Configuration Manual

1. System Requirements

RAM: 32GB
OS: Windows and Google Colab
Processor: i5 Intel
GPU: 4 GB Nvidia, T4- Google Colab

2. Steps to work
Here are the actions you need to take to manually run the image captioning model:

1. We gather information.
Gather any and all cricket-related visuals. Filter out incompatible graphics and download around 1000 photos.
Separate the cricket dataset into a training set of 800 photos and a test set of 200.
Use an external dataset, such as Flickr8k, if you need a more substantial dataset.

2. Images are loaded and features are generated.
To use the InceptionV3 model, you'll need to load the photos and preprocess them to a specified size (for example, 299x299).
Extract 1x2048-dimensional vectors of picture features using a pretrained InceptionV3 model.

3. Textual pre-processing and image caption loading constitute this step.
Incorporate the captions into a dictionary that links the filenames of images with their descriptions.
Get rid of punctuation and capitalise just necessary words when processing text.
Each caption has to have "captionstart" and "captionend" added so the model can read them.

4. Loading Data Consisting of Processed Images and Captions
Encode the sanitised captions by employing a tokenizer to convert the words into a series of integers.
Get the data ready for model fitting by splitting it into input and output pairs.

5. Model Fitting and Definition
An image feature extractor, sequence processor, and decoder make up the image captioning model, which you must define.
Train the model by giving it data in batches using a generator function.
Keep track of the several trained models over time and pick the best one according to how well it does on the development dataset.

6. Model Completion
Choose the best-performing model in an unexplored dataset (Flickr's training data for BLEU-1, cricket's test data for BLEU-2).
The completed model should be saved for later usage.

7. Assessing the Model

Put the model to the test with real data. Compute BLEU scores by comparing expected and reference captions.

8. Create Test or New Captions for Images

Bring up the tokenizer and trained model you made before.
Caption start word ("captionstart") and maximum length must be specified.
With each fresh or test picture:
  >> Make use of InceptionV3 to produce picture features.
  >> The trained model should be used to guess the next caption word until the "captionend" keyword is encountered.
  >> To get the anticipated caption, just take off the first and last words.
Check the quality of the produced captions to see how well the model performed.

2. Dataset

First step is to find suitable dataset to build an image captioning problem. As we are more interested towards sport specific captioning, we will be choosing cricket domain related images. We have downloaded around 1000 images from internet via python code. For cricket data, as we have very small dataset, we will use 800 images as train data and 200 images as test dataset. Although some of the images from cricket data will be filtered out due to non-supported visuals like gif files. In our final cricket captioning model, we will have total 720 images, out of which 575 will be used for model training and remaining 145 will be used for model evaluation (test data). There are many open datasets like Flickr8k dataset, Flickr30k dataset & Microsoft COCO dataset which contains 180k images. We want to build an image captioning model using normal hardware and building model using large image dataset is not feasible on normal laptops. Therefore, we will be using Flickr8k dataset for this model development. This data can be downloaded from Kaggle platform without any cost. This dataset is a benchmark collection of 8000 images from different domains with 5 captions per image. All these captions provide major content info of any image. This dataset has pre-defined 6000 images as training data, 1000 images as development data and remaining 1000 images a test data. A sample of Flickr image captioning is provided below:

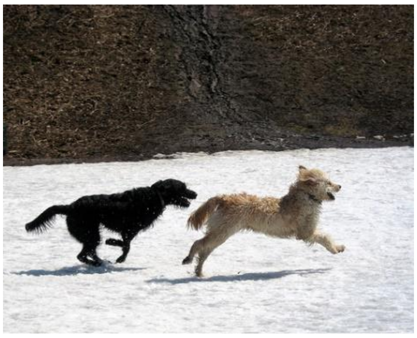


Figure 1. Sample image from Flickr8k data

For above Fig 8, below are the 5 different captions provided by experts:
Caption1: a black dog is running after a white dog in the snow
Caption2: black dog chasing brown dog through snow
Caption3: two dogs chase each other across the snowy ground
Caption4: two dogs play together in the snow
Caption5: two dogs running through a low lying body of water
Cleaned descriptions of last step need to be encoded as numbers or each word should be assigned a number. This step will be helpful in upcoming sequence processing step. Keras

provides a class named as Tokenizer which can learn the mapping for each word of a loaded description and assign a unique number to each word. Now we need to prepare the cleaned text data and image feature vectors as per model fitting requirements. We know that we cannot pass complete caption as target variable of image captioning model. We need to pass word-by-word. To perform that, we need to encode the captions. Each image caption will be divided into words. Model will be provided image feature vector and one word at a time and it will generate next word. Then first two words and image feature vectors will be given as inputs to generate next word. This will the process of model training. Below example (Fig 9 and Table 9) provides an easy to understand explanation of the training data preparation:

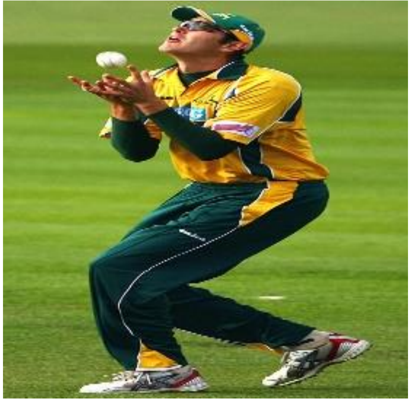


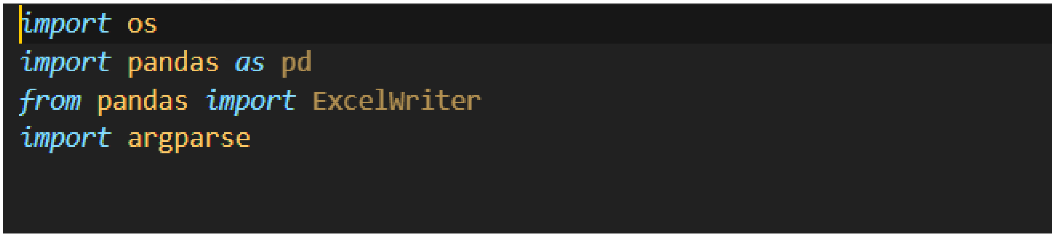
Figure 2. A sample image of cricket data

Caption Generated: A fielder is catching the ball

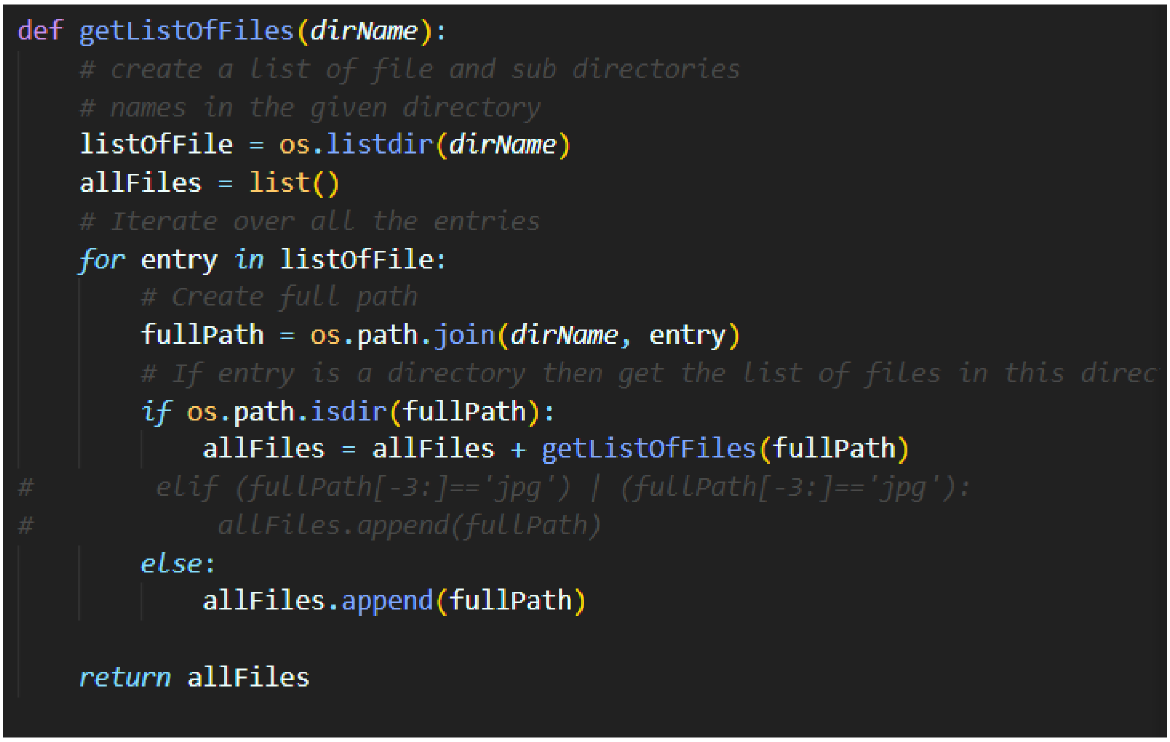
Table 1. Data preparation for image captioning training

S.N.	*Image feature vector (Input1)*	*Word embeddings (Input2)*	*Output*
1	*Image_1*	*captionstart*	*a*
2	*Image_1*	*captionstart a*	*fielder*
3	*Image_1*	*captionstart a fielder*	*is*
4	*Image_1*	*captionstart a fielder is*	*catching*
5	*Image_1*	*captionstart a fielder is catching*	*the*
6	*Image_1*	*captionstart a fielder is catching the*	*ball*
7	*Image_1*	*captionstart a fielder is catching the ball*	*captionend*

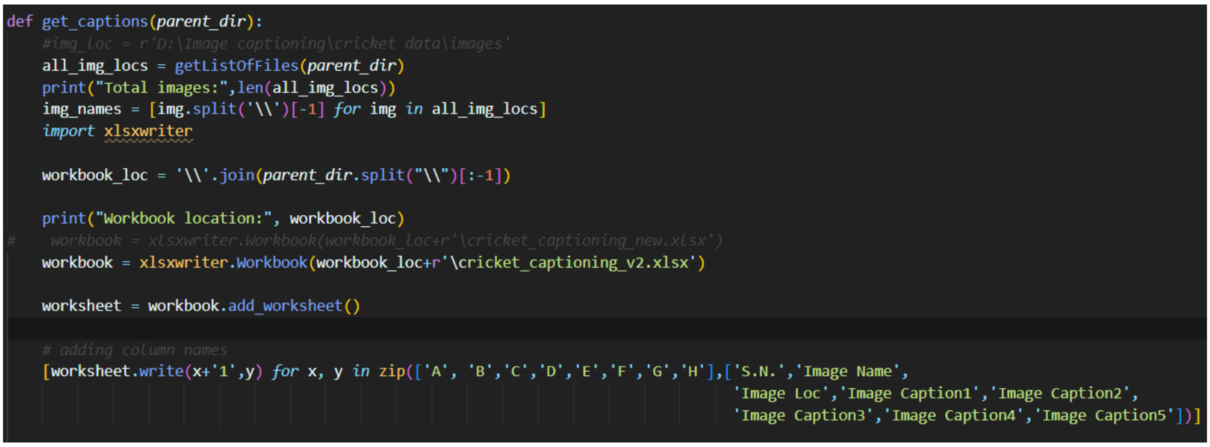
3.Code Snippets



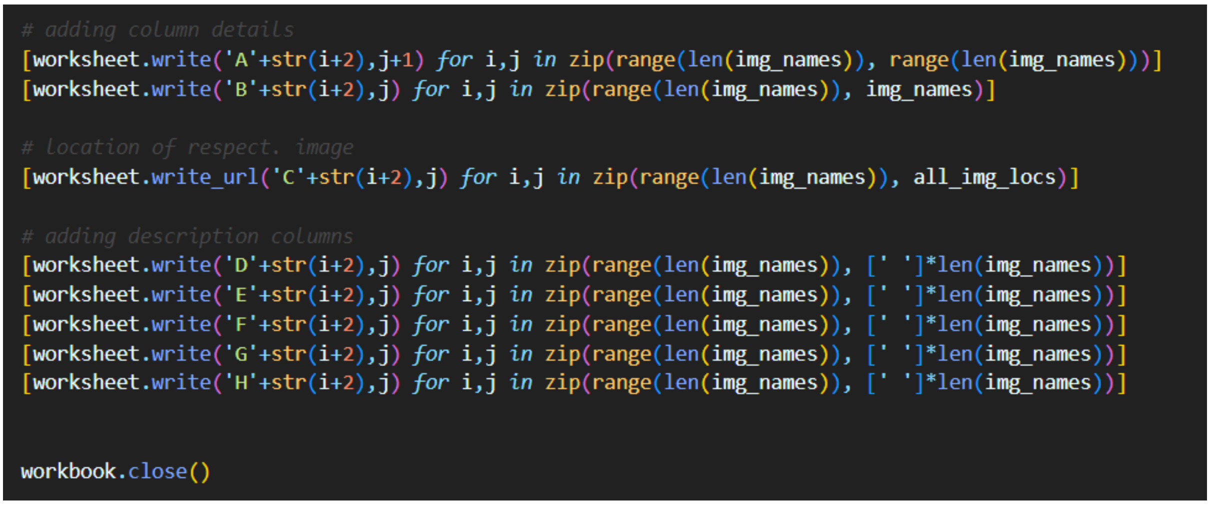
```
import os
import pandas as pd
from pandas import ExcelWriter
import argparse
```



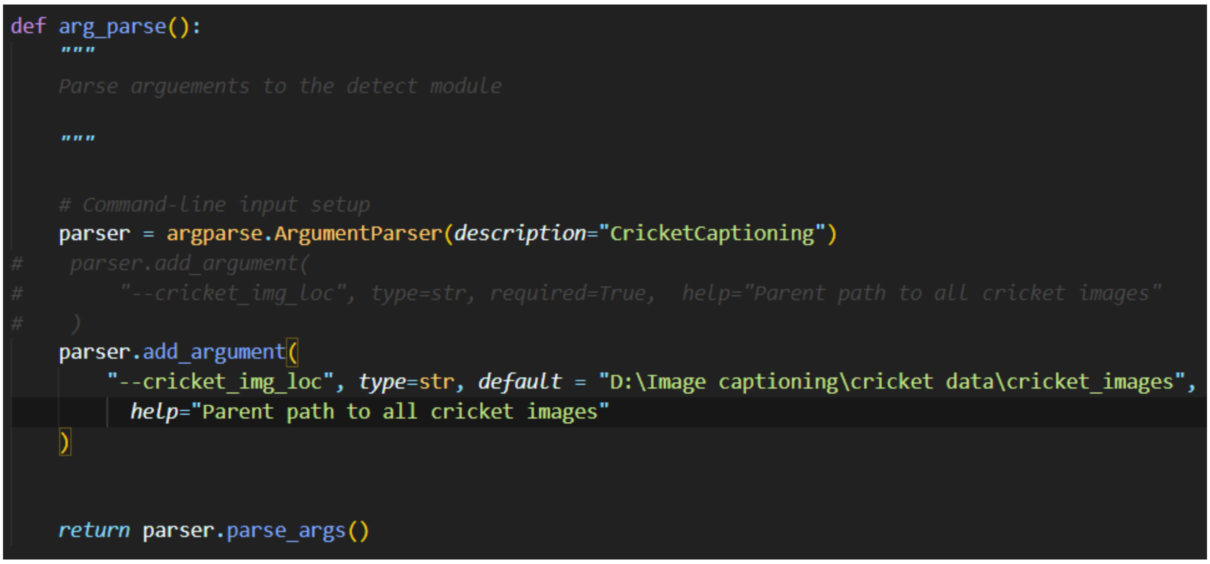
```python
def getListOfFiles(dirName):
    # create a list of file and sub directories
    # names in the given directory
    listOfFile = os.listdir(dirName)
    allFiles = list()
    # Iterate over all the entries
    for entry in listOfFile:
        # Create full path
        fullPath = os.path.join(dirName, entry)
        # If entry is a directory then get the list of files in this direc
        if os.path.isdir(fullPath):
            allFiles = allFiles + getListOfFiles(fullPath)
#         elif (fullPath[-3:]=='jpg') | (fullPath[-3:]=='jpg'):
#             allFiles.append(fullPath)
        else:
            allFiles.append(fullPath)

    return allFiles
```



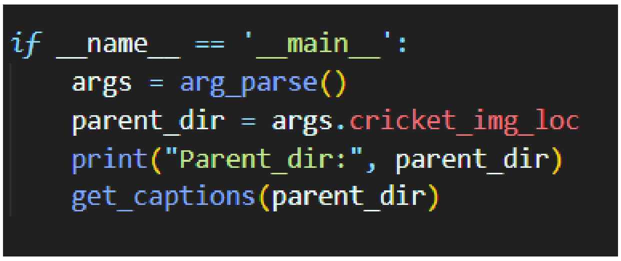
```python
def get_captions(parent_dir):
    #img_loc = r'D:\Image captioning\cricket data\images'
    all_img_locs = getListOfFiles(parent_dir)
    print("Total images:",len(all_img_locs))
    img_names = [img.split('\\')[-1] for img in all_img_locs]
    import xlsxwriter

    workbook_loc = '\\'.join(parent_dir.split("\\")[:-1])

    print("Workbook location:", workbook_loc)
#    workbook = xlsxwriter.Workbook(workbook_loc+r'\cricket_captioning_new.xlsx')
    workbook = xlsxwriter.Workbook(workbook_loc+r'\cricket_captioning_v2.xlsx')

    worksheet = workbook.add_worksheet()

    # adding column names
    [worksheet.write(x+'1',y) for x, y in zip(['A', 'B','C','D','E','F','G','H'],['S.N.','Image Name',
                                                                                     'Image Loc','Image Caption1','Image Caption2',
                                                                                     'Image Caption3','Image Caption4','Image Caption5'])]
```



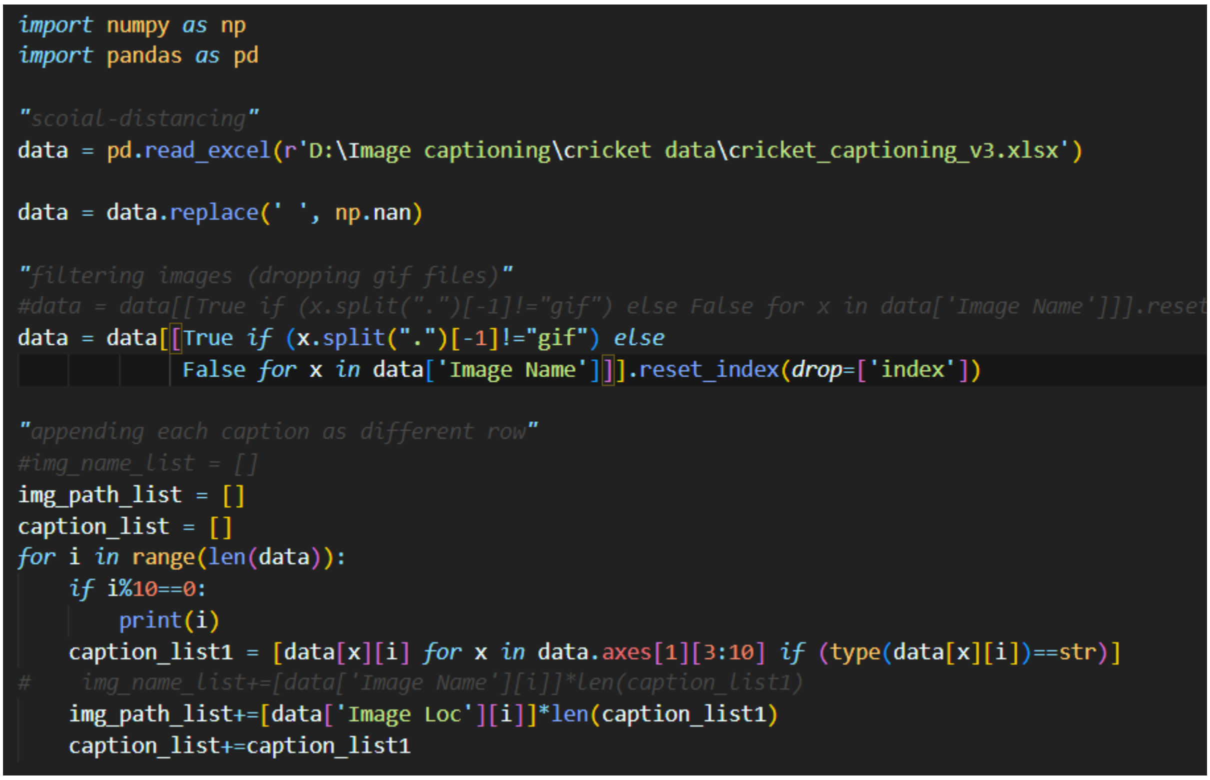
```python
# adding column details
[worksheet.write('A'+str(i+2),j+1) for i,j in zip(range(len(img_names)), range(len(img_names)))]
[worksheet.write('B'+str(i+2),j) for i,j in zip(range(len(img_names)), img_names)]

# location of respect. image
[worksheet.write_url('C'+str(i+2),j) for i,j in zip(range(len(img_names)), all_img_locs)]

# adding description columns
[worksheet.write('D'+str(i+2),j) for i,j in zip(range(len(img_names)), [' ']*len(img_names))]
[worksheet.write('E'+str(i+2),j) for i,j in zip(range(len(img_names)), [' ']*len(img_names))]
[worksheet.write('F'+str(i+2),j) for i,j in zip(range(len(img_names)), [' ']*len(img_names))]
[worksheet.write('G'+str(i+2),j) for i,j in zip(range(len(img_names)), [' ']*len(img_names))]
[worksheet.write('H'+str(i+2),j) for i,j in zip(range(len(img_names)), [' ']*len(img_names))]


workbook.close()
```



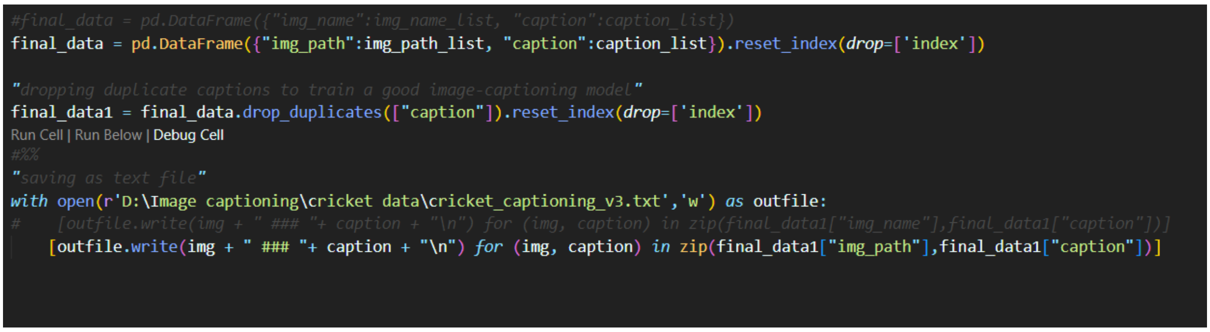
```
def arg_parse():
    """
    Parse arguements to the detect module

    """

    # Command-line input setup
    parser = argparse.ArgumentParser(description="CricketCaptioning")
#    parser.add_argument(
#        "--cricket_img_loc", type=str, required=True,  help="Parent path to all cricket images"
#    )
    parser.add_argument(
        "--cricket_img_loc", type=str, default = "D:\Image captioning\cricket data\cricket_images",
          help="Parent path to all cricket images"
    )


    return parser.parse_args()
```



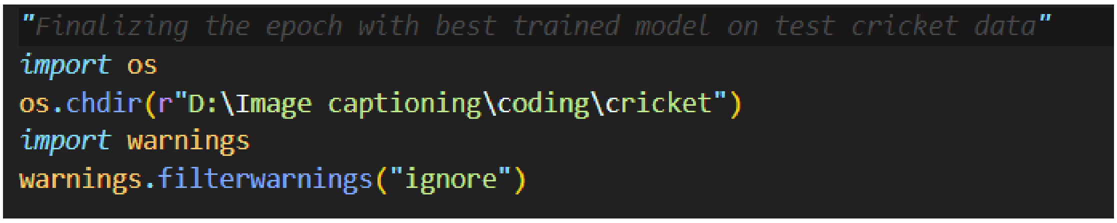
```
if __name__ == '__main__':
    args = arg_parse()
    parent_dir = args.cricket_img_loc
    print("Parent_dir:", parent_dir)
    get_captions(parent_dir)
```



