In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
% matplotlib inline
```

READ TEXT DATA

In [4]:
```
# read text captions
def readTextFile(path):
    with open(path) as f:
        captions = f.read()
    return captions
```

In [5]:
```
captions = readTextFile("Flickr_Data/Flickr_TextData/Flickr8k.token.txt")
print(len(captions.split("\n")))
```

```
40461
```

In [6]:
```
captions = captions.split("\n")[0:-1] # last line in file is empty
```

In [7]:
```
captions[0] # 0th line
```

Out[7]:
```
'1000268201_693b08cb0e.jpg#0\tA child in a pink dress is climbing up a set of stairs in an entry way .'
```

In [8]:
```
captions
```

Out[8]:
```
['1000268201_693b08cb0e.jpg#0\tA child in a pink dress is climbing up a set of stairs in an entry way
.',
 '1000268201_693b08cb0e.jpg#1\tA girl going into a wooden building .',
 '1000268201_693b08cb0e.jpg#2\tA little girl climbing into a wooden playhouse .',
 '1000268201_693b08cb0e.jpg#3\tA little girl climbing the stairs to her playhouse .',
 '1000268201_693b08cb0e.jpg#4\tA little girl in a pink dress going into a wooden cabin .',
 '1001773457_577c3a7d70.jpg#0\tA black dog and a spotted dog are fighting',
 '1001773457_577c3a7d70.jpg#1\tA black dog and a tri-colored dog playing with each other on the road
.',
 '1001773457_577c3a7d70.jpg#2\tA black dog and a white dog with brown spots are staring at each other
in the street .',
 '1001773457_577c3a7d70.jpg#3\tTwo dogs of different breeds looking at each other on the road .',
 '1001773457_577c3a7d70.jpg#4\tTwo dogs on pavement moving toward each other .',
 '1002674143_1b742ab4b8.jpg#0\tA little girl covered in paint sits in front of a painted rainbow with
her hands in a bowl .',
 '1002674143_1b742ab4b8.jpg#1\tA little girl is sitting in front of a large painted rainbow .',
 '1002674143_1b742ab4b8.jpg#2\tA small girl in the grass plays with fingerpaints in front of a white
canvas with a rainbow on it .',
 '1002674143_1b742ab4b8.jpg#3\tThere is a girl with pigtails sitting in front of a rainbow painting
 '
```

In [9]:
```
print(len(captions))
```

```
40460
```

In [10]:
```
# make a dictionary mapping each image to the captions it corresponds to.
d = {}
```

In [11]:
```
for cap in captions:
    first, second = cap.split("\t")
    img_id = first.split(".")[0]
    if d.get(img_id) is None: # check if image_id is already present in dictionary
        d[img_id] = []
    d[img_id].append(second)
```

In [12]:
```
d["1009434119_febe49276a"]
```

Out[12]:
```
['A black and white dog is running in a grassy garden surrounded by a white fence .',
 'A black and white dog is running through the grass .',
 'A Boston terrier is running in the grass .',
 'A Boston Terrier is running on lush green grass in front of a white fence .',
 'A dog runs on the green grass near a wooden fence .']
```

In [13]:
```
d
```

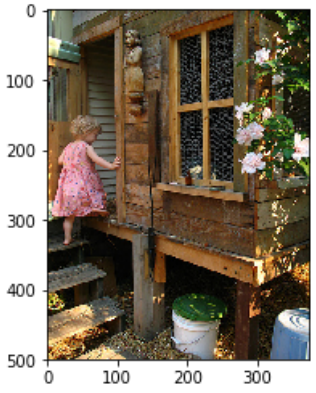
Out[13]:
```
{'1000268201_693b08cb0e': ['A child in a pink dress is climbing up a set of stairs in an entry way
.',
  'A girl going into a wooden building .',
  'A little girl climbing into a wooden playhouse .',
  'A little girl climbing the stairs to her playhouse .',
  'A little girl in a pink dress going into a wooden cabin .'],
 '1001773457_577c3a7d70': ['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 in the street .',
  'Two dogs of different breeds looking at each other on the road .',
  'Two dogs on pavement moving toward each other .'],
 '1002674143_1b742ab4b8': ['A little girl covered in paint sits in front of a painted rainbow with he
r hands in a bowl .',
  'A little girl is sitting in front of a large painted rainbow .',
  'A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it
.',
  'There is a girl with pigtails sitting in front of a rainbow painting .',
  'Young girl with pigtails painting outside in the grass .'],
 '1003163366_44323f5815': ['A man lays on a bench while his dog sits by him .',
```

In [14]:
```
len(d)*5
```

Out[14]:
```
40460
```

In [15]:
```
IMG_PATH = "/Flickr_Data/Images/"
import cv2
import matplotlib.pyplot as plt
```

In [16]:
```
img = cv2.imread("C:\ANACONDA\Scripts\MINOR PROJECT IMAGE CAPTIONING" + IMG_PATH+"1000268201_693b08cb0e.j
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()
```



In [17]:
```
d["1000268201_693b08cb0e"]
```

Out[17]:
```
['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 .']
```

DATA CLEANING

In [18]:
```
import re
```

In [19]:
```
def clean_text(sentence):
    sentence = sentence.lower()
    sentence = re.sub("[^a-z]+"," ",sentence)
    sentence = sentence.split()

