

Python based Project – Learn to Build Image Caption Generator with CNN & LSTM

Free Python course with 57 real-time projects - [Learn Python in Hindi](#) / [Learn Python in English](#)

Project based on Python – Image Caption Generator

You saw an image and your brain can easily tell what the image is about, but can a computer tell what the image is representing? Computer vision researchers worked on this a lot and they considered it impossible until now! With the advancement in Deep learning techniques, availability of huge datasets and computer power, we can build models that can generate captions for an image.

This is what we are going to implement in this Python based project where we will use deep learning techniques of Convolutional Neural Networks and a type of Recurrent Neural Network (LSTM) together.

What is Image Caption Generator?

Image caption generator is a task that involves computer vision and natural language processing concepts to recognize the context of an image and describe them in a natural language like English.

Image Caption Generator with CNN – About the Python based Project

The objective of our project is to learn the concepts of a CNN and LSTM model and build a working model of Image caption generator by implementing CNN with LSTM.

In this Python project, we will be implementing the caption generator using [*CNN \(Convolutional Neural Networks\)*](#) and LSTM (Long short term memory). The image features will be extracted from Xception which is a CNN model trained on the imagenet dataset and then we feed the features into the LSTM model which will be responsible for generating the image captions.

The Dataset of Python based Project

For the image caption generator, we will be using the Flickr_8K dataset. There are also other big datasets like Flickr_30K and MSCOCO dataset but it can take weeks just to train the network so we will be using a small Flickr8k dataset. The advantage of a huge dataset is that we can build better models.

Thanks to Jason Brownlee for providing a direct link to download the dataset (Size: 1GB).

- [Flicker8k Dataset](#)
- [Flickr 8k text](#)

The Flickr_8k_text folder contains file Flickr8k.token which is the main file of our dataset that contains image name and their respective captions separated by newline("\n").

Pre-requisites

This project requires good knowledge of Deep learning, Python, working on Jupyter notebooks, Keras library, Numpy, and [*Natural language processing*](#). Make sure you have installed all the following necessary libraries:

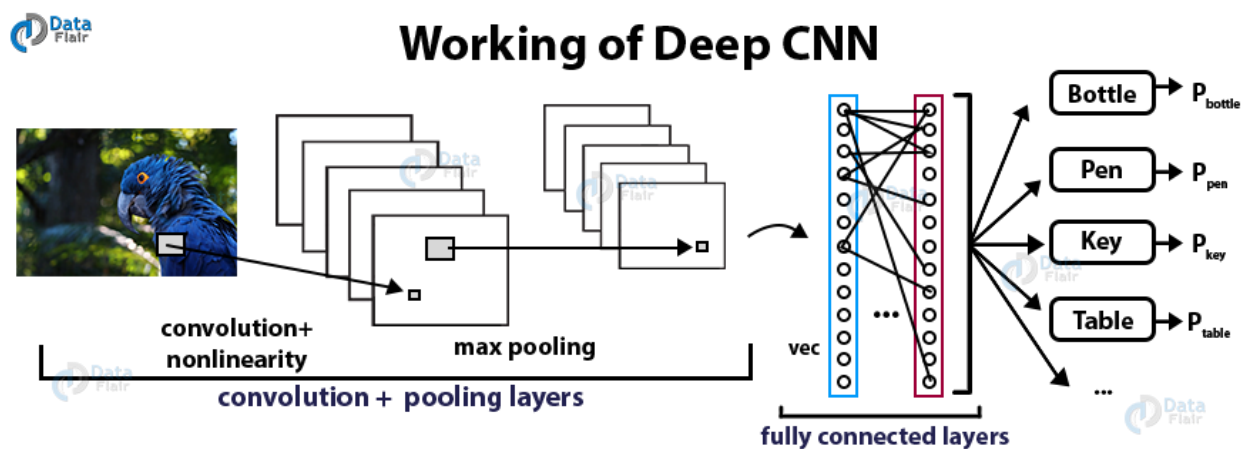
- pip install tensorflow
- keras
- pillow
- numpy
- tqdm
- jupyterlab

Image Caption Generator – Python based Project

What is CNN?

Convolutional Neural networks are specialized deep neural networks which can process the data that has input shape like a 2D matrix. Images are easily represented as a 2D matrix and CNN is very useful in working with images.

CNN is basically used for image classifications and identifying if an image is a bird, a plane or Superman, etc.



It scans images from left to right and top to bottom to pull out important features from the image and combines the feature to classify images. It can handle the images that have been translated, rotated, scaled and changes in perspective.

What is LSTM?

LSTM stands for **Long short term memory**, they are a type of RNN (**recurrent neural network**) which is well suited for sequence prediction problems. Based on the previous text, we can predict what the next word will be. It has proven itself effective from the traditional RNN by overcoming the limitations of RNN which had short term memory. LSTM can carry out relevant information throughout the processing of inputs and with a forget gate, it discards non-relevant information.