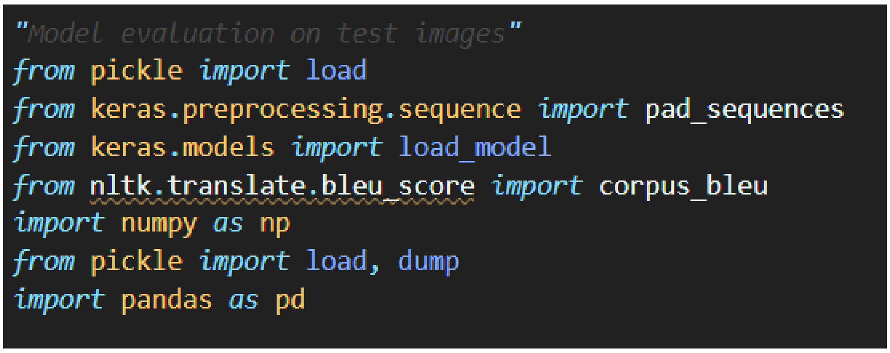
```
import numpy as np
import pandas as pd

"scoial-distancing"
data = pd.read_excel(r'D:\Image captioning\cricket data\cricket_captioning_v3.xlsx')

data = data.replace(' ', np.nan)

"filtering images (dropping gif files)"
#data = data[[True if (x.split(".")[-1]!="gif") else False for x in data['Image Name']]].reset
data = data[[True if (x.split(".")[-1]!="gif") else
             False for x in data['Image Name']]].reset_index(drop=['index'])

"appending each caption as different row"
#img_name_list = []
img_path_list = []
caption_list = []
for i in range(len(data)):
    if i%10==0:
        print(i)
    caption_list1 = [data[x][i] for x in data.axes[1][3:10] if (type(data[x][i])==str)]
#    img_name_list+=[data['Image Name'][i]]*len(caption_list1)
    img_path_list+=[data['Image Loc'][i]]*len(caption_list1)
    caption_list+=caption_list1
```



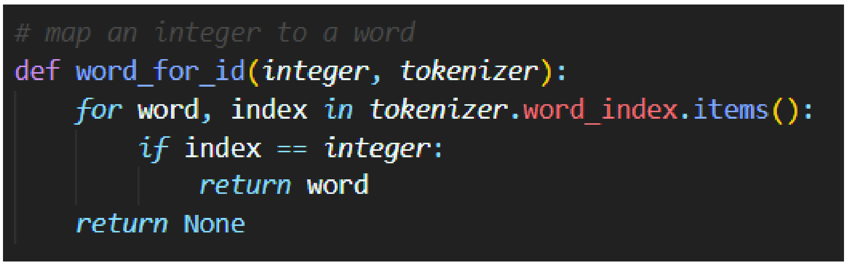
```
#final_data = pd.DataFrame({"img_name":img_name_list, "caption":caption_list})
final_data = pd.DataFrame({"img_path":img_path_list, "caption":caption_list}).reset_index(drop=['index'])

"dropping duplicate captions to train a good image-captioning model"
final_data1 = final_data.drop_duplicates(["caption"]).reset_index(drop=['index'])
Run Cell | Run Below | Debug Cell
#%%
"saving as text file"
with open(r'D:\Image captioning\cricket data\cricket_captioning_v3.txt','w') as outfile:
#    [outfile.write(img + " ### "+ caption + "\n") for (img, caption) in zip(final_data1["img_name"],final_data1["caption"])]
    [outfile.write(img + " ### "+ caption + "\n") for (img, caption) in zip(final_data1["img_path"],final_data1["caption"])]
```



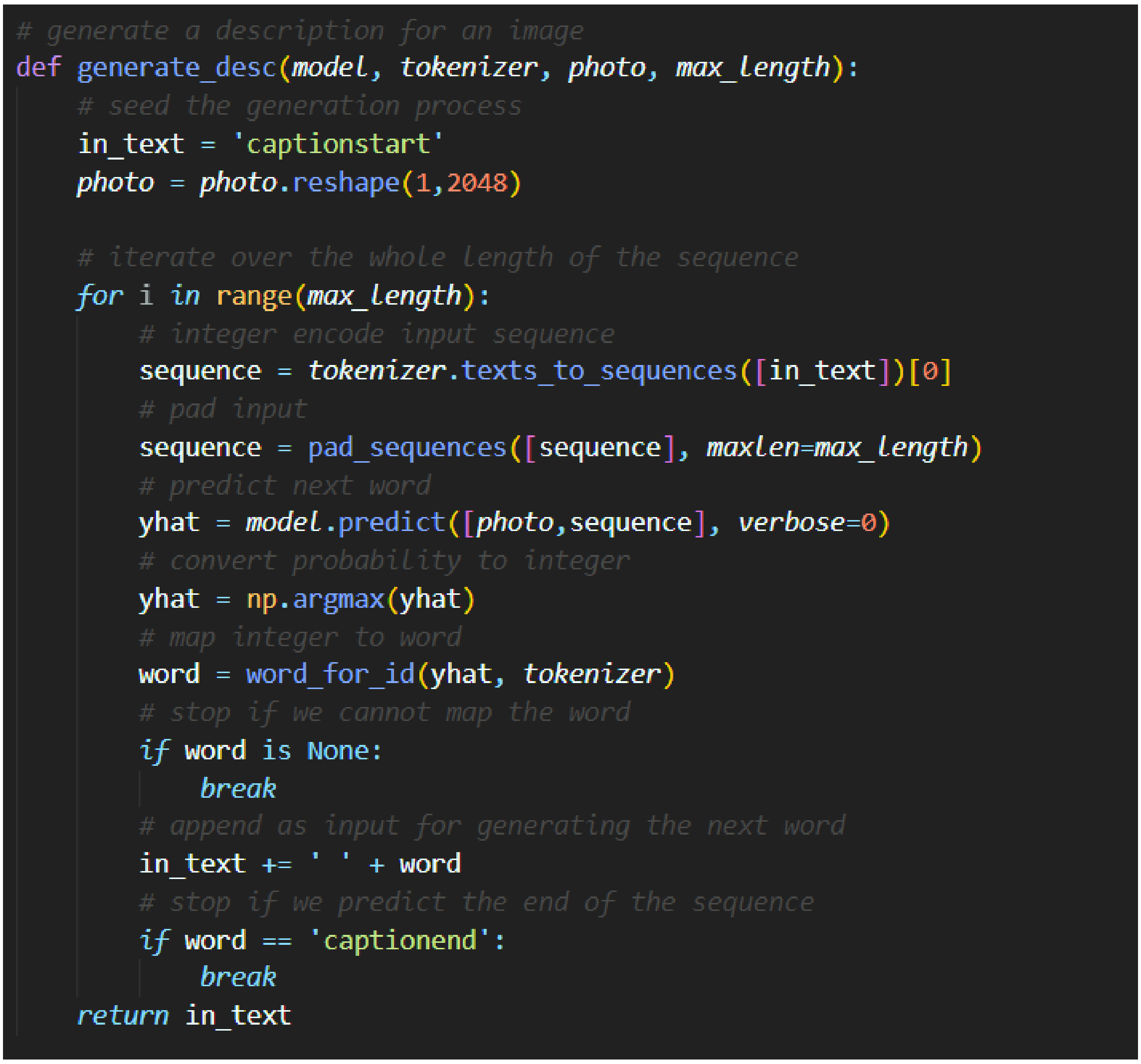
```
"Finalizing the epoch with best trained model on test cricket data"
import os
os.chdir(r"D:\Image captioning\coding\cricket")
import warnings
warnings.filterwarnings("ignore")
```



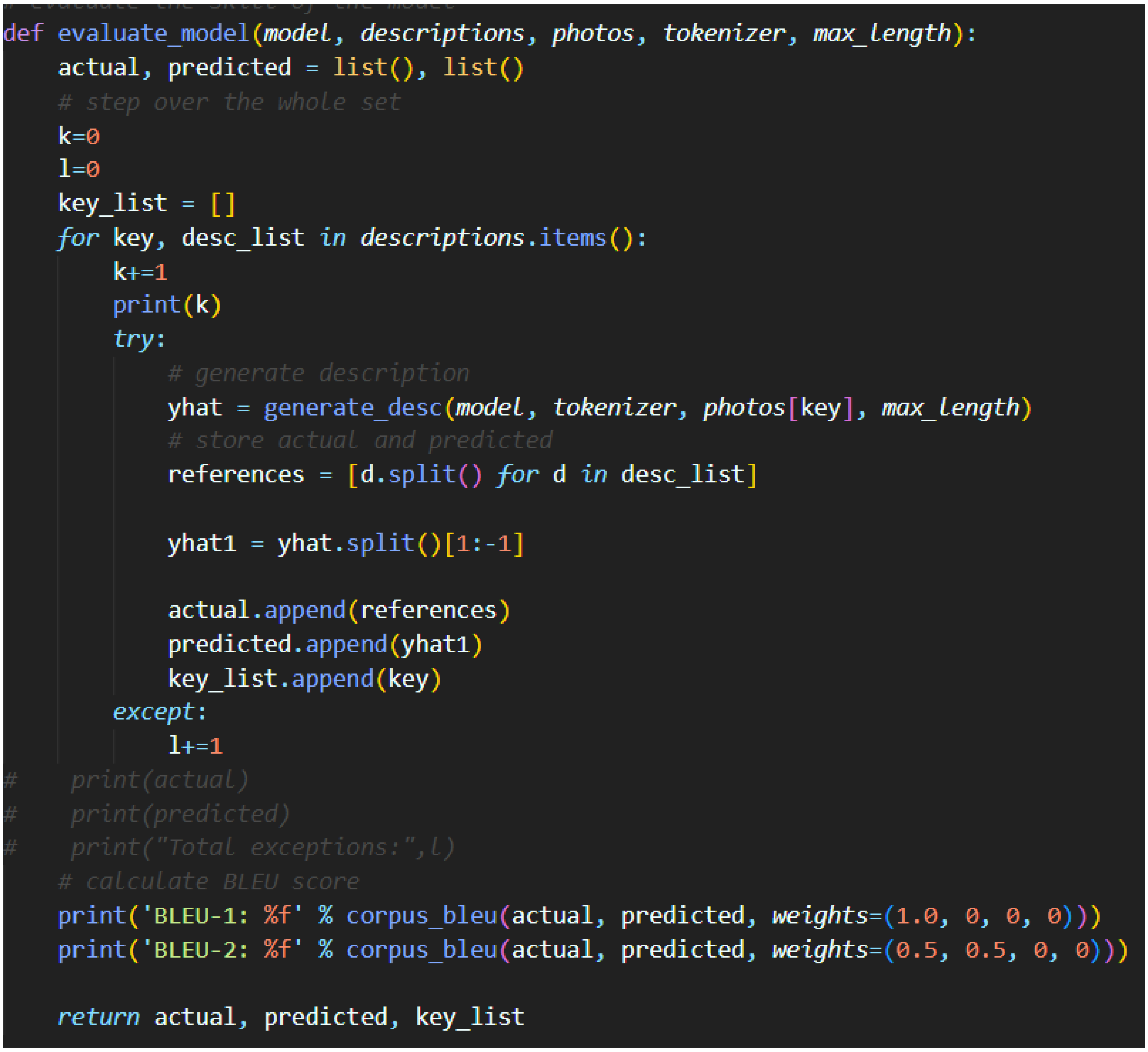
```
"Model evaluation on test images"
from pickle import load
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu
import numpy as np
from pickle import load, dump
import pandas as pd
```



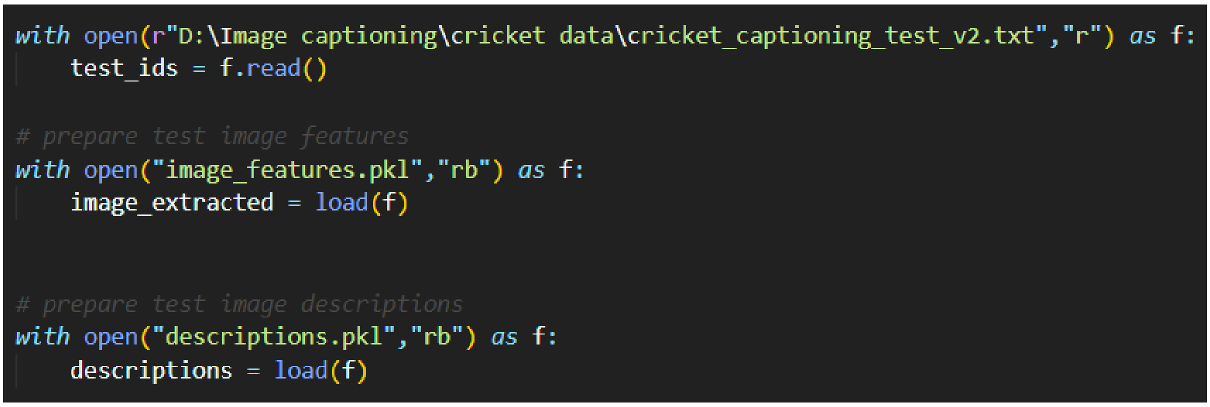
```
# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```



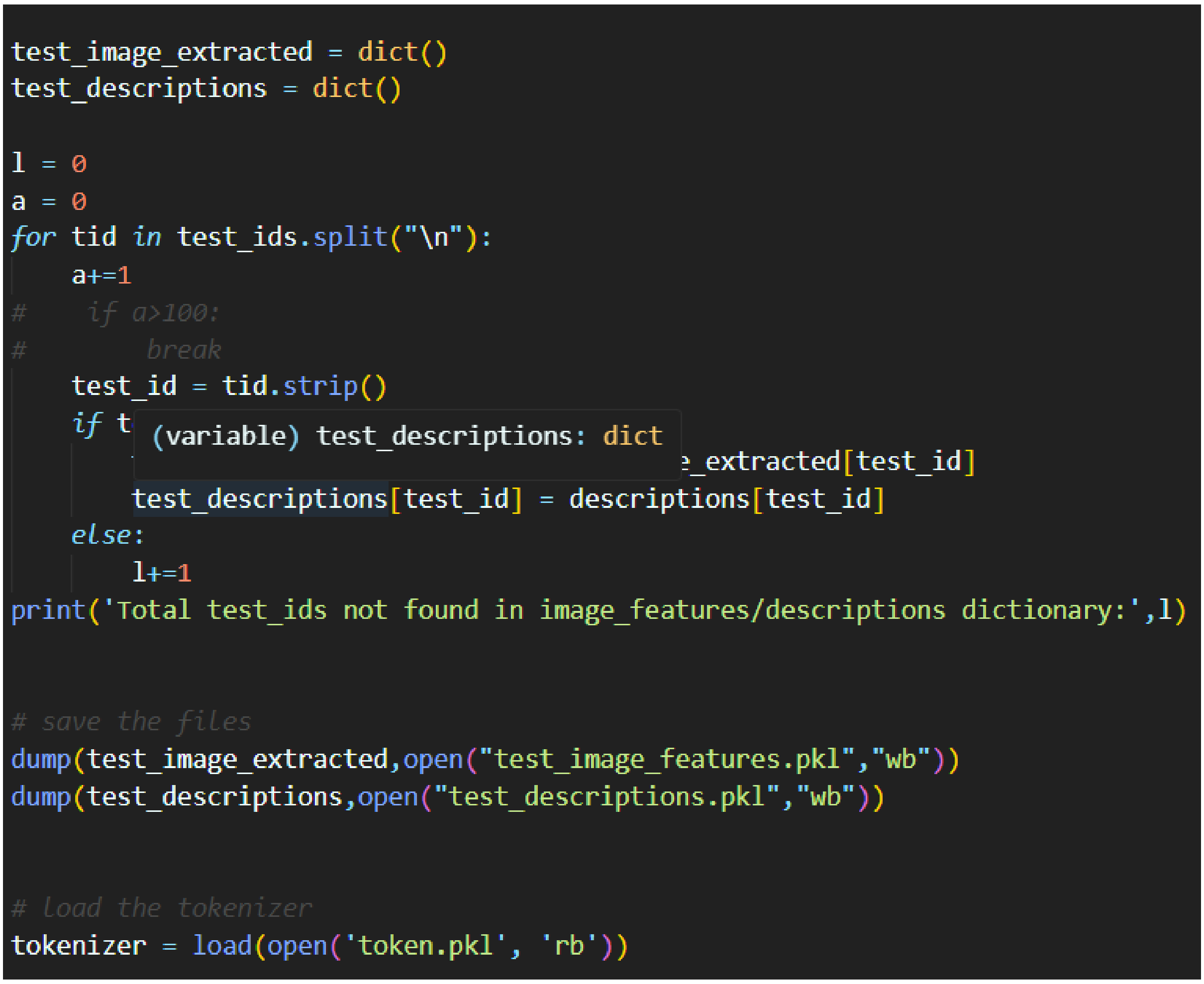
```python
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'captionstart'
    photo = photo.reshape(1,2048)

    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
        # predict next word
        yhat = model.predict([photo,sequence], verbose=0)
        # convert probability to integer
        yhat = np.argmax(yhat)
        # map integer to word
        word = word_for_id(yhat, tokenizer)
        # stop if we cannot map the word
        if word is None:
            break
        # append as input for generating the next word
        in_text += ' ' + word
        # stop if we predict the end of the sequence
        if word == 'captionend':
            break
    return in_text
```



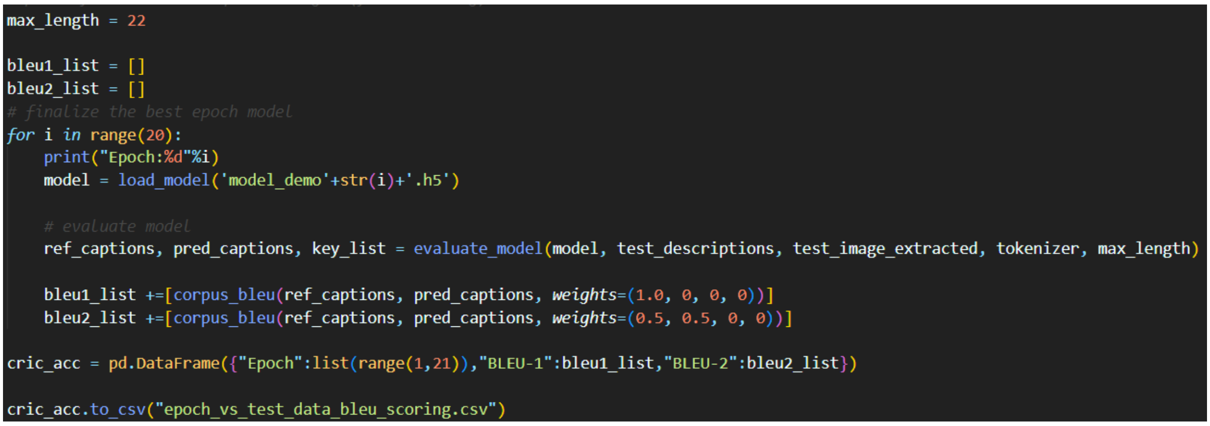
```
# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
    actual, predicted = list(), list()
    # step over the whole set
    k=0
    l=0
    key_list = []
    for key, desc_list in descriptions.items():
        k+=1
        print(k)
        try:
            # generate description
            yhat = generate_desc(model, tokenizer, photos[key], max_length)
            # store actual and predicted
            references = [d.split() for d in desc_list]

            yhat1 = yhat.split()[1:-1]

            actual.append(references)
            predicted.append(yhat1)
            key_list.append(key)
        except:
            l+=1
#     print(actual)
#     print(predicted)
#     print("Total exceptions:",l)
    # calculate BLEU score
    print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))

    return actual, predicted, key_list
```



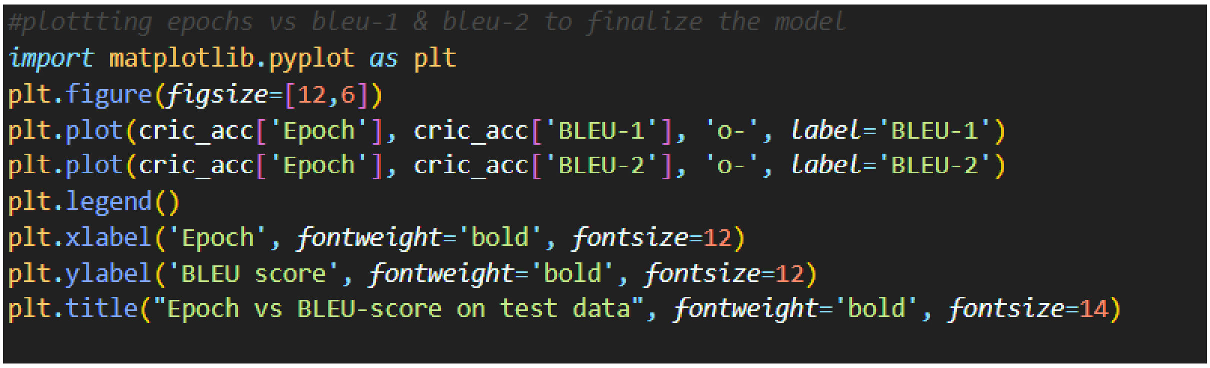
```
with open(r"D:\Image captioning\cricket data\cricket_captioning_test_v2.txt","r") as f:
    test_ids = f.read()

# prepare test image features
with open("image_features.pkl","rb") as f:
    image_extracted = load(f)


# prepare test image descriptions
with open("descriptions.pkl","rb") as f:
    descriptions = load(f)
```



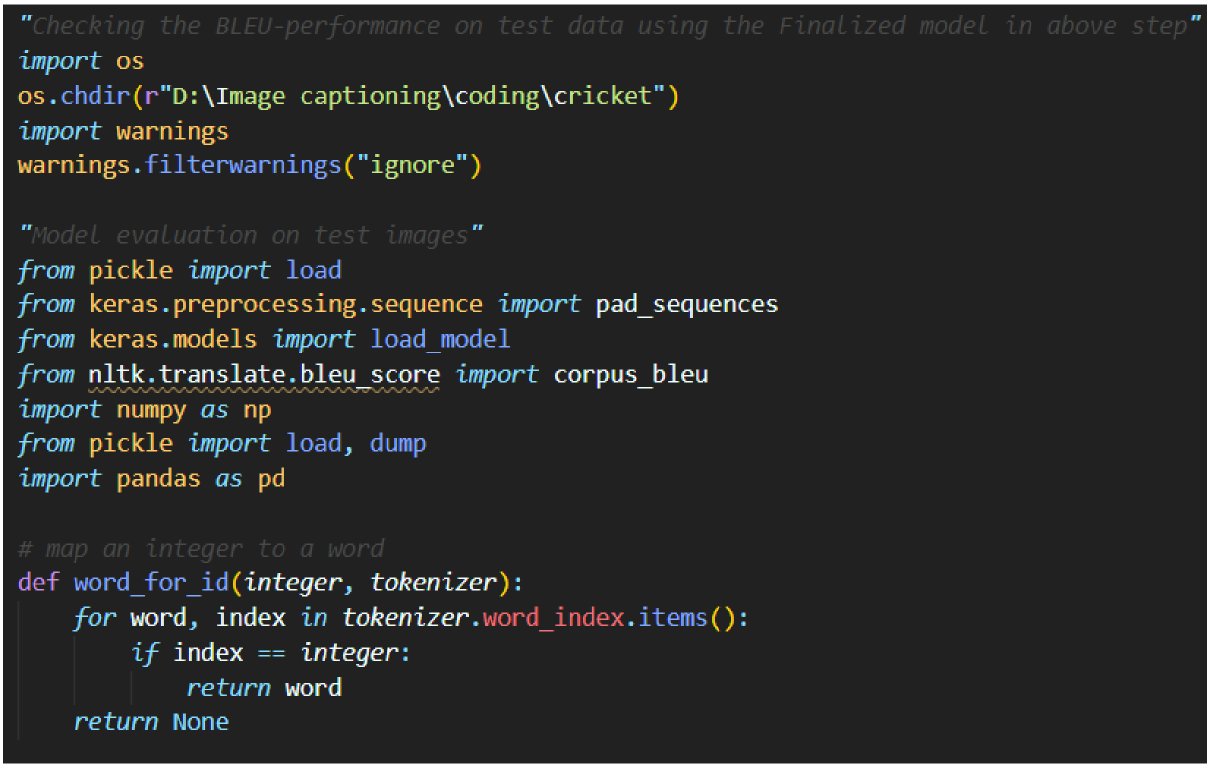
```
test_image_extracted = dict()
test_descriptions = dict()

l = 0
a = 0
for tid in test_ids.split("\n"):
    a+=1
#    if a>100:
#        break
    test_id = tid.strip()
    if t (variable) test_descriptions: dict
                                             e_extracted[test_id]
        test_descriptions[test_id] = descriptions[test_id]
    else:
        l+=1
print('Total test_ids not found in image_features/descriptions dictionary:',l)


# save the files
dump(test_image_extracted,open("test_image_features.pkl","wb"))
dump(test_descriptions,open("test_descriptions.pkl","wb"))


