In [1]:
```
import os
import string
import glob
from tensorflow.keras.applications import MobileNet
import tensorflow.keras.applications.mobilenet
from gtts import gTTS

from tensorflow.keras.applications.inception_v3 import InceptionV3
import tensorflow.keras.applications.inception_v3


from tqdm import tqdm
import tensorflow.keras.preprocessing.image
import pickle
from time import time
import numpy as np
from PIL import Image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Embedding, TimeDistributed, Dense, Repe
                            Activation, Flatten, Reshape, concatenate, Dropout, Bate
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras import Input, layers
from tensorflow.keras import optimizers

from tensorflow.keras.models import Model

from tensorflow.keras.layers import add
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

START = "startseq"
STOP = "endseq"
EPOCHS = 10
USE_INCEPTION = True
```

In [2]:
```
def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return f"{h}:{m:>02}:{s:>05.2f}"
```

In [3]:
```
root_captioning = "E:\captions"
```

In [4]:
```
null_punct = str.maketrans('', '', string.punctuation)
lookup = dict()

with open( os.path.join(root_captioning,'Flickr8k_text','Flickr8k.token.txt'), '

  max_length = 0
  for line in fp.read().split('\n'):
    tok = line.split()
    if len(line) >= 2:
      id = tok[0].split('.')[0]
      desc = tok[1:]

      # Cleanup description
      desc = [word.lower() for word in desc]
      desc = [w.translate(null_punct) for w in desc]
      desc = [word for word in desc if len(word)>1]
      desc = [word for word in desc if word.isalpha()]
      max_length = max(max_length,len(desc))

      if id not in lookup:
        lookup[id] = list()
      lookup[id].append(' '.join(desc))

lex = set()
for key in lookup:
  [lex.update(d.split()) for d in lookup[key]]
```

In [5]:
```
print(len(lookup)) # How many unique words
print(len(lex)) # The dictionary
print(max_length) # Maximum length of a caption (in words)
```

```
8092
8763
32
```

In [6]:
```
# Just rerun if len(img) = 0
img = glob.glob(os.path.join(root_captioning,'Flicker8k_Dataset', '*.jpg'))
```

In [7]:
```
len(img)
```

Out[7]:
```
8091
```

In [8]:
```
train_images_path = os.path.join(root_captioning,'Flickr8k_text','Flickr_8k.trai
train_images = set(open(train_images_path, 'r').read().strip().split('\n'))
test_images_path = os.path.join(root_captioning,'Flickr8k_text','Flickr_8k.testI
test_images = set(open(test_images_path, 'r').read().strip().split('\n'))

train_img = []
test_img = []

for i in img:
  f = os.path.split(i)[-1]
  if f in train_images:
    train_img.append(f)
  elif f in test_images:
    test_img.append(f)
```

In [9]:
```
print(len(train_images))
print(len(test_images))
```

```
6000
1000
```

In [10]:
```
train_descriptions = {k:v for k,v in lookup.items() if f'{k}.jpg' in train_image
for n,v in train_descriptions.items():
  for d in range(len(v)):
    v[d] = f'{START} {v[d]} {STOP}'
```

In [11]:
```
len(train_descriptions)
```

Out[11]:
```
6000
```

In [ ]:
```

```

In [12]:
```
if USE_INCEPTION:
  encode_model = InceptionV3(weights='imagenet')
  encode_model = Model(encode_model.input, encode_model.layers[-2].output)
  WIDTH = 299
  HEIGHT = 299
  OUTPUT_DIM = 2048
  preprocess_input = tensorflow.keras.applications.inception_v3.preprocess_input
else:
  encode_model = MobileNet(weights='imagenet',include_top=False)
  WIDTH = 224
  HEIGHT = 224
  OUTPUT_DIM = 50176
  preprocess_input = tensorflow.keras.applications.mobilenet.preprocess_input
```

```
WARNING:tensorflow:From C:\Users\anupa\Anaconda3\envs\TensorFlow-GPU\lib\site-p
ackages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [13]:
```
encode_model.summary()
```