    sentence = [s for s in sentence if len(s) > 1]
    sentence = " ".join(sentence)
    return sentence
```

In [20]:
```
# clean all captions
for key, caption_list in d.items():
    for i in range(len(caption_list)):
        caption_list[i] = clean_text(caption_list[i])
```

In [21]:
```
d["1000268201_693b08cb0e"]
```

Out[21]:
```
['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 [22]:
```
#store the data in text file
with open("descriptions_1.txt", "w") as f:
    f.write(str(d))
```

CREATE VOCAB OF UNIQUE WORDS

In [23]:
```
import json
descriptions = None
with open("descriptions_1.txt") as f:
    descriptions = f.read()

json_acceptable_string = descriptions.replace("'", "\"")
descriptions = json.loads(json_acceptable_string)
```

In [24]:
```
print(type(descriptions))
```

```
<class 'dict'>
```

In [25]:
```
descriptions['1000268201_693b08cb0e']
```

Out[25]:
```
['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 [26]:
```
vocab = set()
for key in descriptions.keys():
    [vocab.update(sentence.split()) for sentence in descriptions[key]]
```

In [27]:
```
vocab
```

Out[27]:
```
{'profile',
 'took',
 'paleontologist',
 'military',
 'super',
 'assorted',
 'boatload',
 'eyebrows',
 'shire',
 'puma',
 'composure',
 'sightseeing',
 'shooting',
 'beagle',
 'forests',
 'if',
 'container',
 'fit',
 'viewer',
```

In [28]:
```
print(len(vocab))
```

```
8424
```

In [29]:
```
# total no of words across all sentences
total_words = []
for key in descriptions.keys():
    [total_words.append(i) for des in descriptions[key] for i in des.split()]
print(len(total_words))
```

```
373837
```

In [30]:
```
print(total_words[:10])
```

```
['child', 'in', 'pink', 'dress', 'is', 'climbing', 'up', 'set', 'of', 'stairs']
```

REMOVE INFREQUENT WORDS - pick only those words out of vocab that appear atleast 10 times in total_words

In [31]:
```
# find out frequency counts in total_words
import collections
counter = collections.Counter(total_words)
freq_cnt = dict(counter)
print(freq_cnt)
```

