Step by Step Guide to Build Image Caption Generator using Deep Learning

ADVANCED COMPUTER VISION DEEP LEARNING GUIDE IMAGE IMAGE ANALYSIS NLP PYTHON

Whenever an image appears in front of us, our brain can annotate or label it. But what about computers? How can a machine process and label an image with a highly relevant and accurate caption? It seemed quite impossible a few years back. Still, with the enhancement of Computer Vision and Deep learning algorithms, the availability of relevant datasets, and AI models, it becomes easier to build a relevant caption generator for an image. Even Caption generation is growing worldwide, and many data annotation firms are earning billions. In this guide, we will build one such annotation tool capable of generating relevant captions for the image with the help of datasets. Basic knowledge of two Deep learning techniques, including LSTM and CNN, is required.

This article was published as a part of the Data Science Blogathon

Table of contents

What is Image to Caption Generator?

Image caption generator is a process of recognizing the context of an image and annotating it with relevant captions using deep learning and computer vision. It includes labeling an image with English keywords with the help of datasets provided during model training. The imagenet dataset trains the CNN model called Xception. Xception is responsible for image feature extraction. These extracted features will be fed to the LSTM model, which generates the image caption.

enter image description here Image Caption Generator

Source: Stackexchange

What is CNN?

CNN is a subfield of Deep learning and specialized deep neural networks used to recognize and classify images. It processes the data represented as 2D matrix-like images. CNN can deal with scaled, translated, and rotated imagery. It analyzes the visual imagery by scanning them from left to right and top to bottom and extracting relevant features. Finally, it combines all the parts for image classification.

What is LSTM?

Being a type of RNN (recurrent neural network), LSTM (Long short-term memory) is capable of working with sequence prediction problems. It is mostly used for the next word prediction purposes, as in Google search our system is showing the next word based on the previous text. Throughout the processing of inputs, LSTM is used to carry out the relevant information and to discard non-relevant information.

To build an image caption generator model we have to merge CNN with LSTM. We can drive that:

```
Image Caption Generator Model (CNN-RNN model) = CNN + LSTM
```

- CNN – To extract features from the image. A pre-trained model called Xception is used for this.
- LSTM – To generate a description from the extracted information of the image.

Dataset for Image Caption Generator

The Flickr_8K dataset represents the model training of image caption generators. The dataset is downloaded directly from the below links. The downloading process takes some time due to the dataset's large size(1GB). In the image below, you can check all the files in the Flickr_8k_text folder. The most important file is Flickr 8k.token, which stores all the image names with captions. 8091 images are stored inside the Flicker8k_Dataset folder and the text files with captions of images are stored in the Flickr_8k_text folder.

- Flicker8k_Dataset
- Flickr_8k_text

Pre-requisites

We will use Jupyter notebooks to run our caption generator. You can download Jupyter notebooks from here. A good understanding of Python, Deep learning, and NLP is required for the implementation. If you're not familiar with these techniques. Please refer to the link below first.

- Python

- DeepLearning
- NLP

Install below libraries, to begin with, the project:

```
pip install TensorFlow pip install Keras pip install pillow pip install NumPy Pip install tqdm Pip install jupyterlab
```

Building the Image Caption Generator

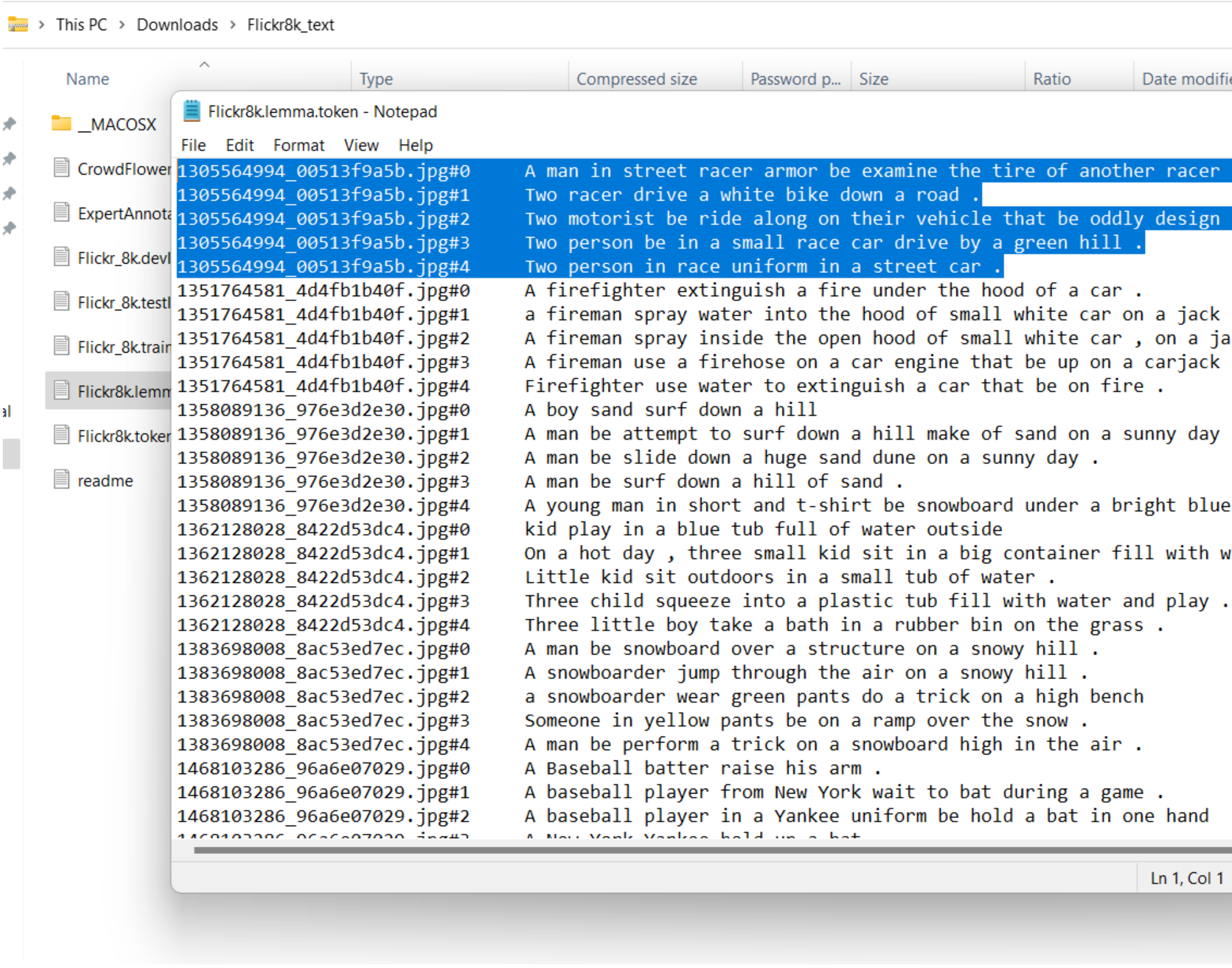
Let’s start by opening the jupyter notebook to create our Python3 project. Name your python3 file with train_caption_generate.ipynb

Import all the Required Packages

```
import numpy as np from PIL import Image import os import string from pickle import dump from pickle import load from keras.applications.xception import Xception #to get pre-trained model Xception from keras.applications.xception import preprocess_input from keras.preprocessing.image import load_img from keras.preprocessing.image import img_to_array from keras.preprocessing.text import Tokenizer #for text tokenization 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#Keras to build our CNN and LSTM from keras.layers import LSTM, Embedding, Dropout from tqdm import tqdm_notebook as tqdm #to check loop progress tqdm().pandas()
```

