```
pip install --upgrade keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.14.0)
```

```
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 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()
```

```
<ipython-input-2-8feb03adf18e>:19: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  tqdm().pandas()
```

0/? [00:11<?, ?it/s]

```
import string  # Add this line to import the 'string' module


# 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_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 uppercase alphabets to lowercase, 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()
            # uppercase to lowercase
            desc = [wrd.lower() for wrd in desc]
            # remove punctuation from each token
            desc = [wrd.translate(table) for wrd in desc]
            # remove hanging 's and 'a'
            desc = [wrd for wrd in desc if (len(wrd) > 1)]
            # remove words containing numbers
            desc = [wrd for wrd in desc if (wrd.isalpha())]
            # converting back to string
            img_caption = ' '.join(desc)
            captions[img][i] = img_caption
    return captions

# ... (rest of your code)


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 = "/content/drive/MyDrive/Flickr8k_text"
dataset_images = "/content/drive/MyDrive/Classroom/images"

#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 = img_capt(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")
```

```
     Length of descriptions = 8092
     Length of vocabulary =  8763
```

```
import pickle
from PIL import Image
from tqdm import tqdm
import os
import numpy as np
from keras.applications.xception import Xception, preprocess_input
# Add the import statement for Xception model

# Function to extract features using 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))  # Resize image to the input size expected by Xception
        image = np.expand_dims(image, axis=0)
        image = preprocess_input(image)
        feature = model.predict(image)
        features[img] = feature
    return features
features = extract_features(dataset_images)
dump(features, open("features.p","wb"))


# Define the load_doc function to read the file
def load_doc(filename):
    with open(filename, 'r') as file:
        document = file.read()
    return document

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)
```

```
from google.colab import drive
drive.mount('/content/drive')


#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
```

```
7577
```

```
#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
```

```
32
```

```
#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))
```

```
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


# 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")
```

```
Dataset:  6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length:  32
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_9 (InputLayer)        [(None, 32)]                 0         []

 input_8 (InputLayer)        [(None, 2048)]               0         []

 embedding_3 (Embedding)     (None, 32, 256)              1939712   ['input_9[0][0]']

 dropout_6 (Dropout)         (None, 2048)                 0         ['input_8[0][0]']

 dropout_7 (Dropout)         (None, 32, 256)              0         ['embedding_3[0][0]']

 dense_9 (Dense)             (None, 256)                  524544    ['dropout_6[0][0]']

 lstm_3 (LSTM)               (None, 256)                  525312    ['dropout_7[0][0]']

 add_15 (Add)                (None, 256)                  0         ['dense_9[0][0]',
                                                                     'lstm_3[0][0]']

 dense_10 (Dense)            (None, 256)                  65792     ['add_15[0][0]']

 dense_11 (Dense)            (None, 7577)                 1947289   ['dense_10[0][0]']

==================================================================================================
Total params: 5002649 (19.08 MB)
Trainable params: 5002649 (19.08 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
None
<ipython-input-13-67e5cbd2f904>:15: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please
```

```
  model.fit_generator(generator, epochs=1, steps_per_epoch= steps, verbose=1)
6000/6000 [==============================] - 1307s 217ms/step - loss: 4.5228
6000/6000 [==============================] - 1283s 214ms/step - loss: 3.6701
6000/6000 [==============================] - 1289s 215ms/step - loss: 3.3895
6000/6000 [==============================] - 1273s 212ms/step - loss: 3.2204
6000/6000 [==============================] - 1285s 214ms/step - loss: 3.0996
6000/6000 [==============================] - 1284s 214ms/step - loss: 3.0140
6000/6000 [==============================] - 1252s 209ms/step - loss: 2.9460
6000/6000 [==============================] - 1237s 206ms/step - loss: 2.8904
6000/6000 [==============================] - 1246s 208ms/step - loss: 2.8450
6000/6000 [==============================] - 1233s 206ms/step - loss: 2.8079
```

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import argparse


img_path = '/content/drive/MyDrive/Classroom/images/1015584366_dfcec3c85a.jpg'  # Replace with your image path


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_1.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)
```