In [1]:

```
import numpy as np
from numpy import array
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import string
import os
from PIL import Image
import glob
import pickle
import json
from pickle import dump, load
from time import time
import tensorflow as tf
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVect
or,\
                        Activation, Flatten, Reshape, concatenate, Dropout, BatchNorma
lization,Bidirectional

from tensorflow.keras.optimizers import Adam, RMSprop
from keras.layers.merge import add
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras import Input, layers
from tensorflow.keras import optimizers
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
```

Mount Drive

In [2]:

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

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

In [3]:

```
import os
ROOT = r'/content/drive/MyDrive/College/CSCE 5214 - Software Development for Artificial
Intelligence/P2/Project '
os.chdir(ROOT)
assert os.getcwd() == ROOT
```

In [4]:

```
# load doc into memory
def load_doc(filename):
        # open the file as read only
        file = open(filename, 'r')
        # read all text
        text = file.read()
        # close the file
        file.close()
        return text

filename = "Dataset/Flickr8k_text/Flickr8k.token.txt"
# load descriptions
doc = load_doc(filename)
print(doc[:300])
```

```
1000268201_693b08cb0e.jpg#0     A child in a pink dress is climbing up a s
et of stairs in an entry way .
1000268201_693b08cb0e.jpg#1     A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2     A little girl climbing into a wooden playh
ouse .
1000268201_693b08cb0e.jpg#3     A little girl climbing the s
```

In [5]:

```
def load_descriptions(doc):
        mapping = dict()
        # process lines
        for line in doc.split('\n'):
                # split line by white space
                tokens = line.split()
                if len(line) < 2:
                        continue
                # take the first token as the image id, the rest as the description
                image_id, image_desc = tokens[0], tokens[1:]
                # extract filename from image id
                image_id = image_id.split('.')[0]
                # convert description tokens back to string
                image_desc = ' '.join(image_desc)
                # create the list if needed
                if image_id not in mapping:
                        mapping[image_id] = list()
                # store description
                mapping[image_id].append(image_desc)
        return mapping

# parse descriptions
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))
```

```
Loaded: 8092
```

In [6]:

```
list(descriptions.keys())[:5]
```

Out[6]:

```
['1000268201_693b08cb0e',
 '1001773457_577c3a7d70',
 '1002674143_1b742ab4b8',
 '1003163366_44323f5815',
 '1007129816_e794419615']
```

In [7]:

```
descriptions['1000268201_693b08cb0e']
```

Out[7]:

```
['A child in a pink dress is climbing up a set of stairs in an entry way
.',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']
```

In [8]:

```
descriptions['1001773457_577c3a7d70']
```

Out[8]:

```
['A black dog and a spotted dog are fighting',
 'A black dog and a tri-colored dog playing with each other on the road
.',
 'A black dog and a white dog with brown spots are staring at each other i
n the street .',
 'Two dogs of different breeds looking at each other on the road .',
 'Two dogs on pavement moving toward each other .']
```

In [9]:

```
def clean_descriptions(descriptions):
        # prepare translation table for removing punctuation
        table = str.maketrans('', '', string.punctuation)
        for key, desc_list in descriptions.items():
                for i in range(len(desc_list)):
                        desc = desc_list[i]
                        # tokenize
                        desc = desc.split()
                        # convert to lower case
                        desc = [word.lower() for word in desc]
                        # remove punctuation from each token
                        desc = [w.translate(table) for w in desc]
                        # remove hanging 's' and 'a'
                        desc = [word for word in desc if len(word)>1]
                        # remove tokens with numbers in them
                        desc = [word for word in desc if word.isalpha()]
                        # store as string
                        desc_list[i] =  ' '.join(desc)

# clean descriptions
clean_descriptions(descriptions)
```

In [10]:

```
descriptions['1000268201_693b08cb0e']
```

Out[10]:

```
['child in pink dress is climbing up set of stairs in an entry way',
 'girl going into wooden building',
 'little girl climbing into wooden playhouse',
 'little girl climbing the stairs to her playhouse',
 'little girl in pink dress going into wooden cabin']
```

In [11]:

```
descriptions['1001773457_577c3a7d70']
```

Out[11]:

```
['black dog and spotted dog are fighting',
 'black dog and tricolored dog playing with each other on the road',
 'black dog and white dog with brown spots are staring at each other in th
e street',
 'two dogs of different breeds looking at each other on the road',
 'two dogs on pavement moving toward each other']
```

In [12]:

```
# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
        # build a list of all description strings
        all_desc = set()
        for key in descriptions.keys():
                [all_desc.update(d.split()) for d in descriptions[key]]
        return all_desc

# summarize vocabulary
vocabulary = to_vocabulary(descriptions)
print('Original Vocabulary Size: %d' % len(vocabulary))
```

```
Original Vocabulary Size: 8763
```