Perform Data Cleaning

As we see all image captions are available in the Flickr 8k.token file of the Flickr_8k_text folder. If you analyze this file carefully, you can drive the format of image storing, each image and caption separated by a new line and carry 5 captions numbered from 0 to 4 along with.



```
1305564994_00513f9a5b.jpg#0    A man in street racer armor be examine the tire of another racer
1305564994_00513f9a5b.jpg#1    Two racer drive a white bike down a road .
1305564994_00513f9a5b.jpg#2    Two motorist be ride along on their vehicle that be oddly design
1305564994_00513f9a5b.jpg#3    Two person be in a small race car drive by a green hill .
1305564994_00513f9a5b.jpg#4    Two person in race uniform in a street car .
1351764581_4d4fb1b40f.jpg#0    A firefighter extinguish a fire under the hood of a car .
1351764581_4d4fb1b40f.jpg#1    a fireman spray water into the hood of small white car on a jack
1351764581_4d4fb1b40f.jpg#2    A fireman spray inside the open hood of small white car , on a ja
1351764581_4d4fb1b40f.jpg#3    A fireman use a firehose on a car engine that be up on a carjack
1351764581_4d4fb1b40f.jpg#4    Firefighter use water to extinguish a car that be on fire .
1358089136_976e3d2e30.jpg#0    A boy sand surf down a hill
1358089136_976e3d2e30.jpg#1    A man be attempt to surf down a hill make of sand on a sunny day
1358089136_976e3d2e30.jpg#2    A man be slide down a huge sand dune on a sunny day .
1358089136_976e3d2e30.jpg#3    A man be surf down a hill of sand .
1358089136_976e3d2e30.jpg#4    A young man in short and t-shirt be snowboard under a bright blue
1362128028_8422d53dc4.jpg#0    kid play in a blue tub full of water outside
1362128028_8422d53dc4.jpg#1    On a hot day , three small kid sit in a big container fill with w
1362128028_8422d53dc4.jpg#2    Little kid sit outdoors in a small tub of water .
1362128028_8422d53dc4.jpg#3    Three child squeeze into a plastic tub fill with water and play .
1362128028_8422d53dc4.jpg#4    Three little boy take a bath in a rubber bin on the grass .
1383698008_8ac53ed7ec.jpg#0    A man be snowboard over a structure on a snowy hill .
1383698008_8ac53ed7ec.jpg#1    A snowboarder jump through the air on a snowy hill .
1383698008_8ac53ed7ec.jpg#2    a snowboarder wear green pants do a trick on a high bench
1383698008_8ac53ed7ec.jpg#3    Someone in yellow pants be on a ramp over the snow .
1383698008_8ac53ed7ec.jpg#4    A man be perform a trick on a snowboard high in the air .
1468103286_96a6e07029.jpg#0    A Baseball batter raise his arm .
1468103286_96a6e07029.jpg#1    A baseball player from New York wait to bat during a game .
1468103286_96a6e07029.jpg#2    A baseball player in a Yankee uniform be hold a bat in one hand
```

Now we are going to define 5 functions for cleaning:

1. load_fp(filename) – To load the document file and read the contents of the file into a string.
2. mg_capt(filename) – To create a description dictionary that will map images with all 5 captions.
3. txt_cleaning(descriptions) – This method is used to clean the data by taking all descriptions as input. While dealing with textual data we need to perform several types of cleaning including uppercase to lowercase conversion, punctuation removal, and removal of the number containing words.
4. txt_vocab(descriptions) – This is used to create a vocabulary from all the unique words extracted out from descriptions.
5. save_descriptions(descriptions, filename) – This function is used to store all the preprocessed descriptions into a file.
6.

```
'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'
                              ]
```

Screenshot of Description Dictionary

Code :

```
# Load the document file into memory def load_fp(filename): # Open file to read file = open(filename, 'r') text = file.read() file.close() return text # get all images with their captions def img_capt(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 function will convert all upper case alphabets to lowercase, removing punctuations and words containing numbers def txt_clean(captions): table = str.maketrans('','',string.punctuation) for img,caps in captions.items(): for i,img_caption in enumerate(caps): img_caption.replace("-"," ") descp = img_caption.split() #uppercase to lowercase descp = [wrd.lower() for wrd in descp] #remove punctuation from each token descp = [wrd.translate(table) for wrd in descp] #remove hanging 's and a descp = [wrd for wrd in descp if(len(wrd)>1)] #remove words containing numbers with them descp = [wrd for wrd in descp if(wrd.isalpha())] #converting back to string img_caption = ' '.join(desc) captions[img][i]= img_caption return captions def txt_vocab(descriptions): # To build vocab of all unique words vocab = set() for key in descriptions.keys(): [vocab.update(d.split()) for d in descriptions[key]] return vocab #To save 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, like i create a folder with my name shikha inside D-drive dataset_text = "D:shikhaProject - Image Caption GeneratorFlickr_8k_text" dataset_images = "D:shikhaProject - Image Caption GeneratorFlicker8k_Dataset" #to prepare our text data filename = dataset_text + "/" + "Flickr8k.token.txt" #loading the file that contains all data #map them into descriptions dictionary descriptions = img_capt(filename) print("Length of descriptions =" ,len(descriptions)) #cleaning the descriptions clean_descriptions = txt_clean(descriptions) #to build vocabulary vocabulary = txt_vocab(clean_descriptions) print("Length of vocabulary = ", len(vocabulary)) #saving all descriptions in one file save_descriptions(clean_descriptions, "descriptions.txt")
```

Extract the Feature Vector

Now we are going to use the pre-trained model called Xception which is already trained with large datasets to extract the features from these models. Xception was trained on an imagenet dataset with 1000 different classes to classify the images. We can use keras.applications to import this model directly. We need to do a few changes to the Xception model to integrate it with our model. The xception model takes 299*299*3 image size as input so we need to delete the last classification layer and extract out the 2048 feature vectors.

```
model = Xception( include_top=False, pooling=’avg’ )
```

Extract_features() function is used to extract these features for all images. At the end we will put the features dictionary into a pickle file named “features.p”.

```
def extract_features(directory): model = Xception( include_top=False, pooling='avg' ) features = {} for pic in tqdm(os.listdir(dirc)): file = dirc + "/" + pic image = Image.open(file) 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")) #to directly load the features from the pickle file. features = load(open("features.p","rb"))
```

Loading dataset for model training

A file named “Flickr_8k.trainImages.txt” is present in our Flickr_8k_test folder. This file carries a list of 6000 image names that are used for the sake of training.

Functions required to load the training datasets:

- load_photos(fname) – This function will take a file name as a parameter and return the list of image names by loading the text file into a string.
- load_clean_descriptions(fname, image) – This function stores the captions for every image from the list of photos to a dictionary. For the ease of the LSTM model in identifying the beginning and ending of a caption, we append the and identifier with each caption.
- load_features(photos) – The extracted feature vectors from the Xception model and the dictionary for photos are returned by this function.

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 = ' ' + " ".join(image_caption) + ' ' 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)
```

Tokenizing the Vocabulary

Machines are not familiar with complex English words so, to process model’sdata they need a simple numerical representation. That’s why we map every word of the vocabulary with a separate unique index value. An in-built tokenizer function is present in the Keras library to create tokens from our vocabulary. We can save them to a pickle file named “tokenizer.p”.

Code:

```
#convert dictionary to clear 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 #The size of our vocabulary is 7577 words. #calculate maximum length
of descriptions to decide the model structure parameters. 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 #Max_length of description is 32
```

Create a Data generator

For training the model as a supervised learning task we need to feed it with input and output sequences. Total 6000 images with 2048 length feature vector and the caption represented as numbers are present in our training sets. It's not possible to hold such a large amount of data into memory so we are going to use a generator method that will yield batches.

For example: [x1, x2] are the input of our model, and y act as output, where x1 shows 2048 feature vectors of the image, x2 shows the input text sequence and y shows the output text sequence that is predicted by the model.

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

```
#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] inp_image, inp_seq, op_word = create_sequences(tokenizer, max_length, description_list,
feature) yield [[inp_image, inp_sequence], op_word] def create_sequences(tokenizer, max_length, desc_list,
feature): x_1, x_2, y = list(), list(), list() # move through each description for the image for desc in
desc_list: # encode the sequence seq = tokenizer.texts_to_sequences([desc])[0] # divide one sequence into
various X,y pairs for i in range(1, len(seq)): # divide 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 x_1.append(feature) x_2.append(in_seq)
y.append(out_seq) return np.array(X_1), np.array(X_2), np.array(y) #To 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))
```

Define the CNN-RNN model

From the Functional API, we will use the Keras Model in order to define the structure of the model. It includes:

- Feature Extractor –With a dense layer, it will extract the feature from the images of size 2048 and we will decrease the dimensions to 256 nodes.
- Sequence Processor – Followed by the LSTM layer, the textual input is handled by this embedded layer.
- Decoder – We will merge the output of the above two layers and process the dense layer to make the final prediction.

```
from keras.utils import plot_model # define the captioning model def define_model(vocab_size, max_length): #
features from the CNN model compressed 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) # merge it [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
```

Training the Image Caption Generator model

We will generate the input and output sequences to train our model with 6000 training images. We create a function named model.fit_generator() to fit the batches to the model. At last, we save the model to our models folder.

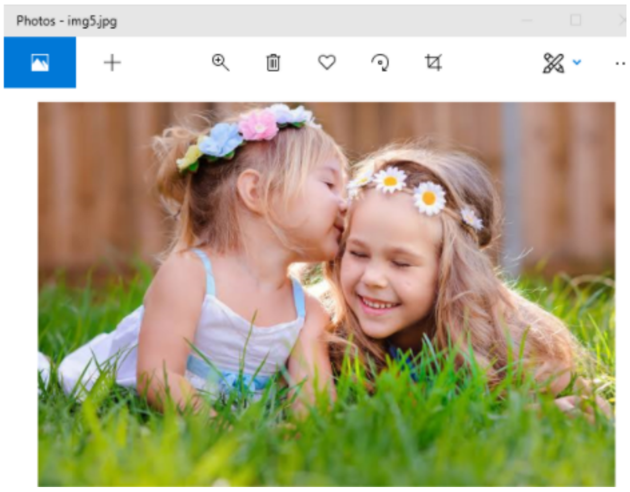
```
# train our model print('Dataset: ', len(train_imgs)) print('Descriptions: train=', len(train_descriptions))
print('Photos: train=', len(train_features)) print('Vocabulary Size:', vocab_size) print('Description Length:
', max_length) model = define_model(vocab_size, max_length) epochs = 10 steps = len(train_descriptions) #
creating a directory named 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")
```

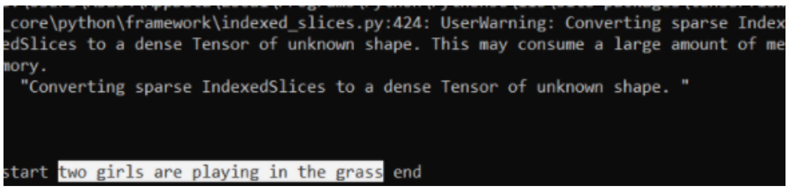
Testing the Image Caption Generator model

After successful model training, our task is to test the model accuracy by inputting test image data. Let's create a python file named test_caption.py to load the model and generate predictions.

```
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: Can't open image! Ensure that image path and extension is
correct") image = image.resize((299,299)) image = np.array(image) # for 4 channels images, we need to 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 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("nn") print(description) plt.imshow(img)
```

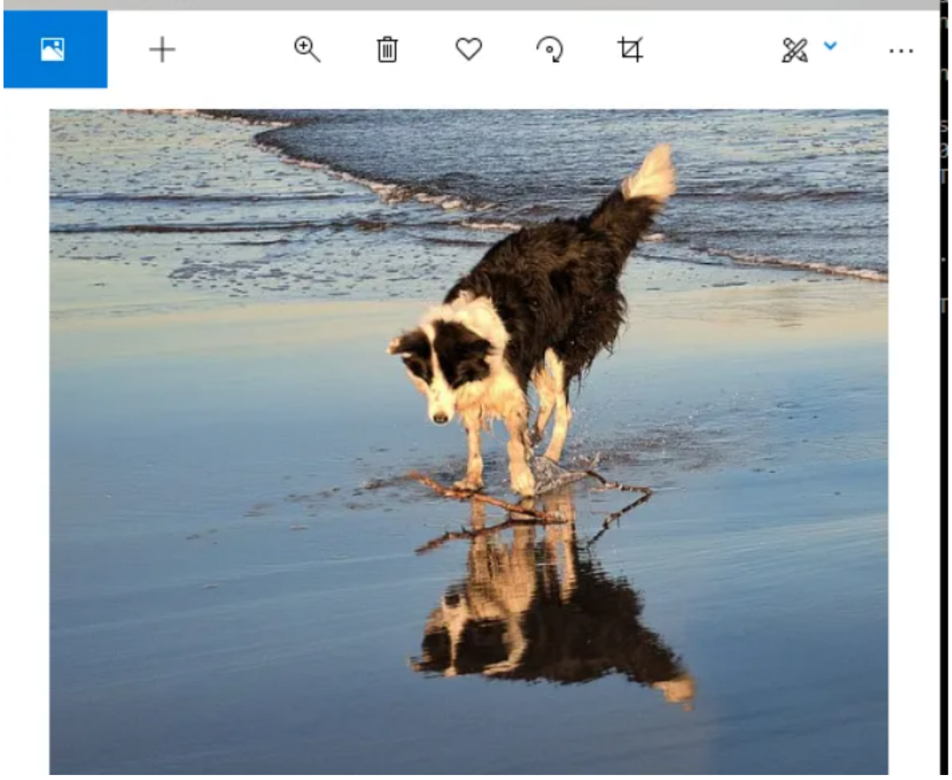
Output:

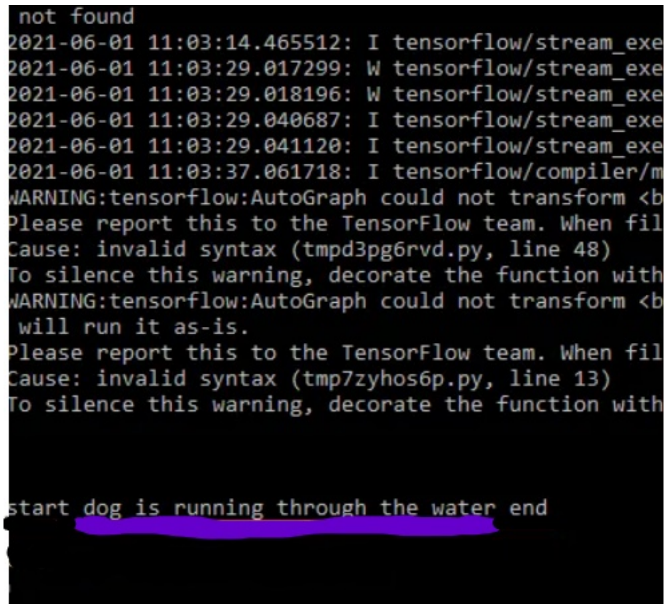




```
_core\python\framework\indexed_slices.py:424: UserWarning: Converting sparse Index
edSlices to a dense Tensor of unknown shape. This may consume a large amount of me
mory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "


start two girls are playing in the grass end
```





```
 not found
2021-06-01 11:03:14.465512: I tensorflow/stream_exe
2021-06-01 11:03:29.017299: W tensorflow/stream_exe
2021-06-01 11:03:29.018196: W tensorflow/stream_exe
2021-06-01 11:03:29.040687: I tensorflow/stream_exe
2021-06-01 11:03:29.041120: I tensorflow/stream_exe
2021-06-01 11:03:37.061718: I tensorflow/compiler/m
WARNING:tensorflow:AutoGraph could not transform <b
Please report this to the TensorFlow team. When fil
Cause: invalid syntax (tmpd3pg6rvd.py, line 48)
To silence this warning, decorate the function with
WARNING:tensorflow:AutoGraph could not transform <b
 will run it as-is.
Please report this to the TensorFlow team. When fil
Cause: invalid syntax (tmp7zyhos6p.py, line 13)
To silence this warning, decorate the function with


start dog is running through the water end
```

End Note

In this guide, we build a deep learning model with the help of CNN and LSTM. We used a very small dataset of 8000 images to train our model, but the business level model used larger datasets of more than 100,000 images for better accuracy. The larger the datasets are higher the accuracy. So, if you want to build a more accurate caption generator you can try this model with large datasets.

Frequently Asked Questions

Q1. What is an image caption generator?

A. An image caption generator is a system that generates textual descriptions or captions for images automatically. It combines computer vision techniques to understand the visual content of an image and natural language processing (NLP) techniques to generate descriptive captions.

Q2. What is image captioning in the project description?

A. In a project description, image captioning refers to generating textual descriptions or captions for images using machine learning and NLP techniques. It involves analyzing the visual features of an image and generating a coherent and relevant caption that describes the image's content.

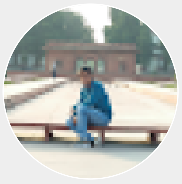
Q3. What is the dataset for image caption generation?

A. The dataset for image caption generation typically consists of pairs of images and corresponding captions. These datasets are manually annotated, where human annotators provide descriptive captions for a given set of images. Commonly used datasets include MSCOCO, Flickr8K, and Flickr30K.

Q4. What is the use of LSTM in image captioning?

A. Long Short-Term Memory (LSTM) is a recurrent neural network (RNN) architecture commonly used in image captioning. LSTM networks can capture long-term dependencies in sequential data, making them suitable for generating coherent and contextually relevant captions by modeling the sequential nature of language.

Article Url - https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/



Shikha Gupta