This is what an LSTM cell looks like –

LSTM Cell Structure

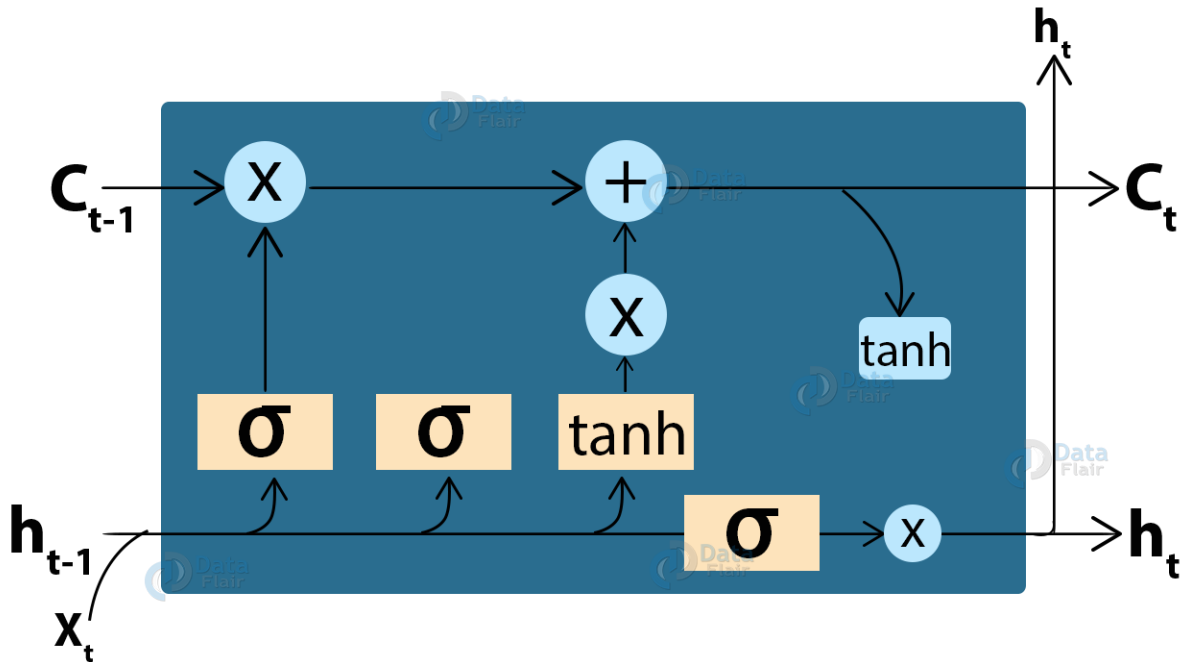
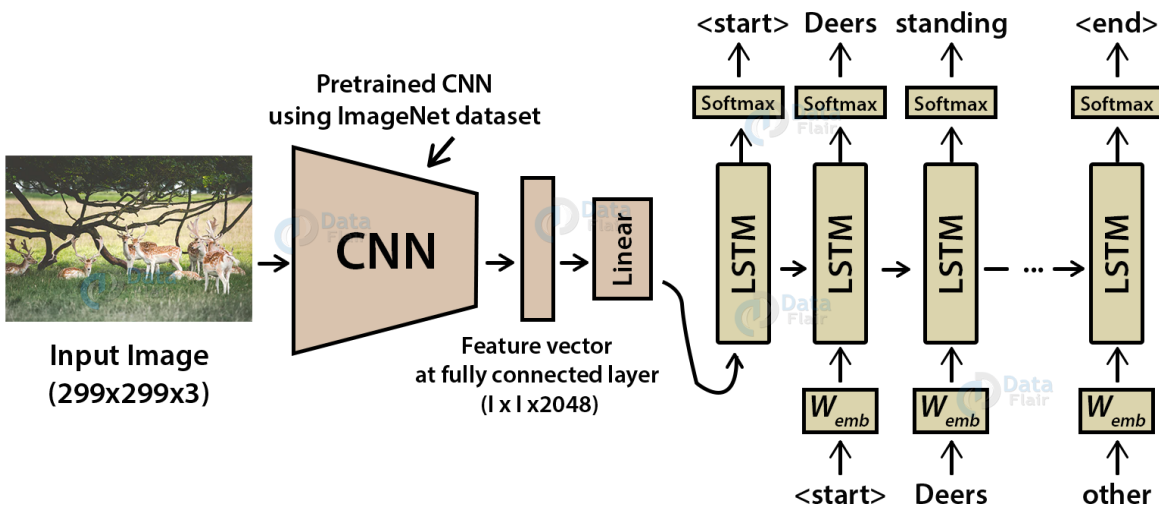


Image Caption Generator Model

So, to make our image caption generator model, we will be merging these architectures. It is also called a CNN-RNN model.

- CNN is used for extracting features from the image. We will use the pre-trained model Xception.
- LSTM will use the information from CNN to help generate a description of the image.

Model - Image Caption Generator



Project File Structure

Downloaded from dataset:

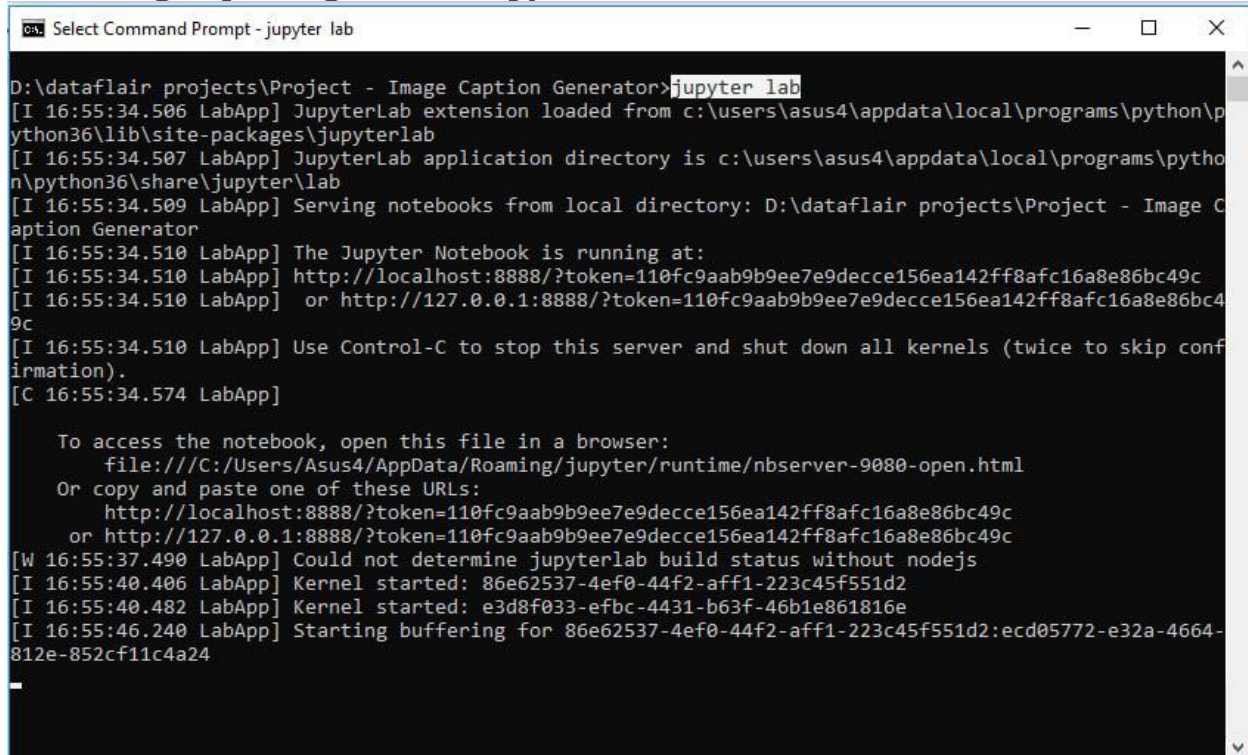
- **Flicker8k_Dataset** – Dataset folder which contains 8091 images.
- **Flickr_8k_text** – Dataset folder which contains text files and captions of images.

The below files will be created by us while making the project.

- **Models** – It will contain our trained models.
- **Descriptions.txt** – This text file contains all image names and their captions after preprocessing.
- **Features.p** – Pickle object that contains an image and their feature vector extracted from the Xception pre-trained CNN model.
- **Tokenizer.p** – Contains tokens mapped with an index value.
- **Model.png** – Visual representation of dimensions of our project.
- **Testing_caption_generator.py** – Python file for generating a caption of any image.
- **Training_caption_generator.ipynb** – Jupyter notebook in which we train and build our image caption generator.

Building the Python based Project

Let's start by initializing the jupyter notebook server by typing `jupyter lab` in the console of your project folder. It will open up the interactive Python notebook where you can run your code. Create a Python3 notebook and name it **training_caption_generator.ipynb**



```
Select Command Prompt - jupyter lab

D:\dataflair projects\Project - Image Caption Generator>jupyter lab
[I 16:55:34.506 LabApp] JupyterLab extension loaded from c:\users\asus4\appdata\local\programs\python\python36\lib\site-packages\jupyterlab
[I 16:55:34.507 LabApp] JupyterLab application directory is c:\users\asus4\appdata\local\programs\python\python36\share\jupyter\lab
[I 16:55:34.509 LabApp] Serving notebooks from local directory: D:\dataflair projects\Project - Image Caption Generator
[I 16:55:34.510 LabApp] The Jupyter Notebook is running at:
[I 16:55:34.510 LabApp] http://localhost:8888/?token=110fc9aab9b9ee7e9decce156ea142ff8afc16a8e86bc49c
[I 16:55:34.510 LabApp] or http://127.0.0.1:8888/?token=110fc9aab9b9ee7e9decce156ea142ff8afc16a8e86bc49c
[I 16:55:34.510 LabApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 16:55:34.574 LabApp]

To access the notebook, open this file in a browser:
    file:///C:/Users/Asus4/AppData/Roaming/jupyter/runtime/nbserver-9080-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=110fc9aab9b9ee7e9decce156ea142ff8afc16a8e86bc49c
    or http://127.0.0.1:8888/?token=110fc9aab9b9ee7e9decce156ea142ff8afc16a8e86bc49c
[W 16:55:37.490 LabApp] Could not determine jupyterlab build status without nodejs
[I 16:55:40.406 LabApp] Kernel started: 86e62537-4ef0-44f2-aff1-223c45f551d2
[I 16:55:40.482 LabApp] Kernel started: e3d8f033-efbc-4431-b63f-46b1e861816e
[I 16:55:46.240 LabApp] Starting buffering for 86e62537-4ef0-44f2-aff1-223c45f551d2:ecd05772-e32a-4664-812e-852cf11c4a24
```

1. First, we import all the necessary packages

import string

import numpy as np

from PIL import Image

import os

from pickle import dump, load

import numpy as np

from keras.applications.xception import Xception, preprocess_input

from keras.preprocessing.image import load_img, img_to_array

from keras.preprocessing.text import Tokenizer

```

from keras.preprocessing.sequence import pad_sequences

from keras.utils import to_categorical

from keras.layers.merge import add

from keras.models import Model, load_model

from keras.layers import Input, Dense, LSTM, Embedding, Dropout

# small library for seeing the progress of loops.

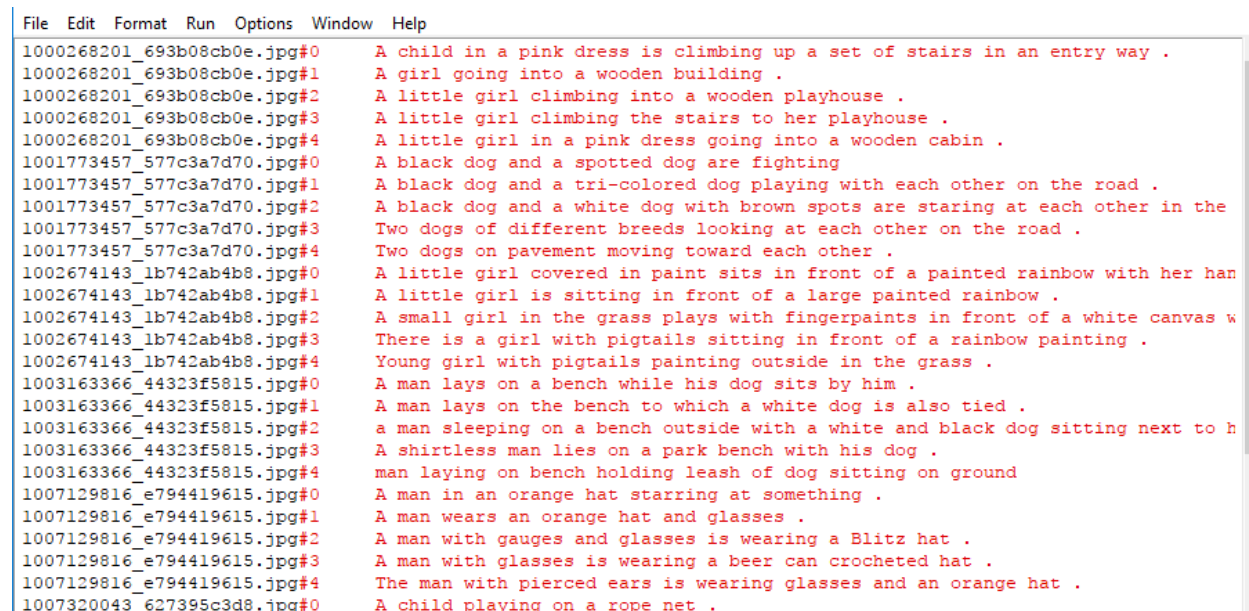
from tqdm import tqdm_notebook as tqdm

tqdm().pandas()

```

2. Getting and performing data cleaning

The main text file which contains all image captions is **Flickr8k.token** in our **Flickr_8k_text** folder.
Have a look at the file –



```

File Edit Format Run Options Window Help
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 .
1001773457_577c3a7d70.jpg#0 A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg#1 A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg#2 A black dog and a white dog with brown spots are staring at each other in the
1001773457_577c3a7d70.jpg#3 Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg#4 Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg#0 A little girl covered in paint sits in front of a painted rainbow with her han
1002674143_1b742ab4b8.jpg#1 A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg#2 A small girl in the grass plays with fingerpaints in front of a white canvas w
1002674143_1b742ab4b8.jpg#3 There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg#4 Young girl with pigtails painting outside in the grass .
1003163366_44323f5815.jpg#0 A man lays on a bench while his dog sits by him .
1003163366_44323f5815.jpg#1 A man lays on the bench to which a white dog is also tied .
1003163366_44323f5815.jpg#2 a man sleeping on a bench outside with a white and black dog sitting next to h
1003163366_44323f5815.jpg#3 A shirtless man lies on a park bench with his dog .
1003163366_44323f5815.jpg#4 man laying on bench holding leash of dog sitting on ground
1007129816_e794419615.jpg#0 A man in an orange hat starring at something .
1007129816_e794419615.jpg#1 A man wears an orange hat and glasses .
1007129816_e794419615.jpg#2 A man with gauges and glasses is wearing a Blitz hat .
1007129816_e794419615.jpg#3 A man with glasses is wearing a beer can crocheted hat .
1007129816_e794419615.jpg#4 The man with pierced ears is wearing glasses and an orange hat .
1007320043_627395c3d8.jpg#0 A child playing on a rope net .

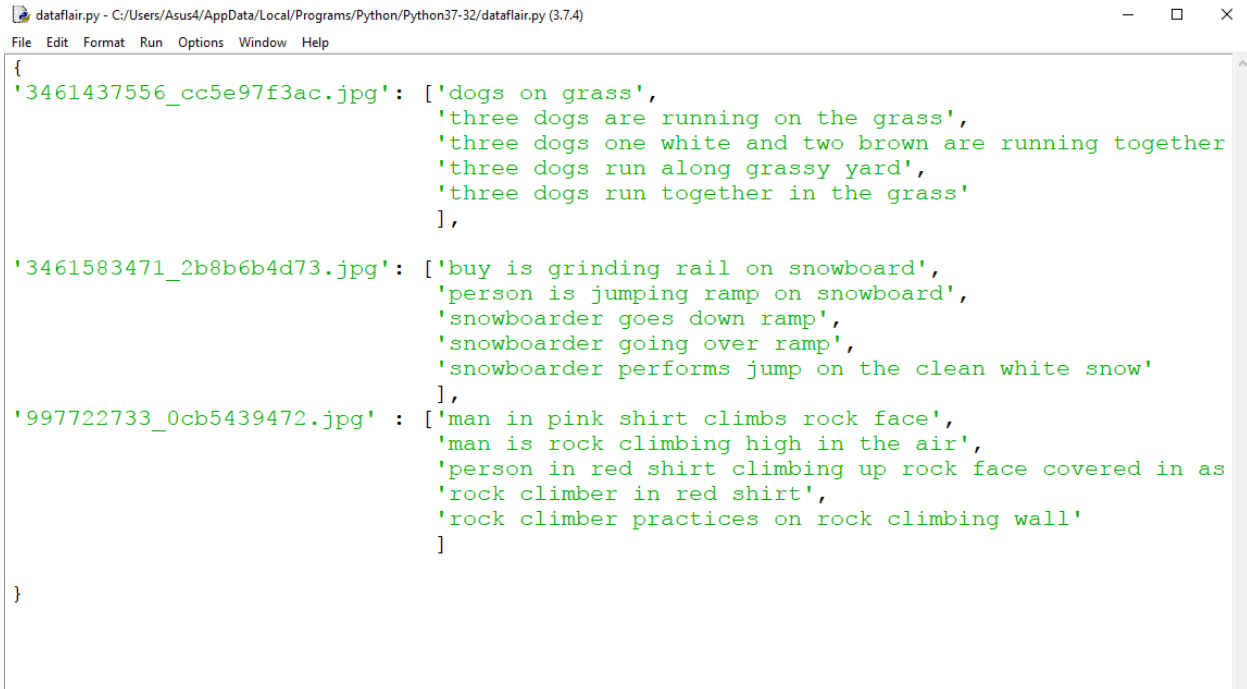
```

The format of our file is image and caption separated by a new line (“\n”).

Each image has 5 captions and we can see that #(0 to 5)number is assigned for each caption.

We will define 5 functions:

- **load_doc(filename)** – For loading the document file and reading the contents inside the file into a string.
- **all_img_captions(filename)** – This function will create a **descriptions** dictionary that maps images with a list of 5 captions. The descriptions dictionary will look something like this:



```
dataflair.py - C:/Users/Asus4/AppData/Local/Programs/Python/Python37-32/dataflair.py (3.7.4)
File Edit Format Run Options Window Help
{
'3461437556_cc5e97f3ac.jpg': ['dogs on grass',
                              'three dogs are running on the grass',
                              'three dogs one white and two brown are running together',
                              'three dogs run along grassy yard',
                              'three dogs run together in the grass'
                              ],
'3461583471_2b8b6b4d73.jpg': ['buy is grinding rail on snowboard',
                              'person is jumping ramp on snowboard',
                              'snowboarder goes down ramp',
                              'snowboarder going over ramp',
                              'snowboarder performs jump on the clean white snow'
                              ],
'997722733_0cb5439472.jpg' : ['man in pink shirt climbs rock face',
                              'man is rock climbing high in the air',
                              'person in red shirt climbing up rock face covered in as',
                              'rock climber in red shirt',
                              'rock climber practices on rock climbing wall'
                              ]
}
```

- **cleaning_text(descriptions)** – This function takes all descriptions and performs data cleaning. This is an important step when we work with textual data, according to our goal, we decide what type of cleaning we want to perform on the text. In our case, we will be removing punctuations, converting all text to lowercase and removing words that contain numbers.
So, a caption like “A man riding on a three-wheeled wheelchair” will be transformed into “man riding on three wheeled wheelchair”
- **text_vocabulary(descriptions)** – This is a simple function that will separate all the unique words and create the vocabulary from all the descriptions.
- **save_descriptions(descriptions, filename)** – This function will create a list of all the descriptions that have been preprocessed and store them into a file. We will create a descriptions.txt file to store all the captions. It will look something like this:

File Edit Format Run Options Window Help

```
1000268201_693b08cb0e.jpg      child in pink dress is climbing up set of stairs in
1000268201_693b08cb0e.jpg      girl going into wooden building
1000268201_693b08cb0e.jpg      little girl climbing into wooden playhouse
1000268201_693b08cb0e.jpg      little girl climbing the stairs to her playhouse
1000268201_693b08cb0e.jpg      little girl in pink dress going into wooden cabin
1001773457_577c3a7d70.jpg      black dog and spotted dog are fighting
1001773457_577c3a7d70.jpg      black dog and tricolored dog playing with each other
1001773457_577c3a7d70.jpg      black dog and white dog with brown spots are staring
1001773457_577c3a7d70.jpg      two dogs of different breeds looking at each other
1001773457_577c3a7d70.jpg      two dogs on pavement moving toward each other
1002674143_1b742ab4b8.jpg      little girl covered in paint sits in front of paint
1002674143_1b742ab4b8.jpg      little girl is sitting in front of large painted re
1002674143_1b742ab4b8.jpg      small girl in the grass plays with fingerpaints in
1002674143_1b742ab4b8.jpg      there is girl with pigtails sitting in front of rai
1002674143_1b742ab4b8.jpg      young girl with pigtails painting outside in the gr
1003163366_44323f5815.jpg      man lays on bench while his dog sits by him
```

Code :

Loading a text file into memory

```
def load_doc(filename):
```

Opening the file as read only

```
file = open(filename, 'r')
```

```
text = file.read()
```

```
file.close()
```

```
return text
```

get all imgs with their captions

```
def all_img_captions(filename):
```

```
file = load_doc(filename)
```

```
captions = file.split('\n')
```

```
descriptions = {}
```

```
for caption in captions[:-1]:
```

```
img, caption = caption.split('\t')
```

```
if img[:-2] not in descriptions:
```

```
descriptions[img[:-2]] = [ caption ]
```

else:

descriptions[img[:-2]].append(caption)

return descriptions

#Data cleaning- lower casing, removing punctuations and words containing numbers

def cleaning_text(captions):

table = str.maketrans("",string.punctuation)

for img,caps **in** captions.items():

for i,img_caption **in** enumerate(caps):

img_caption.replace("-", " ")

desc = img_caption.split()

#converts to lowercase

desc = [word.lower() **for** word **in** desc]

#remove punctuation from each token

desc = [word.translate(table) **for** word **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())]

#convert back to string

img_caption = ' '.join(desc)

captions[img][i]= img_caption

return captions

def text_vocabulary(descriptions):

build vocabulary of all unique words

vocab = set()

```

for key in descriptions.keys():

[vocab.update(d.split()) for d in descriptions[key]]

return vocab

#All descriptions in one file

def save_descriptions(descriptions, filename):

lines = list()

for key, desc_list in descriptions.items():

for desc in desc_list:

lines.append(key + '\t' + desc )

data = "\n".join(lines)

file = open(filename,"w")

file.write(data)

file.close()

# Set these path according to project folder in you system

dataset_text = "D:\dataflair projects\Project - Image Caption Generator\Flickr_8k_text"

dataset_images = "D:\dataflair projects\Project - Image Caption Generator\Flicker8k_Dataset"

#we prepare our text data

filename = dataset_text + "/" + "Flickr8k.token.txt"

#loading the file that contains all data

#mapping them into descriptions dictionary img to 5 captions

descriptions = all_img_captions(filename)

print("Length of descriptions =",len(descriptions))

#cleaning the descriptions

clean_descriptions = cleaning_text(descriptions)

#building vocabulary

```

```

vocabulary = text_vocabulary(clean_descriptions)

print("Length of vocabulary = ", len(vocabulary))

#saving each description to file

save_descriptions(clean_descriptions, "descriptions.txt")

```

3. Extracting the feature vector from all images

This technique is also called transfer learning, we don't have to do everything on our own, we use the pre-trained model that have been already trained on large datasets and extract the features from these models and use them for our tasks. We are using the Xception model which has been trained on imagenet dataset that had 1000 different classes to classify. We can directly import this model from the keras.applications . Make sure you are connected to the internet as the weights get automatically downloaded. Since the Xception model was originally built for imagenet, we will do little changes for integrating with our model. One thing to notice is that the Xception model takes $299 \times 299 \times 3$ image size as input. We will remove the last classification layer and get the 2048 feature vector.

```

model = Xception( include_top=False, pooling='avg' )

```

The function **extract_features()** will extract features for all images and we will map image names with their respective feature array. Then we will dump the features dictionary into a "features.p" pickle file.

Code:

```

def extract_features(directory):

    model = Xception( include_top=False, pooling='avg' )

    features = {}

    for img in tqdm(os.listdir(directory)):

        filename = directory + "/" + img

        image = Image.open(filename)

        image = image.resize((299,299))

        image = np.expand_dims(image, axis=0)

        #image = preprocess_input(image)

        image = image/127.5

```

```

image = image - 1.0

feature = model.predict(image)

features[img] = feature

return features

#2048 feature vector

features = extract_features(dataset_images)

dump(features, open("features.p", "wb"))

```

```

In [44]: # Now let's extract the features from our xception model
def extract_features(directory):
    model = Xception( include_top=False, pooling='avg' )
    features = {}
    for img in tqdm(os.listdir(directory)):
        filename = directory + "/" + img
        image = Image.open(filename)
        image = image.resize((299,299))
        image = np.expand_dims(image, axis=0)
        #image = preprocess_input(image)
        image = image/127.5
        image = image - 1.0

        feature = model.predict(image)
        features[img] = feature
    return features

```

```

In [45]: #2048 feature vector
features = extract_features(dataset_images)
dump(features, open("features.p", "wb"))

```

100%  8091/8091 [06:29<00:00, 20.78it/s]

```

In [21]: features = load(open("features.p", "rb"))

```

This process can take a lot of time depending on your system. I am using an Nvidia 1050 GPU for training purpose so it took me around 7 minutes for performing this task. However, if you are using CPU then this process might take 1-2 hours. You can comment out the code and directly load the features from our pickle file.

```
features = load(open("features.p","rb"))
```

4. Loading dataset for Training the model

In our **Flickr_8k_test** folder, we have **Flickr_8k.trainImages.txt** file that contains a list of 6000 image names that we will use for training.

For loading the training dataset, we need more functions:

- **load_photos(filename)** – This will load the text file in a string and will return the list of image names.
- **load_clean_descriptions(filename, photos)** – This function will create a dictionary that contains captions for each photo from the list of photos. We also append the <start> and <end> identifier for each caption. We need this so that our LSTM model can identify the starting and ending of the caption.
- **load_features(photos)** – This function will give us the dictionary for image names and their feature vector which we have previously extracted from the Xception model.

Code :

```
#load the data
```

```
def load_photos(filename):
```

```
    file = load_doc(filename)
```

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

```
    return photos
```

```
def load_clean_descriptions(filename, photos):
```

```
    #loading clean_descriptions
```

```
    file = load_doc(filename)
```

```
    descriptions = {}
```

```
    for line in file.split("\n"):
```

```
        words = line.split()
```

```
        if len(words)<1 :
```

```
            continue
```

```
        image, image_caption = words[0], words[1:]
```

```

if image in photos:

if image not in descriptions:

descriptions[image] = []

desc = '<start> ' + " ".join(image_caption) + ' <end>'

descriptions[image].append(desc)

return descriptions

def load_features(photos):

#loading all features

all_features = load(open("features.p","rb"))

#selecting only needed features

features = {k:all_features[k] for k in photos}

return features

filename = dataset_text + "/" + "Flickr_8k.trainImages.txt"

#train = loading_data(filename)

train_imgs = load_photos(filename)

train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)

train_features = load_features(train_imgs)

```

5. Tokenizing the vocabulary

Computers don't understand English words, for computers, we will have to represent them with numbers. So, we will map each word of the vocabulary with a unique index value. Keras library provides us with the tokenizer function that we will use to create tokens from our vocabulary and save them to a **"tokenizer.p"** pickle file.

Code:

```

#converting dictionary to clean list of descriptions

def dict_to_list(descriptions):

all_desc = []

```



```

for key in descriptions.keys():

[all_desc.append(d) for d in descriptions[key]]

return all_desc

#creating tokenizer class

#this will vectorise text corpus

#each integer will represent token in dictionary

from keras.preprocessing.text import Tokenizer

def create_tokenizer(descriptions):

desc_list = dict_to_list(descriptions)

tokenizer = Tokenizer()

tokenizer.fit_on_texts(desc_list)

return tokenizer

# give each word an index, and store that into tokenizer.p pickle file

tokenizer = create_tokenizer(train_descriptions)

dump(tokenizer, open('tokenizer.p', 'wb'))

vocab_size = len(tokenizer.word_index) + 1

vocab_size

```

Our vocabulary contains 7577 words.

We calculate the maximum length of the descriptions. This is important for deciding the model structure parameters. Max_length of description is 32.

```

#calculate maximum length of descriptions

def max_length(descriptions):

desc_list = dict_to_list(descriptions)

return max(len(d.split()) for d in desc_list)

max_length = max_length(descriptions)

```

max_length

6. Create Data generator

Let us first see how the input and output of our model will look like. To make this task into a supervised learning task, we have to provide input and output to the model for training. We have to train our model on 6000 images and each image will contain 2048 length feature vector and caption is also represented as numbers. This amount of data for 6000 images is not possible to hold into memory so we will be using a generator method that will yield batches.

The generator will yield the input and output sequence.

For example:

The input to our model is [x1, x2] and the output will be y, where x1 is the 2048 feature vector of that image, x2 is the input text sequence and y is the output text sequence that the model has to predict.

x1(feature vector)	x2(Text sequence)	y(word)
feature	start,	
feature	start, two	
feature	start, two, dogs	
feature	start, two, dogs, drink	
feature	start, two, dogs, drink, water	

```
#create input-output sequence pairs from the image description.
```

```
#data generator, used by model.fit_generator()
```

```
def data_generator(descriptions, features, tokenizer, max_length):
```

```
    while 1:
```

```
        for key, description_list in descriptions.items():
```

```
            #retrieve photo features
```

```
            feature = features[key][0]
```

```
input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list,
feature)
```

```
yield [[input_image, input_sequence], output_word]
```

```
def create_sequences(tokenizer, max_length, desc_list, feature):
```

```
    X1, X2, y = list(), list(), list()
```

```
    # 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(feature)
```

```
            X2.append(in_seq)
```

```
            y.append(out_seq)
```

```
    return np.array(X1), np.array(X2), np.array(y)
```

```
    #You can check the shape of the input and output for your model
```

```
[a,b],c = next(data_generator(train_descriptions, features, tokenizer, max_length))
```

```
a.shape, b.shape, c.shape
```

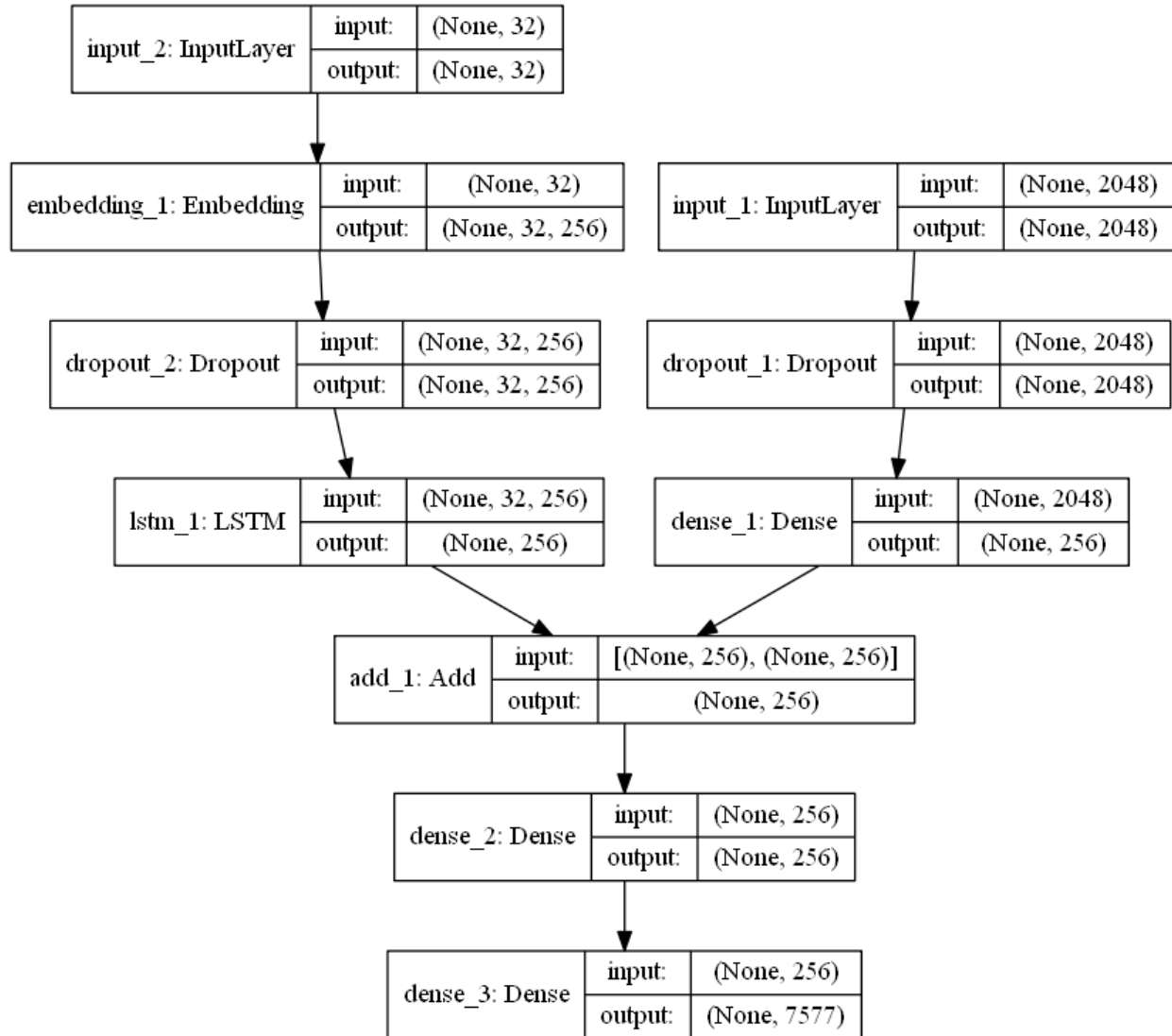
#((47, 2048), (47, 32), (47, 7577))

7. Defining the CNN-RNN model

To define the structure of the model, we will be using the Keras Model from Functional API. It will consist of three major parts:

- **Feature Extractor** – The feature extracted from the image has a size of 2048, with a dense layer, we will reduce the dimensions to 256 nodes.
- **Sequence Processor** – An embedding layer will handle the textual input, followed by the LSTM layer.
- **Decoder** – By merging the output from the above two layers, we will process by the dense layer to make the final prediction. The final layer will contain the number of nodes equal to our vocabulary size.

Visual representation of the final model is given below –



```
from keras.utils import plot_model
```

```
# define the captioning model
```

```
def define_model(vocab_size, max_length):
```

```
# features from the CNN model squeezed from 2048 to 256 nodes
```

```
inputs1 = Input(shape=(2048,))
```

```
fe1 = Dropout(0.5)(inputs1)
```

```
fe2 = Dense(256, activation='relu')(fe1)
```

```
# LSTM sequence model
```

```
inputs2 = Input(shape=(max_length,))
```

```

se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)

se2 = Dropout(0.5)(se1)

se3 = LSTM(256)(se2)

# Merging both models

decoder1 = add([fe2, se3])

decoder2 = Dense(256, activation='relu')(decoder1)

outputs = Dense(vocab_size, activation='softmax')(decoder2)

# 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())

plot_model(model, to_file='model.png', show_shapes=True)

return model

```

8. Training the model

To train the model, we will be using the 6000 training images by generating the input and output sequences in batches and fitting them to the model using `model.fit_generator()` method. We also save the model to our models folder. This will take some time depending on your system capability.

```

# train our model

print('Dataset: ', len(train_imgs))

print('Descriptions: train=', len(train_descriptions))

print('Photos: train=', len(train_features))

print('Vocabulary Size:', vocab_size)

print('Description Length: ', max_length)

model = define_model(vocab_size, max_length)

```

```

epochs = 10

steps = len(train_descriptions)

# making a directory models to save our models

os.mkdir("models")

for i in range(epochs):

generator = data_generator(train_descriptions, train_features, tokenizer, max_length)

model.fit_generator(generator, epochs=1, steps_per_epoch= steps, verbose=1)

model.save("models/model_" + str(i) + ".h5")

```

9. Testing the model

The model has been trained, now, we will make a separate file `testing_caption_generator.py` which will load the model and generate predictions. The predictions contain the max length of index values so we will use the same `tokenizer.p` pickle file to get the words from their index values.

Code:

```

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt

import argparse

ap = argparse.ArgumentParser()

ap.add_argument('-i', '--image', required=True, help="Image Path")

args = vars(ap.parse_args())

img_path = args['image']

def extract_features(filename, model):

try:

image = Image.open(filename)

except:

print("ERROR: Couldn't open image! Make sure the image path and extension is correct")

```



```

image = image.resize((299,299))

image = np.array(image)

# for images that has 4 channels, we convert them into 3 channels

if image.shape[2] == 4:

    image = image[:, :3]

    image = np.expand_dims(image, axis=0)

    image = image/127.5

    image = image - 1.0

    feature = model.predict(image)

    return feature

def word_for_id(integer, tokenizer):

    for word, index in tokenizer.word_index.items():

        if index == integer:

            return word

    return None

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

    in_text = 'start'

    for i in range(max_length):

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

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

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

        pred = np.argmax(pred)

        word = word_for_id(pred, tokenizer)

        if word is None:

            break

```

```

in_text += ' ' + word

if word == 'end':

break

return in_text

#path = 'Flicker8k_Dataset/111537222_07e56d5a30.jpg'

max_length = 32

tokenizer = load(open("tokenizer.p","rb"))

model = load_model('models/model_9.h5')

xception_model = Xception(include_top=False, pooling="avg")

photo = extract_features(img_path, xception_model)

img = Image.open(img_path)

description = generate_desc(model, tokenizer, photo, max_length)

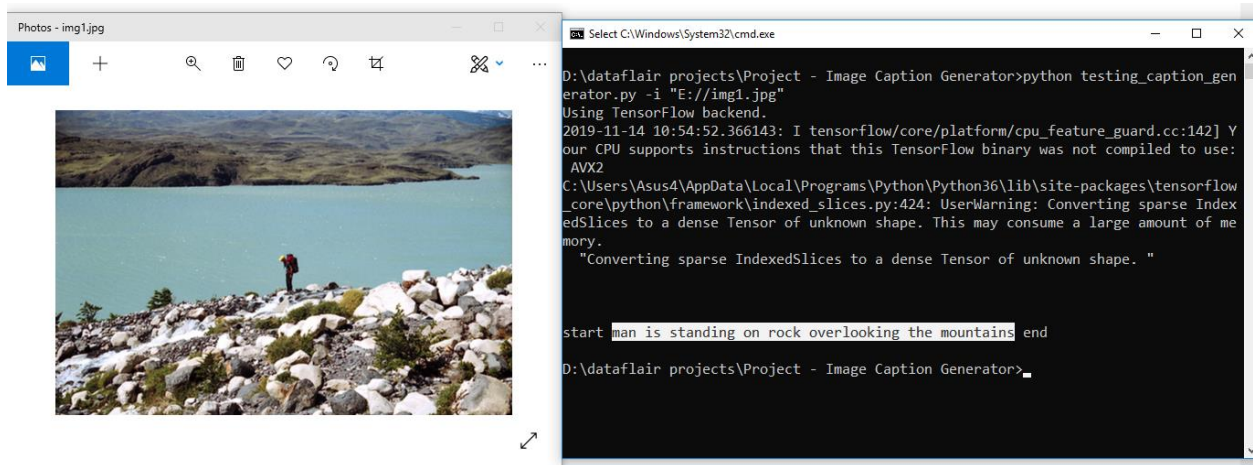
print("\n\n")

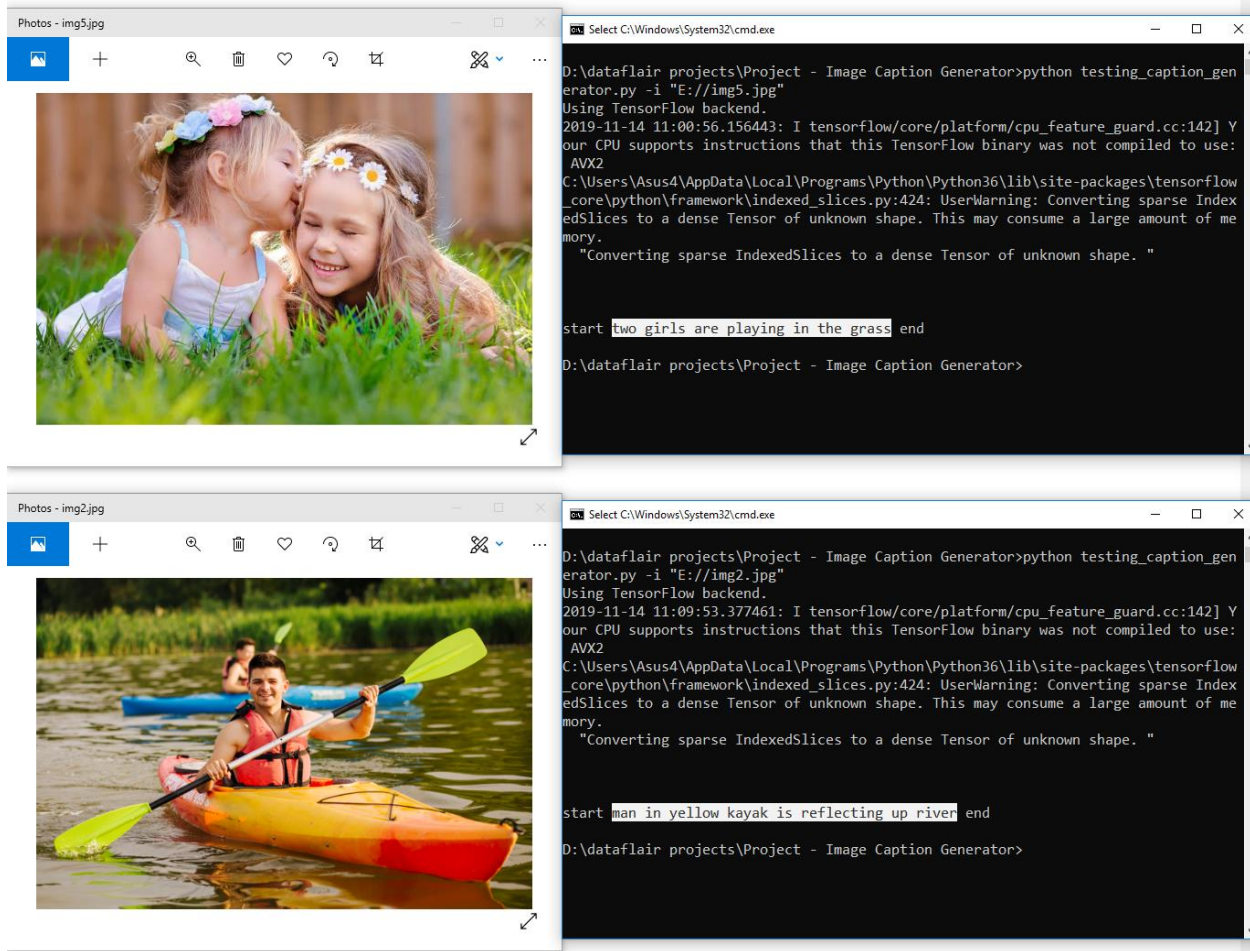
print(description)

plt.imshow(img)

```

Results:





Summary

In this advanced Python project, we have implemented a CNN-RNN model by building an image caption generator. Some key points to note are that our model depends on the data, so, it cannot predict the words that are out of its vocabulary. We used a small dataset consisting of 8000 images. For production-level models, we need to train on datasets larger than 100,000 images which can produce better accuracy models.