In [13]:

```
# save descriptions to file, one per line
def save_descriptions(descriptions, filename):
        lines = list()
        for key, desc_list in descriptions.items():
                for desc in desc_list:
                        lines.append(key + ' ' + desc)
        data = '\n'.join(lines)
        file = open(filename, 'w')
        file.write(data)
        file.close()

save_descriptions(descriptions, 'descriptions.txt')
```

In [14]:

```
# load a pre-defined list of photo identifiers
def load_set(filename):
        doc = load_doc(filename)
        dataset = list()
        # process line by line
        for line in doc.split('\n'):
                # skip empty lines
                if len(line) < 1:
                        continue
                # get the image identifier
                identifier = line.split('.')[0]
                dataset.append(identifier)
        return set(dataset)

# load training dataset (6K)
filename = 'Dataset/Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
```

```
Dataset: 6000
```

In [15]:

```
# Below path contains all the images
images = 'Dataset/Flickr8k_Dataset/'
# Create a list of all image names in the directory
img = glob.glob(images+'*.jpg')
```

In [16]:

```
# Below file conatains the names of images to be used in train data
train_images_file = 'Dataset/Flickr8k_text/Flickr_8k.trainImages.txt'
# Read the train image names in a set
train_images = set(open(train_images_file, 'r').read().strip().split('\n'))

# Create a list of all the training images with their full path names
train_img = []
for i in img: # img is list of full path names of all images
    if i[len(images):] in train_images: # Check if the image belongs to training set
        train_img.append(i) # Add it to the list of train images
```

In [17]:

```
# Below file conatains the names of images to be used in test data
test_images_file = 'Dataset/Flickr8k_text/Flickr_8k.testImages.txt'
# Read the validation image names in a set# Read the test image names in a set
test_images = set(open(test_images_file, 'r').read().strip().split('\n'))

# Create a list of all the test images with their full path names
test_img = []

for i in img: # img is list of full path names of all images
    if i[len(images):] in test_images: # Check if the image belongs to test set
        test_img.append(i) # Add it to the list of test images
```

In [18]:

```
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
        # load document
        doc = load_doc(filename)
        descriptions = dict()
        for line in doc.split('\n'):
                # split line by white space
                tokens = line.split()
                # split id from description
                image_id, image_desc = tokens[0], tokens[1:]
                # skip images not in the set
                if image_id in dataset:
                        # create list
                        if image_id not in descriptions:
                                descriptions[image_id] = list()
                        # wrap description in tokens
                        desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
                        # store
                        descriptions[image_id].append(desc)
        return descriptions

# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
```

```
Descriptions: train=6000
```

In [19]:

```
def preprocess(image_path):
    # Convert all the images to size 299x299 as expected by the inception v3 model
    img = image.load_img(image_path, target_size=(299, 299))
    # Convert PIL image to numpy array of 3-dimensions
    x = image.img_to_array(img)
    # Add one more dimension
    x = np.expand_dims(x, axis=0)
    # preprocess the images using preprocess_input() from inception module
    x = preprocess_input(x)
    return x
```

In [20]:

```
# Load the inception v3 model
model = InceptionV3(weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-appl
ications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96116736/96112376 [==============================] - 1s 0us/step
96124928/96112376 [==============================] - 1s 0us/step
```

In [21]:

```
# Create a new model, by removing the last layer (output layer) from the inception v3
model_new = Model(model.input, model.layers[-2].output)
```

In [22]:

```
# Function to encode a given image into a vector of size (2048, )
def encode(image):
    image = preprocess(image) # preprocess the image
    fea_vec = model_new.predict(image) # Get the encoding vector for the image
    fea_vec = np.reshape(fea_vec, fea_vec.shape[1]) # reshape from (1, 2048) to (2048,
 )
    return fea_vec
```

In [23]:

```
# Call the funtion to encode all the train images
# This will take a while on CPU - Execute this only once
start = time()
encoding_train = {}
for img in train_img:
    encoding_train[img[len(images):]] = encode(img)
print("Time taken in seconds =", time()-start)
```

```
Time taken in seconds = 1930.2793369293213
```

In [24]:

```
# Save the bottleneck train features to disk
with open("saved_models/encoded_train_images.pkl", "wb") as encoded_pickle:
    pickle.dump(encoding_train, encoded_pickle)
```

In [25]:

```
# Call the funtion to encode all the test images - Execute this only once
start = time()
encoding_test = {}
for img in test_img:
    encoding_test[img[len(images):]] = encode(img)
print("Time taken in seconds =", time()-start)
```

```
Time taken in seconds = 324.2758436203003
```

In [26]:

```
# Save the bottleneck test features to disk
with open("saved_models/encoded_test_images.pkl", "wb") as encoded_pickle:
    pickle.dump(encoding_test, encoded_pickle)
```

In [27]:

```
train_features = load(open("saved_models/encoded_train_images.pkl", "rb"))
print('Photos: train=%d' % len(train_features))
```

```
Photos: train=6000
```

In [28]:

```
# Create a list of all the training captions
all_train_captions = []
for key, val in train_descriptions.items():
    for cap in val:
        all_train_captions.append(cap)
len(all_train_captions)
```

Out[28]:

```
30000
```

In [29]:

```
# Consider only words which occur at least 10 times in the corpus
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 [30]:

```
ixtoword = {}
wordtoix = {}

ix = 1
for w in vocab:
    wordtoix[w] = ix
    ixtoword[ix] = w
    ix += 1
```

In [31]:

```
vocab_size = len(ixtoword) + 1 # one for appended 0's
vocab_size
```

Out[31]:

```
1652
```

In [32]:

```
# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
        all_desc = list()
        for key in descriptions.keys():
                [all_desc.append(d) for d in descriptions[key]]
        return all_desc

# calculate the length of the description with the most words
def max_length(descriptions):
        lines = to_lines(descriptions)
        return max(len(d.split()) for d in lines)

# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
```

```
Description Length: 34
```

In [33]:

```
# data generator, intended to be used in a call to model.fit_generator()
def data_generator(descriptions, photos, wordtoix, max_length, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    # print(descriptions)
    # print(photos)
    # loop for ever over images
    while 1:
        for key, desc_list in descriptions.items():
            n+=1
            # retrieve the photo feature
            photo = photos[key+'.jpg']
            for desc in desc_list:
                # encode the sequence
                seq = [wordtoix[word] for word in desc.split(' ') if word in wordtoix]
                # split one sequence into multiple X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pair
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    # store
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(out_seq)
            # yield the batch data
            if n==num_photos_per_batch:
                yield [[array(X1), array(X2)], array(y)]
                X1, X2, y = list(), list(), list()
                n=0
```

In [34]:

```
# Load Glove vectors
glove_dir = ''
embeddings_index = {} # empty dictionary
f = open(os.path.join(glove_dir, 'glove.6B.200d.txt'), encoding="utf-8")

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 400000 word vectors.
```

In [35]:

```
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 wordtoix.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 [36]:

```
embedding_matrix.shape
```

Out[36]:

```
(1652, 200)
```

In [37]:

```
inputs1 = Input(shape=(2048,))
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)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
```

In [38]:

```
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________
_____________________
 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           ['embeddi
ng[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 [39]:

```
model.layers[2]
```

Out[39]:

```
<keras.layers.embeddings.Embedding at 0x7f3d98bf7bd0>
```

In [40]:

```
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False
```

In [41]:

```
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

In [42]:

```
epochs = 10
number_pics_per_bath = 3
steps = len(train_descriptions)//number_pics_per_bath
```

In [43]:

```
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, wordtoix, max_length
, number_pics_per_bath)
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    model.save('saved_models/model_' + str(i) + '.h5')
```

```
2000/2000 [==============================] - 143s 70ms/step - loss: 4.1305
   1/2000 [..............................] - ETA: 2:29 - loss: 3.6378

/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: Cu
stomMaskWarning: Custom mask layers require a config and must override get
_config. When loading, the custom mask layer must be passed to the custom_
objects argument.
  layer_config = serialize_layer_fn(layer)

2000/2000 [==============================] - 139s 70ms/step - loss: 3.4240
2000/2000 [==============================] - 138s 69ms/step - loss: 3.2023
2000/2000 [==============================] - 137s 69ms/step - loss: 3.0694
2000/2000 [==============================] - 139s 69ms/step - loss: 2.9767
2000/2000 [==============================] - 139s 69ms/step - loss: 2.9040
2000/2000 [==============================] - 139s 69ms/step - loss: 2.8465
2000/2000 [==============================] - 139s 69ms/step - loss: 2.8023
2000/2000 [==============================] - 140s 70ms/step - loss: 2.7611
2000/2000 [==============================] - 139s 70ms/step - loss: 2.7268
```

In [44]:

```
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, wordtoix, max_length
, number_pics_per_bath)
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    model.save('saved_models/model_' + str(i) + '.h5')
```

```
   2/2000 [..............................] - ETA: 2:20 - loss: 2.7729

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarnin
g: `Model.fit_generator` is deprecated and will be removed in a future ver
sion. Please use `Model.fit`, which supports generators.
  This is separate from the ipykernel package so we can avoid doing import
s until

2000/2000 [==============================] - 140s 70ms/step - loss: 2.6975
   1/2000 [..............................] - ETA: 2:23 - loss: 2.9138

/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: Cu
stomMaskWarning: Custom mask layers require a config and must override get
_config. When loading, the custom mask layer must be passed to the custom_
objects argument.
  layer_config = serialize_layer_fn(layer)


2000/2000 [==============================] - 139s 70ms/step - loss: 2.6716
2000/2000 [==============================] - 139s 70ms/step - loss: 2.6447
2000/2000 [==============================] - 139s 70ms/step - loss: 2.6247
2000/2000 [==============================] - 139s 70ms/step - loss: 2.6087
2000/2000 [==============================] - 139s 70ms/step - loss: 2.5927
2000/2000 [==============================] - 139s 70ms/step - loss: 2.5749
2000/2000 [==============================] - 140s 70ms/step - loss: 2.5620
2000/2000 [==============================] - 139s 70ms/step - loss: 2.5463
2000/2000 [==============================] - 140s 70ms/step - loss: 2.5359
```

In [ ]:

```
model.optimizer.lr = 0.0001
epochs = 10
number_pics_per_bath = 6
steps = len(train_descriptions)//number_pics_per_bath
```

In [46]:

```
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, wordtoix, max_length
, number_pics_per_bath)
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    #model.save('./model_weights/model_' + str(i) + '.h5')
```

```
   1/1000 [..............................] - ETA: 1:19 - loss: 2.6750

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: UserWarnin
g: `Model.fit_generator` is deprecated and will be removed in a future ver
sion. Please use `Model.fit`, which supports generators.
  This is separate from the ipykernel package so we can avoid doing import
s until

1000/1000 [==============================] - 72s 72ms/step - loss: 2.5052
1000/1000 [==============================] - 72s 72ms/step - loss: 2.4554
1000/1000 [==============================] - 72s 72ms/step - loss: 2.4353
1000/1000 [==============================] - 72s 72ms/step - loss: 2.4245
1000/1000 [==============================] - 72s 72ms/step - loss: 2.4143
1000/1000 [==============================] - 72s 72ms/step - loss: 2.4065
1000/1000 [==============================] - 72s 72ms/step - loss: 2.3947
1000/1000 [==============================] - 72s 72ms/step - loss: 2.3897
1000/1000 [==============================] - 72s 72ms/step - loss: 2.3830
1000/1000 [==============================] - 72s 72ms/step - loss: 2.3796
```

In [ ]:

```
model.save_weights('saved_models/model_30.h5')
```

In [ ]:

```
model.load_weights('saved_models/model_30.h5')
```

In [49]:

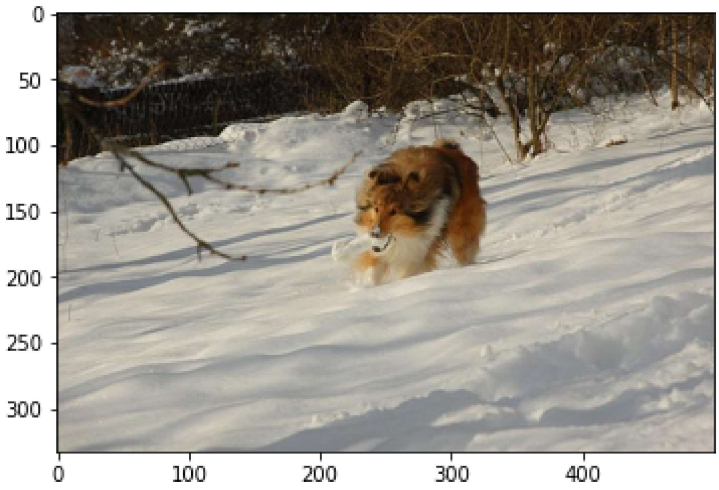
```
with open("saved_models/encoded_test_images.pkl", "rb") as encoded_pickle:
    encoding_test = load(encoded_pickle)
```

In [50]:

```
def greedySearch(photo):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = [wordtoix[w] for w in in_text.split() if w in wordtoix]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = ixtoword[yhat]
        in_text += ' ' + word
        if word == 'endseq':
            break
    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final
```

In [52]:

```
z=0
z+=1
pic = list(encoding_test.keys())[z]
image = encoding_test[pic].reshape((1,2048))
x=plt.imread(images+pic)
plt.imshow(x)
plt.show()
print("Greedy:",greedySearch(image))
```



Greedy: dog running through the snow