

```
!wget https://github.com/jbrownlee/Datasets/releases/download/Flickr8K/Flickr8K_Dataset.zip
!wget https://github.com/jbrownlee/Datasets/releases/download/Flickr8K/Flickr8K_text.zip

--2021-07-24 08:20:53-- https://github.com/jbrownlee/Datasets/releases/download/Flickr8K/Flickr8K_Dataset.zip
Resolving github.com (github.com)... 140.82.112.3
Connecting to github.com (github.com):140.82.112.3... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-releases.githubusercontent.com/124585957/47f52b80-3501-11e9-8f49-4515a2a339b0?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20210724%2Fus-east-1%2F%3A%2Faws4_request&X-Amz-Date=20210724T082053Z&X-Amz-Expires=300&X-Amz-Signature=2c58f1492454b9700861efaf3d77ddf63c2bb6fb3819eb52c1077ebeb0306944&X-Amz-SignedHeaders=host&actor_id=&key_id=&repo_id=124585957&response-content-disposition=attachment;filename=Flickr8K_Dataset.zip&response-content-type=application%2Foctet-stream [following]
--2021-07-24 08:20:53-- https://github-releases.githubusercontent.com/124585957/47f52b80-3501-11e9-8f49-4515a2a339b0?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20210724%2Fus-east-1%2F%3A%2Faws4_request&X-Amz-Date=20210724T082053Z&X-Amz-Expires=300&X-Amz-Signature=2c58f1492454b9700861efaf3d77ddf63c2bb6fb3819eb52c1077ebeb0306944&X-Amz-SignedHeaders=host&actor_id=&key_id=&repo_id=124585957&response-content-disposition=attachment;filename=Flickr8K_Dataset.zip&response-content-type=application%2Foctet-stream
Resolving github-releases.githubusercontent.com (github-releases.githubusercontent.com)... 185.199.108.154, 185.199.109.154, 185.199.110.154, ...
Connecting to github-releases.githubusercontent.com (github-releases.githubusercontent.com):185.199.108.154:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1115419746 (1.0G) [application/octet-stream]
Saving to: 'Flickr8K_Dataset.zip'

Flickr8K_Dataset.zi 100%[=====] 1.04G 50.4MB/s in 23s

--2021-07-24 08:21:15 (46.9 MB/s) - 'Flickr8K_Dataset.zip' saved [1115419746/1115419746]

--2021-07-24 08:21:15-- https://github.com/jbrownlee/Datasets/releases/download/Flickr8K/Flickr8K_text.zip
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com):140.82.113.4:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-releases.githubusercontent.com/124585957/47f52b80-3501-11e9-8d2e-dd69a21a4362?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20210724%2Fus-east-1%2F%3A%2Faws4_request&X-Amz-Date=20210724T082115Z&X-Amz-Expires=300&X-Amz-Signature=cd7328624cae7b8b9058019b11ad766bda77a1c8b90eb5a22c66a386758464&X-Amz-SignedHeaders=host&actor_id=&key_id=&repo_id=124585957&response-content-disposition=attachment;filename=Flickr8K_text.zip&response-content-type=application%2Foctet-stream [following]
--2021-07-24 08:21:16-- https://github-releases.githubusercontent.com/124585957/47f52b80-3501-11e9-8d2e-dd69a21a4362?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20210724%2Fus-east-1%2F%3A%2Faws4_request&X-Amz-Date=20210724T082115Z&X-Amz-Expires=300&X-Amz-Signature=cd7328624cae7b8b9058019b11ad766bda77a1c8b90eb5a22c66a386758464&X-Amz-SignedHeaders=host&actor_id=&key_id=&repo_id=124585957&response-content-disposition=attachment;filename=Flickr8K_text.zip&response-content-type=application%2Foctet-stream
Resolving github-releases.githubusercontent.com (github-releases.githubusercontent.com)... 185.199.108.154, 185.199.109.154, 185.199.110.154, ...
Connecting to github-releases.githubusercontent.com (github-releases.githubusercontent.com):185.199.108.154:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2340801 (2.3M) [application/octet-stream]
Saving to: 'Flickr8K_text.zip'

Flickr8K_text.zip 100%[=====] 2.23M --.-KB/s in 0.06s

2021-07-24 08:21:16 (38.7 MB/s) - 'Flickr8K_text.zip' saved [2340801/2340801]

In [ ]:
!unzip Flickr8K_Dataset.zip
!unzip Flickr8K_text.zip

In [ ]:
import numpy as np
from numpy import array
import matplotlib.pyplot as plt
%matplotlib inline

import string
import os
import glob
from PIL import Image
from time import time

from keras import Input, layers
from keras import optimizers
from keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing import image
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import LSTM, Embedding, Dense, Activation, Flatten, Reshape, Dropout
from keras.layers.wrappers import Bidirectional
from keras.layers.merge import add
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input
from keras.models import Model
from keras.callbacks import ModelCheckpoint
import pickle

In [ ]:
token_path = "Flickr8K.token.txt"
train_images_path = 'Flickr_8k.trainImages.txt'
test_images_path = 'Flickr_8k.testImages.txt'
images_path = 'Flickr8K_Dataset/'

doc = open(token_path, 'r').read()
print(doc[:410])

1000268201_693b08cb0e.jpg#0 A child in a pink dress is climbing up a set 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 playhouse .
1000268201_693b08cb0e.jpg#3 A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4 A little girl in a pink dress going into a wooden cabin .

In [ ]:
descriptions = dict()
for line in doc.split('\n'):
    tokens = line.split()
    if len(line) > 2:
        image_id = tokens[0].split('.')[0]
        image_desc = ' '.join(tokens[1:])
        if image_id not in descriptions:
            descriptions[image_id] = list()
            descriptions[image_id].append(image_desc)

In [ ]:
table = str.maketrans(' ', string.punctuation)
for key, desc_list in descriptions.items():
    for i in range(len(desc_list)):
        desc = desc_list[i]
        desc = desc.strip()
        desc = [word.lower() for word in desc]
        desc = [w.translate(table) for w in desc]
        desc_list[i] = ' '.join(desc)

In [ ]:
pic = '1000268201_693b08cb0e.jpg'
x=plt.imread(images_path+pic)
plt.imshow(x)
plt.show()
descriptions['1000268201_693b08cb0e']

0
100
200
300
400
0 100 200 300

Out[ ]:
['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 [ ]:
vocabulary = set()
for key in descriptions.keys():
    [vocabulary.update(d.split()) for d in descriptions[key]]
print('Original Vocabulary Size: %d' % len(vocabulary))

Original Vocabulary Size: 8828

In [ ]:
lines = list()
for key, desc_list in descriptions.items():
    for desc in desc_list:
        lines.append(key + ' ' + desc)
    new_descriptions = '\n'.join(lines)

In [ ]:
doc = open(train_images_path, 'r').read()
dataset = list()
for line in doc.split('\n'):
    if len(line) > 1:
        identifier = line.split('.')[0]
        dataset.append(identifier)

train = set(dataset)

In [ ]:
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip

--2021-07-21 15:08:06-- http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu):171.64.67.140:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2021-07-21 15:08:07-- https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu):171.64.67.140:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2021-07-21 15:08:07-- http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu):171.64.64.22:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

glove.6B.zip 100%[=====] 822.24M 5.05MB/s in 2m 41s

2021-07-21 15:10:49 (5.10 MB/s) - 'glove.6B.zip' saved [862182613/862182613]

Archive: glove.6B.zip
  inflating: glove.6B.600.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt

In [ ]:
img = glob.glob(images_path + '*.jpg')
train_images = set(open(train_images_path, 'r').read().strip().split('\n'))
train_img = []
for i in img:
    if i[len(images_path):] in train_images:
        train_img.append(i)

test_images = set(open(test_images_path, 'r').read().strip().split('\n'))
test_img = []
for i in img:
    if i[len(images_path):] in test_images:
        test_img.append(i)

In [ ]:
train_descriptions = dict()
for line in new_descriptions.split('\n'):
    tokens = line.split()
    image_id, image_desc = tokens[0], tokens[1:]
    if image_id in train:
        if image_id not in train_descriptions:
            train_descriptions[image_id] = list()
        desc = 'startseq ' + ' '.join(image_desc) + ' ' + 'endseq'
        train_descriptions[image_id].append(desc)

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

In [ ]:
word_count_threshold = 10
word_counts = {}
for sent in all_train_captions:
    ntokens = 0
    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('Vocabulary Size: %d' % len(vocab))

Vocabulary = 1659

In [ ]:
ixtoword = {}
wordtoix = {}
ix = 1
for w in vocab:
    wordtoix[w] = ix
    ixtoword[ix] = w
    ix += 1

vocab_size = len(ixtoword) + 1

In [ ]:
all_desc = list()
for key in train_descriptions.keys():
    [all_desc.append(d) for d in train_descriptions[key]]
lines = all_desc
max_length = max(len(d.split()) for d in lines)

print('Description Length: %d' % max_length)

Description Length: 38

In [ ]:
embeddings_index = {}
f = open(os.path.join('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

In [ ]:
embedding_dim = 200
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in ixtoword.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

In [ ]:
model = InceptionV3(weights='imagenet')

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

In [ ]:
def preprocess(image_path):
    img = image.load_img(image_path, target_size=(299, 299))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    return x

In [ ]:
def encode(image):
    image = preprocess(image)
    fea_vec = model_new.predict(image)
    fea_vec = np.reshape(fea_vec, fea_vec.shape[1])
    return fea_vec

encoding_train = {}
for img in train_img:
    encoding_train[img[len(images_path):]] = encode(img)

train_features = encoding_train

encoding_test = {}
for img in test_img:
    encoding_test[img[len(images_path):]] = encode(img)

In [ ]:
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)

inputs2 = Input(shape=(max_length,))
se2 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se2)
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)
model.summary()

Model: "model_1"

Layer (type) Output Shape Param # Connected to
-----
input_2 (InputLayer) [(None, 2048)] 0
input_3 (InputLayer) [(None, 2048)] 0 embedding[0][0]
dropout (Dropout) (None, 2048) 0 input_2[0][0]
dropout_1 (Dropout) (None, 38, 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]
dense_1 (Dense) (None, 256) 65792 add[0][0]
dense_2 (Dense) (None, 1660) 426620 dense_1[0][0]
Total params: 1,816,924
Trainable params: 1,816,924
Non-trainable params: 0

In [ ]:
with open('content/drive/MyDrive/encoded_train_images.pkl', "wb") as encoded_pickle:
    pickle.dump(encoding_train, encoded_pickle)

In [ ]:
# Save the bottleneck test features to disk
with open('content/drive/MyDrive/encoded_test_images.pkl', "wb") as encoded_pickle:
    pickle.dump(encoding_test, encoded_pickle)

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

In [ ]:
model.compile(loss='categorical_crossentropy', optimizer='adam')

In [ ]:
def data_generator(descriptions, photos, wordtoix, max_length, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    while 1:
        for key, desc_list in descriptions.items():
            # 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
                in_seq = pad_sequences([seq], maxlen=max_length, p
                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)

            if n==num_photos_per_batch:
                yield ([array(X1), array(X2)], array(y))
                X1, X2, y = list(), list(), list()
                n=0

In [ ]:
epochs = 30
batch_size = 3
steps = len(train_descriptions)/batch_size

generator = data_generator(train_descriptions, train_features, wordtoix, max_length, batch_size)
model.fit(generator, epochs=epochs, steps_per_epoch=steps, verbose=1)

Epoch 1/30
2000/2000 [=====] - 308s 146ms/step - loss: 4.1669
Epoch 2/30
2000/2000 [=====] - 293s 146ms/step - loss: 3.0937
Epoch 3/30
2000/2000 [=====] - 293s 147ms/step - loss: 2.8661
Epoch 4/30
2000/2000 [=====] - 292s 146ms/step - loss: 2.7350
Epoch 5/30
2000/2000 [=====] - 292s 146ms/step - loss: 2.6422
Epoch 6/30
2000/2000 [=====] - 297s 148ms/step - loss: 2.5727
Epoch 7/30
2000/2000 [=====] - 292s 146ms/step - loss: 2.5217
Epoch 8/30
2000/2000 [=====] - 289s 145ms/step - loss: 2.4776
Epoch 9/30
2000/2000 [=====] - 288s 144ms/step - loss: 2.4403
Epoch 10/30
2000/2000 [=====] - 288s 144ms/step - loss: 2.4089
Epoch 11/30
2000/2000 [=====] - 293s 146ms/step - loss: 2.3796
Epoch 12/30
2000/2000 [=====] - 291s 145ms/step - loss: 2.3555
Epoch 13/30
2000/2000 [=====] - 290s 145ms/step - loss: 2.3372
Epoch 14/30
2000/2000 [=====] - 289s 145ms/step - loss: 2.3150
Epoch 15/30
2000/2000 [=====] - 291s 146ms/step - loss: 2.3000
Epoch 16/30
2000/2000 [=====] - 293s 147ms/step - loss: 2.2883
Epoch 17/30
2000/2000 [=====] - 292s 146ms/step - loss: 2.2718
Epoch 18/30
2000/2000 [=====] - 289s 145ms/step - loss: 2.2588
Epoch 19/30
2000/2000 [=====] - 289s 145ms/step - loss: 2.2459
Epoch 20/30
2000/2000 [=====] - 282s 141ms/step - loss: 2.2322
Epoch 21/30
2000/2000 [=====] - 283s 141ms/step - loss: 2.2257
Epoch 22/30
2000/2000 [=====] - 281s 140ms/step - loss: 2.2172
Epoch 23/30
2000/2000 [=====] - 280s 140ms/step - loss: 2.2063
Epoch 24/30
2000/2000 [=====] - 288s 148ms/step - loss: 2.2000
Epoch 25/30
2000/2000 [=====] - 279s 139ms/step - loss: 2.1926
Epoch 26/30
2000/2000 [=====] - 279s 139ms/step - loss: 2.1877
Epoch 27/30
2000/2000 [=====] - 279s 139ms/step - loss: 2.1794
Epoch 28/30
2000/2000 [=====] - 278s 139ms/step - loss: 2.1724
Epoch 29/30
2000/2000 [=====] - 278s 139ms/step - loss: 2.1683
Epoch 30/30
2000/2000 [=====] - 278s 139ms/step - loss: 2.1636
<keras.callbacks.History at 0x7f36cd2b9d0>

In [ ]:
model.save('content/drive/MyDrive/model.h5')

/usr/local/lib/python3.7/dist-packages/keras/utils/generic_utils.py:497: CustomMaskWarning: 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.
category=CustomMaskWarning)

In [ ]:
model.save_weights('content/drive/MyDrive/model_30.h5')

In [ ]:
def greedySearch(photo):
    in_text = start_word
    for i in range(max_length):
        sequence = [wordtoix[w] for w in in_text.split()] if w in wordtoix
        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:]
        final = ' '.join(final)
    return final

In [ ]:
def beam_search_predictions(image, beam_index = 3):
    start = wordtoix['startseq']
    start_word = [start, 0.0]
    while len(start_word[0][0]) < max_length:
        for s in start_word:
            par_caps = sequence.pad_sequences([s[0]], maxlen=max_length, padding='post')
            preds = model.predict([image, par_caps], verbose=0)
            word_preds = np.argmax(preds[0])[0]-beam_index
            # Getting the top <beam_index>(n) predictions and creating a
            # new list so as to put them into the model again
            for w in word_preds:
                next_cap, prob = s[0][1], s[1]
                next_cap.append(w)
                prob += preds[0][w]
            temp.append((next_cap, prob))

            start_word = temp
        # Sorting according to the probabilities
        start_word = sorted(start_word, reverse=False, key=lambda l: l[1])
        # Getting the top words
        start_word = start_word[-beam_index:]

    start_word = start_word[0][0]
    intermediate_caption = [ixtoword[i] for i in start_word]
    final_caption = []

    for i in intermediate_caption:
        final = 'endseq'
        final = final.split()
        final = final[:-1]
        final = ' '.join(final)
    return final_caption

In [ ]:
pic = '2399605960_1d0c9e6a20.jpg'
image = encoding_test[pic].reshape((1,2048))
x=plt.imread(images_path+pic)
plt.imshow(x)
plt.show()

print("Greedy Search:", greedySearch(image))
print("Beam Search, K = 3:", beam_search_predictions(image, beam_index = 3))
print("Beam Search, K = 5:", beam_search_predictions(image, beam_index = 5))
print("Beam Search, K = 7:", beam_search_predictions(image, beam_index = 7))
print("Beam Search, K = 10:", beam_search_predictions(image, beam_index = 10))

0
50
100
150
200
250
300
0 100 200 300 400

Greedy Search: a dog is running through the snow
Beam Search, K = 3: a white dog runs through the snow
Beam Search, K = 5: a brown and white dog is playing in the snow
Beam Search, K = 7: a brown and white dog is playing in the snow
Beam Search, K = 10: a brown and white dog is playing in the snow

In [ ]:
```