# load the tokenizer
tokenizer = load(open('token.pkl', 'rb'))
```



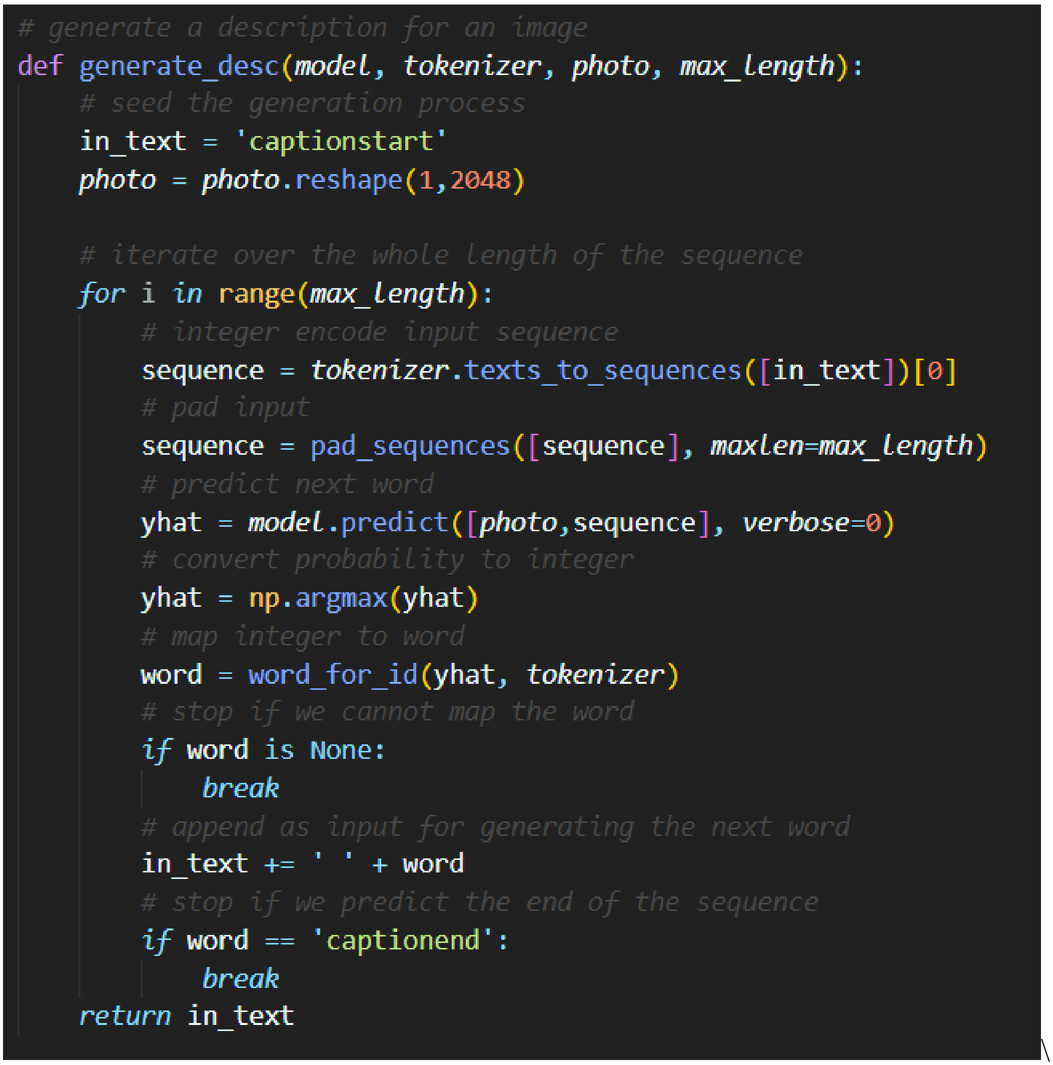
```
max_length = 22

bleu1_list = []
bleu2_list = []
# finalize the best epoch model
for i in range(20):
    print("Epoch:%d"%i)
    model = load_model('model_demo'+str(i)+'.h5')

    # evaluate model
    ref_captions, pred_captions, key_list = evaluate_model(model, test_descriptions, test_image_extracted, tokenizer, max_length)

    bleu1_list +=[corpus_bleu(ref_captions, pred_captions, weights=(1.0, 0, 0, 0))]
    bleu2_list +=[corpus_bleu(ref_captions, pred_captions, weights=(0.5, 0.5, 0, 0))]

cric_acc = pd.DataFrame({"Epoch":list(range(1,21)),"BLEU-1":bleu1_list,"BLEU-2":bleu2_list})

cric_acc.to_csv("epoch_vs_test_data_bleu_scoring.csv")
```



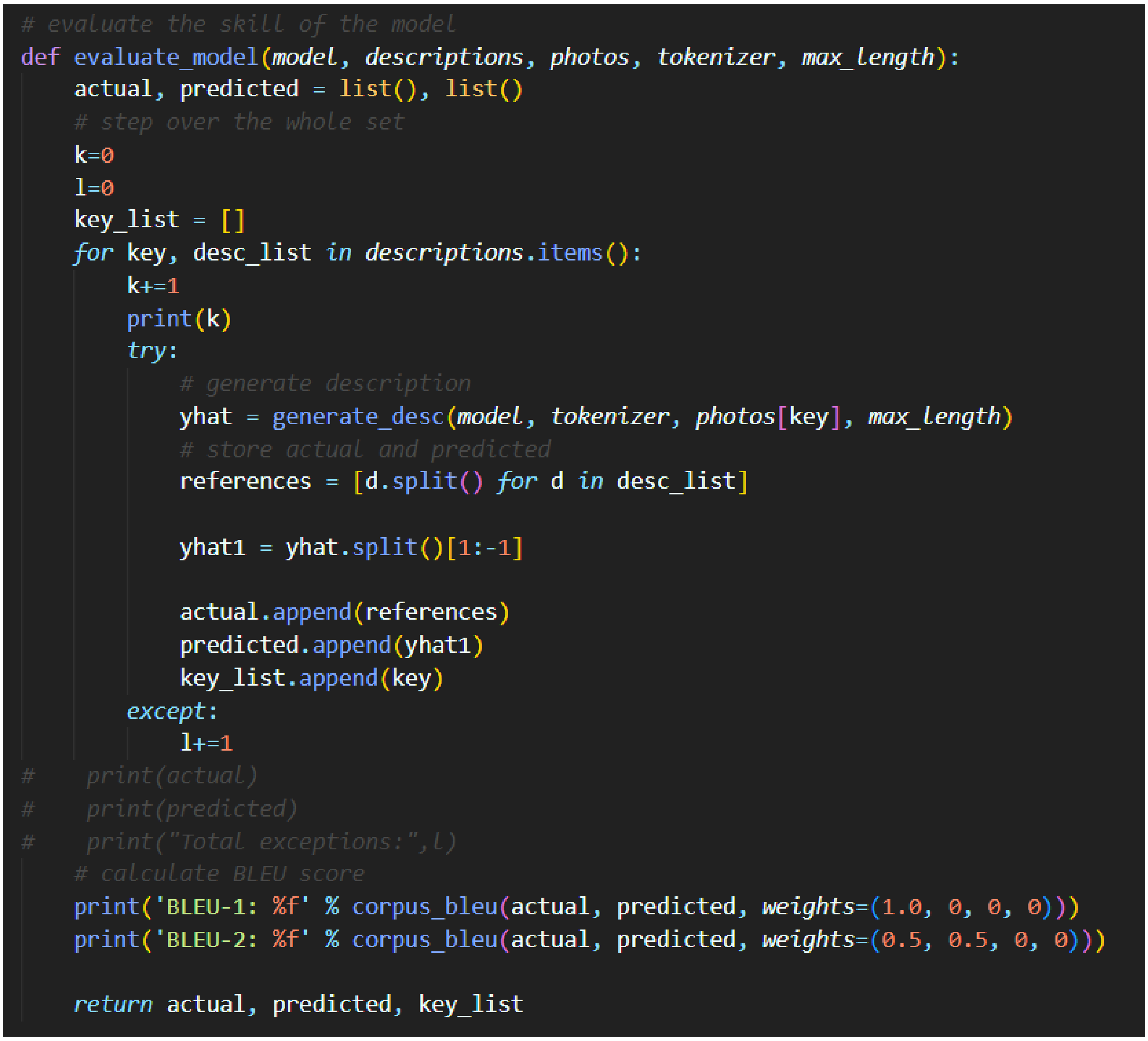
```
#plottting epochs vs bleu-1 & bleu-2 to finalize the model
import matplotlib.pyplot as plt
plt.figure(figsize=[12,6])
plt.plot(cric_acc['Epoch'], cric_acc['BLEU-1'], 'o-', label='BLEU-1')
plt.plot(cric_acc['Epoch'], cric_acc['BLEU-2'], 'o-', label='BLEU-2')
plt.legend()
plt.xlabel('Epoch', fontweight='bold', fontsize=12)
plt.ylabel('BLEU score', fontweight='bold', fontsize=12)
plt.title("Epoch vs BLEU-score on test data", fontweight='bold', fontsize=14)
```



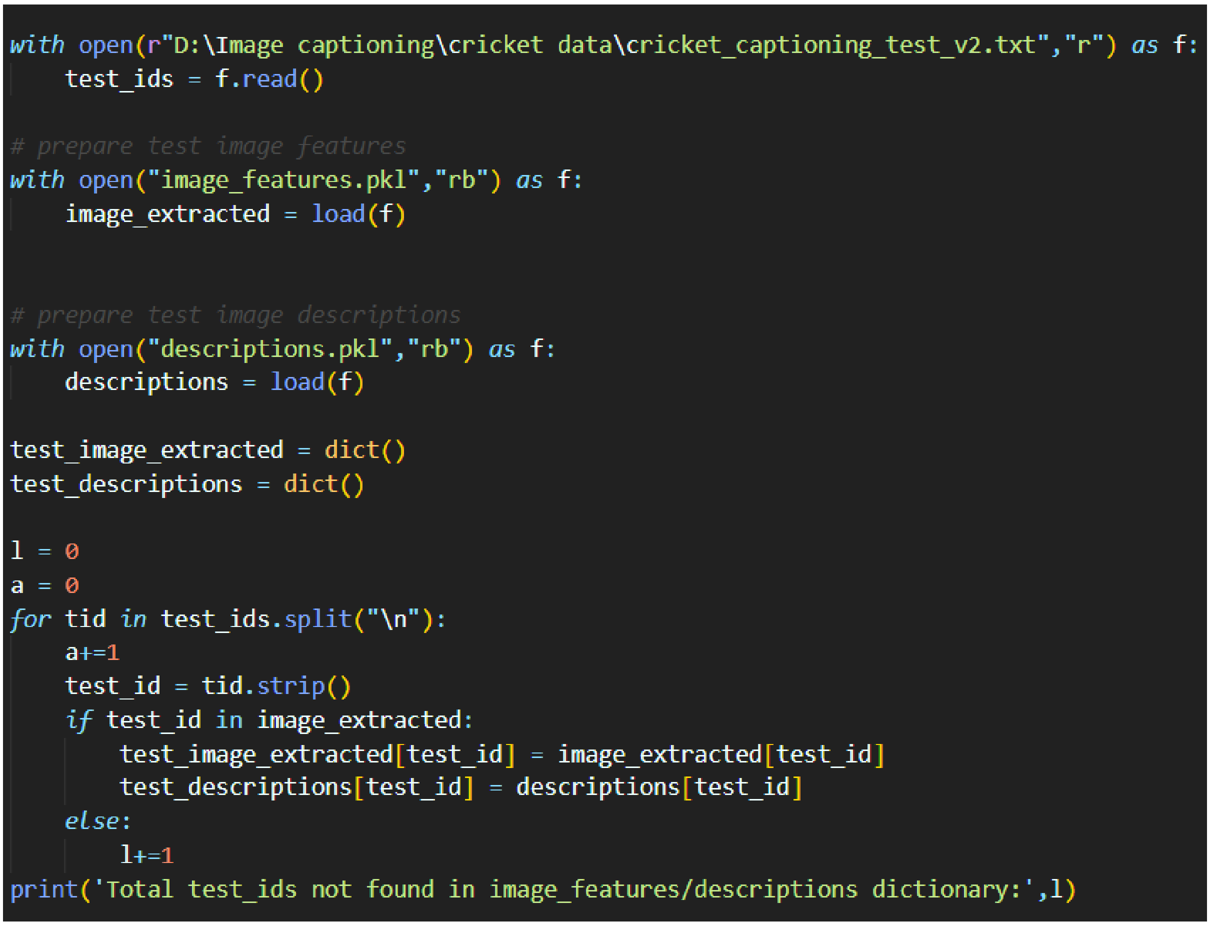
```
”Checking the BLEU-performance on test data using the Finalized model in above step”
import os
os.chdir(r"D:\Image captioning\coding\cricket")
import warnings
warnings.filterwarnings("ignore")

”Model evaluation on test images”
from pickle import load
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu
import numpy as np
from pickle import load, dump
import pandas as pd

# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```



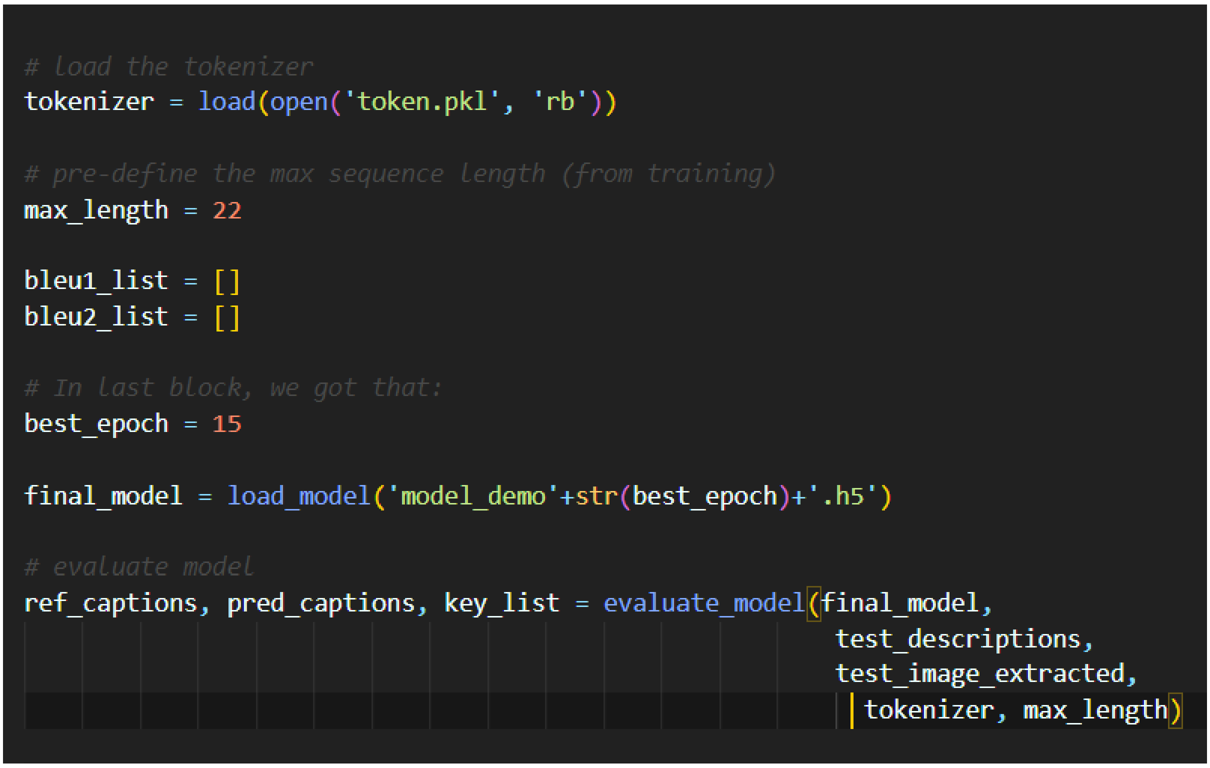
```
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'captionstart'
    photo = photo.reshape(1,2048)

    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
        # predict next word
        yhat = model.predict([photo,sequence], verbose=0)
        # convert probability to integer
        yhat = np.argmax(yhat)
        # map integer to word
        word = word_for_id(yhat, tokenizer)
        # stop if we cannot map the word
        if word is None:
            break
        # append as input for generating the next word
        in_text += ' ' + word
        # stop if we predict the end of the sequence
        if word == 'captionend':
            break
    return in_text
```



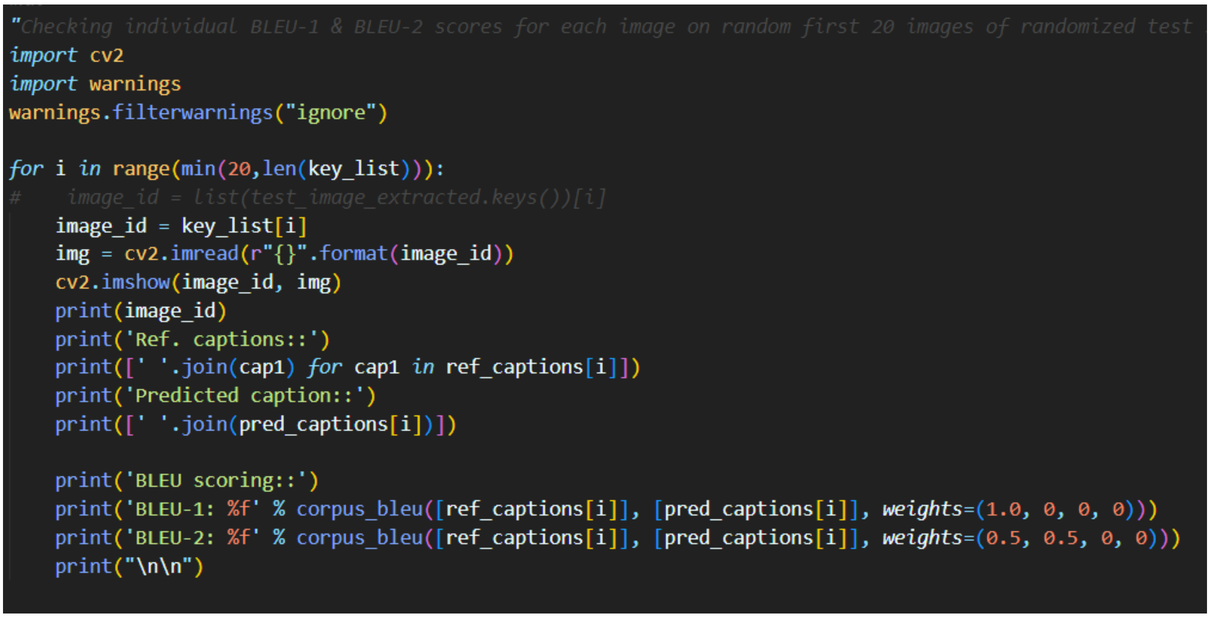
```python
# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
    actual, predicted = list(), list()
    # step over the whole set
    k=0
    l=0
    key_list = []
    for key, desc_list in descriptions.items():
        k+=1
        print(k)
        try:
            # generate description
            yhat = generate_desc(model, tokenizer, photos[key], max_length)
            # store actual and predicted
            references = [d.split() for d in desc_list]

            yhat1 = yhat.split()[1:-1]

            actual.append(references)
            predicted.append(yhat1)
            key_list.append(key)
        except:
            l+=1
#     print(actual)
#     print(predicted)
#     print("Total exceptions:",l)
    # calculate BLEU score
    print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))

    return actual, predicted, key_list
```



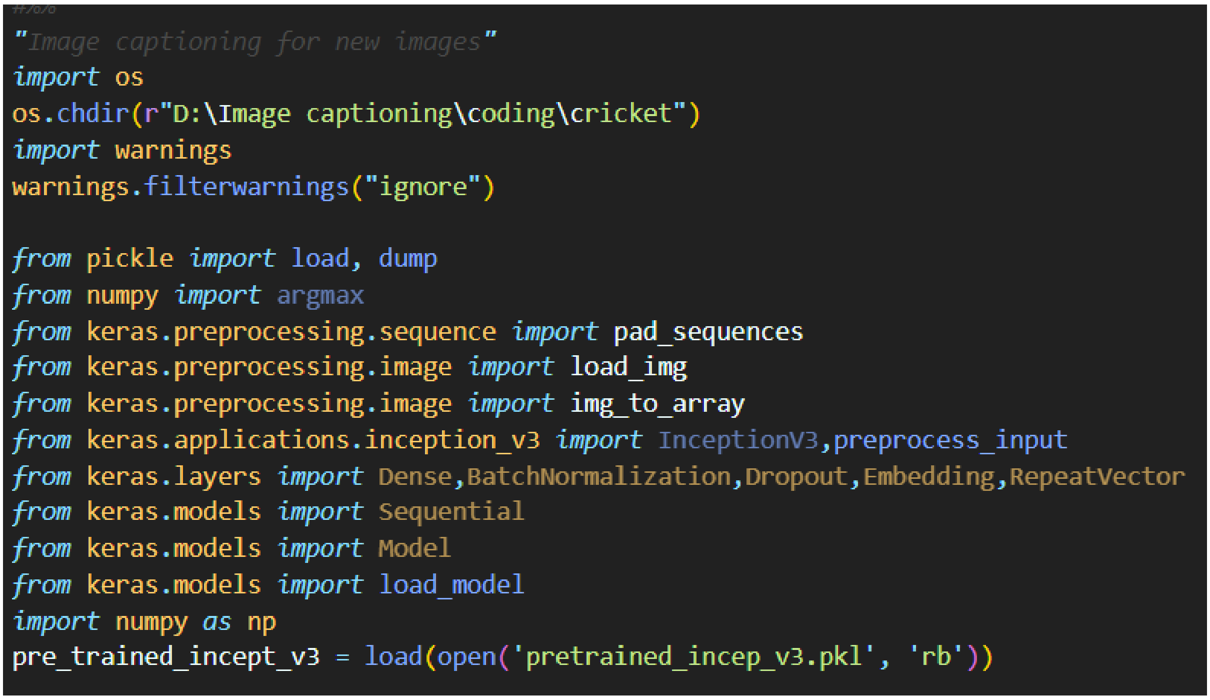
```
with open(r"D:\Image captioning\cricket data\cricket_captioning_test_v2.txt","r") as f:
    test_ids = f.read()

# prepare test image features
with open("image_features.pkl","rb") as f:
    image_extracted = load(f)


# prepare test image descriptions
with open("descriptions.pkl","rb") as f:
    descriptions = load(f)

test_image_extracted = dict()
test_descriptions = dict()

l = 0
a = 0
for tid in test_ids.split("\n"):
    a+=1
    test_id = tid.strip()
    if test_id in image_extracted:
        test_image_extracted[test_id] = image_extracted[test_id]
        test_descriptions[test_id] = descriptions[test_id]
    else:
        l+=1
print('Total test_ids not found in image_features/descriptions dictionary:',l)
```



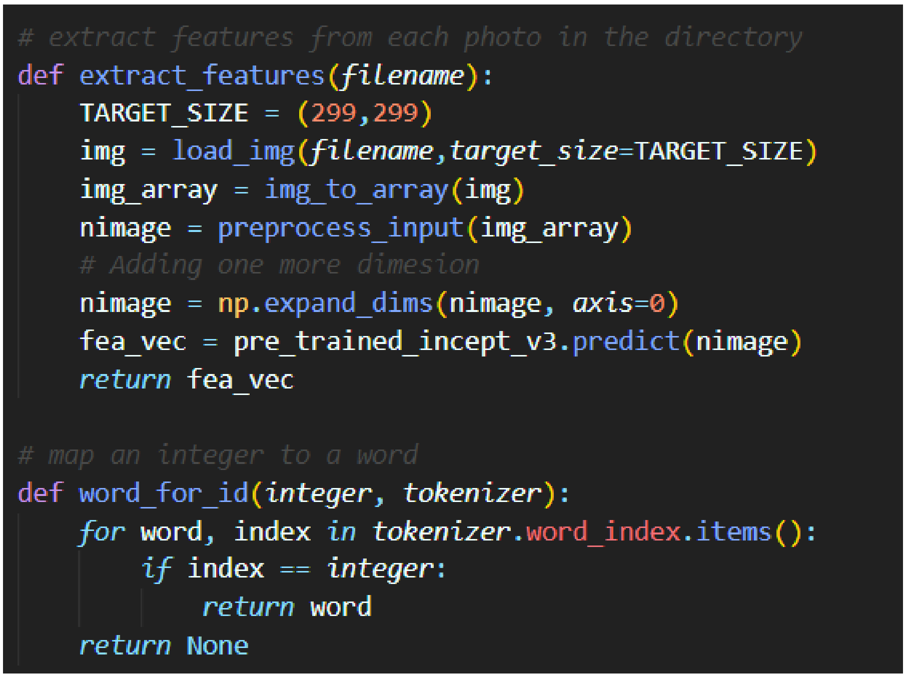
```
# load the tokenizer
tokenizer = load(open('token.pkl', 'rb'))

# pre-define the max sequence length (from training)
max_length = 22

bleu1_list = []
bleu2_list = []

# In last block, we got that:
best_epoch = 15

final_model = load_model('model_demo'+str(best_epoch)+'.h5')

# evaluate model
ref_captions, pred_captions, key_list = evaluate_model(final_model,
                                                        test_descriptions,
                                                        test_image_extracted,
                                                        tokenizer, max_length)
```