```
__________________________________________________________________________________________________
batch_normalization_v1_4 (Batch (None, 71, 71, 192)  576         conv2d_4[0]
[0]
__________________________________________________________________________________________________
activation_4 (Activation)       (None, 71, 71, 192)  0           batch_normal
ization_v1_4[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 35, 35, 192)  0           activation_4
[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 35, 35, 64)   12288       max_pooling2
d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_v1_8 (Batch (None, 35, 35, 64)   192         conv2d_8[0]
[0]
```

In [14]:
```
def encodeImage(img):
  # Resize all images to a standard size (specified bythe image encoding network)
  img = img.resize((WIDTH, HEIGHT), Image.ANTIALIAS)
  # Convert a PIL image to a numpy array
  x = tensorflow.keras.preprocessing.image.img_to_array(img)
  # Expand to 2D array
  x = np.expand_dims(x, axis=0)
  # Perform any preprocessing needed by InceptionV3 or others
  x = preprocess_input(x)
  # Call InceptionV3 (or other) to extract the smaller feature set for the image
  x = encode_model.predict(x) # Get the encoding vector for the image
  # Shape to correct form to be accepted by LSTM captioning network.
  x = np.reshape(x, OUTPUT_DIM )
  return x
```

In [15]:
```
train_path = os.path.join(root_captioning,"data",f'train{OUTPUT_DIM}.pkl')
if not os.path.exists(train_path):
  start = time()
  encoding_train = {}
  for id in tqdm(train_img):
    image_path = os.path.join(root_captioning,'Flicker8k_Dataset', id)
    img = tensorflow.keras.preprocessing.image.load_img(image_path, target_size=
    encoding_train[id] = encodeImage(img)
  with open(train_path, "wb") as fp:
    pickle.dump(encoding_train, fp)
  print(f"\nGenerating training set took: {hms_string(time()-start)}")
else:
  with open(train_path, "rb") as fp:
    encoding_train = pickle.load(fp)
```

In [16]:
```
test_path = os.path.join(root_captioning,"data",f'test{OUTPUT_DIM}.pkl')
if not os.path.exists(test_path):
  start = time()
  encoding_test = {}
  for id in tqdm(test_img):
    image_path = os.path.join(root_captioning,'Flicker8k_Dataset', id)
    img = tensorflow.keras.preprocessing.image.load_img(image_path, target_size=
    encoding_test[id] = encodeImage(img)
  with open(test_path, "wb") as fp:
    pickle.dump(encoding_test, fp)
  print(f"\nGenerating testing set took: {hms_string(time()-start)}")
else:
  with open(test_path, "rb") as fp:
    encoding_test = pickle.load(fp)
```

In [17]:
```
all_train_captions = []
for key, val in train_descriptions.items():
    for cap in val:
        all_train_captions.append(cap)
len(all_train_captions)
```

Out[17]: 30000

In [18]:
```
word_count_threshold = 10
word_counts = {}
nsents = 0
for sent in all_train_captions:
    nsents += 1
    for w in sent.split(' '):
        word_counts[w] = word_counts.get(w, 0) + 1

vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
print('preprocessed words %d ==> %d' % (len(word_counts), len(vocab)))
```

```
preprocessed words 7578 ==> 1651
```

In [19]:
```
idxtoword = {}
wordtoidx = {}

ix = 1
for w in vocab:
    wordtoidx[w] = ix
    idxtoword[ix] = w
    ix += 1

vocab_size = len(idxtoword) + 1
vocab_size
```

Out[19]: 1652

In [20]:
```
max_length +=2
print(max_length)
```

```
34
```

In [21]:
```
def data_generator(descriptions, photos, wordtoidx, max_length, num_photos_per_ba
  # x1 - Training data for photos
  # x2 - The caption that goes with each photo
  # y - The predicted rest of the caption
  x1, x2, y = [], [], []
  n=0
  while True:
    for key, desc_list in descriptions.items():
      n+=1
      photo = photos[key+'.jpg']
      # Each photo has 5 descriptions
      for desc in desc_list:
        # Convert each word into a list of sequences.
        seq = [wordtoidx[word] for word in desc.split(' ') if word in wordtoidx]
        # Generate a training case for every possible sequence and outcome
        for i in range(1, len(seq)):
          in_seq, out_seq = seq[:i], seq[i]
          in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
          out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
          x1.append(photo)
          x2.append(in_seq)
          y.append(out_seq)
      if n==num_photos_per_batch:
        yield ([np.array(x1), np.array(x2)], np.array(y))
        x1, x2, y = [], [], []
        n=0
```

In [22]:
```
glove_dir = os.path.join(root_captioning,'glove.6B')
embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.200d.txt'), encoding="utf-8")

for line in tqdm(f):
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs

f.close()
print(f'Found {len(embeddings_index)} word vectors.')
```

```
400000it [00:20, 19554.36it/s]