```
{'child': 1545, 'in': 18987, 'pink': 739, 'dress': 348, 'is': 9345, 'climbing': 507, 'up': 1302, 'se
t': 109, 'of': 6723, 'stairs': 109, 'an': 2432, 'entry': 1, 'way': 53, 'girl': 3328, 'going': 149, 'i
nto': 1074, 'wooden': 284, 'building': 511, 'little': 1768, 'playhouse': 6, 'the': 18420, 'to': 3176,
'her': 1178, 'cabin': 4, 'black': 3848, 'dog': 8138, 'and': 8863, 'spotted': 38, 'are': 3505, 'fighti
ng': 133, 'tri': 14, 'colored': 221, 'playing': 2008, 'with': 7765, 'each': 430, 'other': 773, 'on':
10746, 'road': 398, 'white': 3959, 'brown': 2578, 'spots': 29, 'staring': 57, 'at': 2916, 'street': 9
44, 'two': 5643, 'dogs': 2125, 'different': 46, 'breeds': 5, 'looking': 744, 'pavement': 48, 'movin
g': 41, 'toward': 146, 'covered': 372, 'paint': 62, 'sits': 577, 'front': 1386, 'painted': 64, 'rainb
ow': 22, 'hands': 246, 'bowl': 30, 'sitting': 1368, 'large': 1237, 'small': 1278, 'grass': 1622, 'pla
ys': 526, 'fingerpaints': 3, 'canvas': 6, 'it': 401, 'there': 304, 'pigtails': 14, 'painting': 43, 'y
oung': 2630, 'outside': 791, 'man': 7275, 'lays': 56, 'bench': 375, 'while': 1968, 'his': 2357, 'by':
1249, 'him': 403, 'which': 51, 'also': 20, 'tied': 15, 'sleeping': 60, 'next': 749, 'shirtless': 104,
'lies': 43, 'park': 508, 'laying': 189, 'holding': 1324, 'leash': 131, 'ground': 357, 'orange': 745,
'hat': 682, 'starring': 8, 'something': 346, 'wears': 115, 'glasses': 206, 'gauges': 2, 'wearing': 30
62, 'blitz': 1, 'beer': 45, 'can': 39, 'crocheted': 1, 'pierced': 6, 'ears': 69, 'rope': 251, 'net':
58, 'red': 2691, 'roping': 2, 'climbs': 201, 'bridge': 141, 'grips': 2, 'onto': 211, 'ropes': 38, 'pl
ayground': 201, 'running': 2073, 'grassy': 474, 'garden': 54, 'surrounded': 178, 'fence': 340, 'throu
gh': 2032, 'boston': 9, 'terrier': 31, 'lush': 8, 'green': 1234, 'runs': 925, 'near': 1026, 'shakes':
37, 'its': 925, 'head': 377, 'shore': 170, 'ball': 1783, 'edge': 170, 'beach': 1046, 'feet': 87, 'sta
nds': 869, 'shaking': 71, 'off': 766, 'water': 2790, 'standing': 1789, 'turned': 20, 'one': 1223, 'si
```

In [32]:
```
len(freq_cnt.keys())
```

Out[32]: 8424

In [33]:
```
# sort the dictionary acc to freq counts
sorted_freq_cnt = sorted(freq_cnt.items(), reverse = True, key = lambda x : x[1])
sorted_freq_cnt
```

Out[33]:
```
[('in', 18987),
 ('the', 18420),
 ('on', 10746),
 ('is', 9345),
 ('and', 8863),
 ('dog', 8138),
 ('with', 7765),
 ('man', 7275),
 ('of', 6723),
 ('two', 5643),
 ('white', 3959),
 ('black', 3848),
 ('boy', 3581),
 ('are', 3505),
 ('woman', 3403),
 ('girl', 3328),
 ('to', 3176),
 ('wearing', 3062),
 ('at', 2916),
 ('people', 2887)
```

In [34]:
```
threshold = 10
sorted_freq_cnt = [x for x in sorted_freq_cnt if x[1] > threshold]
sorted_freq_cnt
```

Out[34]:
```
[('in', 18987),
 ('the', 18420),
 ('on', 10746),
 ('is', 9345),
 ('and', 8863),
 ('dog', 8138),
 ('with', 7765),
 ('man', 7275),
 ('of', 6723),
 ('two', 5643),
 ('white', 3959),
 ('black', 3848),
 ('boy', 3581),
 ('are', 3505),
 ('woman', 3403),
 ('girl', 3328),
 ('to', 3176),
 ('wearing', 3062),
 ('at', 2916),
 ('people', 2887)
```

In [35]:
```
total_words = [x[0] for x in sorted_freq_cnt]
```

In [36]:
```
total_words
```

Out[36]:
```
['in',
 'the',
 'on',
 'is',
 'and',
 'dog',
 'with',
 'man',
 'of',
 'two',
 'white',
 'black',
 'boy',
 'are',
 'woman',
 'girl',
 'to',
 'wearing',
 'at',
 'people'
```

In [37]:
```
len(total_words)
```

Out[37]:
```
1845
```

PREPARE TRAIN TEST DATA

In [38]:
```
train_file_data = readTextFile("Flickr_Data/Flickr_TextData/Flickr_8k.trainImages.txt")
test_file_data = readTextFile("Flickr_Data/Flickr_TextData/Flickr_8k.testImages.txt")
```

In [39]:
```
print(train_file_data)
```

```
2513260012_03d33305cf.jpg
2903617548_d3e38d7f88.jpg
3338291921_fe7ae0c8f8.jpg
488416045_1c6d903fe0.jpg
2644326817_8f45080b87.jpg
218342358_1755a9cce1.jpg
2501968935_02f2cd8079.jpg
2699342860_5288e203ea.jpg
2638369467_8fc251595b.jpg
2926786902_815a99a154.jpg
2851304910_b5721199bc.jpg
3423802527_94bd2b23b0.jpg
3356369156_074750c6cc.jpg
2294598473_40637b5c04.jpg
1191338263_a4fa073154.jpg
2380765956_6313d8cae3.jpg
3197891333_b1b0fd1702.jpg
3119887967_271a097464.jpg
2276499757_b44dc6f8ce.jpg
```

In [40]:
```
print(type(train_file_data))
```

```
<class 'str'>
```

In [41]:
```
train = [row.split(".")[0] for row in train_file_data.split("\n")[:-1]]
print(train[:10])
```

```
['2513260012_03d33305cf', '2903617548_d3e38d7f88', '3338291921_fe7ae0c8f8', '488416045_1c6d903fe0', '26
44326817_8f45080b87', '218342358_1755a9cce1', '2501968935_02f2cd8079', '2699342860_5288e203ea', '263836
9467_8fc251595b', '2926786902_815a99a154']
```

In [42]:
```
test = [row.split(".")[0] for row in test_file_data.split("\n")[:-1]]
print(test[:10])
```