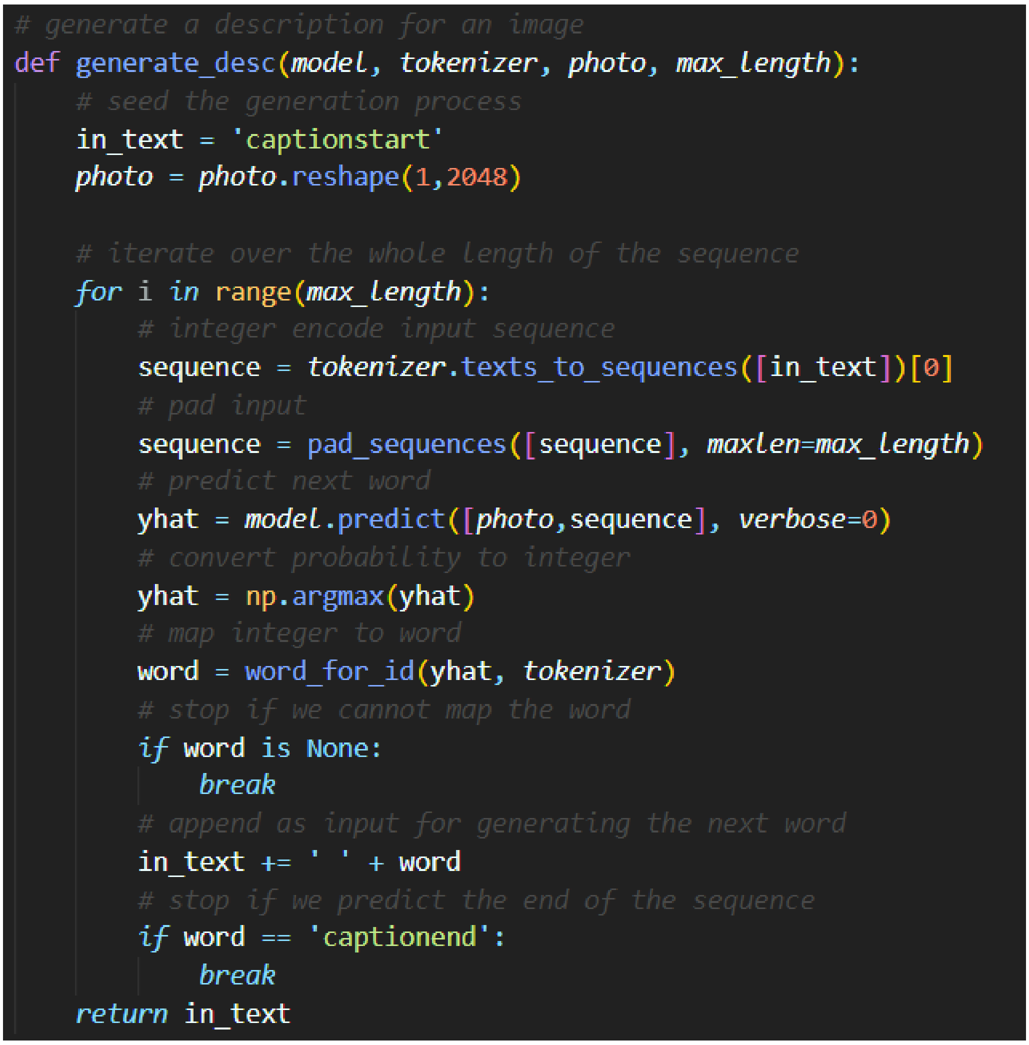
```
"Checking individual BLEU-1 & BLEU-2 scores for each image on random first 20 images of randomized test
import cv2
import warnings
warnings.filterwarnings("ignore")

for i in range(min(20,len(key_list))):
#    image_id = list(test_image_extracted.keys())[i]
    image_id = key_list[i]
    img = cv2.imread(r"{}".format(image_id))
    cv2.imshow(image_id, img)
    print(image_id)
    print('Ref. captions::')
    print([' '.join(cap1) for cap1 in ref_captions[i]])
    print('Predicted caption::')
    print([' '.join(pred_captions[i])])

    print('BLEU scoring::')
    print('BLEU-1: %f' % corpus_bleu([ref_captions[i]], [pred_captions[i]], weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu([ref_captions[i]], [pred_captions[i]], weights=(0.5, 0.5, 0, 0)))
    print("\n\n")
```



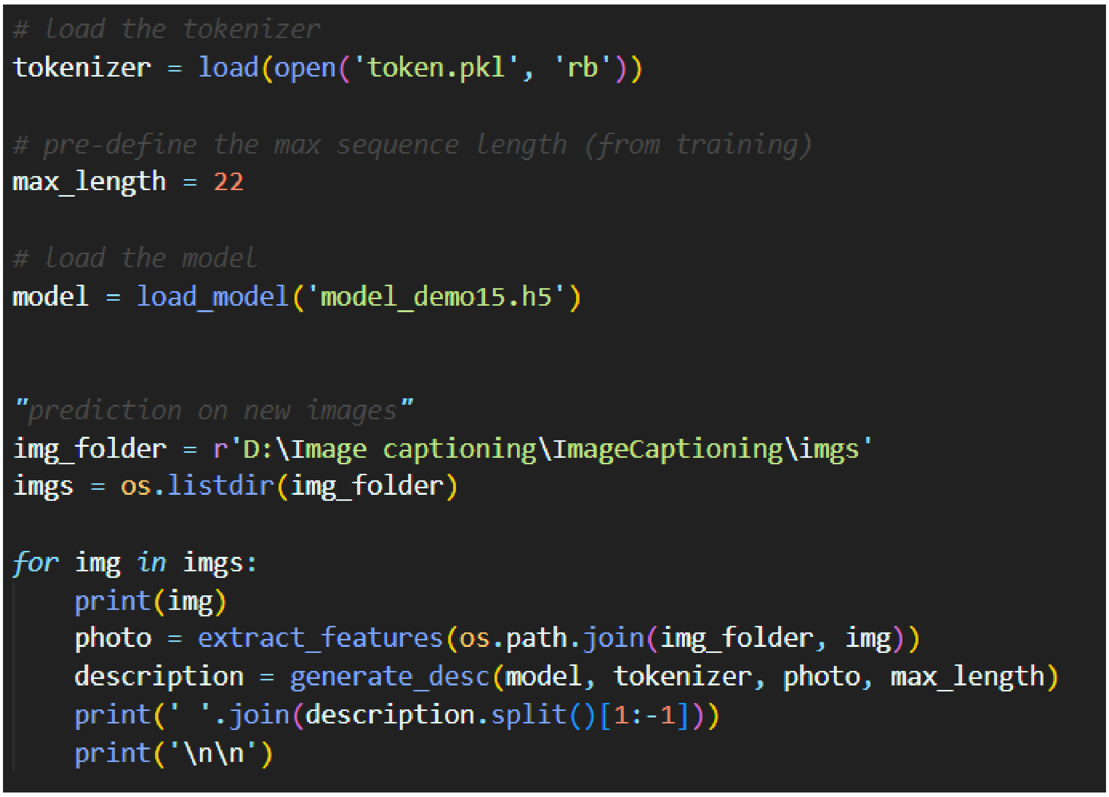
```
#%%
"Image captioning for new images"
import os
os.chdir(r"D:\Image captioning\coding\cricket")
import warnings
warnings.filterwarnings("ignore")

from pickle import load, dump
from numpy import argmax
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.layers import Dense,BatchNormalization,Dropout,Embedding,RepeatVector
from keras.models import Sequential
from keras.models import Model
from keras.models import load_model
import numpy as np
pre_trained_incept_v3 = load(open('pretrained_incep_v3.pkl', 'rb'))
```



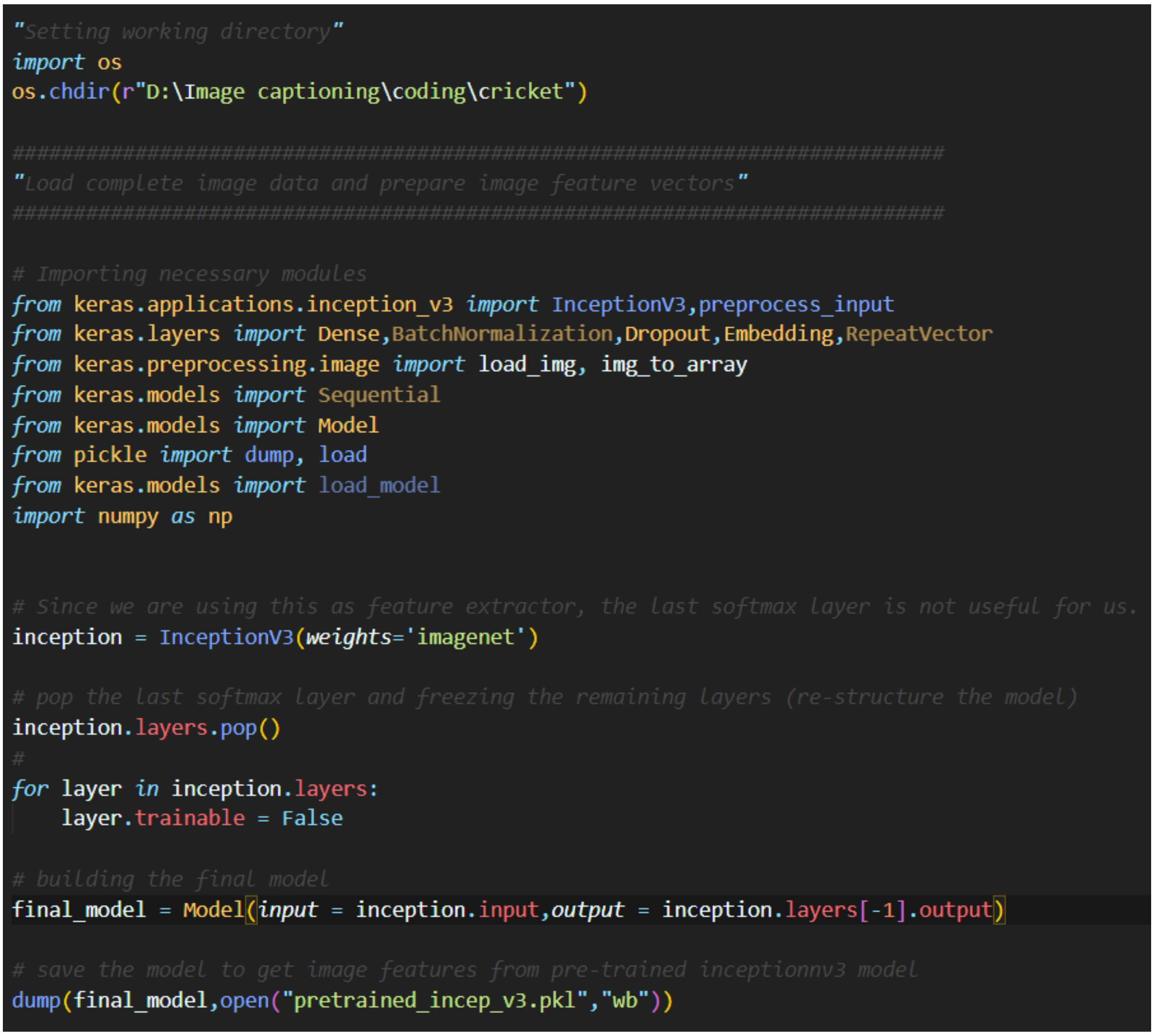
```
# extract features from each photo in the directory
def extract_features(filename):
    TARGET_SIZE = (299,299)
    img = load_img(filename,target_size=TARGET_SIZE)
    img_array = img_to_array(img)
    nimage = preprocess_input(img_array)
    # Adding one more dimesion
    nimage = np.expand_dims(nimage, axis=0)
    fea_vec = pre_trained_incept_v3.predict(nimage)
    return fea_vec

# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```



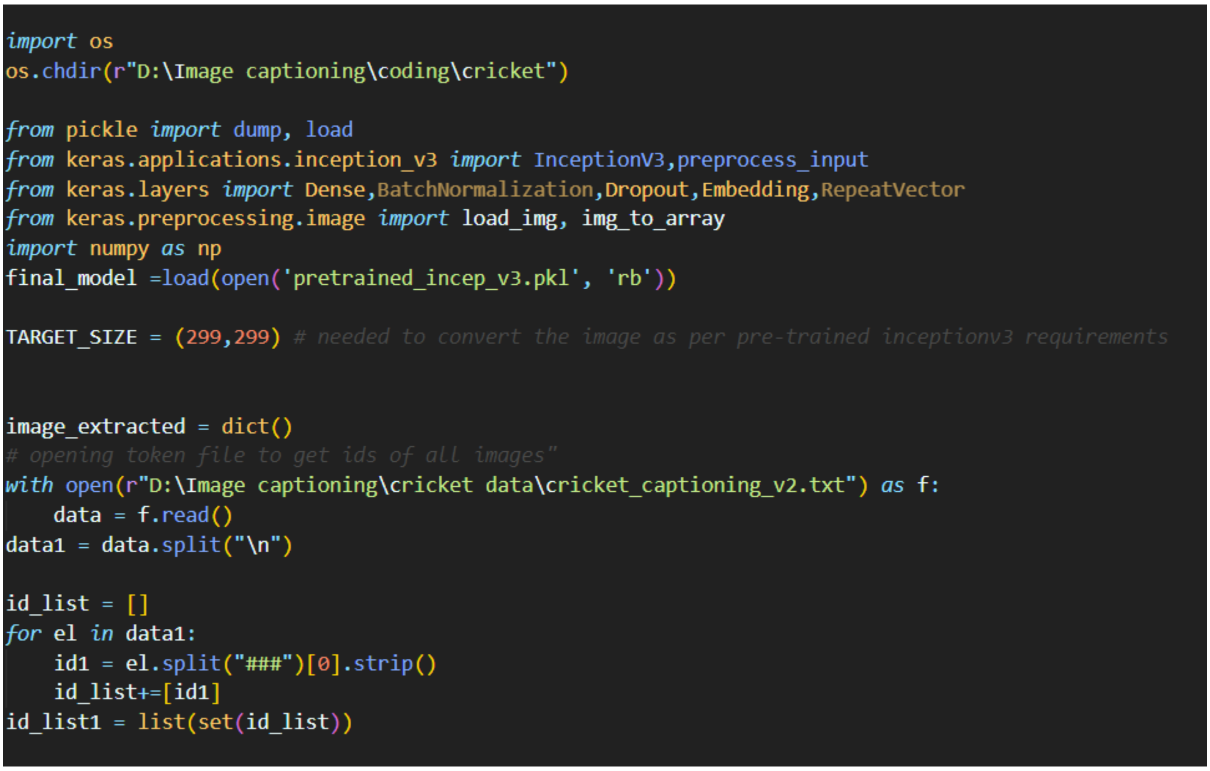
```
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'captionstart'
    photo = photo.reshape(1,2048)

    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
        # predict next word
        yhat = model.predict([photo,sequence], verbose=0)
        # convert probability to integer
        yhat = np.argmax(yhat)
        # map integer to word
        word = word_for_id(yhat, tokenizer)
        # stop if we cannot map the word
        if word is None:
            break
        # append as input for generating the next word
        in_text += ' ' + word
        # stop if we predict the end of the sequence
        if word == 'captionend':
            break
    return in_text
```



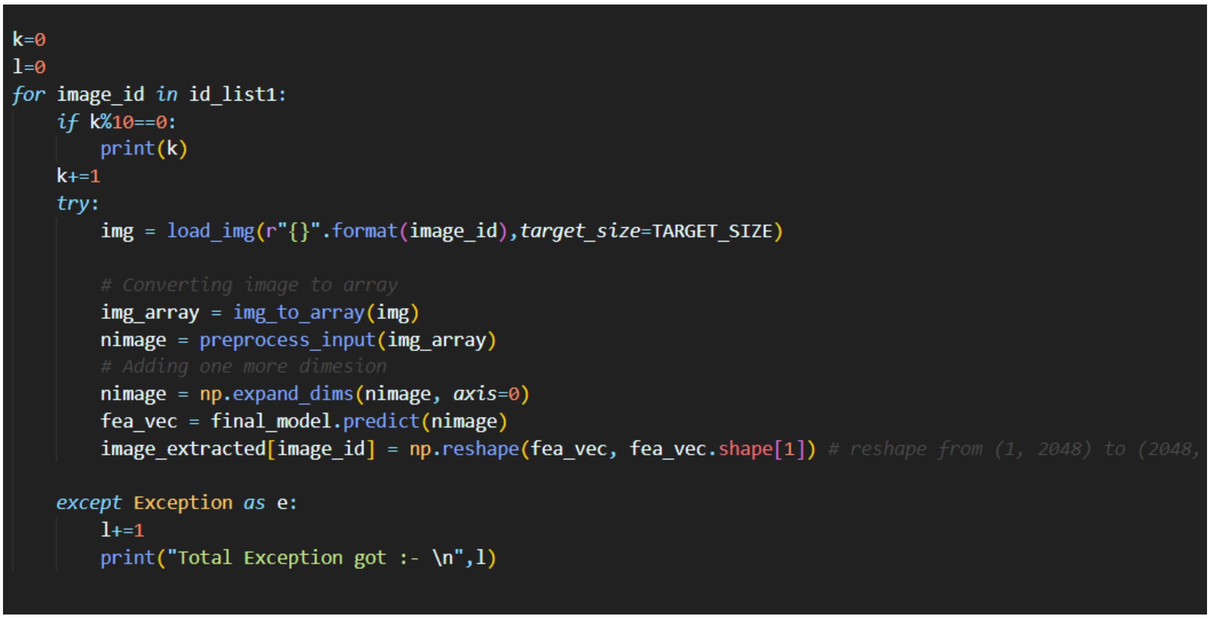
```python
# load the tokenizer
tokenizer = load(open('token.pkl', 'rb'))

# pre-define the max sequence length (from training)
max_length = 22

# load the model
model = load_model('model_demo15.h5')


"prediction on new images"
img_folder = r'D:\Image captioning\ImageCaptioning\imgs'
imgs = os.listdir(img_folder)

for img in imgs:
    print(img)
    photo = extract_features(os.path.join(img_folder, img))
    description = generate_desc(model, tokenizer, photo, max_length)
    print(' '.join(description.split()[1:-1]))
    print('\n\n')
```



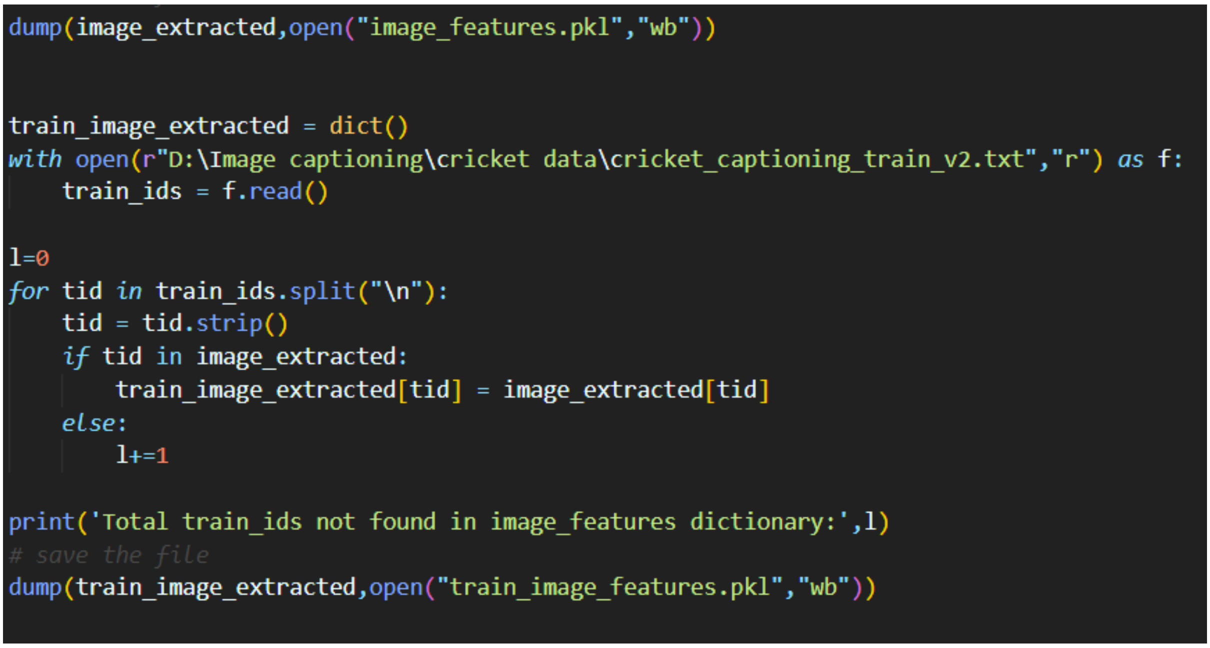
```
"Setting working directory"
import os
os.chdir(r"D:\Image captioning\coding\cricket")

############################################################################
"Load complete image data and prepare image feature vectors"
############################################################################

# Importing necessary modules
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.layers import Dense,BatchNormalization,Dropout,Embedding,RepeatVector
from keras.preprocessing.image import load_img, img_to_array
from keras.models import Sequential
from keras.models import Model
from pickle import dump, load
from keras.models import load_model
import numpy as np


# Since we are using this as feature extractor, the last softmax layer is not useful for us.
inception = InceptionV3(weights='imagenet')

# pop the last softmax layer and freezing the remaining layers (re-structure the model)
inception.layers.pop()
#
for layer in inception.layers:
    layer.trainable = False

# building the final model
final_model = Model(input = inception.input,output = inception.layers[-1].output)

# save the model to get image features from pre-trained inceptionnv3 model
dump(final_model,open("pretrained_incep_v3.pkl","wb"))
```



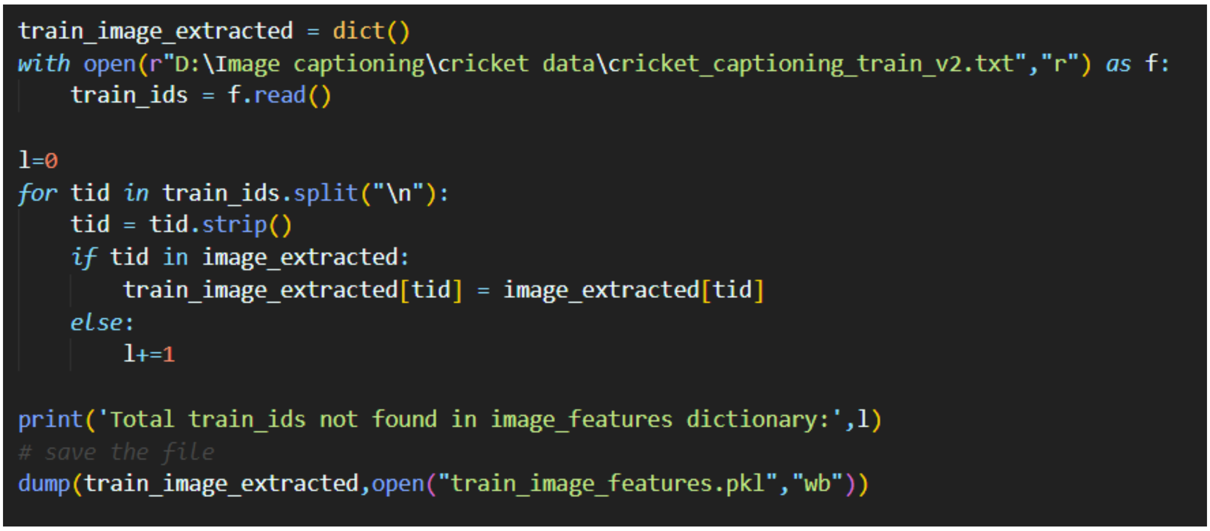
```python
import os
os.chdir(r"D:\Image captioning\coding\cricket")

from pickle import dump, load
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.layers import Dense,BatchNormalization,Dropout,Embedding,RepeatVector
from keras.preprocessing.image import load_img, img_to_array
import numpy as np
final_model =load(open('pretrained_incep_v3.pkl', 'rb'))

TARGET_SIZE = (299,299) # needed to convert the image as per pre-trained inceptionv3 requirements


image_extracted = dict()
# opening token file to get ids of all images"
with open(r"D:\Image captioning\cricket data\cricket_captioning_v2.txt") as f:
    data = f.read()
data1 = data.split("\n")

id_list = []
for el in data1:
    id1 = el.split("###")[0].strip()
    id_list+=[id1]
id_list1 = list(set(id_list))
```



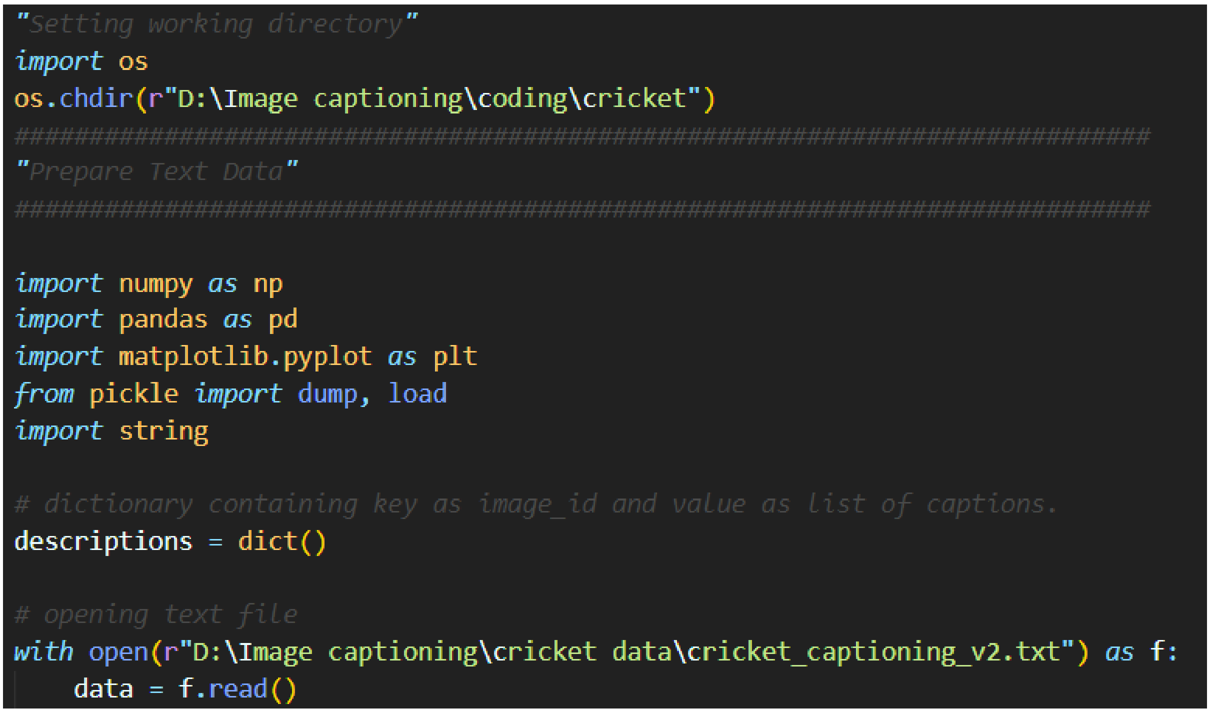
```python
k=0
l=0
for image_id in id_list1:
    if k%10==0:
        print(k)
    k+=1
    try:
        img = load_img(r"{}".format(image_id),target_size=TARGET_SIZE)