Found 400000 word vectors.
```

In [23]:
```
embedding_dim = 200

# Get 200-dim dense vector for each of the 10000 words in out vocabulary
embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in wordtoidx.items():
    #if i < max_words:
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector
```

In [24]:
```
embedding_matrix.shape
```

Out[24]:
```
(1652, 200)
```

In [25]:
```
inputs1 = Input(shape=(OUTPUT_DIM,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
caption_model = Model(inputs=[inputs1, inputs2], outputs=outputs)
```

```
WARNING:tensorflow:From C:\Users\anupa\Anaconda3\envs\TensorFlow-GPU\lib\site-p
ackages\tensorflow\python\keras\layers\core.py:143: calling dropout (from tenso
rflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a
future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - kee
p_prob`.
```

In [26]:
```
embedding_dim
```

Out[26]:
```
200
```

In [27]:
```
caption_model.summary()
```

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            (None, 34)           0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 2048)         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 34, 200)      330400      input_3[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 2048)         0           input_2[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 34, 200)      0           embedding[0]
[0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          524544      dropout[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 256)          467968      dropout_1[0]
[0]
__________________________________________________________________________________________________
add (Add)                       (None, 256)          0           dense[0][0]
                                                                 lstm[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          65792       add[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 1652)         424564      dense_1[0][0]
==================================================================================================
Total params: 1,813,268
Trainable params: 1,813,268
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [28]:
```
caption_model.layers[2].set_weights([embedding_matrix])
caption_model.layers[2].trainable = False
caption_model.compile(loss='categorical_crossentropy', optimizer='adam')
```

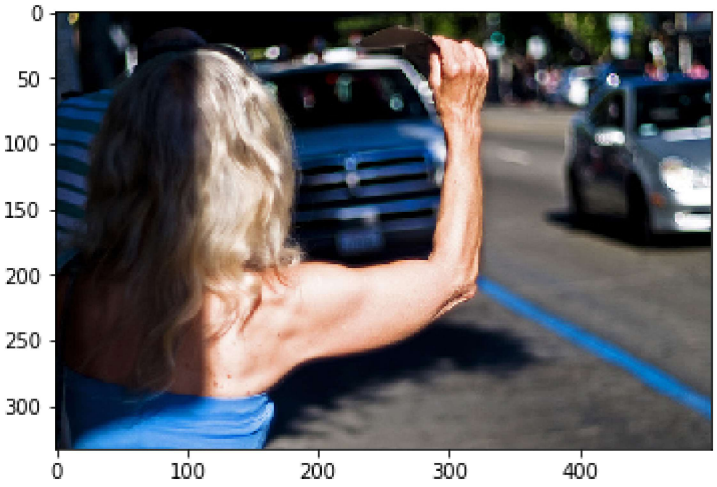
In [29]:
```
number_pics_per_bath = 3
steps = len(train_descriptions)//number_pics_per_bath
```

In [30]:
```
model_path = os.path.join(root_captioning,"data",f'caption-model.hdf5')
if not os.path.exists(model_path):
  for i in tqdm(range(EPOCHS*2)):
      generator = data_generator(train_descriptions, encoding_train, wordtoidx, 
      caption_model.fit_generator(generator, epochs=1, steps_per_epoch=steps, ver

  caption_model.optimizer.lr = 1e-4
  number_pics_per_bath = 6
  steps = len(train_descriptions)//number_pics_per_bath

  for i in range(EPOCHS):
      generator = data_generator(train_descriptions, encoding_train, wordtoidx, 
      caption_model.fit_generator(generator, epochs=1, steps_per_epoch=steps, ver
  caption_model.save_weights(model_path)
  print(f"\Training took: {hms_string(time()-start)}")
else:
  caption_model.load_weights(model_path)
```

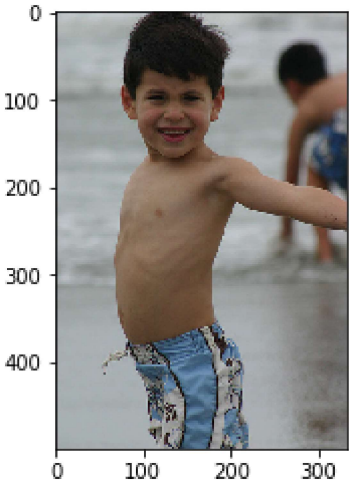
In [31]:
```
def generateCaption(photo):
    in_text = START
    for i in range(max_length):
        sequence = [wordtoidx[w] for w in in_text.split() if w in wordtoidx]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = caption_model.predict([photo,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = idxtoword[yhat]
        in_text += ' ' + word
        if word == STOP:
            break
    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final
```

In [32]:
```
for z in range(5):
  pic = list(encoding_test.keys())[z]
  image = encoding_test[pic].reshape((1,OUTPUT_DIM))
  print(os.path.join(root_captioning,'Flicker8k_Dataset', pic))
  x=plt.imread(os.path.join(root_captioning,'Flicker8k_Dataset', pic))
  plt.imshow(x)
  plt.show()
  print("Caption:",generateCaption(image))
  print("_____________________________________")
```

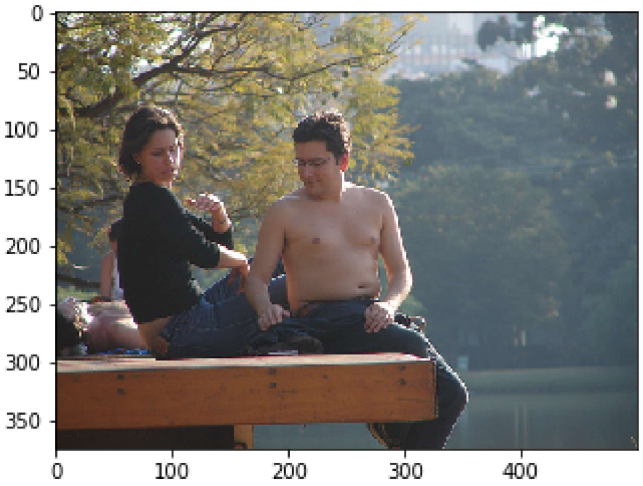
```
E:\captions\Flicker8k_Dataset\1056338697_4f7d7ce270.jpg
```




```
Caption: the woman is wearing black jacket and black hat
_____________________________________
E:\captions\Flicker8k_Dataset\106490881_5a2dd9b7bd.jpg
```

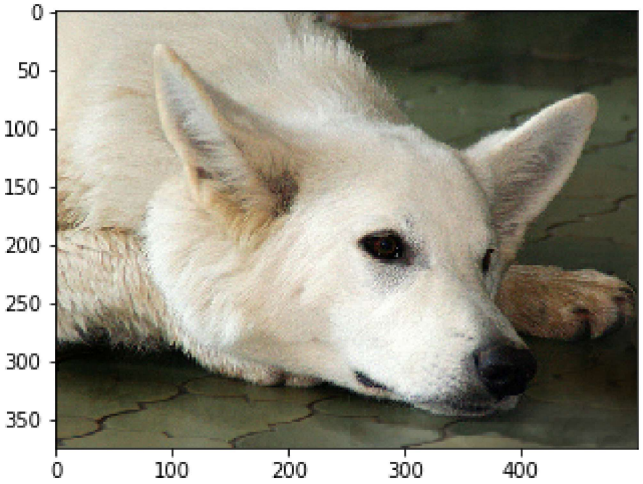



