In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import keras
import re
import nltk
from nltk.corpus import stopwords
import string
import json
from time import time
import pickle
from keras.applications.vgg16 import VGG16
from keras.applications.resnet50 import ResNet50, preprocess_input, decode_predictions
from keras.preprocessing import image
from keras.models import Model, load_model
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import Input, Dense, Dropout, Embedding, LSTM
from keras.layers.merge import add
```

In [3]:

```

with open(r"C:\Users...Flickr_Data\Flickr_Data\Flickr_TextData\Flickr8k.token.txt") as filepath:
    captions = filepath.read()
    filepath.close()
```

In [4]:

```
captions = captions.split("\n")[:-1]
```

In [5]:
```
len(captions)
```

```
40460
```

In [6]:
```

descriptions = {}

for ele in captions:
    i_to_c = ele.split("\t")
    img_name = i_to_c[0].split(".")[0]
    cap = i_to_c[1]

    if descriptions.get(img_name) == None:
        descriptions[img_name] = []

    descriptions[img_name].append(cap)
```

In [7]:
```
descriptions['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 .']
```

In [8]:

```
""" 1. lower each word
    2. remove puntuations
    3. remove words less than length 1 """

def clean_text(sample):
    sample = sample.lower()

    sample = re.sub("[^a-z]+"," ",sample)

    sample = sample.split()

    sample = [s for s in sample if len(s)>1]

    sample = " ".join(sample)

    return sample
```

In [10]:

```

for key, desc_list in descriptions.items():
    for i in range(len(desc_list)):
        desc_list[i] = clean_text(desc_list[i])
```

In [11]:

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

```
['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 [12]:

```
f = open("descriptions.txt","w")
f.write( str(descriptions) )
f.close()
```

In [14]:

```
f = open("descriptions.txt", 'r')
descriptions = f.read()
f.close()

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

In [15]:

```
vocabulary = set()

for key in descriptions.keys():
    [vocabulary.update(i.split()) for i in descriptions[key]]

print('Vocabulary Size: %d' % len(vocabulary))
```

```
Vocabulary Size: 8424
```

In [16]:

```
all_vocab =  []

for key in descriptions.keys():
    [all_vocab.append(i) for des in descriptions[key] for i in des.split()]

print('Vocabulary Size: %d' % len(all_vocab))
print(all_vocab[:15])
```

```
Vocabulary Size: 373837
['child', 'in', 'pink', 'dress', 'is', 'climbing', 'up', 'set', 'of', 'stairs', 'in', 'an', 'entry', 'way', 'girl']
```

In [17]:
```
import collections


counter= collections.Counter(all_vocab)

dic_ = dict(counter)

threshelod_value = 10

sorted_dic = sorted(dic_.items(), reverse=True, key = lambda x: x[1])
sorted_dic = [x for x in sorted_dic if x[1]>threshelod_value]
all_vocab = [x[0] for x in sorted_dic]
```

In [18]:
```
len(all_vocab)
```

```
1845
```

In [20]:
```
# TrainImagesFile
f = open(r"C:\Users...Flickr_Data\Flickr_Data\Flickr_TextData\Flickr_8k.trainImages.txt")
train = f.read()
f.close()
```

In [21]:
```
train  = [e.split(".")[0] for e in train.split("\n")[:-1]]
```

In [24]:
```
1
2
3  train_descriptions = {}
4
5  for t in train:
6      train_descriptions[t] = []
7      for cap in descriptions[t]:
8          cap_to_append = "startseq " + cap + " endseq"
9          train_descriptions[t].append(cap_to_append)
```

In [25]:
```
1  train_descriptions['1000268201_693b08cb0e']
```

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

In [26]:
```
1  model = ResNet50(weights="imagenet", input_shape=(224,224,3))
```

In [27]:
```
1 model.summary()
```