        # Converting image to array
        img_array = img_to_array(img)
        nimage = preprocess_input(img_array)
        # Adding one more dimesion
        nimage = np.expand_dims(nimage, axis=0)
        fea_vec = final_model.predict(nimage)
        image_extracted[image_id] = np.reshape(fea_vec, fea_vec.shape[1]) # reshape from (1, 2048) to (2048,

    except Exception as e:
        l+=1
        print("Total Exception got :- \n",l)
```



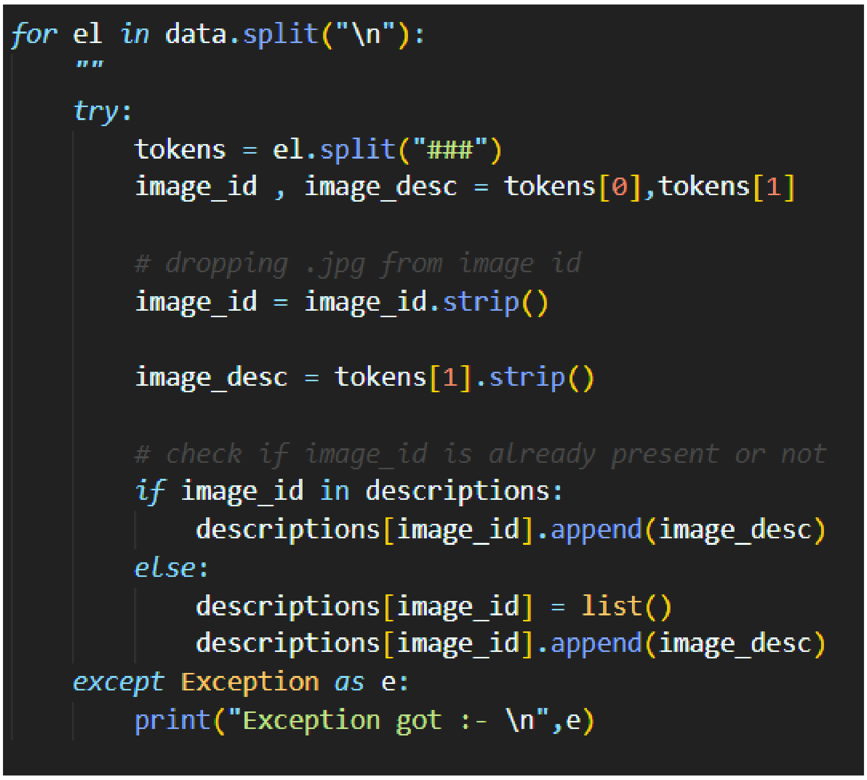
```python
dump(image_extracted,open("image_features.pkl","wb"))


train_image_extracted = dict()
with open(r"D:\Image captioning\cricket data\cricket_captioning_train_v2.txt","r") as f:
    train_ids = f.read()

l=0
for tid in train_ids.split("\n"):
    tid = tid.strip()
    if tid in image_extracted:
        train_image_extracted[tid] = image_extracted[tid]
    else:
        l+=1

print('Total train_ids not found in image_features dictionary:',l)
# save the file
dump(train_image_extracted,open("train_image_features.pkl","wb"))
```



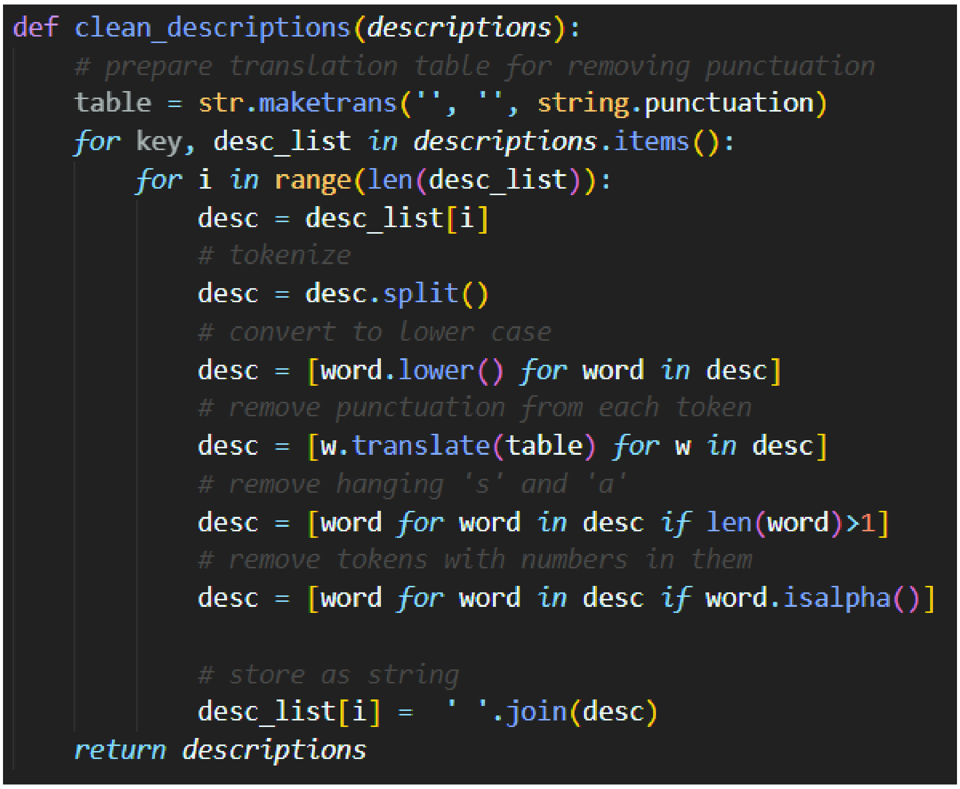
```python
train_image_extracted = dict()
with open(r"D:\Image captioning\cricket data\cricket_captioning_train_v2.txt","r") as f:
    train_ids = f.read()

l=0
for tid in train_ids.split("\n"):
    tid = tid.strip()
    if tid in image_extracted:
        train_image_extracted[tid] = image_extracted[tid]
    else:
        l+=1

print('Total train_ids not found in image_features dictionary:',l)
# save the file
dump(train_image_extracted,open("train_image_features.pkl","wb"))
```



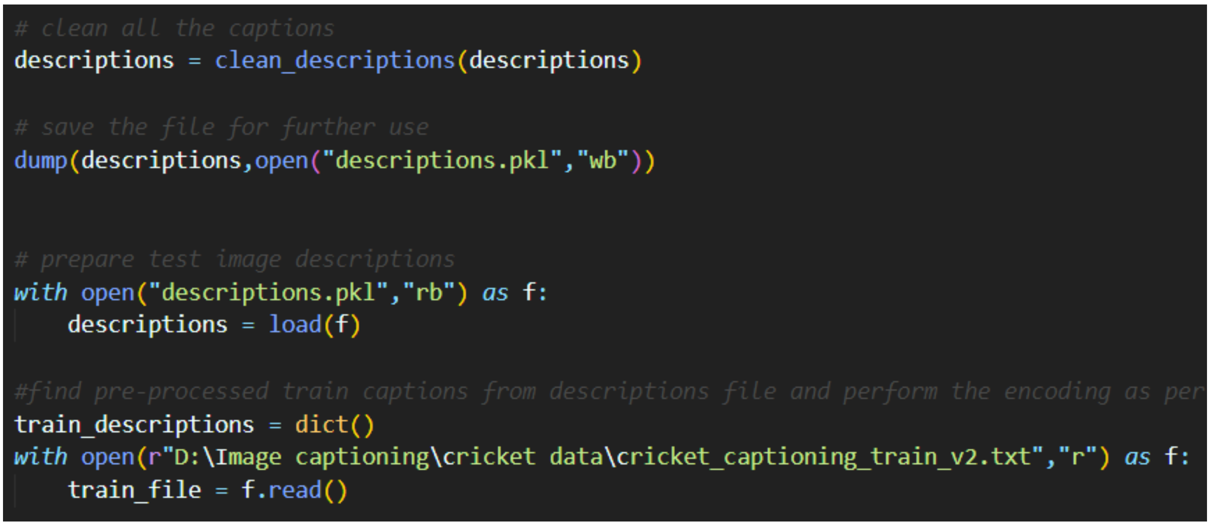
```python
"Setting working directory"
import os
os.chdir(r"D:\Image captioning\coding\cricket")
#############################################################################
"Prepare Text Data"
#############################################################################

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pickle import dump, load
import string

# dictionary containing key as image_id and value as list of captions.
descriptions = dict()

# opening text file
with open(r"D:\Image captioning\cricket data\cricket_captioning_v2.txt") as f:
    data = f.read()
```



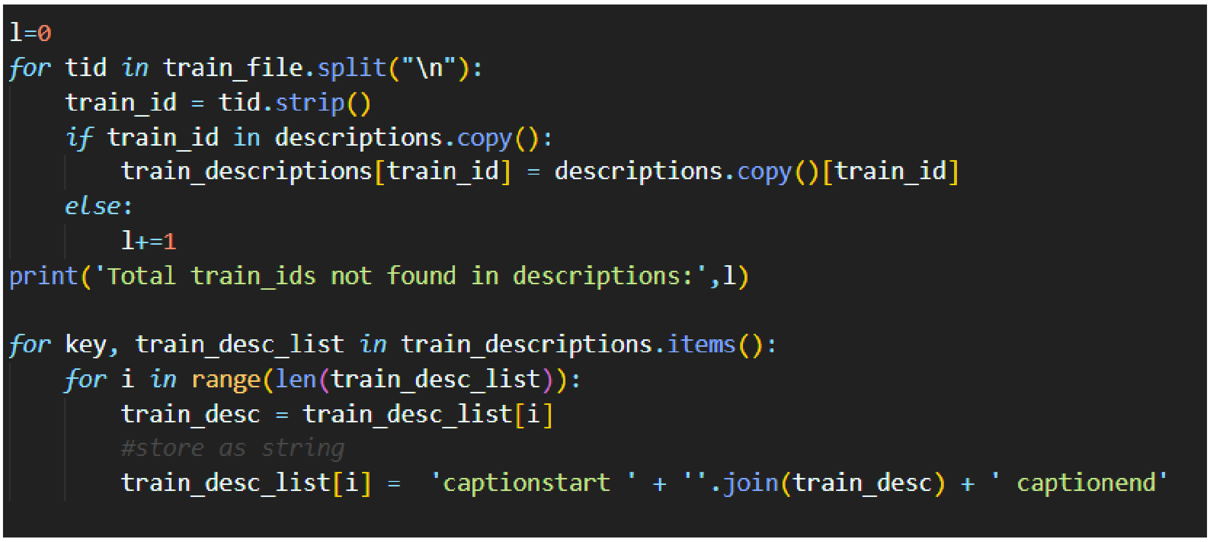
```python
for el in data.split("\n"):
    ""
    try:
        tokens = el.split("###")
        image_id , image_desc = tokens[0],tokens[1]

        # dropping .jpg from image id
        image_id = image_id.strip()

        image_desc = tokens[1].strip()

        # check if image_id is already present or not
        if image_id in descriptions:
            descriptions[image_id].append(image_desc)
        else:
            descriptions[image_id] = list()
            descriptions[image_id].append(image_desc)
    except Exception as e:
        print("Exception got :- \n",e)
```



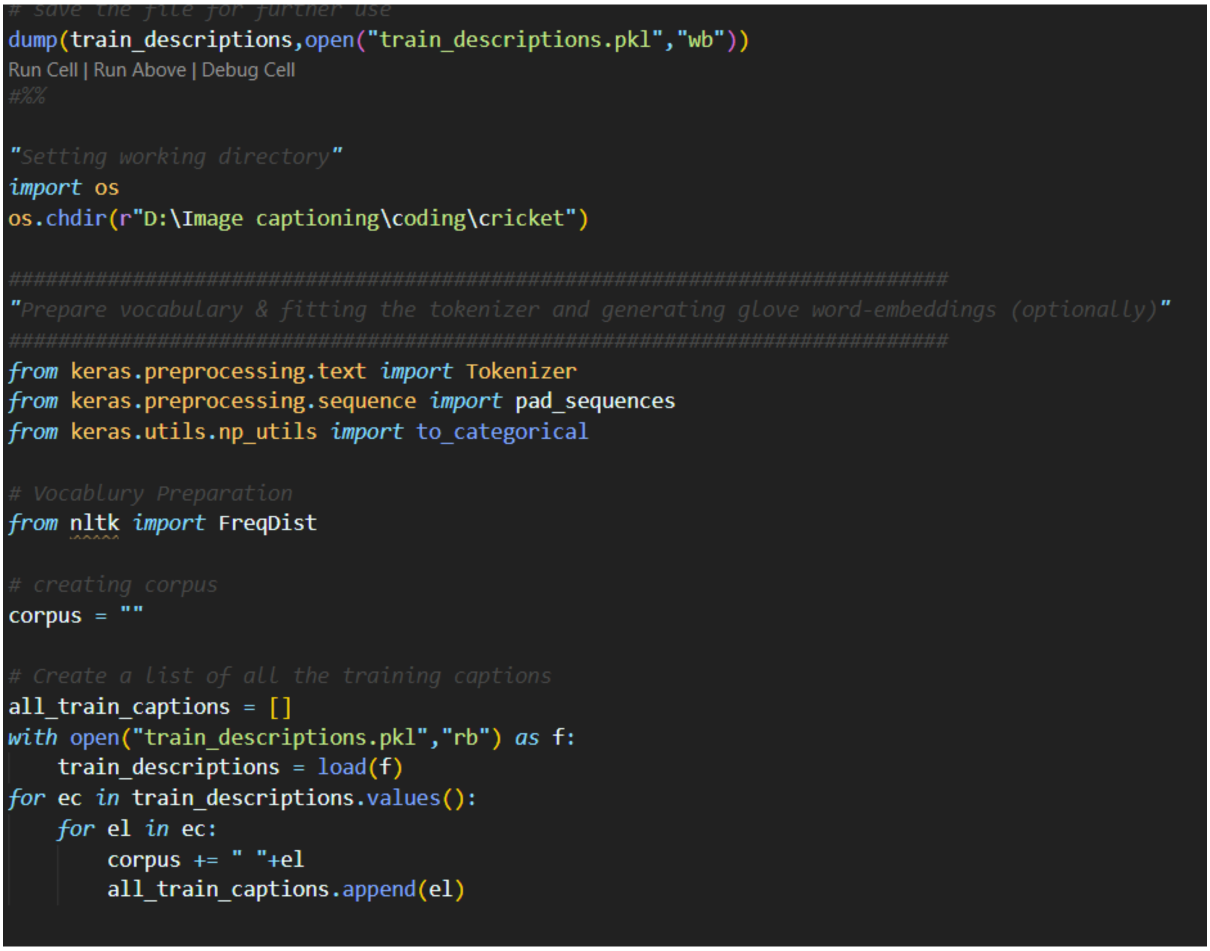
```python
def clean_descriptions(descriptions):
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            # tokenize
            desc = desc.split()
            # convert to lower case
            desc = [word.lower() for word in desc]
            # remove punctuation from each token
            desc = [w.translate(table) for w 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()]

            # store as string
            desc_list[i] =  ' '.join(desc)
    return descriptions
```



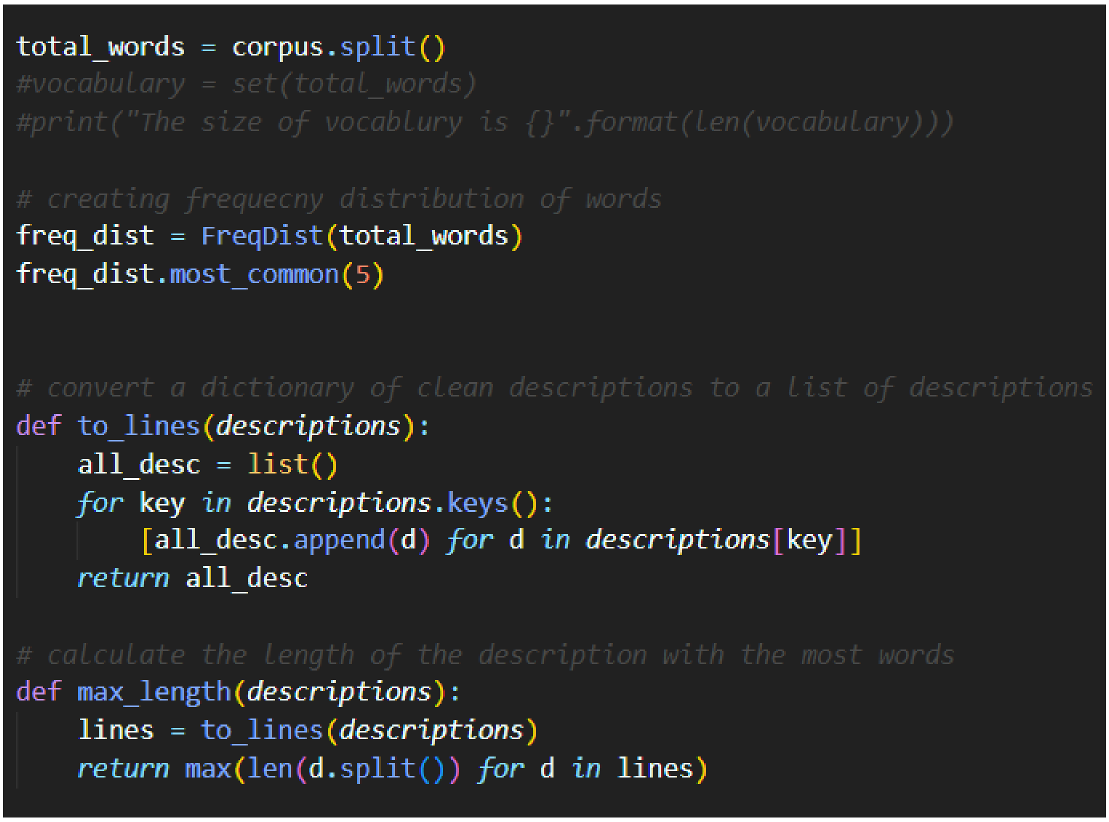
```python
# clean all the captions
descriptions = clean_descriptions(descriptions)

# save the file for further use
dump(descriptions,open("descriptions.pkl","wb"))


# prepare test image descriptions
with open("descriptions.pkl","rb") as f:
    descriptions = load(f)

#find pre-processed train captions from descriptions file and perform the encoding as per
train_descriptions = dict()
with open(r"D:\Image captioning\cricket data\cricket_captioning_train_v2.txt","r") as f:
    train_file = f.read()
```



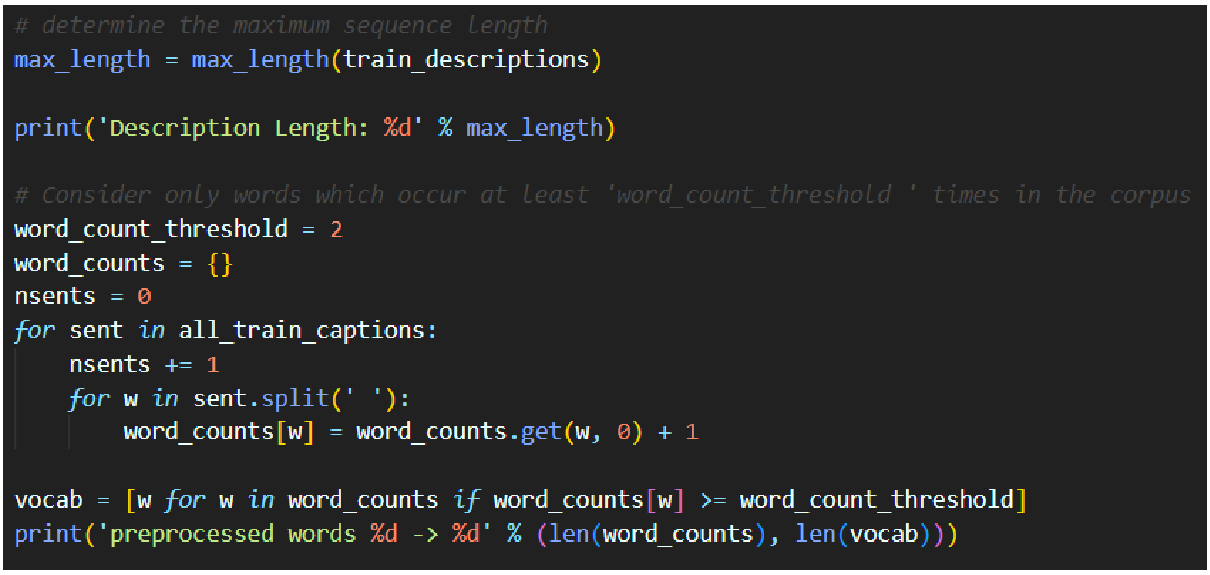
```
l=0
for tid in train_file.split("\n"):
    train_id = tid.strip()
    if train_id in descriptions.copy():
        train_descriptions[train_id] = descriptions.copy()[train_id]
    else:
        l+=1
print('Total train_ids not found in descriptions:',l)

for key, train_desc_list in train_descriptions.items():
    for i in range(len(train_desc_list)):
        train_desc = train_desc_list[i]
        #store as string
        train_desc_list[i] =  'captionstart ' + ''.join(train_desc) + ' captionend'
```



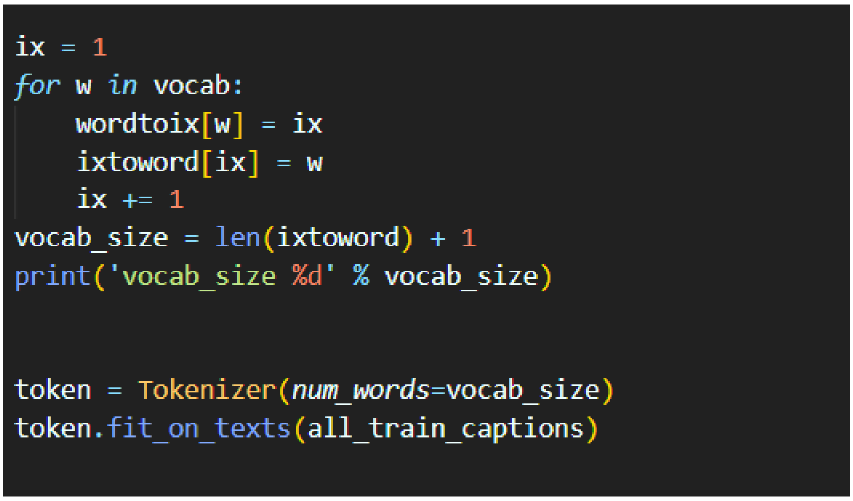
```
# save the file for further use
dump(train_descriptions,open("train_descriptions.pkl","wb"))
Run Cell | Run Above | Debug Cell
#%%

"Setting working directory"
import os
os.chdir(r"D:\Image captioning\coding\cricket")

##############################################################################
"Prepare vocabulary & fitting the tokenizer and generating glove word-embeddings (optionally)"
##############################################################################
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical

# Vocablury Preparation
from nltk import FreqDist

# creating corpus
corpus = ""

# Create a list of all the training captions
all_train_captions = []
with open("train_descriptions.pkl","rb") as f:
    train_descriptions = load(f)
for ec in train_descriptions.values():
    for el in ec:
        corpus += " "+el
        all_train_captions.append(el)
```



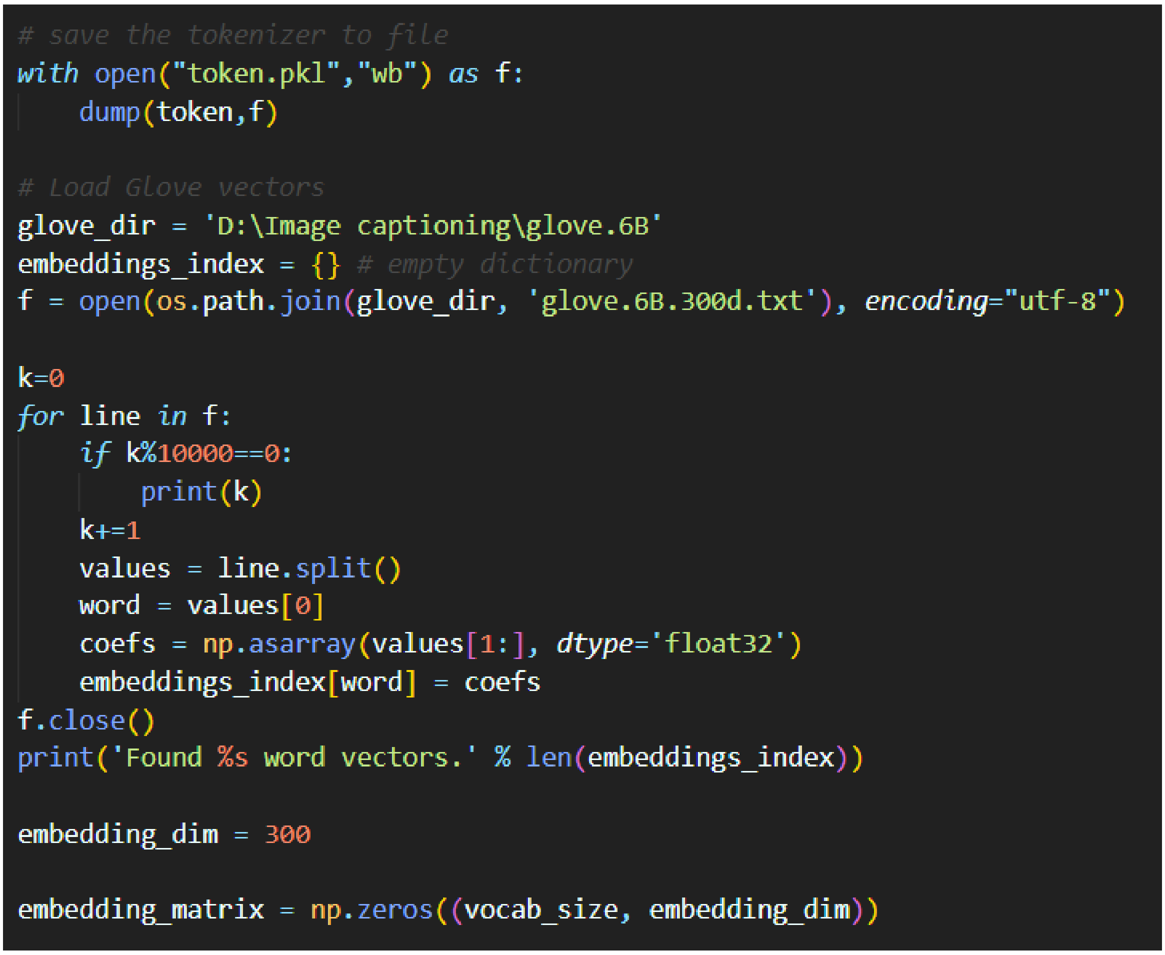
```python
total_words = corpus.split()
#vocabulary = set(total_words)
#print("The size of vocablury is {}".format(len(vocabulary)))

# creating frequecny distribution of words
freq_dist = FreqDist(total_words)
freq_dist.most_common(5)


# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)
```



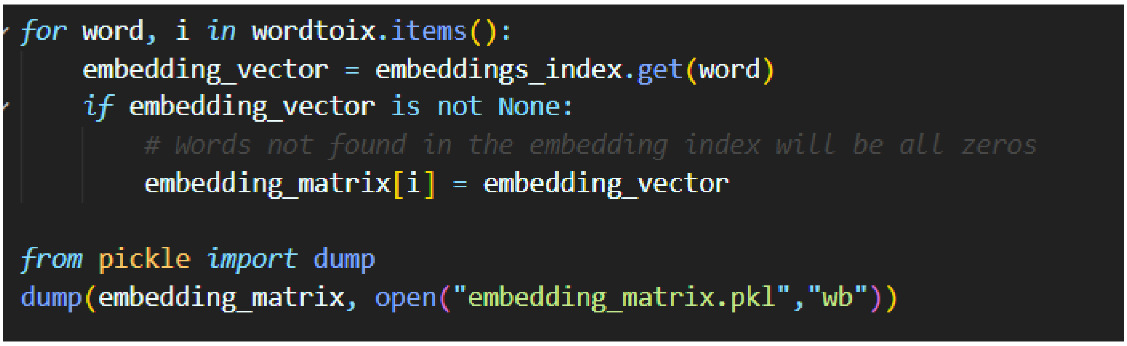
```python
# determine the maximum sequence length
max_length = max_length(train_descriptions)

print('Description Length: %d' % max_length)

# Consider only words which occur at least 'word_count_threshold ' times in the corpus
word_count_threshold = 2
word_counts = {}
nsents = 0
for sent in all_train_captions:
    nsents += 1
    for w in sent.split(' '):
        word_counts[w] = word_counts.get(w, 0) + 1

vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
print('preprocessed words %d -> %d' % (len(word_counts), len(vocab)))
```



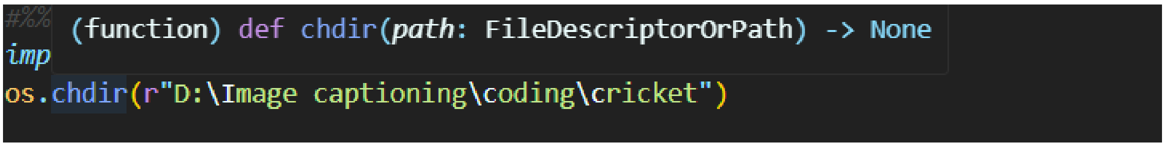
```
ix = 1
for w in vocab:
    wordtoix[w] = ix
    ixtoword[ix] = w
    ix += 1
vocab_size = len(ixtoword) + 1
print('vocab_size %d' % vocab_size)


token = Tokenizer(num_words=vocab_size)
token.fit_on_texts(all_train_captions)
```



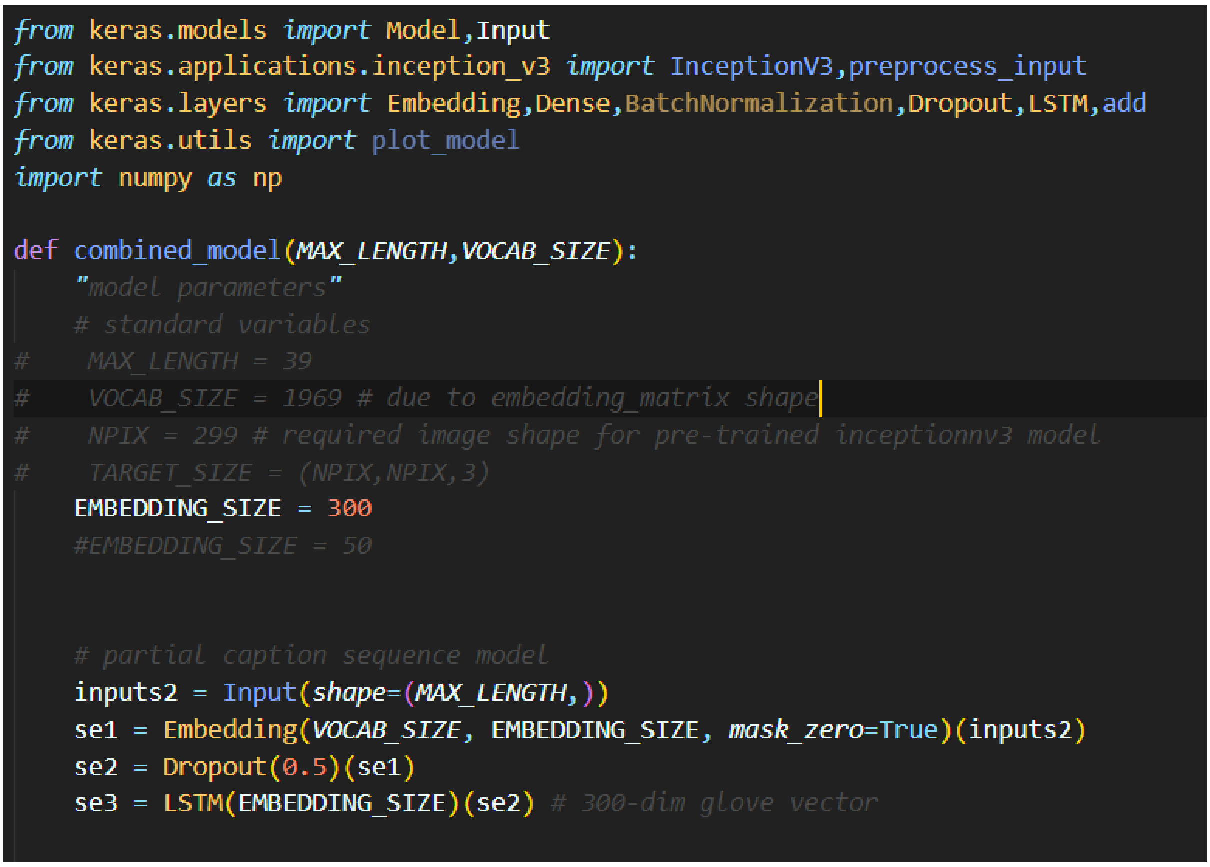
```
# save the tokenizer to file
with open("token.pkl","wb") as f:
    dump(token,f)

# Load Glove vectors
glove_dir = 'D:\Image captioning\glove.6B'
embeddings_index = {} # empty dictionary
f = open(os.path.join(glove_dir, 'glove.6B.300d.txt'), encoding="utf-8")

k=0
for line in f:
    if k%10000==0:
        print(k)
    k+=1
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Found %s word vectors.' % len(embeddings_index))

embedding_dim = 300

embedding_matrix = np.zeros((vocab_size, embedding_dim))
```



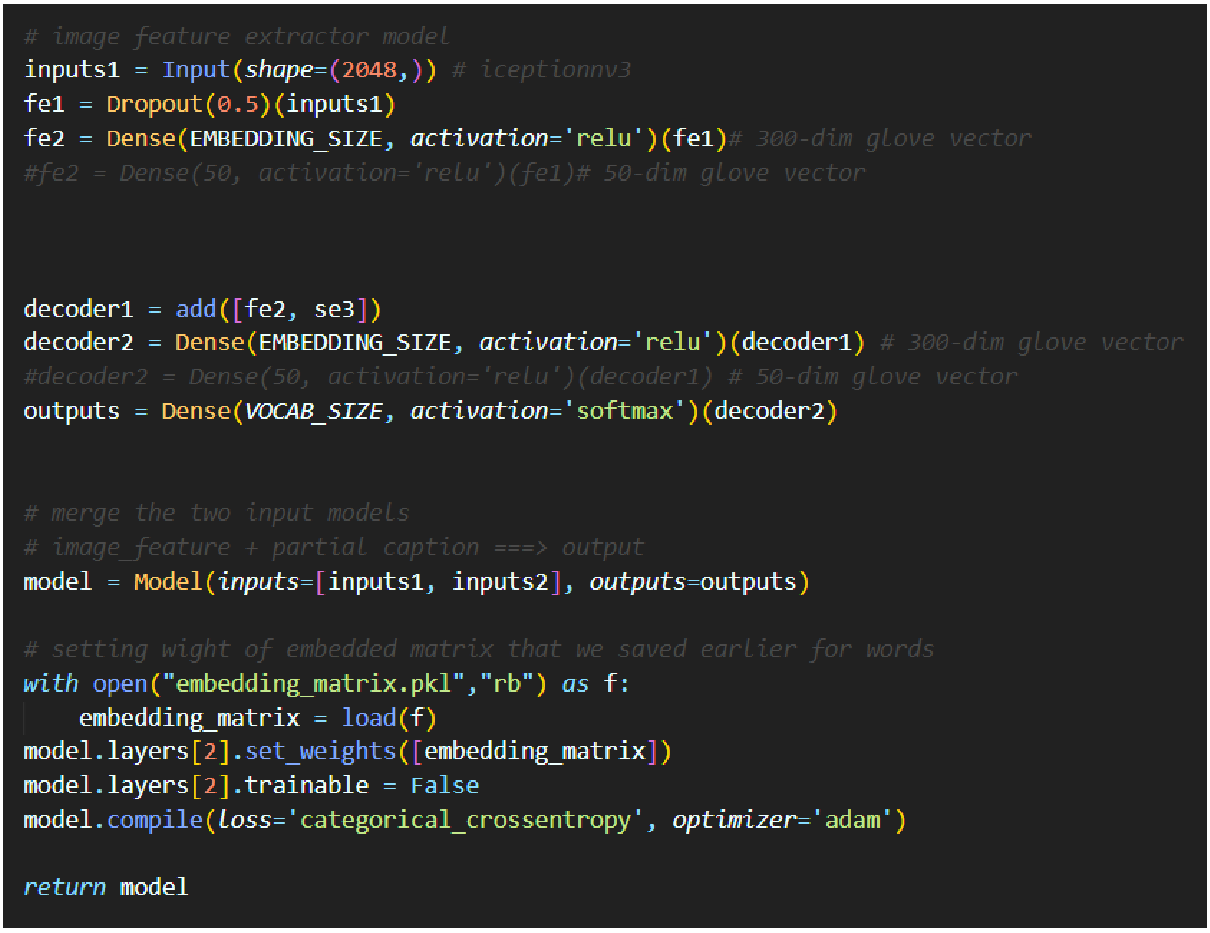
```
for word, i in wordtoix.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector

from pickle import dump
dump(embedding_matrix, open("embedding_matrix.pkl","wb"))
```



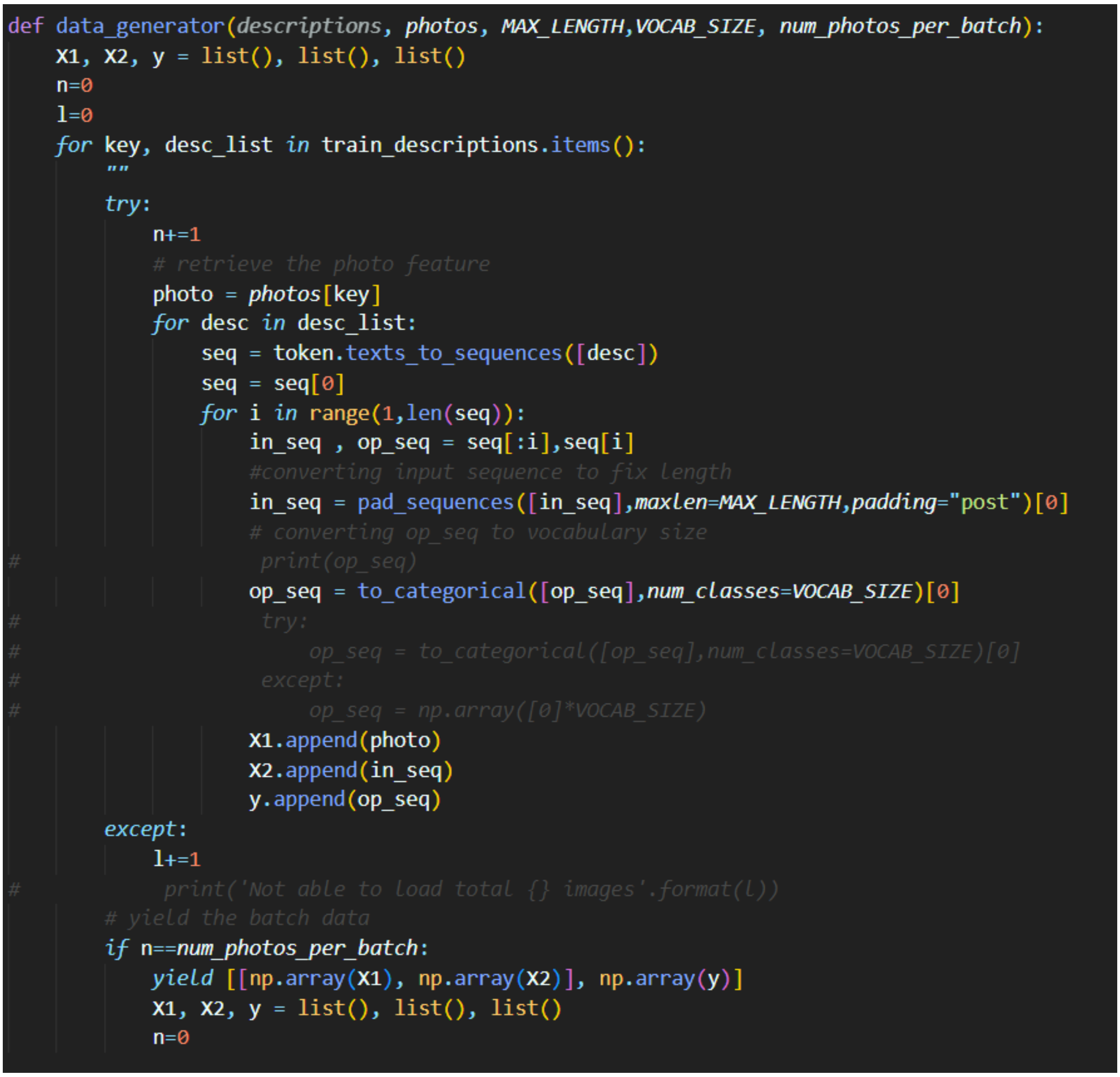
```
#%%  (function) def chdir(path: FileDescriptorOrPath) -> None
imp
os.chdir(r"D:\Image captioning\coding\cricket")
```



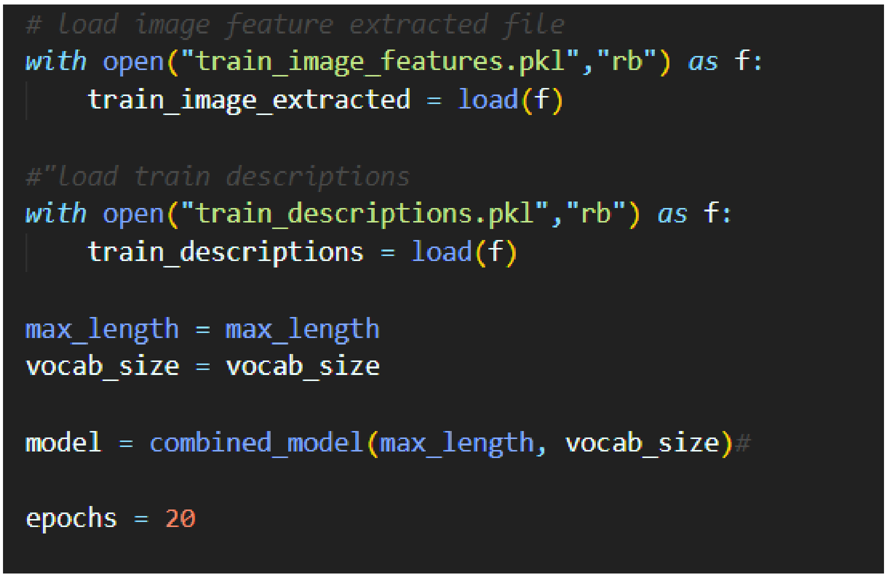
```
from keras.models import Model,Input
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.layers import Embedding,Dense,BatchNormalization,Dropout,LSTM,add
from keras.utils import plot_model
import numpy as np

def combined_model(MAX_LENGTH,VOCAB_SIZE):
    "model parameters"
    # standard variables
#    MAX_LENGTH = 39
#    VOCAB_SIZE = 1969 # due to embedding_matrix shape
#    NPIX = 299 # required image shape for pre-trained inceptionnv3 model
#    TARGET_SIZE = (NPIX,NPIX,3)
    EMBEDDING_SIZE = 300
    #EMBEDDING_SIZE = 50


    # partial caption sequence model
    inputs2 = Input(shape=(MAX_LENGTH,))
    se1 = Embedding(VOCAB_SIZE, EMBEDDING_SIZE, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(EMBEDDING_SIZE)(se2) # 300-dim glove vector
```



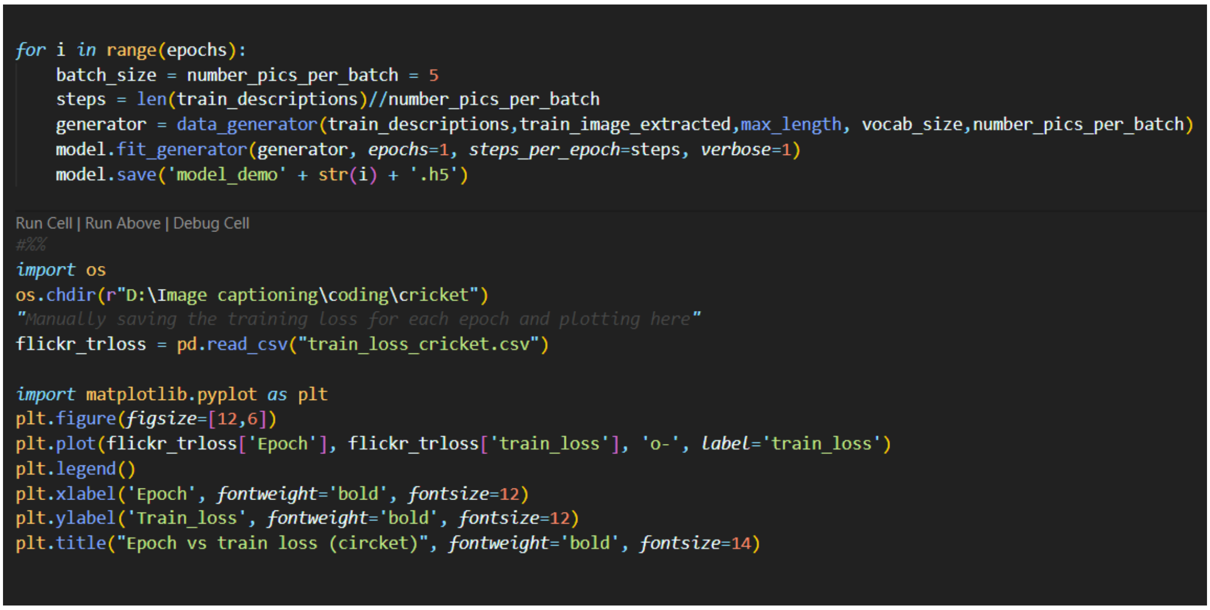
```
# image feature extractor model
inputs1 = Input(shape=(2048,)) # iceptionnv3
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(EMBEDDING_SIZE, activation='relu')(fe1)# 300-dim glove vector
#fe2 = Dense(50, activation='relu')(fe1)# 50-dim glove vector



decoder1 = add([fe2, se3])
decoder2 = Dense(EMBEDDING_SIZE, activation='relu')(decoder1) # 300-dim glove vector
#decoder2 = Dense(50, activation='relu')(decoder1) # 50-dim glove vector
outputs = Dense(VOCAB_SIZE, activation='softmax')(decoder2)


# merge the two input models
# image_feature + partial caption ===> output
model = Model(inputs=[inputs1, inputs2], outputs=outputs)

# setting wight of embedded matrix that we saved earlier for words
with open("embedding_matrix.pkl","rb") as f:
    embedding_matrix = load(f)
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False
model.compile(loss='categorical_crossentropy', optimizer='adam')

return model
```



```
def data_generator(descriptions, photos, MAX_LENGTH,VOCAB_SIZE, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    l=0
    for key, desc_list in train_descriptions.items():
        ""
        try:
            n+=1
            # retrieve the photo feature
            photo = photos[key]
            for desc in desc_list:
                seq = token.texts_to_sequences([desc])
                seq = seq[0]
                for i in range(1,len(seq)):
                    in_seq , op_seq = seq[:i],seq[i]
                    #converting input sequence to fix length
                    in_seq = pad_sequences([in_seq],maxlen=MAX_LENGTH,padding="post")[0]
                    # converting op_seq to vocabulary size
#                    print(op_seq)
                    op_seq = to_categorical([op_seq],num_classes=VOCAB_SIZE)[0]
#                    try:
#                        op_seq = to_categorical([op_seq],num_classes=VOCAB_SIZE)[0]
#                    except:
#                        op_seq = np.array([0]*VOCAB_SIZE)
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(op_seq)
        except:
            l+=1
#            print('Not able to load total {} images'.format(l))
        # yield the batch data
        if n==num_photos_per_batch:
            yield [[np.array(X1), np.array(X2)], np.array(y)]
            X1, X2, y = list(), list(), list()
            n=0
```



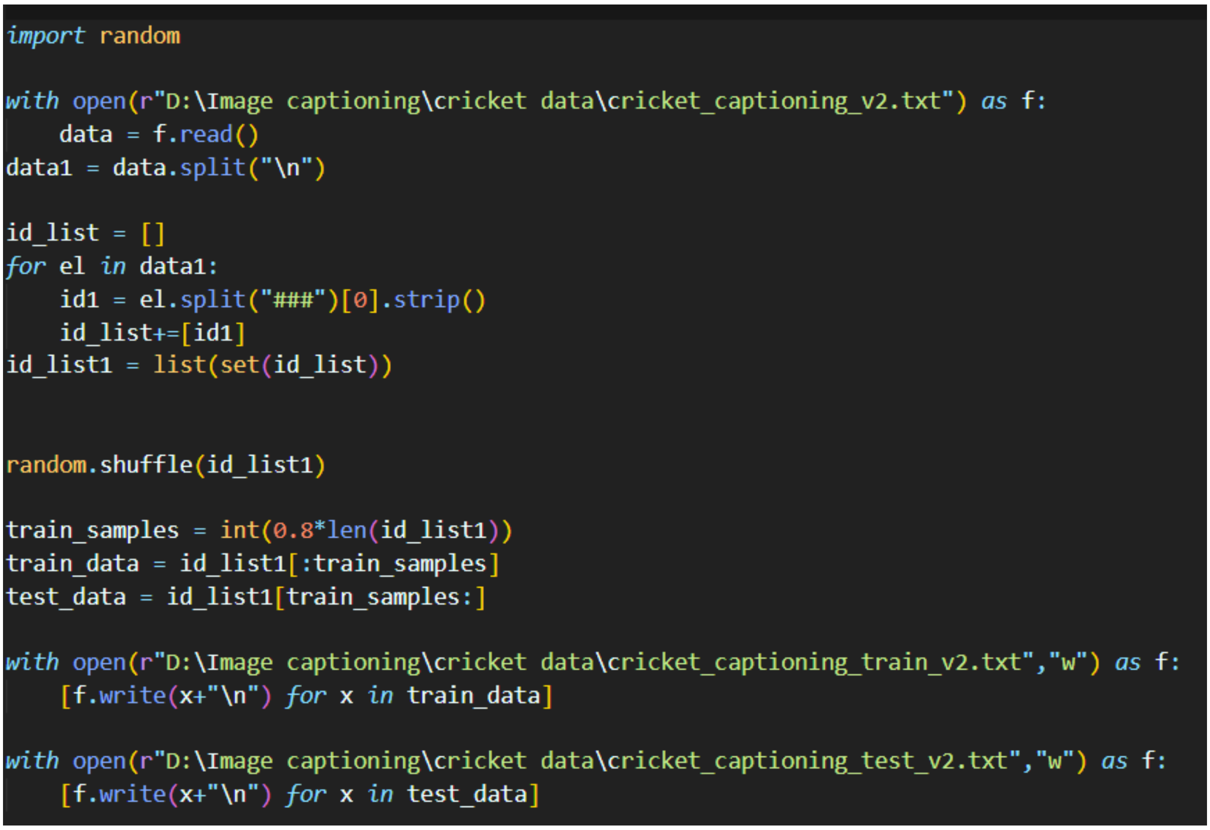
```
# load image feature extracted file
with open("train_image_features.pkl","rb") as f:
    train_image_extracted = load(f)

