Install necessary packages

In [1]:
```
# !pip install tensorflow keras pillow numpy tqdm
```

import all the necessary packages

In [2]:
```
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 tensorflow.keras.utils import to_categorical
# 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()
```

```
C:\Users\Client\AppData\Local\Temp/ipykernel_6484/379097116.py:20: TqdmDeprecationWarning: T
his function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  tqdm().pandas()
```

Getting and performing data cleaning

In [3]:
```
# 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 puntuations 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()
```

In [4]:

```
# Set these path according to project folder in you system
dataset_text = "C:/Users/Client/Downloads/Image-Caption-Generator Flickr8k/Flickr8k_text"
dataset_images = "C:/Users/Client/Downloads/Image-Caption-Generator Flickr8k/Flicker8k_Datas

#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, "C:/Users/Client/Downloads/Image-Caption-Generator Fl
```

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

Extracting the feature vector from all images (TAKES TIME)

In [5]:

```
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("C:/Users/Client/Downloads/Image-Caption-Generator Flickr8k/features.p"
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/
xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 31s 0us/step
83697664/83683744 [==============================] - 31s 0us/step
C:\Users\Client\AppData\Local\Temp/ipykernel_6484/3734661667.py:4: TqdmDeprecationWarning: T
his function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for img in tqdm(os.listdir(directory)):
```

In [6]:

```
features = load(open("C:/Users/Client/Downloads/Image-Caption-Generator Flickr8k/features.p'
```

Loading dataset for Training the model

In [7]:

```
#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("C:/Users/Client/Downloads/Image-Caption-Generator Flickr8k/fea
    #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("C:/Users/Client/Downloads/Image-Caption-Genera
train_features = load_features(train_imgs)
```

Tokenizing the vocabulary

In [8]:
```
#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('C:/Users/Client/Downloads/Image-Caption-Generator Flickr8k/tokenizer.p
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

Out[8]:
```
7577
```

In [9]:
```
#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
```

Out[9]:
```
32
```

Create Data generator

In [10]:
```
#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_leng
            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
```

Out[10]: ((47, 2048), (47, 32), (47, 7577))

Defining the CNN-RNN model

In [11]:

```
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='C:/Users/Client/Downloads/Image-Caption-Generator Flickr8k/m

    return model
```

TRAINING SECTION

In [12]:

```
# 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)
print(model,'model')
epochs = 10
steps = len(train_descriptions)
# making a directory models to save our models
os.mkdir("C:/Users/Client/Downloads/Image-Caption-Generator Flickr8k/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("C:/Users/Client/Downloads/Image-Caption-Generator Flickr8k/models/model_" +
```

```
Dataset:  6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length:  32
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 32)]         0           []

 input_2 (InputLayer)           [(None, 2048)]       0           []

 embedding (Embedding)          (None, 32, 256)      1939712     ['input_3[0][0]']

 dropout (Dropout)              (None, 2048)         0           ['input_2[0][0]']

 dropout_1 (Dropout)            (None, 32, 256)      0           ['embedding[0][0]']

 dense (Dense)                  (None, 256)          524544      ['dropout[0][0]']

 lstm (LSTM)                    (None, 256)          525312      ['dropout_1[0][0]']

 add_12 (Add)                   (None, 256)          0           ['dense[0][0]',
                                                                  'lstm[0][0]']

 dense_1 (Dense)                (None, 256)          65792       ['add_12[0][0]']

 dense_2 (Dense)                (None, 7577)         1947289     ['dense_1[0][0]']

==================================================================================================
Total params: 5,002,649
Trainable params: 5,002,649
Non-trainable params: 0
__________________________________________________________________________________________________
None
('You must install pydot (`pip install pydot`) and install graphviz (see instructions at htt
ps://graphviz.gitlab.io/download/) ', 'for plot_model/model_to_dot to work.')
<keras.engine.functional.Functional object at 0x000001FF70EEDA00> model
```

```
C:\Users\Client\AppData\Local\Temp/ipykernel_6484/409286170.py:16: UserWarning: `Model.fit_g
enerator` is deprecated and will be removed in a future version. Please use `Model.fit`, whi
ch supports generators.
  model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
6000/6000 [==============================] - 2255s 375ms/step - loss: 4.5057

C:\Users\Client\anaconda3\envs\ds\lib\site-packages\keras\engine\functional.py:1410: CustomM
askWarning: Custom mask layers require a config and must override get_config. When loading,
the custom mask layer must be passed to the custom_objects argument.
  layer_config = serialize_layer_fn(layer)
```

```
6000/6000 [==============================] - 2239s 373ms/step - loss: 3.6630
6000/6000 [==============================] - 2247s 375ms/step - loss: 3.3745
6000/6000 [==============================] - 2280s 380ms/step - loss: 3.2001
6000/6000 [==============================] - 2281s 380ms/step - loss: 3.0849
6000/6000 [==============================] - 2263s 377ms/step - loss: 2.9947
6000/6000 [==============================] - 2285s 381ms/step - loss: 2.9279
6000/6000 [==============================] - 2291s 382ms/step - loss: 2.8658
6000/6000 [==============================] - 2277s 379ms/step - loss: 2.8246
6000/6000 [==============================] - 2267s 378ms/step - loss: 2.7829
```