International Journal of Research in Engineering, Science and Management
Volume 5, Issue 1, January 2022
https://www.ijresm.com | ISSN (Online): 2581-5792

Automated Image Capturing Using CNN and RNN

Lakshitaa Sehgal[1*], Simran Mandan[2]

***Abstract*: With the evolution of generation picture captioning is a totally essential issue of virtually all industries regarding information abstraction. To interpret such information by a machine may be very complex and time-consuming. For a device to apprehend the context and surroundings info of a photo, it wishes a higher understanding of the outline projected from the picture. Many deep gaining knowledge of techniques have now not followed conventional strategies however are changing the manner a machine is familiar with and translates. Majorly the usage of Captions and attaining a properly-described vocabulary linked to images. The improvements in technology and ease of computation of extensive information have made it possible for us to without problems observe deep gaining knowledge of in several projects the usage of our non-public computer. A solution calls for each that the content of the photograph is thought and translated to that means within the phrases of words and that the phrases must string collectively to be comprehensible. It combines both laptop imagination and prescient using deep mastering and herbal language processing and marks virtually tough trouble in broader synthetic intelligence. In this project, we create an automatic photo captioning version with the use of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) to provide a series of texts that greatly describe the photograph. Using Flickr 8000 dataset, we have organized our model. As image captioning requires a neural network right here we've got nicely described steps to carry out. To create a deep neural community using CNN and RNN, we first hyperlink the description to the photograph convolutional neural network will take the image and segregate it into a number of traits, the recurrent neural community will make this function into well-described descriptive language. The task makes use of the encoder-decoder model, wherein the CNN which performs the function extraction and the output of the encoder is fed to the decoder which approaches the categorized features into suitable sentences. The characteristic extraction could be finished by using today's Inception V3 module-50 era with the method of switch learning in order that we will adjust the venture-precise to our cause. The language model uses a herbal language toolkit for simple herbal language processing and the structure used for the recurrent neural networks is lengthy-brief term memory.**

***Keywords*: Images, Convolutional Neural Networks (CNN), conventional strategies, CNN and RNN, encoder-decoder model, language.**

1. Introduction

This project aims at creating an auto-captioning model for an image. This can be achieved by using CNN for image feature classification. Then we are using RNN for generating the caption. We have chosen Long Short-Term Memory (LSTM) which is a type of RNN model. The LSTM can learn from sequential data like a series of words and characters. These systems utilize hidden layers that connect the input and output layers. Which makes a memory circle with the goal that the model can learn from the past yields. So basically, CNN works with spatial features and RNN helps in solving sequential data.

This project uses advanced methods of computer vision using Deep Learning and natural language processing using a Recurrent Neural Network. Deep Learning is a machine learning technique with which we can program the computer to learn complex models and understand patterns in a large dataset. The combination of increasing computation speed, the wealth of research and the rapid growth of technology. Deep Learning and AI is experiencing massive growth worldwide and will perhaps be one of the world's biggest industries shortly. The 21st century is the birth of the AI revolution, and data becoming the new 'oil' for it. Every second in today's world large amount of data is being generated. We need to build models that can study these datasets and come up with patterns or find the solution for analysis and research. This can be achieved solely due to deep learning.

Computer Vision is a cutting-edge field of computer science that aims to enable computers to understand what is being seen in an image. Computers don't perceive the world as humans do. For them, the perception is just sets of raw numbers and several limitations like the type of camera, lighting conditions, clarity, scaling, viewpoint variation etc. make computer vision so hard to process as it is very tough to build a robust model that can work on every condition.

The neural network architectures normally we see were trained using the current inputs only. While developing the system, the generating output does not consider the previous inputs. It is because of neglecting any memory elements present. That is why the use of RNN tackles the memory issues that haunt the system. This led us to create an efficient system.

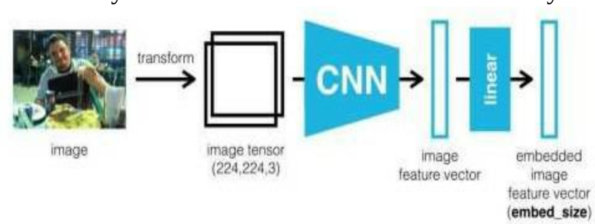


Fig. 1. Architecture diagram

A. Background Study

[1] A Survey on Automatic Image Caption Generation - Shuang Bai, Shan An.

