```
import pandas as pd
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
```

```
Using TensorFlow backend.
```

## Read Captions File

```
# Reading the Description file

with open("./flicker8k-dataset/Flickr8k_text/Flickr8k.token.txt") as filepath:
    captions = filepath.read()
    filepath.close()


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


len(captions)
```

```
40460
```

```
# creating a "descriptions" dictionary  where key is 'img_name' and value is list of captions corresponding to that image_file.

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)


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 .']
```

```
Start coding or generate with AI.
```

## Data Cleaning

```
""" 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


clean_text("My noghsujf si am m cricket101 &8 mphi*&86%%&??,BY6fajdn 213 q rqu243 boy  32 ewr wO>>J DHD 34  asfb HHGY Gvg HgB   231 123'
```

```
'my noghsujf si am cricket mphi by fajdn rqu boy ewr wo dhd asfb hhgy gvg hgb'
```

```
#  modify all the captions i.e - cleaned captions

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


#  clean descriptions

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']
```

```
# writing clean description to .txt file

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


#  reading description file

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

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


Start coding or generate with AI.


Start coding or generate with AI.


# finding the unique vocabulary

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
```

```
Start coding or generate with AI.


#  ALl words in description dictionary
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']
```

```
#  count the frequency of each word, sort them and discard the words having frequency lesser than threshold value

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]


len(all_vocab)
```

```
1845
```

## Loading Training Testing Data

```
# TrainImagesFile
f = open("flicker8k-dataset/Flickr8k_text/Flickr_8k.trainImages.txt")
train = f.read()
f.close()


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


# TestImagesFile
f = open("flicker8k-dataset/Flickr8k_text/Flickr_8k.testImages.txt")
test = f.read()
f.close()


test  = [e.split(".")[0] for e in test.split("\n")[:-1]]


Start coding or generate with AI.


Start coding or generate with AI.


Start coding or generate with AI.


Start coding or generate with AI.


Start coding or generate with AI.


# create train_descriptions dictionary, which will be similar to earlier one, but having only train samples
# add startseq + endseq

train_descriptions = {}

for t in train:
    train_descriptions[t] = []
    for cap in descriptions[t]:
        cap_to_append = "startseq " + cap + " endseq"
        train_descriptions[t].append(cap_to_append)


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']
```

## Data Preprocessing - Images

```
"""
In this section, we will load our images and do some processing so that we can feed it in our network.
"""


model = ResNet50(weights="imagenet", input_shape=(224,224,3))
```

```
model.summary()
```

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            (None, 224, 224, 3)  0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           input_2[0][0]
__________________________________________________________________________________________________
conv1 (Conv2D)                  (None, 112, 112, 64) 9472        conv1_pad[0][0]
__________________________________________________________________________________________________
bn_conv1 (BatchNormalization)   (None, 112, 112, 64) 256         conv1[0][0]
__________________________________________________________________________________________________
activation_50 (Activation)      (None, 112, 112, 64) 0           bn_conv1[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0           activation_50[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 56, 56, 64)   0           pool1_pad[0][0]
__________________________________________________________________________________________________
res2a_branch2a (Conv2D)         (None, 56, 56, 64)   4160        max_pooling2d_2[0][0]
__________________________________________________________________________________________________
bn2a_branch2a (BatchNormalizati (None, 56, 56, 64)   256         res2a_branch2a[0][0]
__________________________________________________________________________________________________
activation_51 (Activation)      (None, 56, 56, 64)   0           bn2a_branch2a[0][0]
__________________________________________________________________________________________________
res2a_branch2b (Conv2D)         (None, 56, 56, 64)   36928       activation_51[0][0]
__________________________________________________________________________________________________
bn2a_branch2b (BatchNormalizati (None, 56, 56, 64)   256         res2a_branch2b[0][0]
__________________________________________________________________________________________________
activation_52 (Activation)      (None, 56, 56, 64)   0           bn2a_branch2b[0][0]
__________________________________________________________________________________________________
res2a_branch2c (Conv2D)         (None, 56, 56, 256)  16640       activation_52[0][0]
__________________________________________________________________________________________________
res2a_branch1 (Conv2D)          (None, 56, 56, 256)  16640       max_pooling2d_2[0][0]
__________________________________________________________________________________________________
bn2a_branch2c (BatchNormalizati (None, 56, 56, 256)  1024        res2a_branch2c[0][0]
__________________________________________________________________________________________________
bn2a_branch1 (BatchNormalizatio (None, 56, 56, 256)  1024        res2a_branch1[0][0]
__________________________________________________________________________________________________
add_17 (Add)                    (None, 56, 56, 256)  0           bn2a_branch2c[0][0]
                                                                 bn2a_branch1[0][0]
__________________________________________________________________________________________________
activation_53 (Activation)      (None, 56, 56, 256)  0           add_17[0][0]
__________________________________________________________________________________________________
res2b_branch2a (Conv2D)         (None, 56, 56, 64)   16448       activation_53[0][0]
__________________________________________________________________________________________________
bn2b_branch2a (BatchNormalizati (None, 56, 56, 64)   256         res2b_branch2a[0][0]
__________________________________________________________________________________________________
activation_54 (Activation)      (None, 56, 56, 64)   0           bn2b_branch2a[0][0]
__________________________________________________________________________________________________
res2b_branch2b (Conv2D)         (None, 56, 56, 64)   36928       activation_54[0][0]
__________________________________________________________________________________________________
bn2b_branch2b (BatchNormalizati (None, 56, 56, 64)   256         res2b_branch2b[0][0]
__________________________________________________________________________________________________
activation_55 (Activation)      (None, 56, 56, 64)   0           bn2b_branch2b[0][0]
__________________________________________________________________________________________________
res2b_branch2c (Conv2D)         (None, 56, 56, 256)  16640       activation_55[0][0]
__________________________________________________________________________________________________
bn2b_branch2c (BatchNormalizati (None, 56, 56, 256)  1024        res2b_branch2c[0][0]
__________________________________________________________________________________________________
```

```
# Create a new model, by removing the last layer (output layer of 1000 classes) from the resnet50
model_new = Model(model.input, model.layers[-2].output)


images = "./flicker8k-dataset/Flickr8k_Dataset/"


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


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


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
```

```
# Save the bottleneck train features to disk

with open("./storage/encoded_train_images.pkl", "wb") as encoded_pickle:
    pickle.dump(encoding_train, encoded_pickle)
```

Start coding or generate with AI.

```
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
```

```
# Save the bottleneck train features to disk

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

Start coding or generate with AI.

```
# Load the train images features from disk

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

```
# Load the test images features from disk

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

Start coding or generate with AI.

## Data Preprocessing - Captions

Start coding or generate with AI.

```
"""
word_to_idx is mapping between each unique word in all_vocab to int value
and idx_to_word is vice-versa
"""


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
```

```
#  need to add these 2 words as well

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

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

```
#  vocab_size is total vocabulary len +1 because we will append 0's as well.

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

```
1848
```

```
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
```

Start coding or generate with AI.

## Data Preparation using Generator Function

Start coding or generate with AI.

```
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+".jpg"]

            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:
                yield [[np.array(X1), np.array(X2)], np.array(y)]
                X1, X2, y = [], [], []
                n=0
```

Start coding or generate with AI.

## Word Embedding

```
f = open("./GloVE/glove.6B.50d.txt", encoding='utf8')
```

```
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()
```

## Converting words into vectors Directly - (Embedding Layer Output)

```
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()


    embedding_output.shape
```

```
(1848, 50)
```

Start coding or generate with AI.

## Model Architecture

```
# image feature extractor model

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


# partial caption sequence model

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)


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

# Merge 2 networks
model = Model(inputs=[input_img_fea, input_cap], outputs=outputs)


model.summary()
```

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_3 (InputLayer)         (None, 35)                0
__________________________________________________________________________________________________
input_2 (InputLayer)         (None, 2048)              0
__________________________________________________________________________________________________
embedding_1 (Embedding)      (None, 35, 50)            92400     input_3[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)          (None, 2048)              0         input_2[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)          (None, 35, 50)            0         embedding_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)              (None, 256)               524544    dropout_2[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                (None, 256)               314368    dropout_3[0][0]
__________________________________________________________________________________________________
add_1 (Add)                  (None, 256)               0         dense_2[0][0]
                                                                 lstm_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)              (None, 256)               65792     add_1[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)              (None, 1848)              474936    dense_3[0][0]
==================================================================================================
Total params: 1,472,040
Trainable params: 1,379,640
Non-trainable params: 92,400
__________________________________________________________________________________________________
```

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


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

## Train Our Model

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


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

```
WARNING:tensorflow:From D:\Program Files\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorfl
Instructions for updating:
Use tf.cast instead.
Epoch 1/1
2000/2000 [==============================] - 189s 95ms/step - loss: 4.2791
Epoch 1/1
2000/2000 [==============================] - 180s 90ms/step - loss: 3.5671
Epoch 1/1
2000/2000 [==============================] - 191s 95ms/step - loss: 3.3153
Epoch 1/1
2000/2000 [==============================] - 191s 96ms/step - loss: 3.1609
Epoch 1/1
2000/2000 [==============================] - 186s 93ms/step - loss: 3.0468
Epoch 1/1
2000/2000 [==============================] - 184s 92ms/step - loss: 2.9644
Epoch 1/1
2000/2000 [==============================] - 186s 93ms/step - loss: 2.8949
Epoch 1/1
2000/2000 [==============================] - 185s 93ms/step - loss: 2.8408
Epoch 1/1
2000/2000 [==============================] - 195s 97ms/step - loss: 2.7902
Epoch 1/1
2000/2000 [==============================] - 182s 91ms/step - loss: 2.7536
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

```
model = load_model("./model_weights/model_9.h5")
```

## Predictions

Start coding or generate with AI.

```
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
```

Start coding or generate with AI.

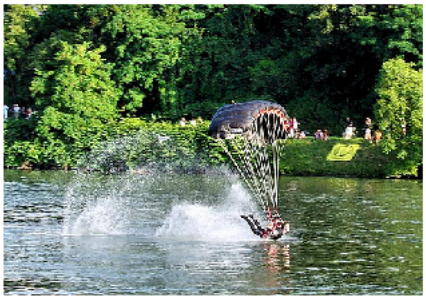
Start coding or generate with AI.

```
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(images+img_name)
    plt.imshow(i)
    plt.axis("off")
    plt.show()

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

little boy in blue shirt is holding child in the bathroom



man in yellow shirt and black shorts is wakeboarding in the water

man in red shirt and khaki helmet is standing on top of mountain

two men in sports uniforms are playing soccer

young boy in blue shorts is jumping into the air

the surfer is leaping into the air