Image Captioning using Python

Submitted in partial fulfilment of the requirements

Of the degree of

Bachelors of Engineering

By

Chinmay Paralkar 7957

Royston Pinto 7963

Kajal Sakpal 8126

Supervisor (s):

Prof. Dipali Koshti

(Assistant Professor, Department of Computer Engineering)

(DEPARTMENT OF COMPUTER SCIENCE)

FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING
BANDRA, MUMBAI 400 050

MUMBAI UNIVERSITY

2020

Certificate

This is to certify that the project entitled **"*Image Captioning using Python*"** is successfully bonafide work of ***"Chinmay Paralkar, Royston Pinto and Kajal Sakpal"*** students of B.E. in a Computer Science academic year (2019-2020) in partial fulfilment for the completion of "**Undergraduate Degree"** in "**Department of Computer Science".**

Supervisor/Guide

Co-supervisor/Guide

Head of Department

Principal

Project Report Approval for B.E.

This project report entitled **Image Captioning using Python** by **Chinmay Paralkar, Royston Pinto** and **Kajal Sakpal** are approved for the degree of **Bachelors of Engineering in Computer Science**.

Examiners:

1.----------

2.----------

Date:

Place: Mumbai

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

---------- (Signature)

Chinmay Paralkar 7957

---------- (Signature)

Royston Pinto 7963

---------- (Signature)

Kajal Sakpal 8126

Date:

Table of Content

Chapter 1

1. Introduction

All of us humans have the ability to recognize things around us and we are able to identify significant objects in our background. This ability can be given to a computer with some effort. Image captioning is a problem where a model must generate a brief human-readable text for a given photograph. Hence, with the help of this problem statement we can program our computer to recognize salient stuff in the image.

Just prior to the recent development of Deep Neural Networks this problem was inconceivable even by the most advanced researchers in Computer Vision. But with the advent of Deep Learning this problem can be solved very easily if we have the required dataset.

1.1. Problem Statement

Image Captioning is the process of generating textual description of an image. If we upload any image so this system is generate relevant caption for it. You have to write a computer program that takes an image as input and produces a relevant caption as output.

1.2 The Motivation

We must first understand how important this problem is to real world scenarios. Let's see few applications where a solution to this problem can be very useful.

Self-driving cars — Automatic driving is one of the biggest challenges and if we can properly caption the scene around the car, it can give a boost to the self driving system.

Aid to the blind — We can create a product for the blind which will guide them travelling on the roads without the support of anyone else. We can do this by first converting the scene into text and then the text to voice.

Both are now famous applications of Deep Learning. CCTV cameras are everywhere today, but along with viewing the world, if we can also generate relevant captions, then we can raise alarms as soon as there is some malicious activity going on somewhere. This could probably help reduce some crime and/or accidents.

1.3 Aim

The aim of our project is to find out a caption for an image, using CNN algorithm. We will predict the caption word by word. Thus, we need to encode each word into a fixed sized vector. However, this part will be seen later when we look at the model design, but for now we will create two Python Dictionaries namely “wordtoix” (pronounced — word to index) and “ixtoword” (pronounced — index to word).

1.4 Scope

People are increasingly discovering that many laws that are difficult to find can be found from a large amount of data. In the image description generation task, there are currently rich and colorful datasets, such as MSCOCO, Flickr8k, Flickr30k, PASCAL 1K, AI Challenger Dataset, and STAIR Captions, and gradually become a trend of contention. In the dataset, each image has five reference descriptions, and Table 2 summarizes the number of images in each dataset. In order to have multiple independent descriptions of each image, the dataset uses different syntax to describe the same image. As illustrated in the example in Figure 10, different descriptions of the same image focus on different aspects of the scene or are constructed using different grammars.

1.5 Applications

There are number of application of image captioning. For instance, image captioning can help visually impaired people to grasp what is happening in a picture. Furthermore, it could enhance the image search of search engines, it could simplify SEO by automatically generating descriptions for the pictures or improve online marketing and customer segmentation by identifying customer interests through interpreting their shared images via social media platforms. Nevertheless, image captioning is a very complex task as it goes beyond the sole classification of objects in pictures. The relation between the objects and the attributes have to be recognized. Finally, these information must be expressed in a natural language like English.

Chapter 2

2. Literature Survey

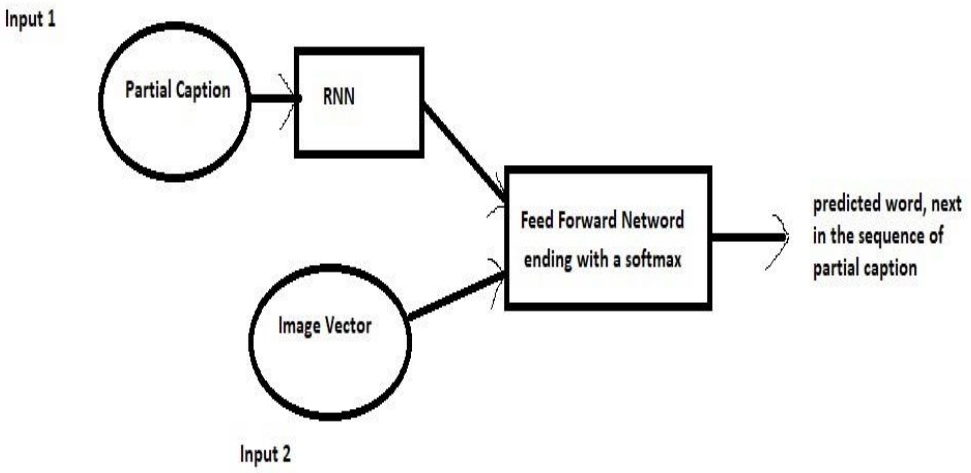
2.1 Previous Work in the field

Accurate image captioning with the use of multimodal neural networks has been a hot topic in the field of Deep Learning. I have been working with several of these approaches and the algorithms seem to give very promising results. But when it comes to using image captioning in real world applications, most of the time only a few are mentioned such as hearing aid for the blind and content generation.

It is really interesting to know if there are any other good applications (already existing or potential) where image captioning can be used either directly or as a support process.

2.2 Existing System

There are many software and apps for tourist booking and management which are used by tourist for easy booking and other help, but there doesn't exists any website or app for tourism prediction which is indeed very much important for marketers, managers and planners in order to reduce the risk of decision making in future.



Research Paper 1: Unified Vision-Language Pre-Training for Image Captioning and VQA

This paper presents a unified Vision-Language Pre-training (VLP) model. The model is unified in that (1) it can be finetuned for either vision-language generation (e.g., image captioning) or understanding (e.g., visual question answering) tasks, and (2) it uses a shared multi-layer transformer network for both encoding and decoding, which differs from many existing methods where the encoder and decoder are implemented using separate models. The unified VLP model is pre-trained on a large amount of image-text pairs using the unsupervised learning objectives of two tasks: bidirectional and sequence-to-sequence (seq2seq) masked vision-language prediction. The two tasks differ solely in what context the prediction conditions on. This is controlled by utilizing specific self-attention masks for the shared transformer network. To the best of our knowledge, VLP is the first reported model that achieves state-of-the-art results on both vision-language generation and understanding tasks, as disparate as image captioning and visual question answering, across three challenging benchmark datasets: COCO Captions, Flickr30k Captions, and VQA 2.0. The code and the pre-trained models are available at https://github.com/LuoweiZhou/VLP.

Research Paper 2: Image Captioning With AI

The problem of writing captions for an image is two-fold: you need to get meaning out of the image by extracting relevant features, and you need to translate these features into a human-readable format. In this tutorial, we're going to look at both phases independently and then connect the pieces together. We'll start with the feature extraction phase.

This tutorial was inspired by the TensorFlow tutorial on image captioning. For this tutorial we are going to use the COCO dataset (Common Ojects in Context), which consists of over 200k labelled images, each paired with five captions.

Chapter 3

3. Proposed System

3.1. The Proposed Solution

We must first understand how important this problem is to real world scenarios. Let's see few applications where a solution to this problem can be very useful.

- Self driving cars — Automatic driving is one of the biggest challenges and if we can properly caption the scene around the car, it can give a boost to the self driving system.
- Aid to the blind — We can create a product for the blind which will guide them travelling on the roads without the support of anyone else. We can do this by first converting the scene into text and then the text to voice. Both are now famous applications of Deep Learning. Refer this **link** where its shown how Nvidia research is trying to create such a product.
- CCTV cameras are everywhere today, but along with viewing the world, if we can also generate relevant captions, then we can raise alarms as soon as there is some malicious activity going on somewhere. This could probably help reduce some crime and/or accidents.
- Automatic Captioning can help, make Google Image Search as good as Google Search, as then every image could be first converted into a caption and then search can be performed based on the caption.

3.2. Methodology Overview

Image Captioning is the process of generating textual description of an image. If we upload any image so this system is generate relevant caption for it.

You have to write a computer program that takes an image as input and produces a relevant caption as output. This assumes familiarity with basic Deep Learning concepts like Multi-layered Perceptrons, Convolution Neural Networks, Recurrent Neural Networks, Transfer Learning, Gradient Descent, Backpropagation, Overfitting, Probability, Text Processing, Python syntax and data structures, Keras library, etc.

3.3 Requirement Gathering

3.3.1 Dataset

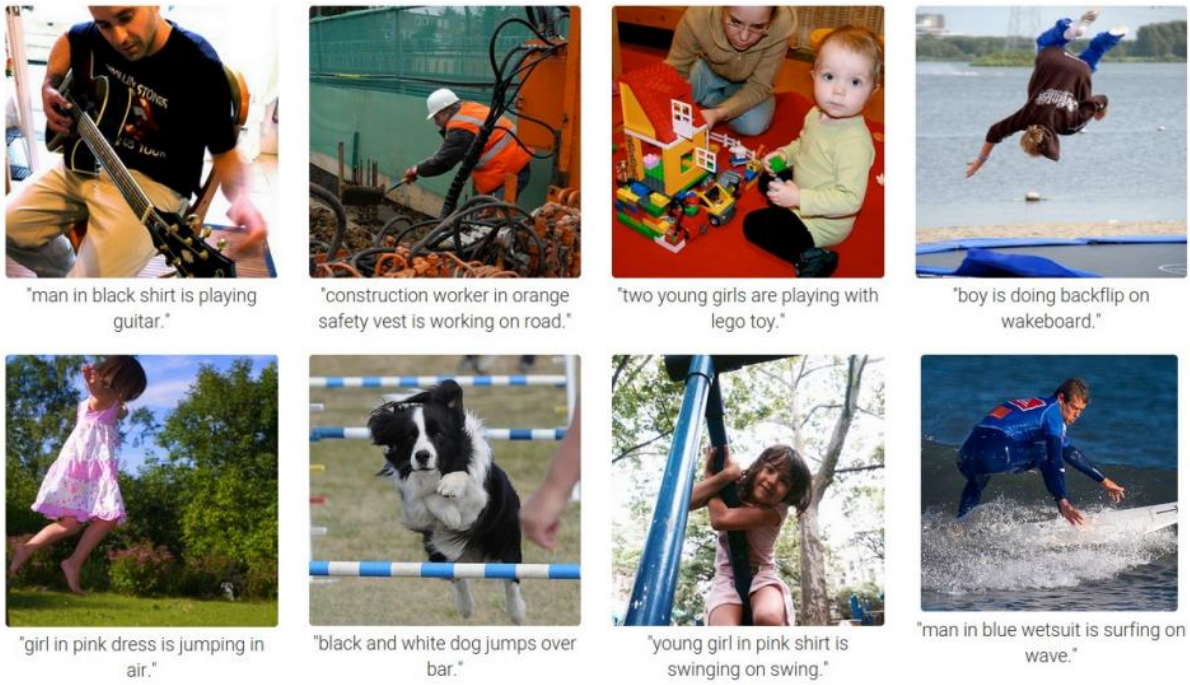
There are many open source datasets available for this problem, like Flickr 8k (containing8k images), Flickr 30k (containing 30k images), MS COCO (containing 180k images), etc. But for the purpose of this project, we have used the **Flickr 8k** dataset which can be easily downloaded by filling the form provided by the University of Illinois at Urbana-Champaign. Also training a model with large number of images may not be feasible on a system which is not a very high end PC/Laptop. This dataset contains **8000 images each with 5 captions**.

These images are bifurcated as follows:

Training Set — 6000 images,

Dev Set — 1000 images,

Test Set — 1000 images.



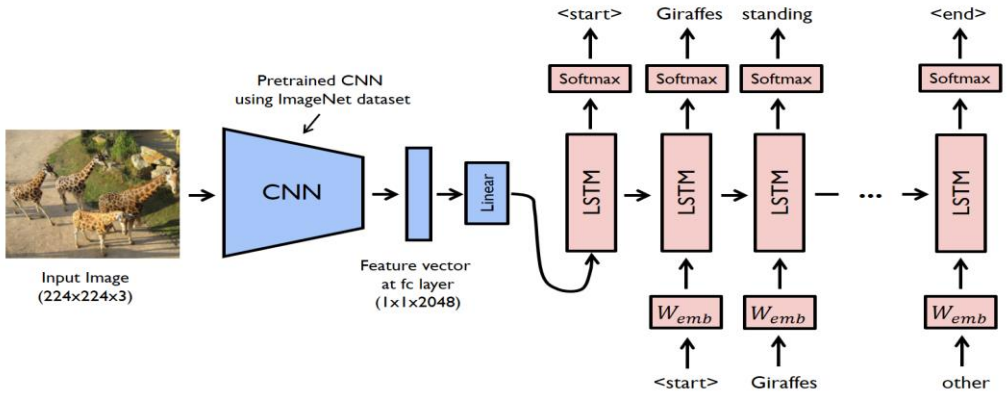
Along with images, the dataset also contains some text files related to the images. One of the files is “Flickr8k.token.txt” which contains the name of each image along with its 5 captions. We can read this file as follows:

```
filename = "/dataset/TextFiles/Flickr8k.token.txt"
file = open(filename, 'r')
doc = file.read()
```

Sample text File

101654506_8eb26cfb60.jpg#0 A brown and white dog is running through the snow .101654506_8eb26cfb60.jpg#1 A dog is running in the snow101654506_8eb26cfb60.jpg#2 A dog running through snow .101654506_8eb26cfb60.jpg#3 a white and brown dog is running through a snow covered field .101654506_8eb26cfb60.jpg#4 The white and brown dog is running over the surface of the snow . 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 .

3.3.2 Step by Step Procedure



Step 1: Data Cleaning

The first step is data cleaning. we generally perform some basic cleaning like lower-casing all the words (otherwise“hello” and “Hello” will be regarded as two separate words), removing special tokens (like ‘%’, ‘$’, ‘#’, etc.), eliminating words which contain numbers (like ‘hey199’, etc.).

We create a vocabulary of all the unique words present across all the 8000*5 (i.e. 40000) image captions (**corpus**) in the data set.

We write all these captions along with their image names in a new file namely, "*descriptions.txt"* and save it on the disk.

We consider only those words which **occur at least 10 times** in the entire corpus.

Step 2: Loading the training set

The text file "Flickr_8k.trainImages.txt" contains the names of the images that belong to the training set. So we load these names into a list "train".

Now, we load the descriptions of these images from "descriptions.txt" (saved on the hard disk) in the Python dictionary "train_descriptions".

However, when we load them, we will add two tokens in every caption as follows (significance explained later):

'**startseq**' -> This is a start sequence token which will be added at the start of every caption.

'**endseq**' -> This is an end sequence token which will be added at the end of every caption.

Step 3: Data Pre processing

We need to process images and captions

<u>Image pre processing:</u>

Next step is to convert every image into a fixed sized vector which can then be fed as input to the neural network.

we pass every image to this model to get the corresponding 2048 length feature vector.

we encode all the test images and save them in the file "**encoded_test_images.pkl**".

Pre processing caption:

Captions are out targets. This is the output we expect form our model. Here we predict the caption word by word. We need to encode each word into a fixed sized vector.

Step 4: Data preperatio using generator:

In this step we prepare our data in a way which will be convenient to be given as input to the deep learning model.

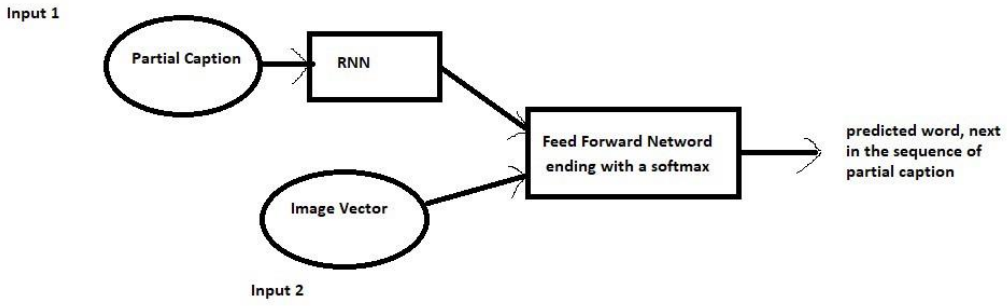
Data Generators are a functionality which is natively implemented in Python. The ImageDataGenerator class provided by the Keras API is nothing but an implementation of generator function in Python.

Step 5 : Word Embedding

In this step we map every word (index) to a 200-long vector and for this purpose,

The Model Architecture:

The following diagram shows high level model architecture.



The **LSTM (Long Short Term Memory)** layer is nothing but a specialized Recurrent Neural Network to process the sequence input .

Model Summary:

```
model.summary()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_4 (InputLayer)         (None, 34)                0
_________________________________________________________________________________________________
input_3 (InputLayer)         (None, 2048)              0
_________________________________________________________________________________________________
embedding_2 (Embedding)      (None, 34, 200)           330400      input_4[0][0]
_________________________________________________________________________________________________
dropout_3 (Dropout)          (None, 2048)              0           input_3[0][0]
_________________________________________________________________________________________________
dropout_4 (Dropout)          (None, 34, 200)           0           embedding_2[0][0]
_________________________________________________________________________________________________
dense_2 (Dense)              (None, 256)               524544      dropout_3[0][0]
_________________________________________________________________________________________________
lstm_2 (LSTM)                (None, 256)               467968      dropout_4[0][0]
_________________________________________________________________________________________________
add_2 (Add)                  (None, 256)               0           dense_2[0][0]
                                                                   lstm_2[0][0]
_________________________________________________________________________________________________
dense_3 (Dense)              (None, 256)               65792       add_2[0][0]
_________________________________________________________________________________________________
dense_4 (Dense)              (None, 1652)              424564      dense_3[0][0]
=================================================================================================
Total params: 1,813,268
Trainable params: 1,813,268
Non-trainable params: 0
```

We have used adam optimizer to optimize our output.

Finally the weights of the model are updated through backpropagation algorithm and the model will learn to output a word, given an image feature vector and a partial caption.

Hyper parameters during training:

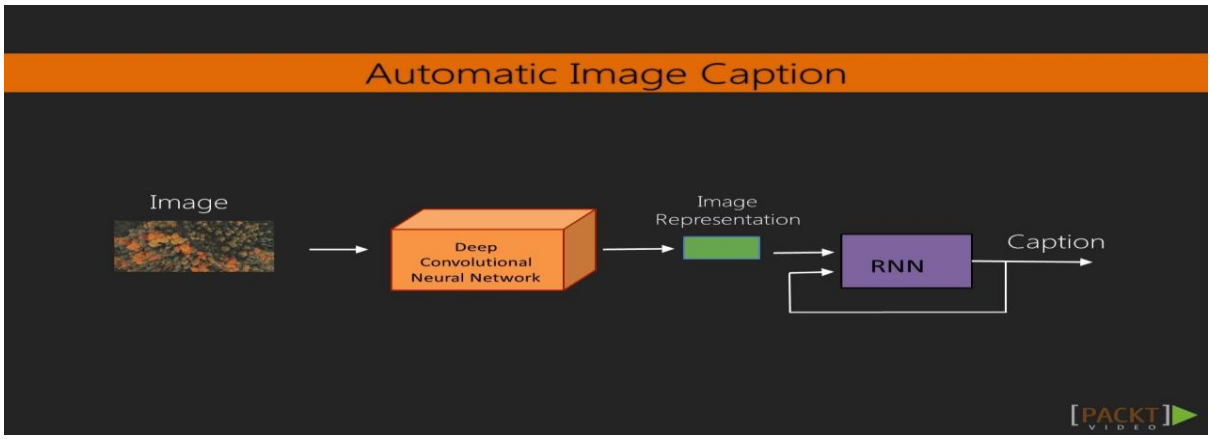
The model was then trained for 30 epochs with the initial learning rate of 0.001 and 3 pictures per batch (batch size). However after 20 epochs, the learning rate was reduced to 0.0001 and the model was trained on 6 pictures per batch.

Chapter 4

4. Implementation Details

4.1 Design Details

The diagram shows high level architecture of our project.




```
from pickle import dump
from os import listdir
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model
# extract features from each photo in the directory
def extract_features(directory):
 # load the model
 model = VGG16()
 # re-structure the model
 model.layers.pop()
 model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
 # summarize
```

```
 print(model.summary())
 # extract features from each photo
 features = dict()
 for name in listdir(directory):
  # load an image from file
  filename = directory + '/' + name
  image = load_img(filename, target_size=(224, 224))
  # convert the image pixels to a numpy array
  image = img_to_array(image)
  # reshape data for the model
  image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
  # prepare the image for the VGG model
  image = preprocess_input(image)
  # get features
  feature = model.predict(image, verbose=0)
  # get image id
  image_id = name.split('.')[0]
  # store feature
  features[image_id] = feature
  # print('>%s' % name)
 return features

# extract features from all images
directory = '/content/drive/My Drive/Image-Captioning-master1/Flickr8K_Data'
print('\nstart')
features = extract_features(directory)
print('\nend')
print('Extracted Features: %d' % len(features))
# save to file
```

```
dump(features, open('/content/drive/My Drive/Image-Captioning-
master1/Flickr8K_Text/features.pkl', 'wb'))

print('pickle')

import string

# load doc into memory

def load_doc(filename):

  # open the file as read only

  file = open(filename, 'r')

  # read all text

  text = file.read()

  # close the file

  file.close()

  return text

# extract descriptions for images

def load_descriptions(doc):

  mapping = dict()

  # process lines

  for line in doc.split('\n'):

    # split line by white space

    tokens = line.split()

    if len(line) < 2:

      continue

    # take the first token as the image id, the rest as the description

    image_id, image_desc = tokens[0], tokens[1:]

    # remove filename from image id

    image_id = image_id.split('.')[0]

    # convert description tokens back to string

    image_desc = ' '.join(image_desc)

    # create the list if needed

    if image_id not in mapping:
```

```
        mapping[image_id] = list()
      # store description
      mapping[image_id].append(image_desc)
    return mapping
def clean_descriptions(descriptions):
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
      for i in range(len(desc_list)):
        desc = desc_list[i]
        # tokenize
        desc = desc.split()
        # convert to lower case
        desc = [word.lower() for word in desc]
        # remove punctuation from each token
        desc = [w.translate(table) for w in desc]
        # remove hanging 's' and 'a'
        desc = [word for word in desc if len(word)>1]
        # remove tokens with numbers in them
        desc = [word for word in desc if word.isalpha()]
        # store as string
        desc_list[i] =  ' '.join(desc)
# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    # build a list of all description strings
    all_desc = set()
    for key in descriptions.keys():
      [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc
# save descriptions to file, one per line
```

```
def save_descriptions(descriptions, filename):
 lines = list()
 for key, desc_list in descriptions.items():
  for desc in desc_list:
   lines.append(key + ' ' + desc)
 data = '\n'.join(lines)
 file = open(filename, 'w')
 file.write(data)
 file.close()
filename = '/content/drive/My Drive/Image-Captioning-master1/Flickr8K_Text/Flickr8k.token.txt'
# load descriptions
doc = load_doc(filename)
# parse descriptions
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))
# clean descriptions
clean_descriptions(descriptions)
# summarize vocabulary
vocabulary = to_vocabulary(descriptions)
print('Vocabulary Size: %d' % len(vocabulary))
print(vocabulary)
# save to file
save_descriptions(descriptions, '/content/drive/My Drive/Image-Captioning-
master1/Flickr8K_Text/descriptions.txt')
################################################################################
from numpy import array
from pickle import load
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
```

```
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense, BatchNormalization
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import add
from tensorflow.keras.layers import Concatenate
from tensorflow.keras.callbacks import ModelCheckpoint
# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text
# load a pre-defined list of photo identifiers
def load_set(filename):
  doc = load_doc(filename)
  dataset = list()
  # process line by line
  for line in doc.split('\n'):
    # skip empty lines
    if len(line) < 1:
      continue
    # get the image identifier
    identifier = line.split('.')[0]
```

```
    dataset.append(identifier)
  return set(dataset)

# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
  # load document
  doc = load_doc(filename)
  descriptions = dict()
  for line in doc.split('\n'):
    # split line by white space
    tokens = line.split()
    # split id from description
    image_id, image_desc = tokens[0], tokens[1:]
    # skip images not in the set
    if image_id in dataset:
      # create list
      if image_id not in descriptions:
        descriptions[image_id] = list()
      # wrap description in tokens
      desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
      # store
      descriptions[image_id].append(desc)
  return descriptions

# load photo features
def load_photo_features(filename, dataset):
  # load all features
  all_features = load(open(filename, 'rb'))
  # filter features
  features = {k: all_features[k] for k in dataset}
```

```
  return features

# covert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
  all_desc = list()
  for key in descriptions.keys():
    [all_desc.append(d) for d in descriptions[key]]
  return all_desc

# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
  lines = to_lines(descriptions)
  tokenizer = Tokenizer()
  tokenizer.fit_on_texts(lines)
  return tokenizer

# calculate the length of the description with the most words
def max_length(descriptions):
  lines = to_lines(descriptions)
  return max(len(d.split()) for d in lines)
 from pickle import dump
from pickle import load
# create sequences of images, input sequences and output words for an image
def create_sequences(tokenizer, max_length, descriptions, photos):
        X1, X2, y = list(), list(), list()
        # walk through each image identifier
        for key, desc_list in descriptions.items():
                # walk through each description for the image
                for desc in desc_list:
                        # encode the sequence
```

```
                seq = tokenizer.texts_to_sequences([desc])[0]
                # split one sequence into multiple X,y pairs
                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(photos[key][0])
                        X2.append(in_seq)
                        y.append(out_seq)
        return array(X1), array(X2), array(y)
# train dataset
# load training dataset (6K)
filename = '/content/drive/My Drive/Image-Captioning-master1/Flickr8K_Text/Flickr_8k.trainImages.txt'
#train = load_set(filename)
train=load_doc(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('/content/drive/My Drive/Image-Captioning-master1/Flickr8K_Text/descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# photo features
train_features = load_photo_features('/content/drive/My Drive/Image-Captioning-master1/Flickr8K_Text/features.pkl', train)
print('Photos: train=%d' % len(train_features))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
```

```
dump(tokenizer, open('/content/drive/My Drive/Image-Captioning-master1/Flickr8K_Text/tokenizer.pkl', 'wb'))

vocab_size = len(tokenizer.word_index) + 1

print('Vocabulary Size: %d' % vocab_size)

# determine the maximum sequence length

max_length = max_length(train_descriptions)

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

# prepare sequences

X1train, X2train, ytrain = create_sequences(tokenizer, max_length, train_descriptions, train_features)

# dev dataset

# load test set

filename = '/content/drive/My Drive/Image-Captioning-master1/Flickr8K_Text/Flickr_8k.devImages.txt'

test = load_set(filename)

print('Dataset: %d' %len(test))

# descriptions

test_descriptions = load_clean_descriptions('/content/drive/My Drive/Image-Captioning-master1/Flickr8K_Text/descriptions.txt', test)

print('Descriptions: test=%d' % len(test_descriptions))

# photo features

test_features = load_photo_features('/content/drive/My Drive/Image-Captioning-master1/Flickr8K_Text/features.pkl', test)

print('Photos: test=%d' % len(test_features))

# prepare sequences

X1test, X2test, ytest = create_sequences(tokenizer, max_length, test_descriptions, test_features)


# fit model

def define_model(vocab_size, max_length):

    # feature extractor model

    inputs1 = Input(shape=(1000,))

    fe1 = Dropout(0.3)(inputs1)
```

```
    fe2 = Dense(256, activation='relu')(fe1)
    # sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.3)(se1)

    #merge layer
    merge = add([fe2, se2])
    #language model
    decoder1 = LSTM(256)(merge)

    #fully connected
    decoder2 = Dense(256, activation='relu')(decoder1)
    bn = BatchNormalization()(decoder2)
    do = Dropout(0.1)(bn)
    decoder3 = Dense(512, activation='relu')(do)
    bn = BatchNormalization()(decoder3)
    do = Dropout(0.1)(bn)
    outputs = Dense(vocab_size, activation='softmax')(do)
    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    # summarize model
    print(model.summary())
    return model

# define the model
model = define_model(vocab_size, max_length)
# define checkpoint callback
filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
```

```
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# fit model

model.fit([X1train, X2train], ytrain, epochs=20, verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest))

###############################################################################

from numpy import argmax

from pickle import load

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences

from keras.models import load_model

from nltk.translate.bleu_score import corpus_bleu


# generate a description for an image

def generate_desc(model, tokenizer, photo, max_length):

        # seed the generation process

        in_text = 'startseq'

        # iterate over the whole length of the sequence

        for i in range(max_length):

                # integer encode input sequence

                sequence = tokenizer.texts_to_sequences([in_text])[0]

                # pad input

                sequence = pad_sequences([sequence], maxlen=max_length)

                # predict next word

                yhat = model.predict([photo,sequence], verbose=0)

                # convert probability to integer

                yhat = argmax(yhat)

                # map integer to word

                word = word_for_id(yhat, tokenizer)

                # stop if we cannot map the word

                if word is None:
```

```
                break
            # append as input for generating the next word
            in_text += ' ' + word
            # stop if we predict the end of the sequence
            if word == 'endseq':
                break
    return in_text
# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None



# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
    actual, predicted = list(), list()
    # step over the whole set
    for key, desc_list in descriptions.items():
        # generate description
        yhat = generate_desc(model, tokenizer, photos[key], max_length)
        # store actual and predicted
        references = [d.split() for d in desc_list]
        actual.append(references)
        predicted.append(yhat.split())
    # calculate BLEU score
    print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
    print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
```

```
        print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
###############################################################################
from numpy import argmax
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.models import load_model
from pickle import load
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

def generate_desc(model, tokenizer, photo, max_length):
        # seed the generation process
        in_text = 'startseq'
        # iterate over the whole length of the sequence
        for i in range(max_length):
                # integer encode input sequence
                sequence = tokenizer.texts_to_sequences([in_text])[0]
                # pad input
                sequence = pad_sequences([sequence], maxlen=max_length)
                # predict next word
                yhat = model.predict([photo,sequence], verbose=0)
                # convert probability to integer
                yhat = argmax(yhat)
                # map integer to word
```

```
            word = word_for_id(yhat, tokenizer)
            # stop if we cannot map the word
            if word is None:
                break
            # append as input for generating the next word
            in_text += ' ' + word
            # stop if we predict the end of the sequence
            if word == 'endseq':
                break
    return in_text

# extract features from each photo in the directory
def extract_features(filename):
  # load the model
  model = VGG16()
  # re-structure the model
  model.layers.pop()
  model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
  # load the photo
  image = load_img(filename, target_size=(224, 224))
  # convert the image pixels to a numpy array
  image = img_to_array(image)
  # reshape data for the model
  image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # prepare the image for the VGG model
  image = preprocess_input(image)
  # get features
  feature = model.predict(image, verbose=0)
  del model
  return feature
```

```
# map an integer to a word
def word_for_id(integer, tokenizer):
	for word, index in tokenizer.word_index.items():
		if index == integer:
			return word
	return None



# load the tokenizer
tokenizer = load(open('tokenizer.pkl', 'rb'))
# pre-define the max sequence length (from training)
max_length = 34
# load the model
model = load_model('/content/model-ep004-loss3.210-val_loss3.454.h5')
# load and prepare the photograph
photos = ['/content/image/test/example2.jpg']
for fname in photos:
  photo = extract_features(fname)

  # generate description
  description = generate_desc(model, tokenizer, photo, max_length)
  print(description[8:-6])
  img=mpimg.imread(fname)
  imgplot = plt.imshow(img)
  plt.axis('off')

  plt.show()
del model
```

4.2. The Hardware Requirements

As we are working on predicting the tourists that visit a country for the given month, the hardware Requirements are not too much. We have designed web pages as front end that would require the user to have a laptop or pc where he/she can go to our website and enter the asked input for which they'll get the output on the same page.
So the hardware requirements for running our project are:

- Laptop for the user to visit the page and get predictions for their inputs.
- Operating System :Windows 10
- RAM : 3 GB/4GB
- Processor : I3/I5
- Hard Disk :100 GB

4.3 The Software Requirements

- Using a larger dataset.
- Changing the model architecture, e.g. include an attention module.
- Doing more hyper parameter tuning (learning rate, batch size, number of layers, number of units, dropout rate, batch normalization etc.).
- Use the cross validation set to understand overfitting.
- Using Beam Search instead of Greedy Search during Inference.
- Inception v3 is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the flicker8k dataset.

Chapter 5

Results and Discussion

To evaluate the model we try to generate caption for out test dataset.

We stop when either of the below two conditions is met:

- We encounter an '**endseq**' token which means the model thinks that this is the end of the caption. (You should now understand the importance of the 'endseq' token)
- We reach a maximum **threshold** of the number of words generated by the model.

With this we get _____% accuracy on our test data set.

The black cat is walking on grass

References

[1] https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8

[2] https://www.tensorflow.org/tutorials/text/image_captioning

[3] https://arxiv.org/pdf/1909.11059v3.pdf

[4] https://arxiv.org/abs/1905.04899v2

[5] https://blog.paperspace.com/image-captioning-with-ai/