#"load train descriptions
with open("train_descriptions.pkl","rb") as f:
    train_descriptions = load(f)

max_length = max_length
vocab_size = vocab_size

model = combined_model(max_length, vocab_size)#

epochs = 20
```



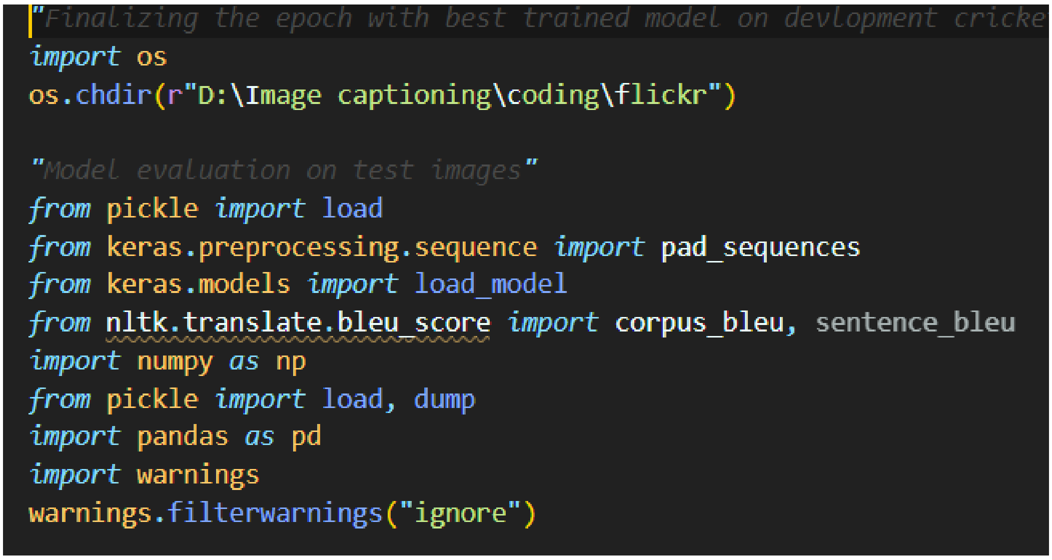
```
for i in range(epochs):
    batch_size = number_pics_per_batch = 5
    steps = len(train_descriptions)//number_pics_per_batch
    generator = data_generator(train_descriptions,train_image_extracted,max_length, vocab_size,number_pics_per_batch)
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    model.save('model_demo' + str(i) + '.h5')
```

Run Cell | Run Above | Debug Cell

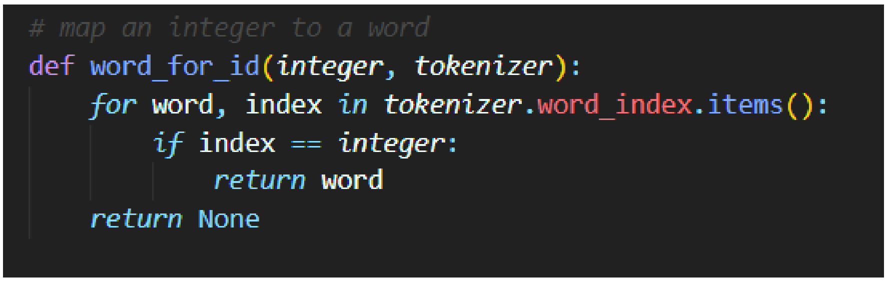
```
#%%
import os
os.chdir(r"D:\Image captioning\coding\cricket")
"Manually saving the training loss for each epoch and plotting here"
flickr_trloss = pd.read_csv("train_loss_cricket.csv")

import matplotlib.pyplot as plt
plt.figure(figsize=[12,6])
plt.plot(flickr_trloss['Epoch'], flickr_trloss['train_loss'], 'o-', label='train_loss')
plt.legend()
plt.xlabel('Epoch', fontweight='bold', fontsize=12)
plt.ylabel('Train_loss', fontweight='bold', fontsize=12)
plt.title("Epoch vs train loss (circket)", fontweight='bold', fontsize=14)
```



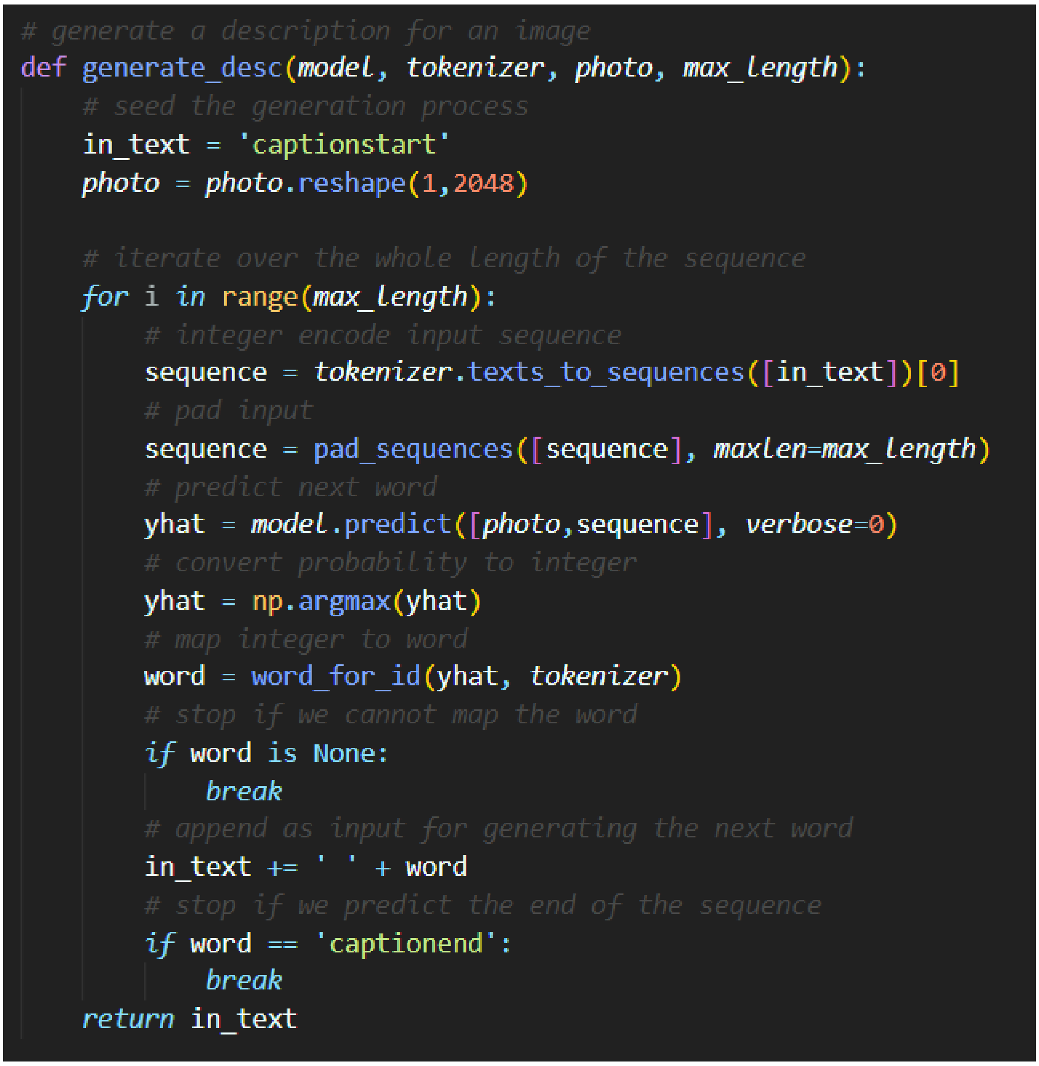
```
import random

with open(r"D:\Image captioning\cricket data\cricket_captioning_v2.txt") as f:
    data = f.read()
data1 = data.split("\n")

id_list = []
for el in data1:
    id1 = el.split("###")[0].strip()
    id_list+=[id1]
id_list1 = list(set(id_list))


random.shuffle(id_list1)

train_samples = int(0.8*len(id_list1))
train_data = id_list1[:train_samples]
test_data = id_list1[train_samples:]

with open(r"D:\Image captioning\cricket data\cricket_captioning_train_v2.txt","w") as f:
    [f.write(x+"\n") for x in train_data]

with open(r"D:\Image captioning\cricket data\cricket_captioning_test_v2.txt","w") as f:
    [f.write(x+"\n") for x in test_data]
```



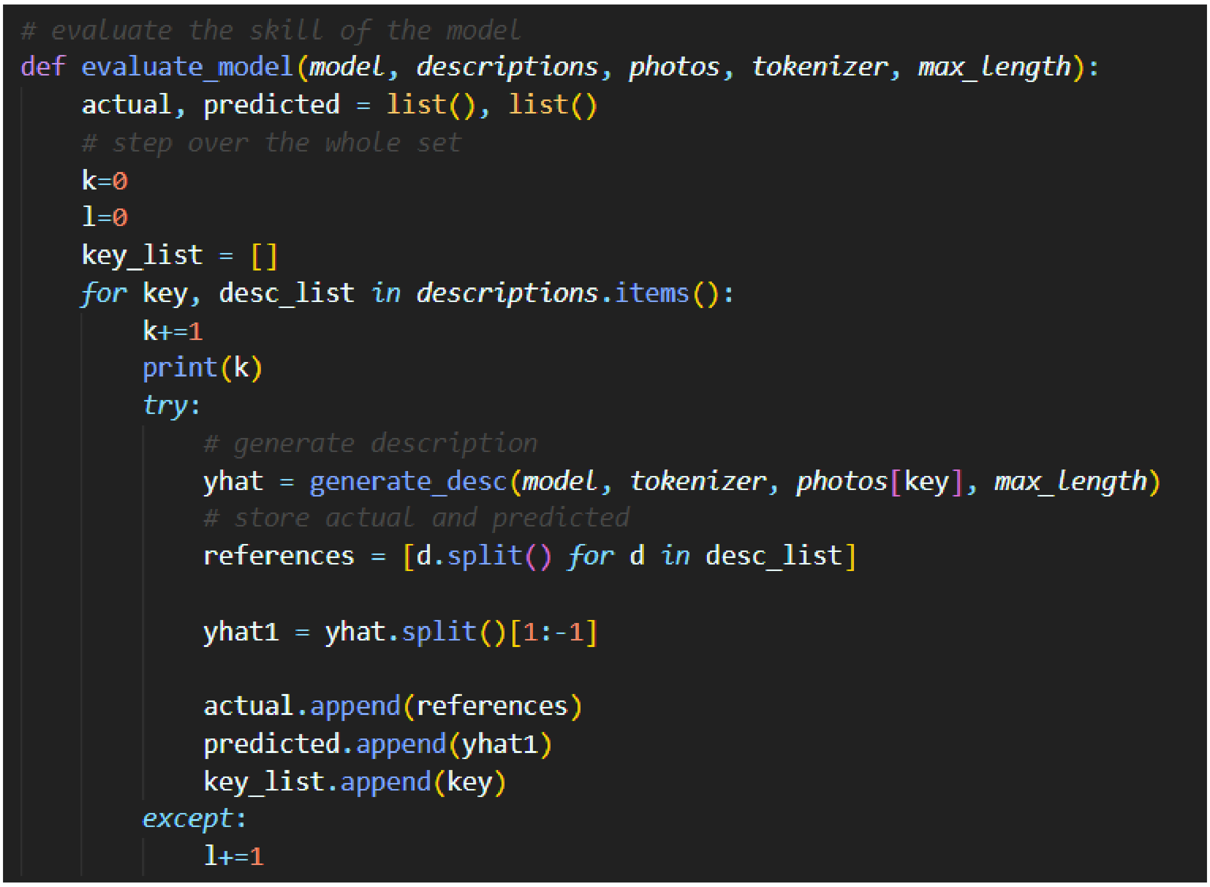
```
"Finalizing the epoch with best trained model on devlopment cricke
import os
os.chdir(r"D:\Image captioning\coding\flickr")

"Model evaluation on test images"
from pickle import load
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu, sentence_bleu
import numpy as np
from pickle import load, dump
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```



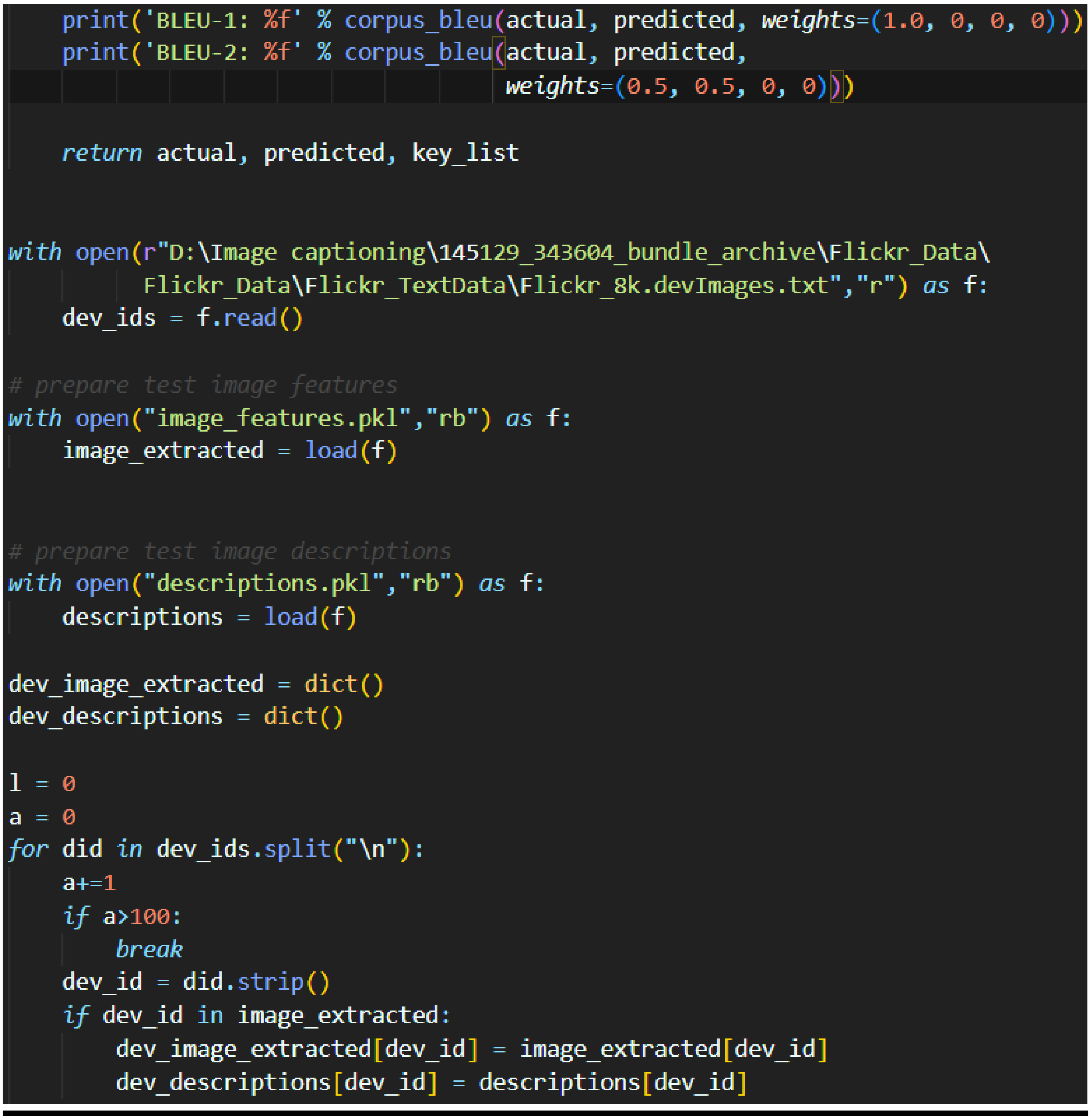
```
# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```



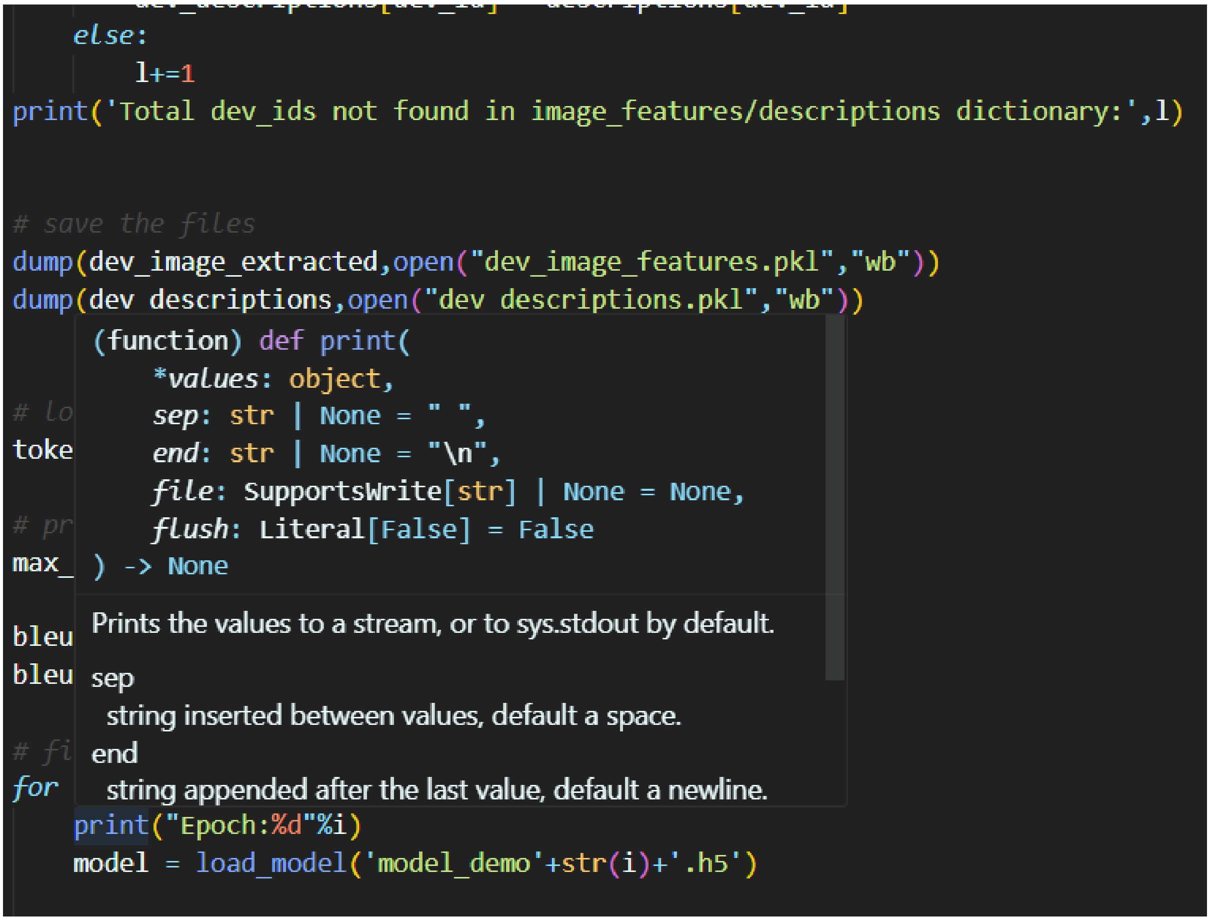
```python
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'captionstart'
    photo = photo.reshape(1,2048)

    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
        # predict next word
        yhat = model.predict([photo,sequence], verbose=0)
        # convert probability to integer
        yhat = np.argmax(yhat)
        # map integer to word
        word = word_for_id(yhat, tokenizer)
        # stop if we cannot map the word
        if word is None:
            break
        # append as input for generating the next word
        in_text += ' ' + word
        # stop if we predict the end of the sequence
        if word == 'captionend':
            break
    return in_text
```



```python
# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
    actual, predicted = list(), list()
    # step over the whole set
    k=0
    l=0
    key_list = []
    for key, desc_list in descriptions.items():
        k+=1
        print(k)
        try:
            # generate description
            yhat = generate_desc(model, tokenizer, photos[key], max_length)
            # store actual and predicted
            references = [d.split() for d in desc_list]

            yhat1 = yhat.split()[1:-1]

            actual.append(references)
            predicted.append(yhat1)
            key_list.append(key)
        except:
            l+=1
```



```
    print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu(actual, predicted,
                                     weights=(0.5, 0.5, 0, 0)))

    return actual, predicted, key_list


with open(r"D:\Image captioning\145129_343604_bundle_archive\Flickr_Data\
          Flickr_Data\Flickr_TextData\Flickr_8k.devImages.txt","r") as f:
    dev_ids = f.read()

# prepare test image features
with open("image_features.pkl","rb") as f:
    image_extracted = load(f)


# prepare test image descriptions
with open("descriptions.pkl","rb") as f:
    descriptions = load(f)

dev_image_extracted = dict()
dev_descriptions = dict()

l = 0
a = 0
for did in dev_ids.split("\n"):
    a+=1
    if a>100:
        break
    dev_id = did.strip()
    if dev_id in image_extracted:
        dev_image_extracted[dev_id] = image_extracted[dev_id]
        dev_descriptions[dev_id] = descriptions[dev_id]
```



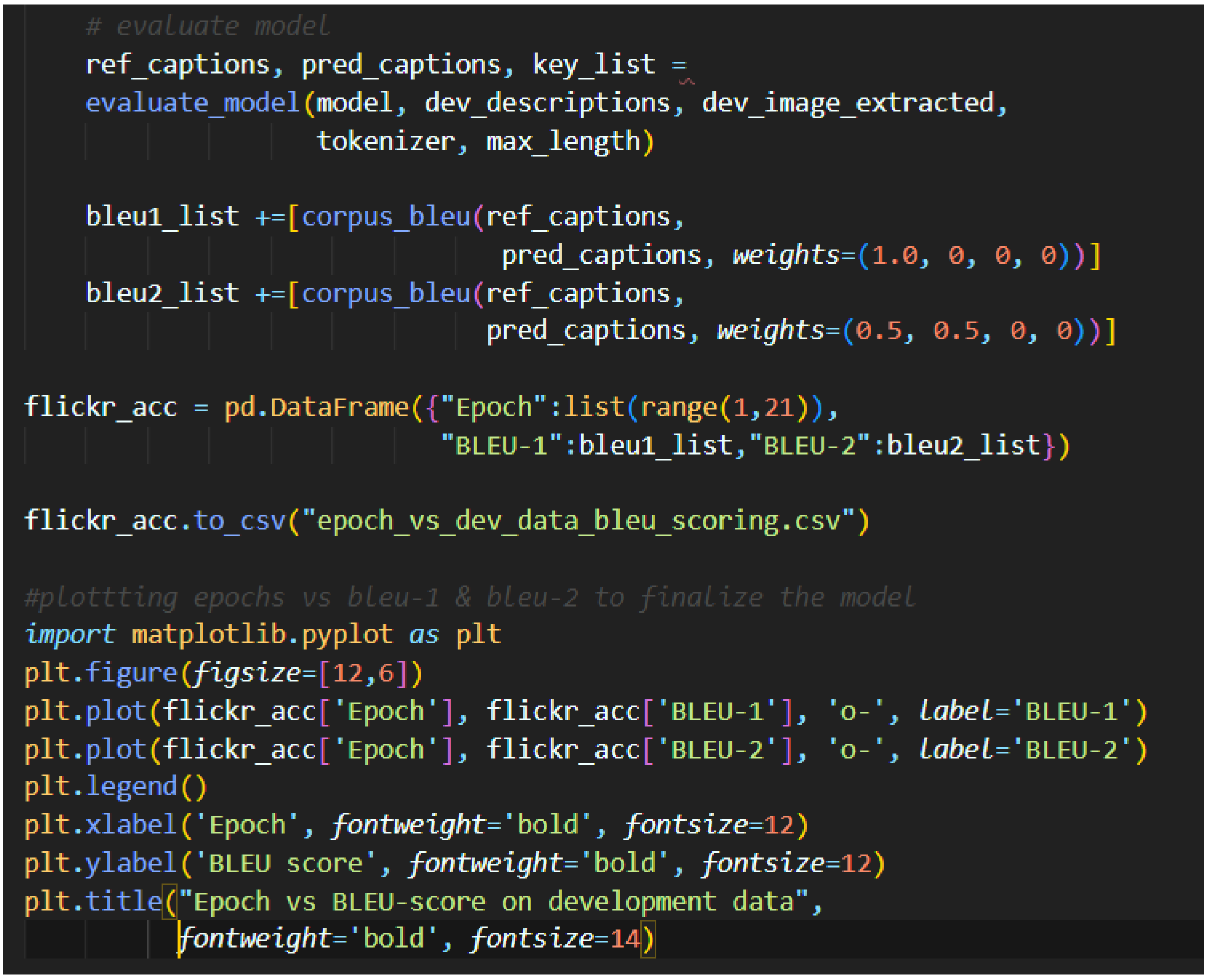
```
        dev_descriptions[dev_id] = descriptions[dev_id]
    else:
        l+=1
print('Total dev_ids not found in image_features/descriptions dictionary:',l)