```
Model: "resnet50"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, 112, 112, 64) 9472        conv1_pad[0][0]
__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 112, 112, 64) 256         conv1_conv[0][0]
__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 112, 112, 64) 0           conv1_bn[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0           conv1_relu[0][0]
__________________________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, 56, 56, 64)   0           pool1_pad[0][0]
__________________________________________________________________________________________________
conv2_block1_1_conv (Conv2D)    (None, 56, 56, 64)   4160        pool1_pool[0][0]
__________________________________________________________________________________________________
conv2_block1_1_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block1_1_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_1_relu (Activation (None, 56, 56, 64)   0           conv2_block1_1_bn[0][0]
```

In [28]:
```
1 model_new = Model(model.input, model.layers[-2].output)
```

In [42]:
```
1 images = "..../flicker8k-dataset/Flickr8k_Dataset/"
```

In [29]:

```
def preprocess_image(img):
    img = image.load_img(img, target_size=(224,224))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    return img
```

In [30]:

```
def encode_image(img):
    img = preprocess_image(img)
    feature_vector = model_new.predict(img)
    feature_vector = feature_vector.reshape(feature_vector.shape[1],)
    return feature_vector
```

In [259]:

```
start = time()

encoding_train = {}

for ix, img in enumerate(train):

    img = "..../flicker8k-dataset/Flickr8k_Dataset/{}.jpg".format(train[ix])
    encoding_train[img[len(images):]] = encode_image(img)

    if ix%100==0:
        print("Encoding image- "+ str(ix))

print("Time taken in seconds =", time()-start)
```

```
Encoding image- 0
Encoding image- 100
Encoding image- 200
Encoding image- 300
Encoding image- 400
Encoding image- 500
Encoding image- 600
Encoding image- 700
Encoding image- 800
Encoding image- 900
Encoding image- 1000
Encoding image- 1100
Encoding image- 1200
Encoding image- 1300
Encoding image- 1400
Encoding image- 1500
Encoding image- 1600
Encoding image- 1700
Encoding image- 1800
Encoding image- 1900
```

```
Encoding image- 2000
Encoding image- 2100
Encoding image- 2200
Encoding image- 2300
Encoding image- 2400
Encoding image- 2500
Encoding image- 2600
Encoding image- 2700
Encoding image- 2800
Encoding image- 2900
Encoding image- 3000
Encoding image- 3100
Encoding image- 3200
Encoding image- 3300
Encoding image- 3400
Encoding image- 3500
Encoding image- 3600
Encoding image- 3700
Encoding image- 3800
Encoding image- 3900
Encoding image- 4000
Encoding image- 4100
Encoding image- 4200
Encoding image- 4300
Encoding image- 4400
Encoding image- 4500
Encoding image- 4600
Encoding image- 4700
Encoding image- 4800
Encoding image- 4900
Encoding image- 5000
Encoding image- 5100
Encoding image- 5200
Encoding image- 5300
Encoding image- 5400
Encoding image- 5500
Encoding image- 5600
Encoding image- 5700
Encoding image- 5800
Encoding image- 5900
Time taken in seconds = 1818.5798392295837
```

In [260]:
```
1
2 with open("..../storage/encoded_train_images.pkl", "wb") as encoded_pickle:
3     pickle.dump(encoding_train, encoded_pickle)
```

In [31]:
```
1 import pickle as pkl
2 with open("encodin_train_features.pkl","rb") as f:
3     encoding_train=pkl.load(f)
```

In [253]:

```
start = time()

encoding_test = {}

for ix, img in enumerate(test):

    img = "..../flicker8k-dataset/Flickr8k_Dataset/{}.jpg".format(test[ix])
    encoding_test[img[len(images):]] = encode_image(img)

    if ix%100==0:
        print("Encoding image- "+ str(ix))

print("Time taken in seconds =", time()-start)
```

```
Encoding image- 0
Encoding image- 100
Encoding image- 200
Encoding image- 300
Encoding image- 400
Encoding image- 500
Encoding image- 600
Encoding image- 700
Encoding image- 800
Encoding image- 900
Time taken in seconds = 303.322877407074
```

In [258]:

```

with open("..../encoded_test_images.pkl", "wb") as encoded_pickle:
    pickle.dump(encoding_test, encoded_pickle)
```

In [71]:
```
with open("encoded_test_features.pkl","rb") as f:
    encoding_test=pkl.load(f)
```

In [15]:
```

with open("..../encoded_train_images.pkl", "rb") as encoded_pickle:
    encoding_train = pickle.load(encoded_pickle)
```

In [ ]:
```

with open("..../storage/encoded_test_features.pkl", "rb") as encoded_pickle:
    encoding_test = pickle.load(encoded_pickle)
```

In [72]:
```
1 encoding_test
```

```
{'3385593926_d3e9c21170': array([0.2823609 , 0.31681862, 0.04513445, ..., 0.74424076, 0.29651454,
       0.920625  ], dtype=float32),
 '2677656448_6b7e7702af': array([0.23350658, 0.05166636, 0.6242709 , ..., 0.00522089, 0.26217806,
       0.08686365], dtype=float32),
 '311146855_0b65fdb169': array([0.00912154, 0.0721353 , 0.1220707 , ..., 0.02203191, 1.1318818 ,
       0.03855684], dtype=float32),
 '1258913059_07c613f7ff': array([0.02427815, 1.2347251 , 0.07595173, ..., 0.08897065, 0.09812832,
       1.9384186 ], dtype=float32),
 '241347760_d44c8d3a01': array([0.0505117 , 6.3199897 , 0.3120082 , ..., 0.05379438, 0.01552999,
       0.02812625], dtype=float32),
 '2654514044_a70a6e2c21': array([1.7662996 , 0.03384979, 0.10334545, ..., 0.00532009, 0.6680156 ,
       0.39294165], dtype=float32),
 '2339106348_2df90aa6a9': array([0.06683154, 1.0869427 , 0.07896088, ..., 0.01411188, 0.1311434 ,
       0.09507965], dtype=float32),
 '256085101_2c2617c5d0': array([0.5742956 , 0.51020324, 0.04079673, ..., 0.3325452 , 0.02118182,
       0.19905947], dtype=float32),
 '280706862_14c30d734a': array([0.4725839 , 1.022034  , 0.3235157 , ..., 0.37484682, 0.05683213,
       0.15788084], dtype=float32),
 '3072172967_630e9c69d0': array([0.6180923 , 1.7388427 , 0.05182064, ..., 0.72864807, 1.188615  ,
       0.46246415], dtype=float32),
 '3482062809_3b694322c4': array([0.3330196 , 0.6153882 , 0.04577607, ..., 0.08085107, 0.98302686,
       0.37899324], dtype=float32),
 '1167669558_87a8a467d6': array([0.29192945, 0.05319565, 0.        , ..., 0.29094884, 0.35756508,
```

In [33]:

```
ix = 1
word_to_idx = {}
idx_to_word = {}

for e in all_vocab:
    word_to_idx[e] = ix
    idx_to_word[ix] = e
    ix +=1
```

In [34]:

```
word_to_idx['startseq'] = 1846
word_to_idx['endseq'] = 1847

idx_to_word[1846] = 'startseq'
idx_to_word[1847] = 'endseq'
```

In [35]:

```
vocab_size = len(idx_to_word)+1
print(vocab_size)
```

```
1848
```

In [36]:

```
all_captions_len = []

for key in train_descriptions.keys():
    for cap in train_descriptions[key]:
        all_captions_len.append(len(cap.split()))

max_len = max(all_captions_len)
print(max_len)
```

```
35
```

In [54]:

```
def data_generator(train_descriptions, encoding_train, word_to_idx, max_len, num_photos_per_batch):

    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)):

                    in_seq = seq[0:i]
                    out_seq = seq[i]

                    in_seq = pad_sequences([in_seq], maxlen=max_len, value=0, padding='post')[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:
                u = np.array(X1)
                v = np.array(X2)
                w = np.array(y)
                yield ([u,v],w)
                X1, X2, y = [], [], []
                n=0
```

In [39]:

```
f = open("glove_6B_50d.txt", encoding='utf8')
```

In [40]:

```
embedding_index = {}

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype="float")

    embedding_index[word] = coefs

f.close()
```

In [41]:
```
def get_embedding_output():

    emb_dim = 50
    embedding_output = 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:
            embedding_output[idx] = embedding_vector

    return embedding_output


embedding_output = get_embedding_output()
```

In [42]:
```
embedding_output.shape
```

```
(1848, 50)
```

In [43]:
```

input_img_fea = Input(shape=(2048,))
inp_img1 = Dropout(0.3)(input_img_fea)
inp_img2 = Dense(256, activation='relu')(inp_img1)
```

In [44]:

```
input_cap = Input(shape=(max_len,))
inp_cap1 = Embedding(input_dim=vocab_size, output_dim=50, mask_zero=True)(input_cap)
inp_cap2 = Dropout(0.3)(inp_cap1)
inp_cap3 = LSTM(256)(inp_cap2)
```

In [45]:

```
decoder1 = add([inp_img2 , inp_cap3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[input_img_fea, input_cap], outputs=outputs)
```

In [46]:
```
model.summary()
```

```
Model: "functional_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 35)]         0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 2048)]       0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 35, 50)       92400       input_3[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 2048)         0           input_2[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 35, 50)       0           embedding[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          524544      dropout[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 256)          314368      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, 1848)         474936      dense_1[0][0]
==================================================================================================
Total params: 1,472,040
Trainable params: 1,472,040
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [47]:
```
model.layers[2].set_weights([embedding_output])
model.layers[2].trainable = False
```

In [48]:
```
model.compile(loss="categorical_crossentropy", optimizer="adam")
```

In [49]:
```
epochs = 10
number_pics_per_bath = 3
steps = len(train_descriptions)//number_pics_per_bath
```

In [ ]:
```
for i in range(epochs):
    generator = data_generator(train_descriptions, encoding_train, word_to_idx, max_len, number_pics_pe
    model.fit_generator(generator, epochs=10, steps_per_epoch=steps, verbose=1)
    model.save('..../model_weights/model_' + str(i) + '.h5')
```

In [57]:
```
import tensorflow as tf
tf.config.experimental_run_functions_eagerly(True)
```

```
WARNING:tensorflow:From <ipython-input-57-cef63d6acc25>:2: experimental_run_functions_eagerly (from tensorflow.python.eager.def_function)
is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.config.run_functions_eagerly` instead of the experimental version.
```

In [59]:
```
model = load_model("model_9.h5")
```

In [60]:

```
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()
    final_caption = final_caption[1:-1]
    final_caption = ' '.join(final_caption)

    return final_caption
```

In [ ]:

```

```

In [61]:

```
img_path='C://Users...Flickr_Data//Flickr_Data//Images//'
```

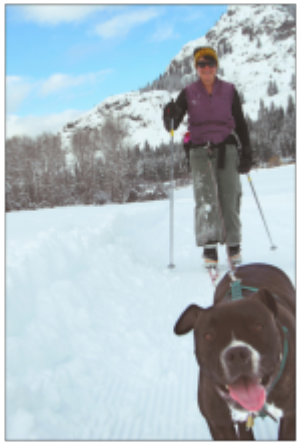
In [74]:

```
for i in range(20):
    rn =  np.random.randint(0, 1000)
    img_name = list(encoding_test.keys())[rn]
    photo = encoding_test[img_name].reshape((1,2048))

    i = plt.imread(img_path+img_name+".jpg")
    plt.imshow(i)
    plt.axis("off")
    plt.show()

    caption = predict_caption(photo)

    print(caption)
```

```
man in blue shirt and jeans is standing in front of some people
```

```
man on bike is riding his bike through the woods
```

```
two children are sitting in bed
```



```
black dog is running through the snow
```

```
two people are walking on the beach
```

```
man in black shirt and jeans is walking past the street
```

```
man in blue shirt and black shorts is jumping over rock
```

```
girl in purple shirt is holding up her nose
```

```
man in red shirt and black hat is standing in front of crowd
```

```
surfer in wetsuit is riding wave
```

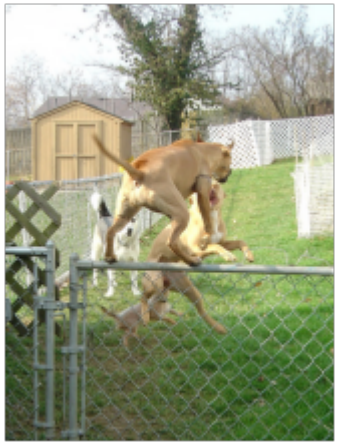

```
two men in white and white are playing basketball
```

```
man in green shirt and jeans is jumping into the air
```

```
woman in black shirt and tie and black hat is standing in front of crowd
```

```
man with glasses and woman in black
```

```
woman with camera is wearing black bandanna
```



```
dog jumping over hurdle
```

```
two dogs are running on the beach
```


```
group of people are standing in front of some adults
```

```
boy in red shirt and blue jeans is holding the wheel of another boy in blue shirt
```

In [77]:

```
img_name = list(encoding_test.keys())[2]
photo = encoding_test[img_name].reshape((1,2048))

i = plt.imread(img_path+img_name+".jpg")
plt.imshow(i)
plt.axis("off")
plt.show()

caption = predict_caption(photo)
description
print(caption)
```

in black shirt and jeans is sitting on bench in front of some houses

two girls in red and white dresses are playing with fireworks

In [81]:
```
import nltk
BLEUscore = nltk.translate.bleu_score.sentence_bleu([reference], hypothesis)
print(BLEUscore)
```

```
0.1899939474068717
```

In [83]:
```
for i in range(5):
    reference=descriptions[img_name][i]
    hypothesis=caption
    BLEUscore += nltk.translate.bleu_score.sentence_bleu([reference], hypothesis)
print(BLEUscore/5)
```

```
0.28111219733146703
```

In [89]:
```
reference2=reference.split()
print(reference2)
```

```
['people', 'some', 'dressed', 'in', 'costumes', 'and', 'dogs', 'on', 'snowy', 'mountain']
```

In [110]:

```
for i in range(20):
    rn =  np.random.randint(0, 1000)
    img_name = list(encoding_test.keys())[rn]
    photo = encoding_test[img_name].reshape((1,2048))

    i = plt.imread(img_path+img_name+".jpg")
    plt.imshow(i)
    plt.axis("off")
    plt.show()

    caption = predict_caption(photo)

    BLEUscore=0
    reference1=descriptions[img_name][0].split()
    reference2=descriptions[img_name][1].split()
    reference3=descriptions[img_name][2].split()
    reference4=descriptions[img_name][3].split()
    reference5=descriptions[img_name][4].split()
    hypothesis=caption.split()
    BLEUscore1 = nltk.translate.bleu_score.corpus_bleu([[reference1,reference2,reference3,reference4,re
    BLEUscore2 = nltk.translate.bleu_score.corpus_bleu([[reference1,reference2,reference3,reference4,re
    BLEUscore3 = nltk.translate.bleu_score.corpus_bleu([[reference1,reference2,reference3,reference4,re
    BLEUscore4 = nltk.translate.bleu_score.corpus_bleu([[reference1,reference2,reference3,reference4,re

    print("Cumulative 1-gram:%.3f"%BLEUscore1)
    print("Cumulative 2-gram:%.3f"%(BLEUscore2))
    print(descriptions[img_name][0])
    print(caption)
```

```
Cumulative 1-gram:0.636
Cumulative 2-gram:0.437
man holds flag next to snowbound campsite
man in red and white coat is standing in the snow
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```