```
Caption: small boy wades in the water
_____________________________________
E:\captions\Flicker8k_Dataset\1082379191_ec1e53f996.jpg
```



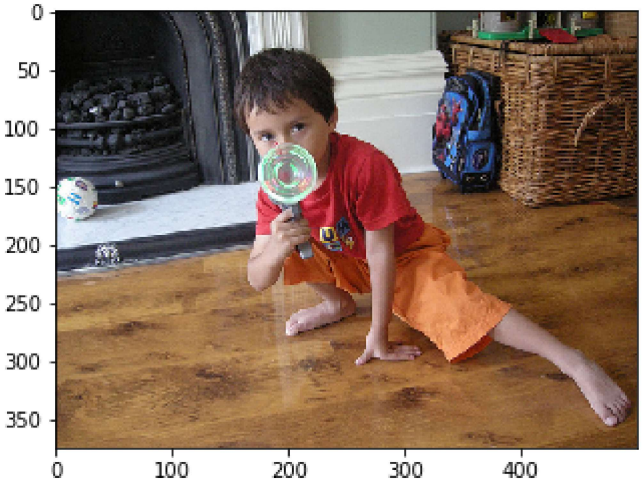
Caption: man in black shirt is sitting on the edge of rock

E:\captions\Flicker8k_Dataset\1084040636_97d9633581.jpg



Caption: dog is running through the snow

E:\captions\Flicker8k_Dataset\1096395242_fc69f0ae5a.jpg



Caption: young boy is licking water from water fountain

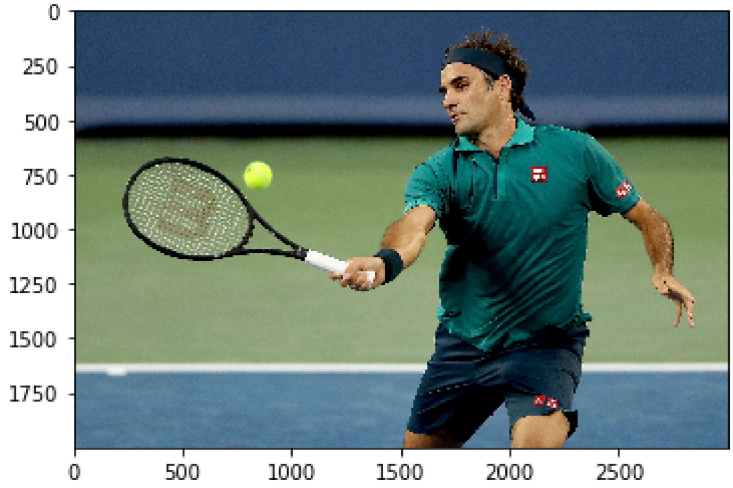
In [33]:
```
encoding_test[pic].shape
```

Out[33]:
```
(2048,)
```

In [44]:
```
from PIL import Image, ImageFile
from matplotlib.pyplot import imshow
import requests
from io import BytesIO
import numpy as np

%matplotlib inline

urls = [
 "https://github.com/Priye-Ranjan/Auto-Image-Captioning-using-ML/blob/master/1.jp
  "https://github.com/Priye-Ranjan/Auto-Image-Captioning-using-ML/blob/master/2.j
 "https://github.com/Priye-Ranjan/Auto-Image-Captioning-using-ML/blob/master/3.jp
  "https://github.com/Priye-Ranjan/Auto-Image-Captioning-using-ML/blob/master/4.j
    "https://github.com/Priye-Ranjan/Auto-Image-Captioning-using-ML/blob/master/m
  ]

l=[]
x=0
for url in urls:
  response = requests.get(url)
  img = Image.open(BytesIO(response.content))
  img.load()


  plt.imshow(img)
  plt.show()

  response = requests.get(url)

  img = encodeImage(img).reshape((1,OUTPUT_DIM))
  print(img.shape)
  print("Caption:",generateCaption(img))
  l.append(generateCaption(img))
  mytext=l[x]
  x=x+1
  language = 'en'
  myobj = gTTS(text=mytext, lang=language, slow=False)
  myobj.save("welcome2.mp3")
  os.system("mpg321 welcome.mp3")
  print("_____")
```

```
(1, 2048)
Caption: dog is running through the grass
```

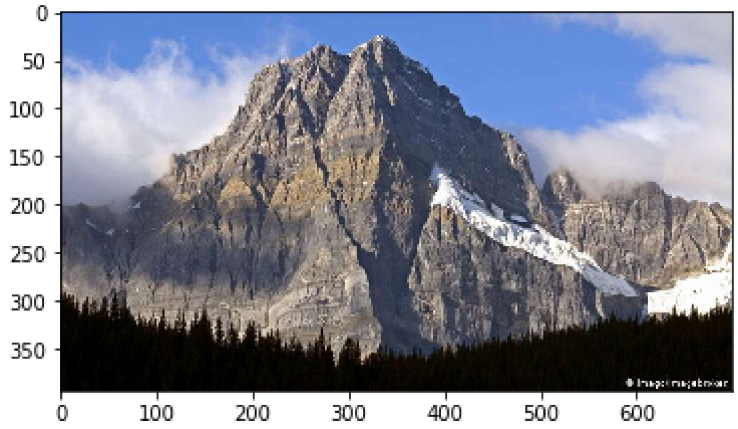

```
(1, 2048)
Caption: two boys play soccer on field
```




```
(1, 2048)
Caption: woman in white and white outfit is swinging tennis ball in her hand
```


```
(1, 2048)
```

```
Caption: man in red shirt is jumping over log in the snow
```




```
(1, 2048)
Caption: man is standing on top of mountain looking at the mountains
```

In [ ]: