loss 0.272510
Time taken for 1 epoch 15.30 sec

Epoch 30 Batch 0 Loss 0.3453
Epoch 30 Batch 100 Loss 0.2494
Epoch 30 Loss 0.263543
Time taken for 1 epoch 15.22 sec

Epoch 31 Batch 0 Loss 0.3078
Epoch 31 Batch 100 Loss 0.2433
Epoch 31 Loss 0.257108
Time taken for 1 epoch 15.49 sec

Epoch 32 Batch 0 Loss 0.3115
Epoch 32 Batch 100 Loss 0.2413
Epoch 32 Loss 0.249230
Time taken for 1 epoch 15.16 sec

Epoch 33 Batch 0 Loss 0.3121
Epoch 33 Batch 100 Loss 0.2498
Epoch 33 Loss 0.241339
Time taken for 1 epoch 15.12 sec

Epoch 34 Batch 0 Loss 0.3202
Epoch 34 Batch 100 Loss 0.1985
Epoch 34 Loss 0.233051
Time taken for 1 epoch 15.12 sec

Epoch 35 Batch 0 Loss 0.2781
Epoch 35 Batch 100 Loss 0.2365
Epoch 35 Loss 0.231847
Time taken for 1 epoch 15.23 sec

Epoch 36 Batch 0 Loss 0.2663
Epoch 36 Batch 100 Loss 0.1989
Epoch 36 Loss 0.226987
Time taken for 1 epoch 15.69 sec

Epoch 37 Batch 0 Loss 0.2968
Epoch 37 Batch 100 Loss 0.2329
Epoch 37 Loss 0.220267
Time taken for 1 epoch 15.28 sec

Epoch 38 Batch 0 Loss 0.2508
Epoch 38 Batch 100 Loss 0.2147
Epoch 38 Loss 0.213822
Time taken for 1 epoch 15.26 sec

Epoch 39 Batch 0 Loss 0.2514
Epoch 39 Batch 100 Loss 0.2263
Epoch 39 Loss 0.210921
Time taken for 1 epoch 15.13 sec

Epoch 40 Batch 0 Loss 0.2926
Epoch 40 Batch 100 Loss 0.2202
Epoch 40 Loss 0.208643
Time taken for 1 epoch 15.12 sec
```

In [32]:

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



In [33]:

```python
# creating a dictionary whic is inverse of vocabulary
invert_vocab={v:k for k ,v in vocabulary.items()}
```

In [34]:

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

    hidden = decoder.reset_state(batch_size=1)

    img = cv2.imread(image)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (224,224))
    img = np.reshape(img, (1,224,224,3))


    img_tensor_val =image_model.predict(img)#.reshape(1,-1,512)
    #print(img_tensor_val.shape)
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0],-1,im
g_tensor_val.shape[3]))


    features = encoder(img_tensor_val)

    dec_input = tf.expand_dims([vocabulary['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.random.categorical(predictions, 1)[0][0].numpy()
        result.append(invert_vocab[predicted_id])

        if invert_vocab[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
```

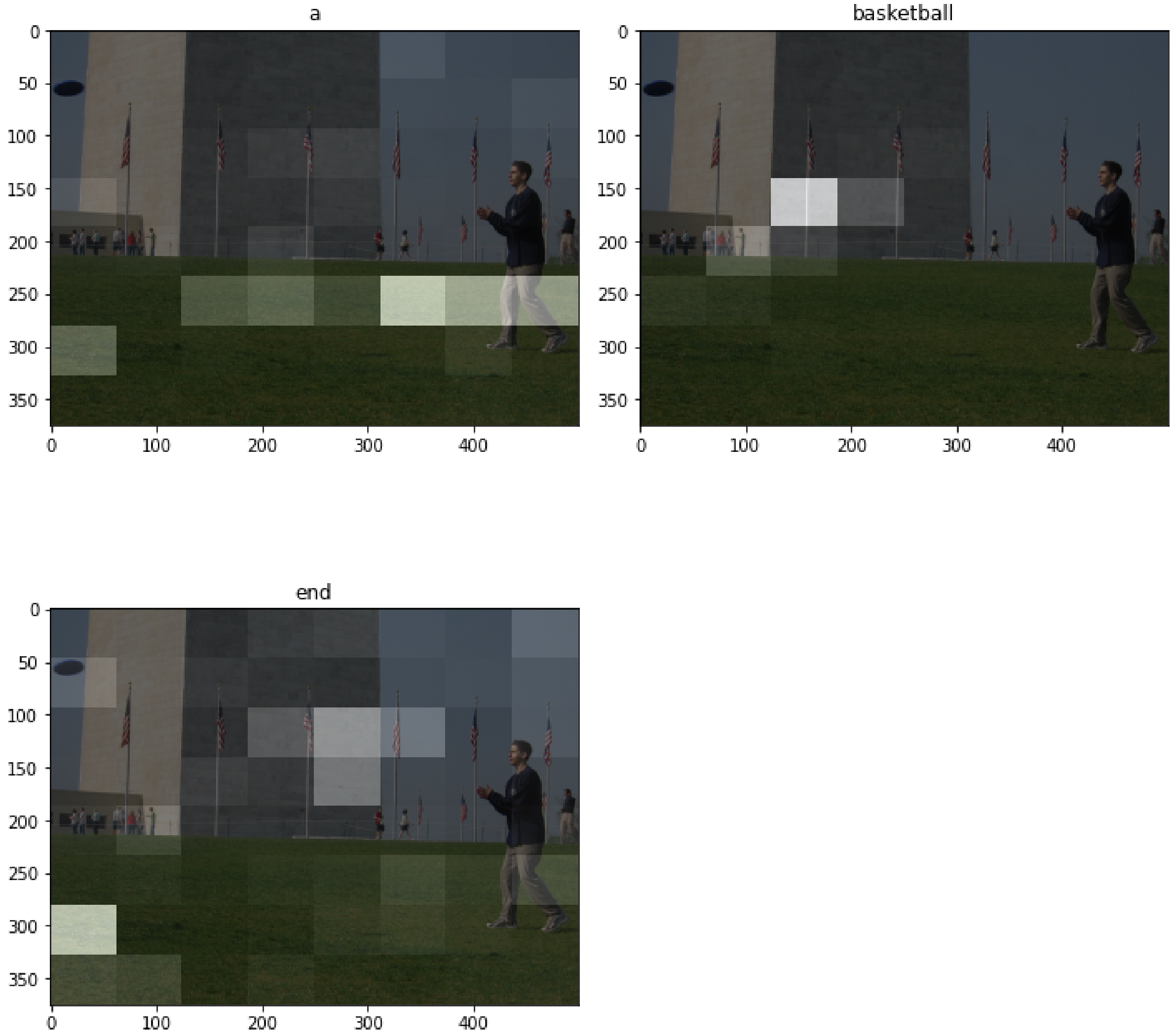
In [35]:

```
from PIL import Image
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 i in range(len_result):
        temp_att = np.resize(attention_plot[i], (8, 8))
        grid_size = max(np.ceil(len_result/2), 2)
        ax = fig.add_subplot(grid_size, grid_size, i+1)
        ax.set_title(result[i])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_extent())

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

In [42]:

```
import nltk.translate.bleu_score as bleu
import warnings
warnings.filterwarnings('ignore')

img=test_images[9]

r,a=evaluate(img)
print("\nGenerated Captions are: ",(" ").join([x for x in r]))
print("\nOriginal captions are :")
key=img.split("/")[1]
score=[]
for i in cap[key]:
  print("\t\t",i)
  reference=i
  translation = (" ").join([x for x in r])
  BLEU_score= bleu.sentence_bleu(reference, translation)
  score.append(BLEU_score)

print("\nThe Best  Blue Score is ",max(score))
print(plot_attention(img, r, a))
```

```
Generated Captions are:  a basketball end

Original captions are :
                 start a man in a field backed by american flags end
                 start a man is being thrown a frisbee in front of a monument
and flags end
                 start a man is clapping at the base of the washington memori
al end
                 start a man stands in front of the gateway arch end
                 start a man tries to catch a football on grass surrounded by
american flags end

The Best  Blue Score is  0.8891397050194614
```

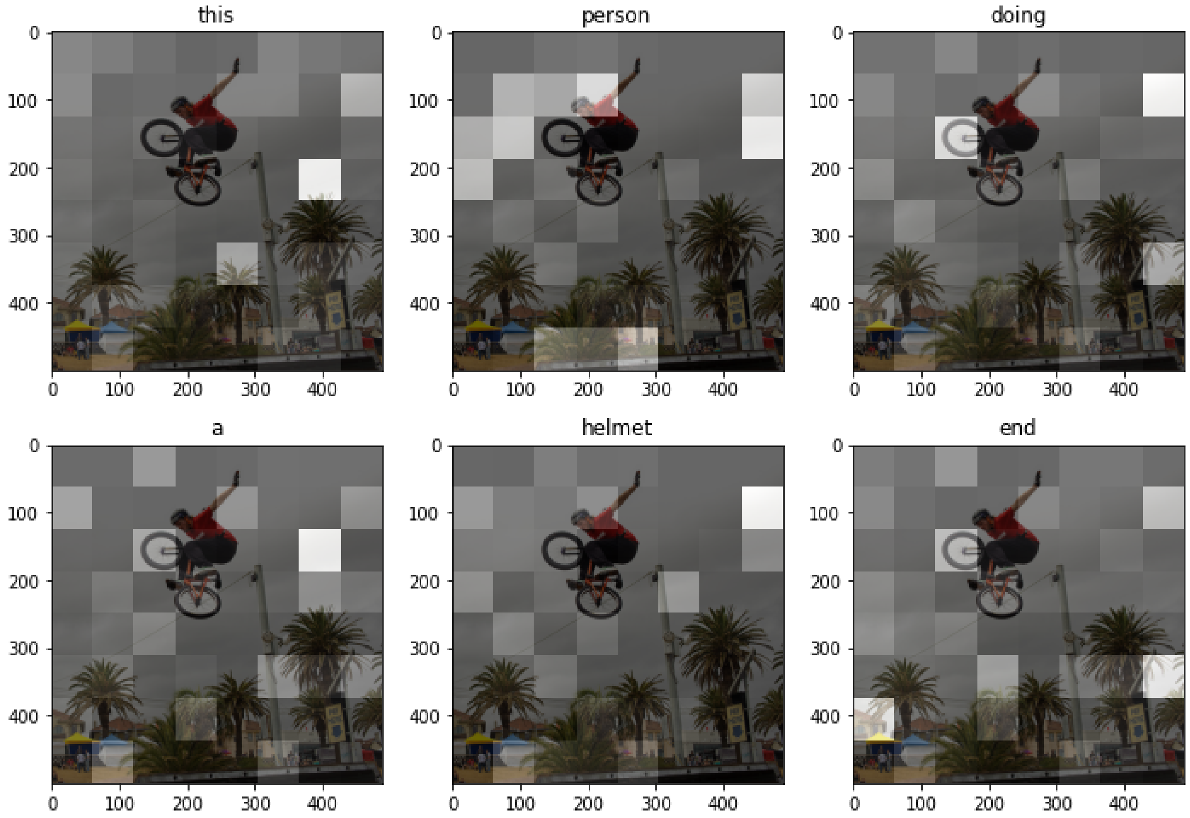


```
None
```

In [37]:

```
import nltk.translate.bleu_score as bleu
import warnings
warnings.filterwarnings('ignore')

img=test_images[0]

r,a=evaluate(img)
print("\nGenerated Captions are: ",(" ").join([x for x in r]))
print("\nOriginal captions are :")
key=img.split("/")[1]
score=[]
for i in cap[key]:
  print("\t\t",i)
  reference=i
  translation = (" ").join([x for x in r])
  BLEU_score= bleu.sentence_bleu(reference, translation)
  score.append(BLEU_score)

print("\nThe Best  Blue Score is ",max(score))
print(plot_attention(img, r, a))
```

```
Generated Captions are:  this person doing a helmet end

Original captions are :
                 start a bicyclist is attempting a trick on a wire in a tropi
cal location end
                 start a biker lets go of the handlebars while flying over th
e ground end
                 start an extreme cyclist in red shirt and helmet jumping his
bike over a guidewire end
                 start a person on a bike leaps high in the air at a beach en
d
                 start person in orange shirt balancing bicycle on wire end

The Best  Blue Score is  0.8408964152537145
```

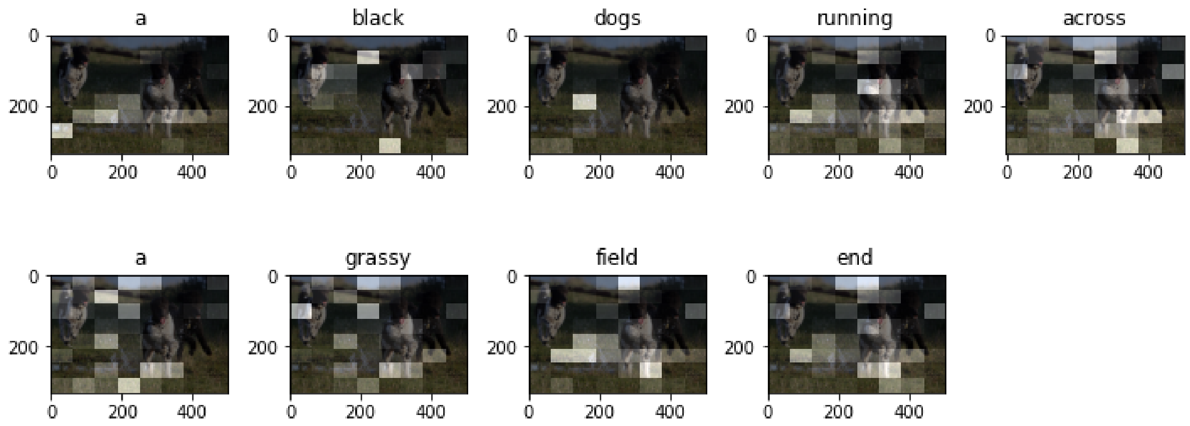


```
None
```

In [38]:

```
import nltk.translate.bleu_score as bleu
import warnings
warnings.filterwarnings('ignore')

img=test_images[40]

r,a=evaluate(img)
print("\nGenerated Captions are: ",(" ").join([x for x in r]))
print("\nOriginal captions are :")
key=img.split("/")[1]
score=[]
for i in cap[key]:
  print("\t\t",i)
  reference=i
  translation = (" ").join([x for x in r])
  BLEU_score= bleu.sentence_bleu(reference, translation)
  score.append(BLEU_score)

print("\nThe Best  Blue Score is ",max(score))
print(plot_attention(img, r, a))
```

```
Generated Captions are:  a black dogs running across a grassy field end

Original captions are :
                 start the dogs are running through the water end
                 start three black and white dogs running through a watery fi
eld end
                 start three dogs running through a puddle end
                 start three dogs run through a soggy field end
                 start three dogs run through water and grass end

The Best  Blue Score is  0.7796914510717229
```



```
None
```

In [46]:

```
import pickle
with open("/content/drive/My Drive/CASE Studies/Image captioning/Attention/voc
abulary.pkl","wb") as f:
  pickle.dump(vocabulary,f)
```

In [47]:

```
import pickle
with open("/content/drive/My Drive/CASE Studies/Image captioning/Attention/invert_vocab.pkl","wb") as f:
  pickle.dump(invert_vocab,f)
```

In [ ]: