1ST CELL

from google.colab import drive

drive.mount('/content/drive')

2ND CELL

!pip install TensorFlow

!pip install Keras

!pip install pillow

!pip install NumPy

!pip install tqdm

3RD CELL

!pip install --upgrade keras

!pip install tensorflow

!pip install --upgrade tensorflow

!pip install --upgrade tensorflow-addons

4TH CELL

import numpy as np

from PIL import Image

import os

import string

from pickle import dump, load

from tensorflow.keras.applications.xception import Xception

from tensorflow.keras.applications.xception import preprocess\_input

from tensorflow.keras.preprocessing import image

from tensorflow.keras.preprocessing.image import img\_to\_array

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.layers import Add

from tensorflow.keras.models import Model, load\_model

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

from tqdm import tqdm

# Set the notebook interface for tqdm

tqdm().pandas()

# Rest of your code...

import os

os.environ['KERAS\_BACKEND'] = 'tensorflow'

5TH CELL

import string

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\_fp(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(descp)

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 paths according to your project folder on your system

dataset\_text = "/content/drive/MyDrive/ML/Flickr8k\_text"

dataset\_images = "/content/drive/MyDrive/ML/Flicker8k\_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, "/content/drive/MyDrive/ML/descriptions.txt")

6TH CELL

from tensorflow.keras.applications import Xception

from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.xception import preprocess\_input

import numpy as np

import os

from tqdm import tqdm

from PIL import Image

from pickle import dump, load

def extract\_features(directory):

model = Xception(include\_top=False, pooling='avg')

features = {}

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

file = os.path.join(directory, pic)

if os.path.isfile(file) and pic.lower().endswith(('.png', '.jpg', '.jpeg')):

img = Image.open(file)

img = img.resize((299, 299))

img = np.expand\_dims(img, axis=0)

img = img / 127.5

img = img - 1.0

feature = model.predict(img)

features[pic] = feature

return features

dataset\_images = "/content/drive/MyDrive/ML/Flicker8k\_Dataset"

features = extract\_features(dataset\_images)

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

7TH CELL

features = load(open("/content/drive/MyDrive/ML/features.p","rb"))

8TH CELL

# Load the data

def load\_doc(filename):

with open(filename, 'r') as file:

text = file.read()

return text

def load\_photos(filename):

file = load\_doc(filename)

photos = file.split("\n")[:-1] # Use "\n" instead of "n"

return photos

def load\_clean\_descriptions(filename, photos):

# Loading clean\_descriptions

file = load\_doc(filename)

descriptions = {}

for line in file.split("\n"): # Use "\n" instead of "n"

words = line.split()

if len(words) < 2: # Use "len(words) < 2" to ensure both image and caption are present

continue

image, image\_caption = words[0], words[1:]

if image in photos:

if image not in descriptions:

descriptions[image] = []

desc = ' '.join(image\_caption) # No need to add extra spaces here

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

# Define your dataset\_text and load\_doc function here

# Load train images list

filename = dataset\_text + "/" + "Flickr\_8k.trainImages.txt"

train\_imgs = load\_photos(filename)

# Load clean descriptions for train images

train\_descriptions = load\_clean\_descriptions("descriptions.txt", train\_imgs)

# Load features for train images

train\_features = load\_features(train\_imgs)

# Now you can use train\_imgs, train\_descriptions, and train\_features

9TH CELL

#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('/content/drive/MyDrive/ML/tokenizer.p', 'wb'))

vocab\_size = len(tokenizer.word\_index) + 1

vocab\_size

10TH CELL

#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

11TH CELL

#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))

12TH CELL

from tensorflow.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='/content/drive/MyDrive/ML/model.png', show\_shapes=True)

return model

13TH CELL

#train our model

import os

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

from tensorflow.keras.models import Model

# Assuming you have defined data\_generator and tokenizer somewhere in your code

# data\_generator(train\_descriptions, train\_features, tokenizer, max\_length)

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")

14TH CELL

from PIL import Image

#img = Image.open('/content/drive/MyDrive/ML/Flicker8k\_Dataset/111537222\_07e56d5a30.jpg')

#imagePath = '/content/drive/MyDrive/ML/Flicker8k\_Dataset/3738685861\_8dfff28760.jpg'

#imagePath = '/content/drive/MyDrive/ML/Flicker8k\_Dataset/241347611\_cb265be138.jpg'

#imagePath = '/content/drive/MyDrive/ML/Flicker8k\_Dataset/242558556\_12f4d1cabc.jpg'

imagePath = '/content/drive/MyDrive/ML/Flicker8k\_Dataset/250892549\_1e06a06a78.jpg'

#imagePath = '/content/drive/MyDrive/ML/Flicker8k\_Dataset/

#imagePath = '/content/drive/MyDrive/ML/Flicker8k\_Dataset/

#imagePath = '/content/drive/MyDrive/ML/Flicker8k\_Dataset/

img = Image.open(imagePath)

img

import numpy as np

from PIL import Image

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from pickle import load

def word\_for\_id(integer, tokenizer):

for word, index in tokenizer.word\_index.items():

if index == integer:

return word

return None

def generate\_caption(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

LAST CELL!!!

# Load the trained model and tokenizer

model = load\_model('models/model\_9.h5') # Change the path to your trained model

tokenizer = load(open('/content/drive/MyDrive/ML/tokenizer.p', 'rb')) # Change the path to your tokenizer

# Load an image for which you want to generate a caption

image\_path = '/content/drive/MyDrive/ML/Flicker8k\_Dataset/250892549\_1e06a06a78.jpg'

xception\_model = Xception(include\_top=False, pooling="avg")

# Extract image features using your function

img\_features = extract\_features(image\_path, xception\_model) # Replace xception\_model with your image feature extraction model

# Generate a caption using the model

caption = generate\_caption(model, tokenizer, img\_features, max\_length)

print("Generated Caption:", caption)