The image captioning approach mechanically produces a caption for a photograph. As a lately emerged research place,

*Corresponding author: lakshitaasehgal29@gmail.com

its miles attracting more and more interest. To obtain the motive of picture captioning, semantic facts of pictures desires to be captured and expressed in natural languages. Connecting both research communities of computer vision and natural language processing, photograph captioning is a pretty tough challenge. Various methods have been proposed to treat this hassle. A survey on advances in photo captioning research is given. Based on the technique accompanied by the photograph captioning procedures are categorised into distinct training. Representative strategies in each class are summarized, and their strengths and boundaries are referred to.

[2] Improving Image Captioning via Leveraging Knowledge Graphs - Yimin Zhou, Yiwei Sun, Vasant Honavar.

In this, the use of knowledge graphs that capture widespread or common-sense knowledge, to reinforce the facts extracted from pics by the state-of-the-artwork techniques for image captioning is explored. The outcomes of the experiments, on several benchmark statistic sets inclusive of MS COCO, as measured by using CIDEr-D, an overall performance metric for picture captioning, display that the variants of the kingdom-of-the-art techniques for photo captioning that make use of the facts extracted from expertise graphs can appreciably outperform people who depend solely on the data extracted from snapshots.

[3] Image Captioning with Object Detection and Localization - Zhongliang Yang, Yu-Jin Zhang, Sadaqat ur Rehman, Yongfeng Huang.

Automatically generating a herbal language description of a photo is a challenge close to the heart of photo understanding. In this paper, a multi-version neural community approach intently associated with the human visual system that routinely learns to describe the content material of snapshots is presented. The version includes two sub-models: an item detection and localization model, which extract the statistics of gadgets and their spatial dating in pictures respectively; Besides, a deep recurrent neural network (RNN) based on lengthy quick-time period memory (LSTM) gadgets with attention mechanism for sentences era. Each phrase of the description could be robotically aligned to unique items of the entered image whilst it is generated. This is similar to the eye mechanism of the human visual gadget. Experimental consequences on the Flickr 8k dataset showcase the merit of the proposed technique, which outperform previous benchmark models

[4] Convolutional Image Captioning - Jyoti Aneja, Aditya Deshpande, Alexander G. Schwing.

In latest years' giant development has been made in photograph captioning, the use of Recurrent Neural Networks powered by means of lengthy-short-time period-reminiscence (LSTM) devices. Despite mitigating the vanishing gradient problem, and in spite of their compelling potential to memorize dependencies, LSTM units are complicated and inherently sequential throughout time. However, the complex addressing and overwriting mechanism blended with inherently sequential processing, and big garage required because of back-propagation thru time (BPTT), poses demanding situations at some point of education.

2. Methodology

A. Working

Now, to create a deep neural network the usage of CNN and RNN. We first hyperlink the description to the photograph convolutional neural network will take the image and segregate it into a number of traits, the recurrent neural network will make this feature into properly-described descriptive language.

3. Proposed Model

A. CNN Encoder

The encoder is based totally on a Convolutional neural network that encodes a picture right into a compact representation. The CNN-Encoder is an Inception V3 module (Residual Network. One-layer activation has using skips series that depend on the device. And wherein it gets inputted to every other layer, going even further into the network, accordingly making the use of inception v3 module viable.

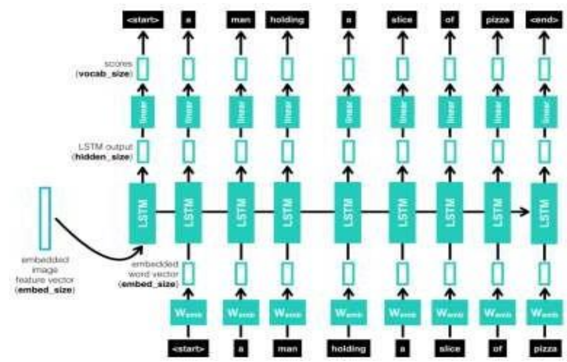


Fig. 2. RNN decoder

The CNN encoder is followed with the aid of a recurrent neural network that generates a corresponding sentence. The RNN-Decoder consists of an unmarried LSTM layer observed through one absolutely-connected (linear) layer. The RNN community is trained on the Flickr 8k dataset. It is used to are expecting the following phrase of a sentence based totally on preceding phrases. The captions are offered as a list of tokenized phrases so that the RNN version can teach and back propagate to lessen mistakes and generate higher and more comprehensible texts describing the photograph.

B. Flickr 8k dataset

This dataset contains 8,000 images where each image has 5 different descriptions based on the features and context. Here they are well defined and used in various different environments.

C. Inception V3 module

There are 4 versions. The first Google Net must be the Inception-v1, but there are numerous typos in Inception-v3 which lead to wrong descriptions about Inception versions. This may be due to the intense ILSVRC competition at that moment. Consequently, there are many reviews within the internet mixing up between v2 and v3. Some of the reviews even think that v2 and v3 are equivalent with just some minor different settings.

Hardware and Software Requirements:

Hardware:

- Processor: Minimum 1 GHz; Recommended 2GHz or more.
- Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
- Hard Drive: Minimum 32 GB; Recommended 64 GB or more.
- Memory (RAM): Minimum 1 GB; Recommended 4 GB or above.

Software:

- Keras
- TensorFlow
- Jupyter Notebook
- Windows/Linux

D. Dataset/Tool Used

Flickr 8k dataset which comes with images and captions for supervised learning.

4. Results

[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
from pickle import dump, load
from time import time
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector,\
                         Activation, Flatten, Reshape, concatenate, Dropout, BatchNormalization
from keras.optimizers import Adam, RMSprop
from keras.layers.wrappers import Bidirectional
from keras.layers.merge import add
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras import Input, layers
from keras import optimizers
from keras.applications.inception_v3 import preprocess_input
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
```

```
Using TensorFlow backend.
```

[9]:
```
descriptions['1000268201_693b08cb0e']
```

Out[9]:
```
['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']
```

[10]:
```
descriptions['1001773457_577c3a7d70']
```

Out[10]:
```
['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 the street',
 'two dogs of different breeds looking at each other on the road',
 'two dogs on pavement moving toward each other']
```

[7]:
```
descriptions['1001773457_577c3a7d70']
```

Out[7]:
```
['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 .']
```

[8]:
```
def clean_descriptions(descriptions):
    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.split()
            desc = [word.lower() for word in desc]
            desc = [w.translate(table) for w in desc]
            desc = [word for word in desc if len(word)>1]
            desc = [word for word in desc if word.isalpha()]
            desc_list[i] =  ' '.join(desc)

clean_descriptions(descriptions)
```

[3]:
```
def loading_doc(file_name):
    file = open(file_name, 'r')
    txt = file.read()
    file.close()
    return txt

file_name = "/kaggle/input/flicker8k-dataset/Flickr8k_text/Flickr8k.token.txt"
doc = loading_doc(file_name)
print(doc[:300])
```

```
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 s
```

[2]:
```
import os
for dirname, _, file_names in os.walk('/kaggle/input'):
    for file_name in file_names:
        print(os.path.join(dirname, file_name))
```

```
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/2075041394_0b3ea1822d.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/3185662156_c877583c53.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/2189181027_a445b13438.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/2234910971_80e0325918.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/3301859683_2d5e4b40a3.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/2070831281_dc04b3e15d.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/172092461_a9a9762e13.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/3621623690_0095e330bc.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/3498482871_4e02f31c35.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/336551609_1385ab139e.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/2472720629_d9a6736356.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/2938316391_97382d14aa.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/3208987435_780ae35ef0.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/3222749441_3bdfe088e3.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/2604305843_ebe3e8a328.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/314940358_ec1958dc1d.jpg
/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/197107117_4b438b1872.jpg
```

[4]:
```
def loading_description(doc):
    mapping = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        if len(line) < 2:
            continue
        image_id, image_desc = tokens[0], tokens[1:]
        image_id = image_id.split('.')[0]
        image_desc = ' '.join(image_desc)
        if image_id not in mapping:
            mapping[image_id] = list()
        mapping[image_id].append(image_desc)
    return mapping

descriptions = loading_description(doc)
print('Loaded: %d ' % len(descriptions))
```

```
Loaded: 8092
```

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

Out[5]:
```
['1000268201_693b08cb0e',
 '1001773457_577c3a7d70',
 '1002674143_1b742ab4b8',
 '1003163366_44323f5815',
 '1007129816_e794419615']
```

[6]:
```
descriptions['1000268201_693b08cb0e']
```

Out[6]:
```
['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 .']
```

[11]:
```
def to_vocabulary(descriptions):

    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc

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

```
Original Vocabulary Size: 8763
```

[12]:
```
def save_descriptions(descriptions, file_name):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + ' ' + desc)
    data = '\n'.join(lines)
    file = open(file_name, 'w')
    file.write(data)
    file.close()

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

[13]:
```
def load_set(file_name):
    doc = loading_doc(file_name)
    dataset = list()
    for line in doc.split('\n'):
        if len(line) < 1:
            continue
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)

file_name = '/kaggle/input/flicker8k-dataset/Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(file_name)
print('Dataset: %d' % len(train))
```

```
Dataset: 6000
```

[14]:
```
images = '/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/'
img = glob.glob(images + '*.jpg')
```

```
[15]:
train_images_file = '/kaggle/input/flicker8k-dataset/Flickr8k_text/Flickr_8k.trainImages.txt'
train_images = set(open(train_images_file, 'r').read().strip().split('\n'))
train_img = []

for i in img:
    if i[len(images):] in train_images:
        train_img.append(i)
```

```
[16]:
test_images_file = '/kaggle/input/flicker8k-dataset/Flickr8k_text/Flickr_8k.testImages.txt'
test_images = set(open(test_images_file, 'r').read().strip().split('\n'))
test_img = []

for i in img:
    if i[len(images):] in test_images:
        test_img.append(i)
```

```
Time taken in seconds = 39.04336953163147
```

```
[26]:
with open("encoded_test_images.pkl", "wb") as encoded_pickle:
    pickle.dump(encoding_test, encoded_pickle)
```

```
[27]:
train_features = load(open("encoded_train_images.pkl", "rb"))
print('Photos: train=%d' % len(train_features))
```

```
Photos: train=6000
```

```
[28]:
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
```

```
[17]:
def load_clean_descriptions(file_name, dataset):
    doc = loading_doc(file_name)
    descriptions = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        image_id, image_desc = tokens[0], tokens[1:]
        if image_id in dataset:
            if image_id not in descriptions:
                descriptions[image_id] = list()
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            descriptions[image_id].append(desc)
    return descriptions

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

```
Descriptions: train=6000
```

```
[18]:
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
```

```
[19]:
model = InceptionV3(weights='imagenet')
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.5/inception_v3_weight
96116736/96112376 [==============================] - 3s 0us/step
```

```
[20]:
model_new = Model(model.input, model.layers[-2].output)
```

```
[21]:
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
```

```
[22]:
start = time()
encoding_train = {}
i=0;
for img in train_img:
    encoding_train[img[len(images):]] = encode(img)
    print(i)
    i=i+1
print("Time taken in seconds =", time()-start)
```

```
5983
5984
5985
5986
5987
5988
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5999
Time taken in seconds = 241.26520681381226
```

```
[23]:
import pickle
```

```
[31]:
vocab_size = len(ixtoword) + 1
vocab_size
```

```
Out[31] 1652
```

```
[32]:
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)

max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
```

```
Description Length: 34
```

```
[33]:
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():
            n+=1
            photo = photos[key+'.jpg']
            for desc in desc_list:
                seq = [wordtoix[word] for word in desc.split(' ') if word in wordtoix]
                for i in range(1, len(seq)):
                    in_seq, out_seq = seq[:i], seq[i]
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(out_seq)
            if n==num_photos_per_batch:
                yield [[array(X1), array(X2)], array(y)]
                X1, X2, y = list(), list(), list()
                n=0
```

```
[38]:
model.summary()
```

Model: "model_2"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 34)	0	
input_2 (InputLayer)	(None, 2048)	0	
embedding_1 (Embedding)	(None, 34, 200)	330400	input_3[0][0]
dropout_1 (Dropout)	(None, 2048)	0	input_2[0][0]
dropout_2 (Dropout)	(None, 34, 200)	0	embedding_1[0][0]
dense_1 (Dense)	(None, 256)	524544	dropout_1[0][0]
lstm_1 (LSTM)	(None, 256)	467968	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, 1652)	424564	dense_2[0][0]

Total params: 1,813,268
Trainable params: 1,813,268
Non-trainable params: 0

```
[34]:
glove_dir = '/kaggle/input/glove6b200d'
embeddings_index = {}
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.
```

```
[35]:
embedding_dim = 200
embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in wordtoix.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
[36]:
embedding_matrix.shape
```

```
Out[36] (1652, 200)
```

```
[ ]:
```

```
[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)
```

[39]:
```
model.layers[2]
```

Out[39]:
```
<keras.layers.embeddings.Embedding at 0x7f7a3f9f8b10>
```

[40]:
```
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False
```

[41]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

[42]:
```
epochs = 10
number_pics_per_bath = 3
steps = len(train_descriptions)//number_pics_per_bath
```

[43]:
```
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_' + str(i) + '.h5')
```

```
Epoch 1/1
2000/2000 [==============================] - 96s 48ms/step - loss: 4.1272
Epoch 1/1
2000/2000 [==============================] - 97s 49ms/step - loss: 3.4257
Epoch 1/1
2000/2000 [==============================] - 95s 48ms/step - loss: 3.2098
Epoch 1/1
2000/2000 [==============================] - 96s 48ms/step - loss: 3.0767
Epoch 1/1
2000/2000 [==============================] - 95s 48ms/step - loss: 2.9832
Epoch 1/1
2000/2000 [==============================] - 96s 48ms/step - loss: 2.9072
Epoch 1/1
2000/2000 [==============================] - 95s 48ms/step - loss: 2.8479
Epoch 1/1
2000/2000 [==============================] - 96s 48ms/step - loss: 2.8001
Epoch 1/1
2000/2000 [==============================] - 97s 49ms/step - loss: 2.7600
Epoch 1/1
2000/2000 [==============================] - 96s 48ms/step - loss: 2.7239
```

[50]:
```
model.save_weights('model_9.h5')
```

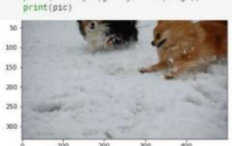
[51]:
```
model.load_weights('model_9.h5')
```

[46]:
```
images = '/kaggle/input/flicker8k-dataset/Flickr8k_Dataset/Flicker8k_Dataset/'
```

[47]:
```
with open("encoded_test_images.pkl", "rb") as encoded_pickle:
    encoding_test = load(encoded_pickle)
```

[48]:
```
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
```

[49]:
```
for i in range(10):
    rn =  np.random.randint(0, 1000)
    pic = list(encoding_test.keys())[rn]
    image = encoding_test[pic].reshape((1,2048))
    x=plt.imread(images+pic)
    plt.imshow(x)
    plt.show()
    print("Greedy:",greedySearch(image))
    print(pic)
```




```
Greedy: two dogs play in the snow
2398605966_1d0c9e6a20.jpg
```


```
Greedy: two dogs play in the snow
2398605966_1d0c9e6a20.jpg
```

5. Conclusion

In this paper, we have presented a multimodal methodology for automatic captioning of image based on InceptionV3 and LSTM. The model proposed had been designed with encoder-decoder architecture which was trained over a huge Flicker 8k dataset consisting of a set of 8000 images with their respective captions. We adopted InceptionV3, a convolutional neural network, as the encoder to encode an image into a compact representation as the feature matrix. Thereafter, a language model LSTM was selected as the decoder to generate the description. The experimental evaluations indicate that the proposed model is able to generate accurate captions for images.

References

[1] Anderson, P., Fernando, B., Johnson, M. and Gould, S., 2016, October. Spice: Semantic propositional image caption evaluation. In European Conference on Computer Vision (pp. 382-398). Springer, Cham.
[2] Aneja, J., Deshpande, A. and Schwing, A.G., 2018. Convolutional image captioning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5561-5570).
[3] Bai, S. and An, S., 2018. A survey on automatic image caption generation. Neurocomputing, 311, pp.291-304.
[4] Chen, L., Zhang, H., Xiao, J., Nie, L., Shao, J., Liu, W. and Chua, T.S., 2017. Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 5659-5667).
[5] Gu, J., Wang, G., Cai, J. and Chen, T., 2017. An empirical study of language cnn for image captioning. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1222-1231).
[6] Kinghorn, P., Zhang, L. and Shao, L., 2019. A hierarchical and regional deep learning architecture for image description generation. Pattern Recognition Letters, 119, pp.77-85.
[7] Mao, J., Xu, W., Yang, Y., Wang, J., Huang, Z. and Yuille, A., 2014. Deep captioning with multimodal recurrent neural networks (m-rnn). arXiv preprint arXiv:1412.6632.
[8] Yang, Z., Zhang, Y.J., ur Rehman, S. and Huang, Y., 2017, September. Image captioning with object detection and localization. In International Conference on Image and Graphics (pp.109-118). Springer, Cham.
[9] Vinyals, O., Toshev, A., Bengio, S. and Erhan, D., 2015. Show and tell: A neural image caption generator. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3156-3164).
[10] Zhou, Y., Sun, Y. and Honavar, V., 2019, January. Improving image captioning by leveraging knowledge graphs. In 2019 IEEE Winter Conference on Applications of Computer Vision (WACV) (pp. 283-293).