```
['3385593926_d3e9c21170', '2677656448_6b7e7702af', '311146855_0b65fdb169', '1258913059_07c613f7ff', '24
1347760_d44c8d3a01', '2654514044_a70a6e2c21', '2339106348_2df90aa6a9', '256085101_2c2617c5d0', '2807068
62_14c30d734a', '3072172967_630e9c69d0']
```

In [43]:
```
len(train), len(test)
```

Out[43]: (6000, 1000)

In [44]:
```
# prepare descriptions for the training data
# add start and end token to the training data
train_descriptions = {}
for img_id in train:
    train_descriptions[img_id] = []
    for cap in descriptions[img_id]:
        cap_to_append = "startseq " + cap + " endseq"
        train_descriptions[img_id].append(cap_to_append)
```

In [45]:
```
train_descriptions
```

Out[45]:
```
{'2513260012_03d33305cf': ['startseq black dog is running after white dog in the snow endseq',
  'startseq black dog chasing brown dog through snow endseq',
  'startseq two dogs chase each other across the snowy ground endseq',
  'startseq two dogs play together in the snow endseq',
  'startseq two dogs running through low lying body of water endseq'],
 '2903617548_d3e38d7f88': ['startseq little baby plays croquet endseq',
  'startseq little girl plays croquet next to truck endseq',
  'startseq the child is playing croquette by the truck endseq',
  'startseq the kid is in front of car with put and ball endseq',
  'startseq the little boy is playing with croquet hammer and ball beside the car endseq'],
 '3338291921_fe7ae0c8f8': ['startseq brown dog in the snow has something hot pink in its mouth endse
q',
  'startseq brown dog in the snow holding pink hat endseq',
  'startseq brown dog is holding pink shirt in the snow endseq',
  'startseq dog is carrying something pink in its mouth while walking through the snow endseq',
  'startseq dog with something pink in its mouth is looking forward endseq'],
 '488416045_1c6d903fe0': ['startseq brown dog is running along beach endseq',
  'startseq brown dog wearing black collar running across the beach endseq',
  'startseq dog walks on the sand near the water endseq',
  'startseq brown dog running on the beach endseq'
```

In [46]:
```
import pickle
with open("train_descriptions.pkl", "wb") as f:
    pickle.dump(train_descriptions, f)
```

TRANFER LEARNING

FEATURE EXTRACTION FROM IMAGES AND TEXT

STEP 1 - IMAGE FEATURE EXTRACTION

In [47]:
```
import keras
import numpy as np
```

```
Using TensorFlow backend.
```

In [48]:
```
from keras.applications.resnet50 import ResNet50
```

```
#Load the ResNet50 model
resnet_model = ResNet50(weights='imagenet',input_shape = (224,224,3))
```

```
resnet_model.summary()
```

```
resnet_model.save("resnet_model.hdf5")
```

In [49]:
```
from keras.models import load_model
```

In [50]:
```
resnet_model = load_model("resnet_model.hdf5")
```

```
C:\ANACONDA\lib\site-packages\keras\engine\saving.py:292: UserWarning: No training configuration found
in save file: the model was *not* compiled. Compile it manually.
  warnings.warn('No training configuration found in save file: '
```

In [51]:
```
resnet_model.summary()
```

```
__________________________________________________________________________________________________
res5c_branch2b (Conv2D)         (None, 7, 7, 512)    2359808     activation_47[0][0]
__________________________________________________________________________________________________
bn5c_branch2b (BatchNormalizati (None, 7, 7, 512)    2048        res5c_branch2b[0][0]
__________________________________________________________________________________________________
activation_48 (Activation)      (None, 7, 7, 512)    0           bn5c_branch2b[0][0]
__________________________________________________________________________________________________
res5c_branch2c (Conv2D)         (None, 7, 7, 2048)   1050624     activation_48[0][0]
__________________________________________________________________________________________________
bn5c_branch2c (BatchNormalizati (None, 7, 7, 2048)   8192        res5c_branch2c[0][0]
__________________________________________________________________________________________________
add_16 (Add)                    (None, 7, 7, 2048)   0           bn5c_branch2c[0][0]
                                                                 activation_46[0][0]
__________________________________________________________________________________________________
activation_49 (Activation)      (None, 7, 7, 2048)   0           add_16[0][0]
__________________________________________________________________________________________________
avg_pool (GlobalAveragePooling2 (None, 2048)         0           activation_49[0][0]
__________________________________________________________________________________________________
fc1000 (Dense)                  (None, 1000)         2049000     avg_pool[0][0]
==================================================================================================
```

In [52]:
```
from keras.models import Model
model_new = Model(resnet_model.input, resnet_model.layers[-2].output)
```

In [53]:
```
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.resnet50 import preprocess_input
```

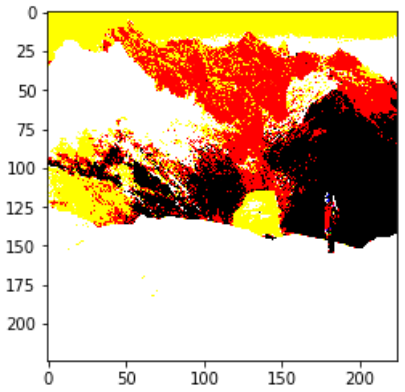
In [188]:
```
def preprocess_img(img):
    img = load_img(img, target_size=(224,224))
    img = img_to_array(img)
    img = np.expand_dims(img, axis = 0)
    img = preprocess_input(img)
    return img
```

In [55]:
```
im = "C:\ANACONDA\Scripts\MINOR PROJECT IMAGE CAPTIONING" + IMG_PATH+"56494233_1824005879.jpg"
```

In [56]:
```
img = preprocess_img(im)
plt.imshow(img[0])
# image after preprocessing
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for inte
gers).
```

Out[56]:
```
<matplotlib.image.AxesImage at 0x14666ba6400>
```



In [57]:
```
print(img)
```

```
[[[[115.061     14.221001 -56.68    ]
   [110.061     13.221001 -59.68    ]
   [112.061     12.221001 -58.68    ]
   ...
   [132.061     34.221    -42.68    ]
   [128.061     37.221    -45.68    ]
   [129.061     34.221    -44.68    ]]

  [[114.061     14.221001 -59.68    ]
   [117.061     17.221    -55.68    ]
   [118.061     16.221    -57.68    ]
   ...
   [138.061     36.221    -37.68    ]
   [135.061     40.221    -43.68    ]
   [134.061     40.221    -41.68    ]]

  [[115.061     17.221    -51.68    ]
   [120.061     22.221    -54.68    ]
   [118.061     20.221    -54.68    ]
```

In [189]:
```
def encode_image(img):
    img = preprocess_img(img)
    feature_vector = model_new.predict(img)
    feature_vector = feature_vector.reshape((-1,))
    #print(feature_vector.shape)
    return feature_vector
```

In [59]:
```
encode_image("C:\ANACONDA\Scripts\MINOR PROJECT IMAGE CAPTIONING" +IMG_PATH+"56494233_1824005879.jpg")
```

Out[59]:
```
array([0.7869591 , 0.6670176 , 0.01559329, ..., 0.        , 0.        ,
       0.        ], dtype=float32)
```

```
encoding_train = {} # will store the image ids with their corresponding feature vectors extracted from
resnet50
```

```
for ix, img_id in enumerate(train):
    img_path = "C:\ANACONDA\Scripts\MINOR PROJECT IMAGE CAPTIONING" + IMG_PATH + "/" + img_id + ".jpg"
    encoding_train[img_id] = encode_image(img_path)
```

```
len(encoding_train['2513260012_03d33305cf'])
# 2048 sized feature arrays for each image
```

```
# save features to disk
import pickle
with open("encoded_train_features.pkl", "wb") as f:
    pickle.dump(encoding_train, f)
```

In [ ]:
```
import pickle
```

In [62]:
```
encoding_train = pickle.load(open("encoded_train_features.pkl", "rb"))
```

In [63]:
```
len(encoding_train)
```

Out[63]: 6000

In [64]:
```
encoding_train
```

Out[64]:
```
{'2513260012_03d33305cf': array([0.28047287, 0.406874  , 0.08379951, ..., 0.91079134, 0.04341812,
        0.09577895], dtype=float32),
 '2903617548_d3e38d7f88': array([0.        , 0.00240233, 0.07777371, ..., 0.32226387, 0.33870834,
        0.82707775], dtype=float32),
 '3338291921_fe7ae0c8f8': array([0.6499905 , 0.45327264, 0.35990563, ..., 0.01184102, 0.17420349,
        0.04910808], dtype=float32),
 '488416045_1c6d903fe0': array([0.5911424 , 0.24315509, 0.        , ..., 1.1071402 , 0.44211772,
        0.41112903], dtype=float32),
 '2644326817_8f45080b87': array([0.0837353 , 0.55374855, 0.03665285, ..., 0.11103823, 1.0484872 ,
        0.        ], dtype=float32),
 '218342358_1755a9cce1': array([1.5750589 , 0.3306428 , 0.34105495, ..., 0.2440799 , 0.03760232,
        0.03851033], dtype=float32),
 '2501968935_02f2cd8079': array([0.08495852, 1.2882218 , 0.03415355, ..., 0.15635864, 0.5512884 ,
        1.3358811 ], dtype=float32),
 '2699342860_5288e203ea': array([0.34909672, 0.30768737, 0.10818278, ..., 0.65115106, 0.        ,
        0.05328921], dtype=float32),
 '2638369467_8fc251595b': array([1.3183991 , 0.00438301, 0.23403354, ..., 0.8236048 , 0.06217944,
        0.86687243], dtype=float32),
 '2926786902_815a99a154': array([1.0064671 , 0.56222415, 1.9197018 , ..., 1.1765666 , 0.4799578 ,
```

```
# extract features for test images also
encoding_test = {}
for ix, img_id in enumerate(test):
    img_path = "C:\ANACONDA\Scripts\MINOR PROJECT IMAGE CAPTIONING" + IMG_PATH + "/" + img_id + ".jpg"
    encoding_test[img_id] = encode_image(img_path)
```

```
with open("encoded_test_features.pkl", "wb") as f:
    pickle.dump(encoding_test, f)
```

In [65]:
```
encoding_test = pickle.load(open("encoded_test_features.pkl", "rb"))
```

In [66]:
```
len(encoding_test)
```

Out[66]: 1000

In [67]:
```
encoding_test
```

Out[67]:
```
{'3385593926_d3e9c21170': array([0.28236082, 0.31681818, 0.04513453, ..., 0.7442413 , 0.2965144 ,
        0.9206255 ], dtype=float32),
 '2677656448_6b7e7702af': array([0.23350716, 0.05166651, 0.6242703 , ..., 0.00522086, 0.2621777 ,
        0.08686393], dtype=float32),
 '311146855_0b65fdb169': array([0.00912151, 0.07213554, 0.1220706 , ..., 0.02203188, 1.131882  ,
        0.03855705], dtype=float32),
 '1258913059_07c613f7ff': array([0.02427822, 1.2347254 , 0.07595192, ..., 0.08897056, 0.09812839,
        1.938419  ], dtype=float32),
 '241347760_d44c8d3a01': array([0.05051163, 6.319989  , 0.312008  , ..., 0.0537944 , 0.01552991,
        0.02812621], dtype=float32),
 '2654514044_a70a6e2c21': array([1.7663001 , 0.03384983, 0.10334536, ..., 0.00532028, 0.6680157 ,
        0.39294168], dtype=float32),
 '2339106348_2df90aa6a9': array([0.06683169, 1.0869431 , 0.07896071, ..., 0.01411186, 0.13114332,
        0.09507984], dtype=float32),
 '256085101_2c2617c5d0': array([0.5742952 , 0.51020306, 0.04079671, ..., 0.3325456 , 0.0211818 ,
        0.19905935], dtype=float32),
 '280706862_14c30d734a': array([0.47258455, 1.0220349 , 0.32351598, ..., 0.37484726, 0.05683207,
        0.15788071], dtype=float32),
 '3072172967_630e9c69d0': array([0.6180927 , 1.7388422 , 0.05182058, ..., 0.7286484 , 1.1886153 ,
```

DATA LOADER

In [68]:
```
from keras.preprocessing.sequence import pad_sequences
```

In [69]:
```
# generator remembers the state of the function in the previous call
def data_generator(train_descriptions, encoding_train, word_to_idx, max_len, batch_size):
    X1, X2, y = [], [], []
    n = 0
    while True:
        for key, desc_list in train_descriptions.items():
            n += 1
            photo = encoding_train[key]
            for desc in desc_list:
                seq = [word_to_idx[word] for word in desc.split() if word in word_to_idx]
                for i in range(1, len(seq)):
                    xi = seq[0:i]
                    yi = seq[i]
                    xi = pad_sequences([xi], maxlen = max_len, value = 0, padding= 'post')[0] # value=0 d
                    yi = to_categorical([yi], num_classes = vocab_size)[0] # one hot encoding of the word

                    # make a mini batch
                    X1.append(photo)
                    X2.append(xi)
                    y.append(yi)

                    if n == batch_size:
                        yield ([np.array(X1), np.array(X2)], np.array(y))

                        # prepare for next iteration
                        X1, X2, y = [],[],[]
                        n = 0
```

PREPROCESSING FOR CAPTIONS

In [70]:
```
len(total_words)
```

Out[70]:
```
1845
```

In [71]:
```
word_to_idx = {}
idx_to_word = {}
for i, word in enumerate(total_words):
    word_to_idx[word] = i+1
    idx_to_word[i+1] = word
```

In [72]:
```
word_to_idx['dog']
```

Out[72]: 6

In [73]:
```
idx_to_word[6]
```

Out[73]: 'dog'

In [74]:
```
print(len(idx_to_word))
```

```
1845
```

In [75]:
```
idx_to_word[1846] = 'startseq'
word_to_idx['startseq'] = 1846
```

In [76]:
```
idx_to_word[1847] = 'endseq'
word_to_idx['endseq'] = 1847
```

In [77]:
```
print(len(idx_to_word))
```

```
1847
```

In [78]:
```
vocab_size = len(word_to_idx) + 1
print(vocab_size)
```

```
1848
```

In [79]:
```
# find out the length of longest sentence
max_len = 0
for key in train_descriptions.keys():
    for cap in train_descriptions[key]:
        max_len = max(max_len,len(cap.split()))
```

In [80]:
```
max_len
```

Out[80]: 35

In [81]:
```
with open("word_to_idx", "wb") as f:
    pickle.dump(word_to_idx, f)
```

TEXT FEATURE EXTRACTION USING TRANSFER LEARNING (GLOVE EMBEDDINGS)

In [82]:
```
f = open("glove.6B.50d.txt", encoding = "utf8")
```

In [83]:
```
# create a dictionary mapping each word in the glove embedding text file to its corresponding glove embedd
embedding_index = {}
```

In [84]:
```
for line in  f:
    values = line.split()
    word = values[0]
    word_embedding = np.array(values[1:], dtype = 'float')
    embedding_index[word] = word_embedding
f.close()
```

In [85]:
```
embedding_index['house']
```

Out[85]:
```
array([ 0.60137  ,  0.28521  , -0.032038 , -0.43026  ,  0.74806  ,
        0.26223  , -0.97361  ,  0.078581 , -0.57588  , -1.188    ,
       -1.8507   , -0.24887  ,  0.055549 ,  0.0086155,  0.067951 ,
        0.40554  , -0.073998 , -0.21318  ,  0.37167  , -0.71791  ,
        1.2234   ,  0.35546  , -0.41537  , -0.21931  , -0.39661  ,
       -1.7831   , -0.41507  ,  0.29533  , -0.41254  ,  0.020096 ,
        2.7425   , -0.9926   , -0.71033  , -0.46813  ,  0.28265  ,
       -0.077639 ,  0.3041   , -0.06644  ,  0.3951   , -0.70747  ,
       -0.38894  ,  0.23158  , -0.49508  ,  0.14612  , -0.02314  ,
        0.56389  , -0.86188  , -1.0278   ,  0.039922 ,  0.20018  ])
```

In [86]:
```
def get_embedding_matrix():
    emb_dim = 50
    matrix = np.zeros((vocab_size, emb_dim))
    for word, idx in word_to_idx.items():
        embedding_vector = embedding_index.get(word)
        if embedding_vector is not None: # if word is not in glove.txt file, take it as all zeros
            matrix[idx] = embedding_vector
    return matrix
```

In [87]:
```
embedding_matrix = get_embedding_matrix()
```

In [88]:
```
embedding_matrix.shape
```

Out[88]:
```
(1848, 50)
```

In [89]:
```
embedding_matrix[1847], embedding_matrix[1846] # glove vector for startseq and endseg - all 0s
```

Out[89]:
```
(array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]))
```

In [90]:
```
type(embedding_matrix)
```

Out[90]:
```
numpy.ndarray
```

In [91]:
```
with open("embedding_matrix.pkl", "wb") as f:
    pickle.dump(embedding_matrix, f)
```

MODEL ARCHITECTURE

In [92]:
```
from keras.layers import Input, Dense, Dropout, Embedding, LSTM, Add
from keras.models import Model
```

In [93]:
```
vocab_size
```

Out[93]:
```
1848
```

In [94]:
```
input_img_features = Input(shape=(2048,))
inp_img1 = Dropout(0.3)(input_img_features)
inp_img2 = Dense(256, activation='relu')(inp_img1)
```

In [95]:
```
input_captions = Input(shape=(max_len,))
inp_cap1 = Embedding(input_dim = vocab_size, output_dim = 50, mask_zero = True)(input_captions)
inp_cap2 = Dropout(0.3)(inp_cap1)
inp_cap3 = LSTM(256)(inp_cap2)
```

In [96]:
```
decoder1 = Add()([inp_img2, inp_cap3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation = 'softmax')(decoder2)
```

In [97]:
```
model = Model(inputs = [input_img_features, input_captions], outputs=outputs) # combined model
```

In [98]:
```
model.summary()
```

```
_____________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=============================================================================================
input_2 (InputLayer)         (None, 35)                0
_____________________________________________________________________________________________
input_1 (InputLayer)         (None, 2048)              0
_____________________________________________________________________________________________
embedding_1 (Embedding)      (None, 35, 50)            92400       input_2[0][0]
_____________________________________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0           input_1[0][0]
_____________________________________________________________________________________________
dropout_2 (Dropout)          (None, 35, 50)            0           embedding_1[0][0]
_____________________________________________________________________________________________
dense_1 (Dense)              (None, 256)               524544      dropout_1[0][0]
_____________________________________________________________________________________________
lstm_1 (LSTM)                (None, 256)               314368      dropout_2[0][0]
_____________________________________________________________________________________________
add_1 (Add)                  (None, 256)               0           dense_1[0][0]
                                                                   lstm_1[0][0]
_____________________________________________________________________________________________
dense_2 (Dense)              (None, 256)               65792       add_1[0][0]
_____________________________________________________________________________________________
dense_3 (Dense)              (None, 1848)              474936      dense_2[0][0]
=============================================================================================
Total params: 1,472,040
Trainable params: 1,472,040
Non-trainable params: 0
_____________________________________________________________________________________________
```

In [99]:
```
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False
```

In [100]:
```
model.compile(loss = "categorical_crossentropy", optimizer= "adam")
```

TRAINING OF MODEL

In [101]:
```
epochs = 20
batch_size = 3
steps = len(train_descriptions)//batch_size
from keras.utils.np_utils import to_categorical
```

In [102]:
```
len(train_descriptions)
```

Out[102]: 6000

In [103]:
```
def train():
    for i in range(epochs):
        generator = data_generator(train_descriptions, encoding_train, word_to_idx, max_len, batch_size) 
        model.fit_generator(generator, epochs = 1, steps_per_epoch = steps, verbose=1)
        model.save('./model_weights/model_' + str(i) + '.h5')
```

```
train() # trained in google colab
```

In [104]:
```
model = load_model('./model_weights/model_19.h5')
```

In [105]:
```
model
```

Out[105]:
```
<keras.engine.training.Model at 0x14615faa198>
```

PREDICTIONS

In [106]:
```
def predict_caption(photo):
    in_text = "startseq"
    for i in range(max_len):
        sequence = [word_to_idx[w] for w in in_text.split() if w in word_to_idx]
        sequence = pad_sequences([sequence], maxlen = max_len, padding = 'post')
        ypred = model.predict([photo, sequence])
        ypred = ypred.argmax()
        word = idx_to_word[ypred]
        in_text += (' '+ word)
        if word == 'endseq':
            break
    final_caption = in_text.split()[1:-1]
    final_caption = ' '.join(final_caption)
    return final_caption
```

In [109]:
```
idx = np.random.randint(0, 1000)
img_name = list(encoding_test.keys())[idx]
photo_2048 = encoding_test[img_name].reshape((1, 2048))
i = plt.imread("Flickr_Data/Images/" + img_name +'.jpg')
caption = predict_caption(photo_2048)
print(caption)
plt.axis("off")
plt.imshow(i)
```

```
man rides his bike on dirt path
```

Out[109]:
```
<matplotlib.image.AxesImage at 0x14666a20e80>
```

In [157]:
```
img_name = list(encoding_test.keys())[5]
photo_2048 = encoding_test[img_name].reshape((1, 2048))
i = plt.imread("Flickr_Data/Images/" + img_name +'.jpg')
caption = predict_caption(photo_2048)
print(caption)
plt.axis("off")
plt.imshow(i)
```

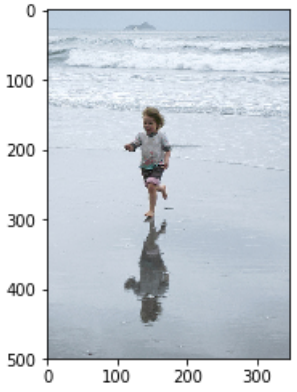
```
dog running through the grass
```

Out[157]:
```
<matplotlib.image.AxesImage at 0x1466690b978>
```

In [168]:
```
img_name = list(encoding_test.keys())[110]
photo_2048 = encoding_test[img_name].reshape((1, 2048))
i = plt.imread("Flickr_Data/Images/" + img_name +'.jpg')
caption = predict_caption(photo_2048)
print(caption)
plt.axis("off")
plt.imshow(i)
```

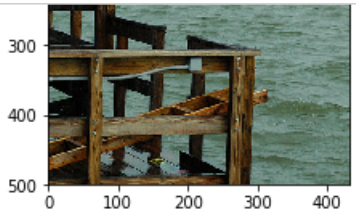
```
boy in wetsuit is running on the beach
```

Out[168]:
```
<matplotlib.image.AxesImage at 0x1466697ab38>
```



In [149]:
```
def display_img_and_caption(img_number):
    img_name = list(encoding_test.keys())[img_number]
    photo_2048 = encoding_test[img_name].reshape((1, 2048))
    i = plt.imread("Flickr_Data/Images/" + img_name +'.jpg')
    caption = predict_caption(photo_2048)
    print(caption)
    plt.axis("off")
    plt.imshow(i)
    plt.show()
```

In [151]:
```
for i in range(0, 100):
    display_img_and_caption(i)
```



boy in red shirt is jumping off tire swing

In [ ]:
```
# end
```

In [ ]:
```
# Load and test on images
```

In [178]:
```
import cv2
```

In [209]:
```
def encode_image_2(img):
    img = preprocess_img_2(img)
    feature_vector = model_new.predict(img)
    feature_vector = feature_vector.reshape((-1,))
    #print(feature_vector.shape)
    return feature_vector
def preprocess_img_2(img):
    #img = load_img(img, target_size=(224,224))
    #img = img_to_array(img)
    img = np.expand_dims(img, axis = 0)
    img = preprocess_input(img)
    return img
```

In [221]:
```
def load_and_predict(im): # takes as input the image name
    name = im
    im = plt.imread(name + ".jpg")
    im = cv2.resize(im, (224,224))
    im = encode_image_2(im)
    photo_2048 = im.reshape((1, 2048))
    i = plt.imread(name+'.jpg')
    caption = predict_caption(photo_2048)
    print(caption)
    plt.axis("off")
    plt.imshow(i)
```

In [234]:
```
load_and_predict("pic3")
```

```
boy in yellow shirt is wakeboarding on large ski
```

In [235]:
```
load_and_predict("pic4")
```

```
two men in uniforms are playing soccer
```

In [ ]:
```
# end
```