# save the files
dump(dev_image_extracted,open("dev_image_features.pkl","wb"))
dump(dev descriptions,open("dev descriptions.pkl","wb"))
     (function) def print(
         *values: object,
# lo     sep: str | None = " ",
toke     end: str | None = "\n",
         file: SupportsWrite[str] | None = None,
# pr     flush: Literal[False] = False
max_ ) -> None
```

Prints the values to a stream, or to sys.stdout by default.

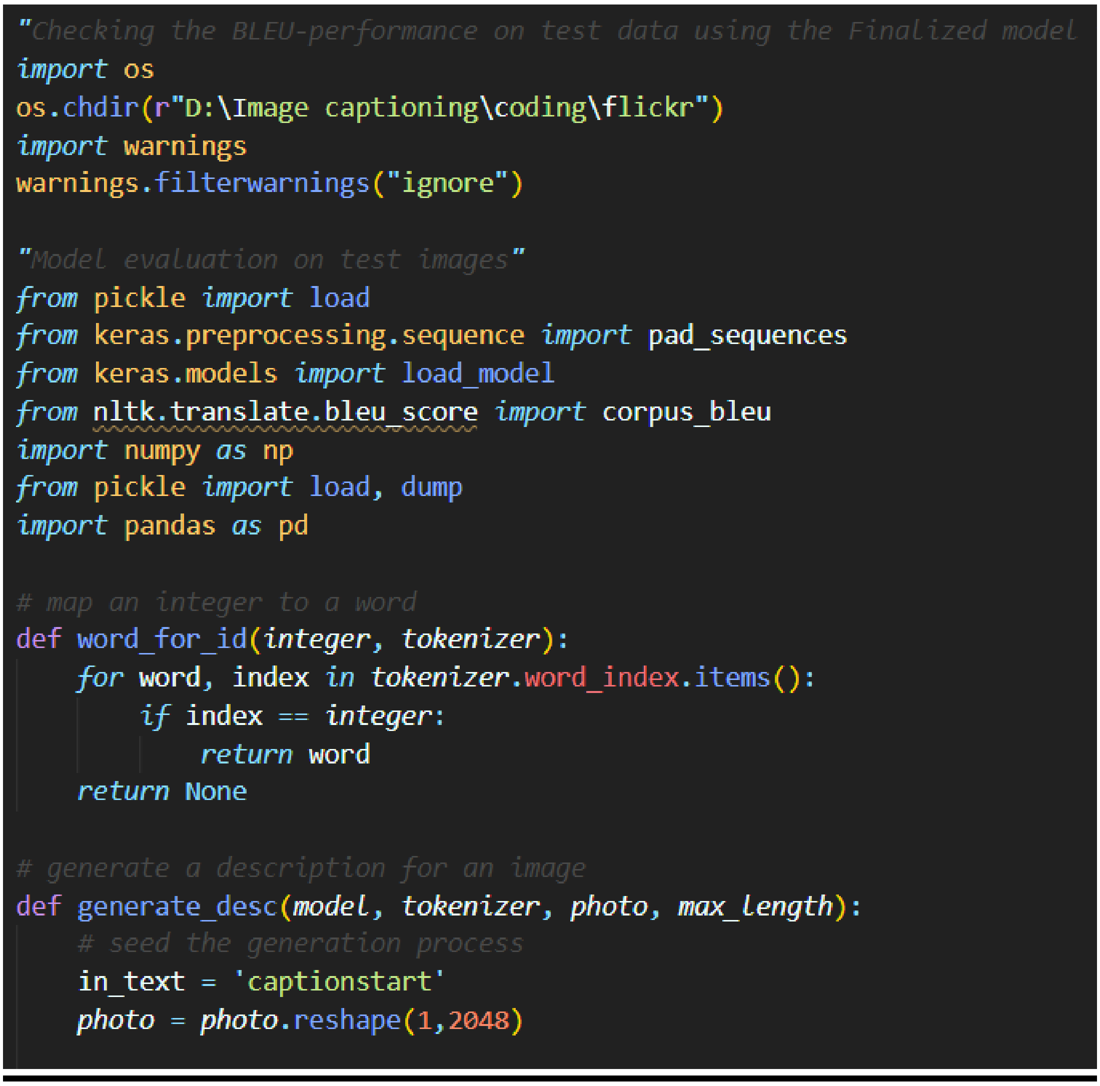
sep
string inserted between values, default a space.

end
string appended after the last value, default a newline.

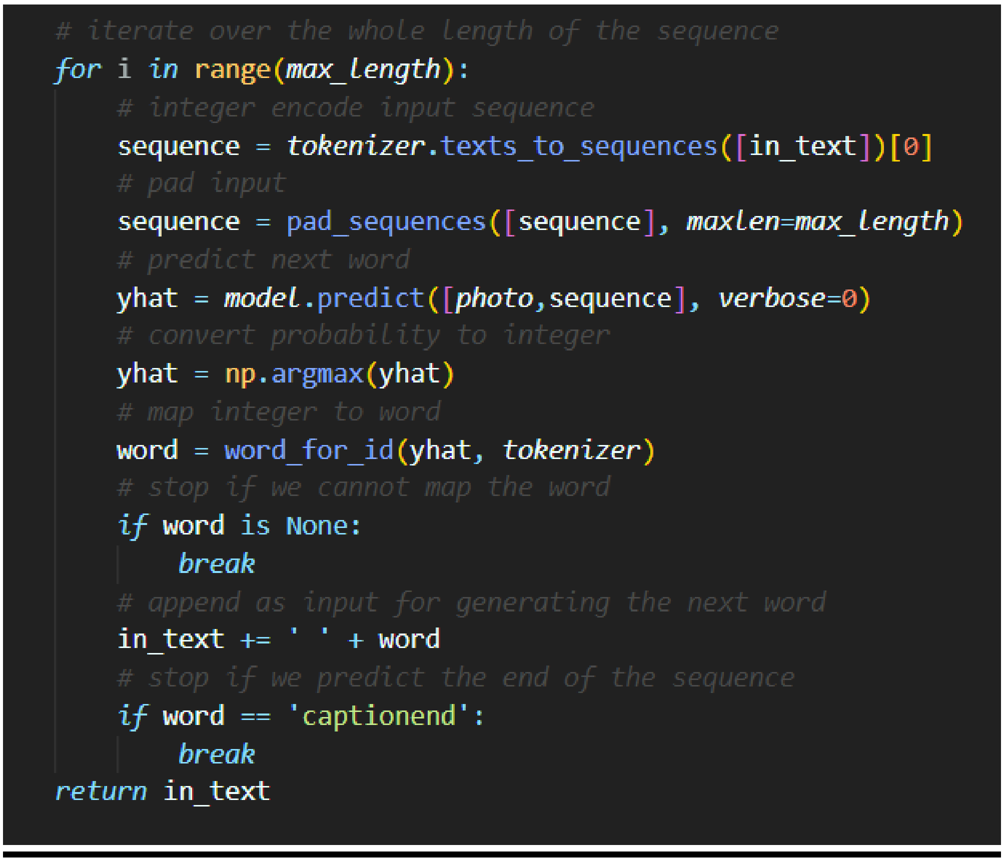
```
bleu
bleu

# fi
for
    print("Epoch:%d"%i)
    model = load_model('model_demo'+str(i)+'.h5')
```



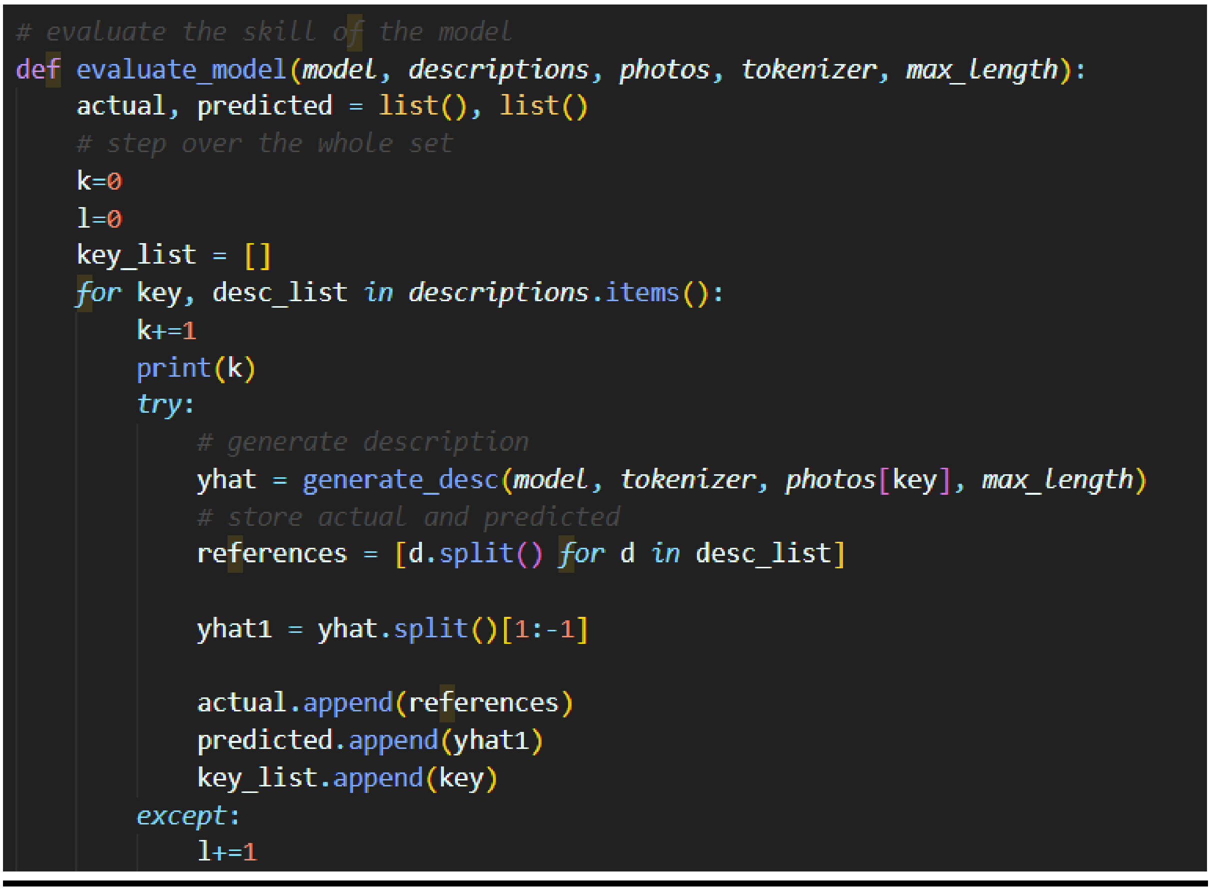
```
    # evaluate model
    ref_captions, pred_captions, key_list =
    evaluate_model(model, dev_descriptions, dev_image_extracted,
                   tokenizer, max_length)

    bleu1_list +=[corpus_bleu(ref_captions,
                               pred_captions, weights=(1.0, 0, 0, 0))]
    bleu2_list +=[corpus_bleu(ref_captions,
                              pred_captions, weights=(0.5, 0.5, 0, 0))]

flickr_acc = pd.DataFrame({"Epoch":list(range(1,21)),
                           "BLEU-1":bleu1_list,"BLEU-2":bleu2_list})

flickr_acc.to_csv("epoch_vs_dev_data_bleu_scoring.csv")

#plotttting epochs vs bleu-1 & bleu-2 to finalize the model
import matplotlib.pyplot as plt
plt.figure(figsize=[12,6])
plt.plot(flickr_acc['Epoch'], flickr_acc['BLEU-1'], 'o-', label='BLEU-1')
plt.plot(flickr_acc['Epoch'], flickr_acc['BLEU-2'], 'o-', label='BLEU-2')
plt.legend()
plt.xlabel('Epoch', fontweight='bold', fontsize=12)
plt.ylabel('BLEU score', fontweight='bold', fontsize=12)
plt.title("Epoch vs BLEU-score on development data",
          fontweight='bold', fontsize=14)
```



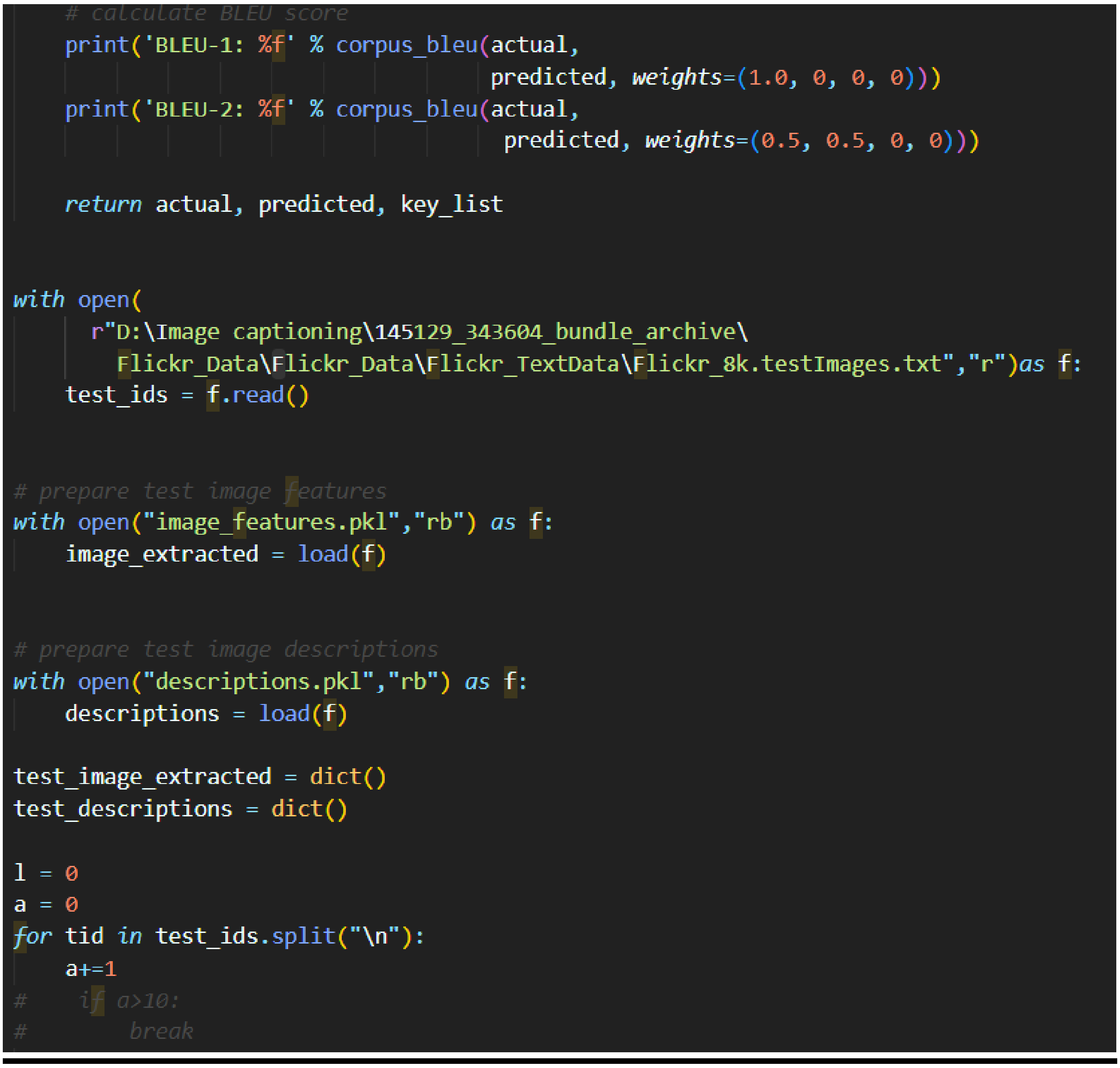
```
"Checking the BLEU-performance on test data using the Finalized model
import os
os.chdir(r"D:\Image captioning\coding\flickr")
import warnings
warnings.filterwarnings("ignore")

"Model evaluation on test images"
from pickle import load
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu
import numpy as np
from pickle import load, dump
import pandas as pd

# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'captionstart'
    photo = photo.reshape(1,2048)
```



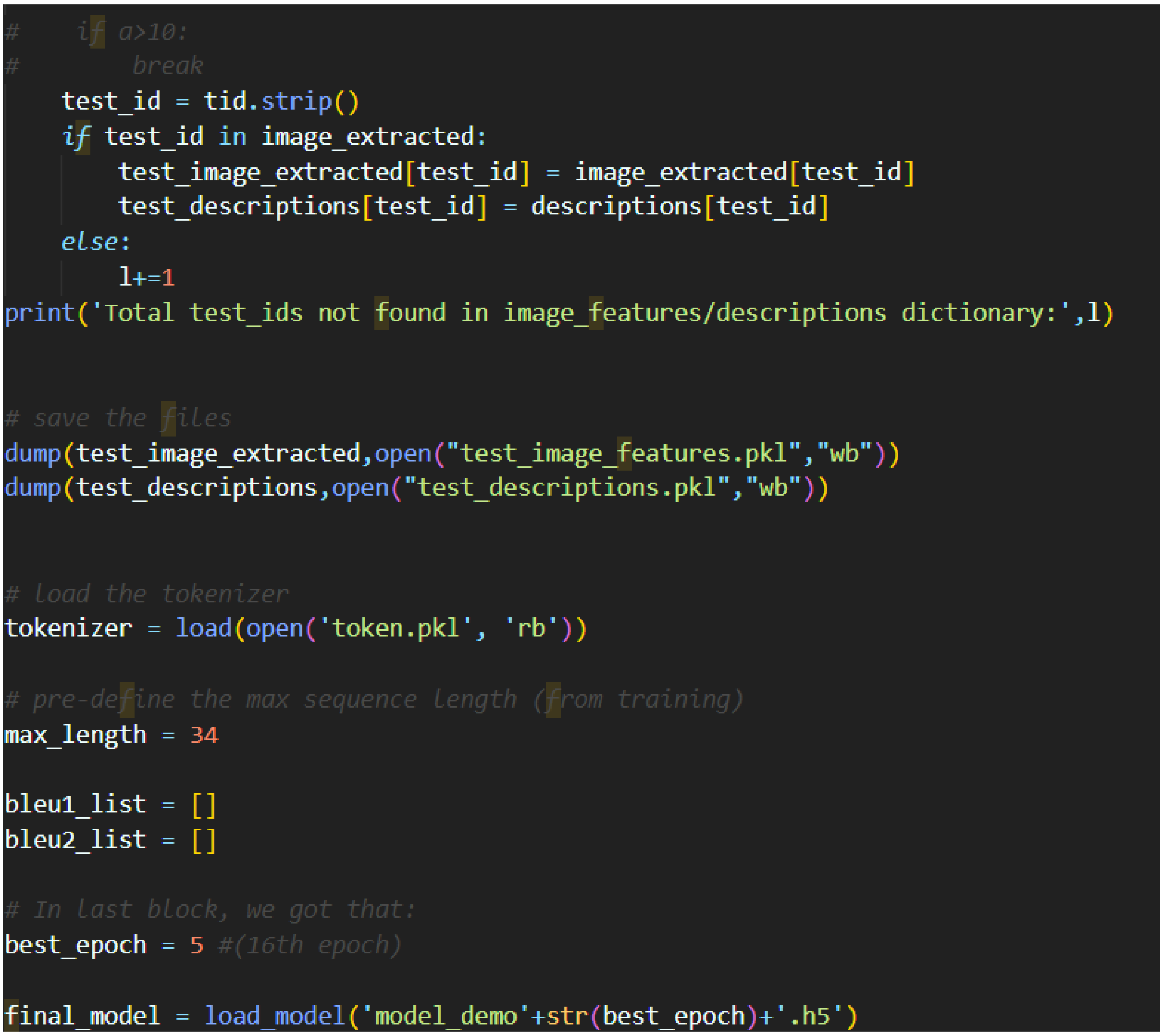
```python
    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
        # predict next word
        yhat = model.predict([photo,sequence], verbose=0)
        # convert probability to integer
        yhat = np.argmax(yhat)
        # map integer to word
        word = word_for_id(yhat, tokenizer)
        # stop if we cannot map the word
        if word is None:
            break
        # append as input for generating the next word
        in_text += ' ' + word
        # stop if we predict the end of the sequence
        if word == 'captionend':
            break
    return in_text
```



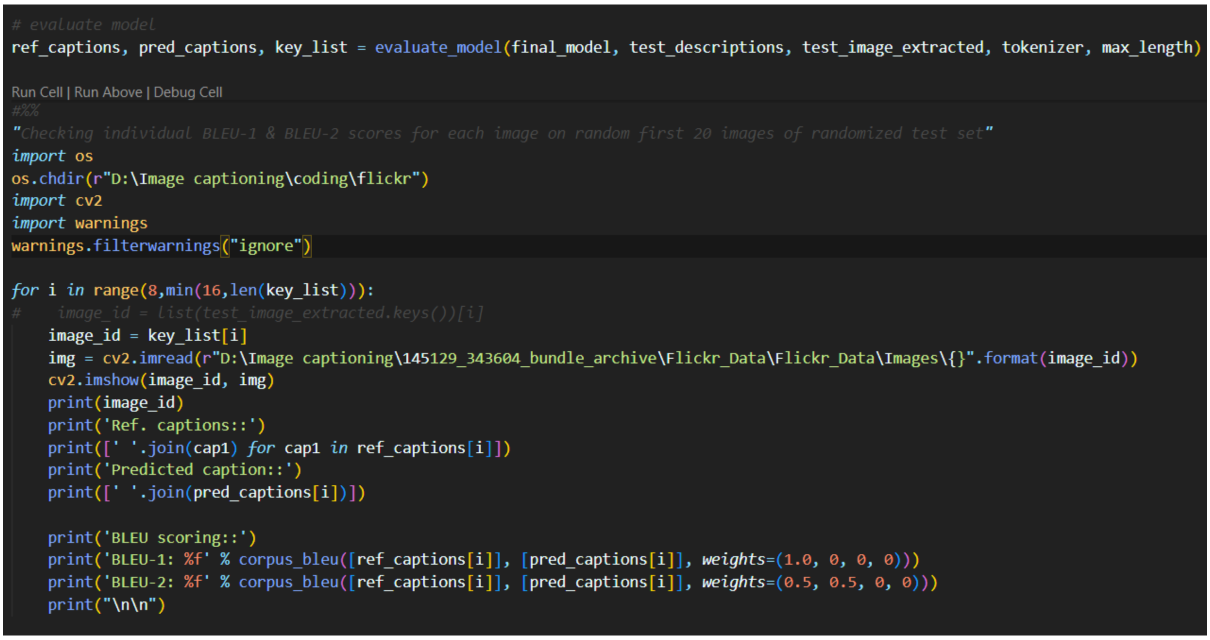
```python
# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
    actual, predicted = list(), list()
    # step over the whole set
    k=0
    l=0
    key_list = []
    for key, desc_list in descriptions.items():
        k+=1
        print(k)
        try:
            # generate description
            yhat = generate_desc(model, tokenizer, photos[key], max_length)
            # store actual and predicted
            references = [d.split() for d in desc_list]

            yhat1 = yhat.split()[1:-1]

            actual.append(references)
            predicted.append(yhat1)
            key_list.append(key)
        except:
            l+=1
```



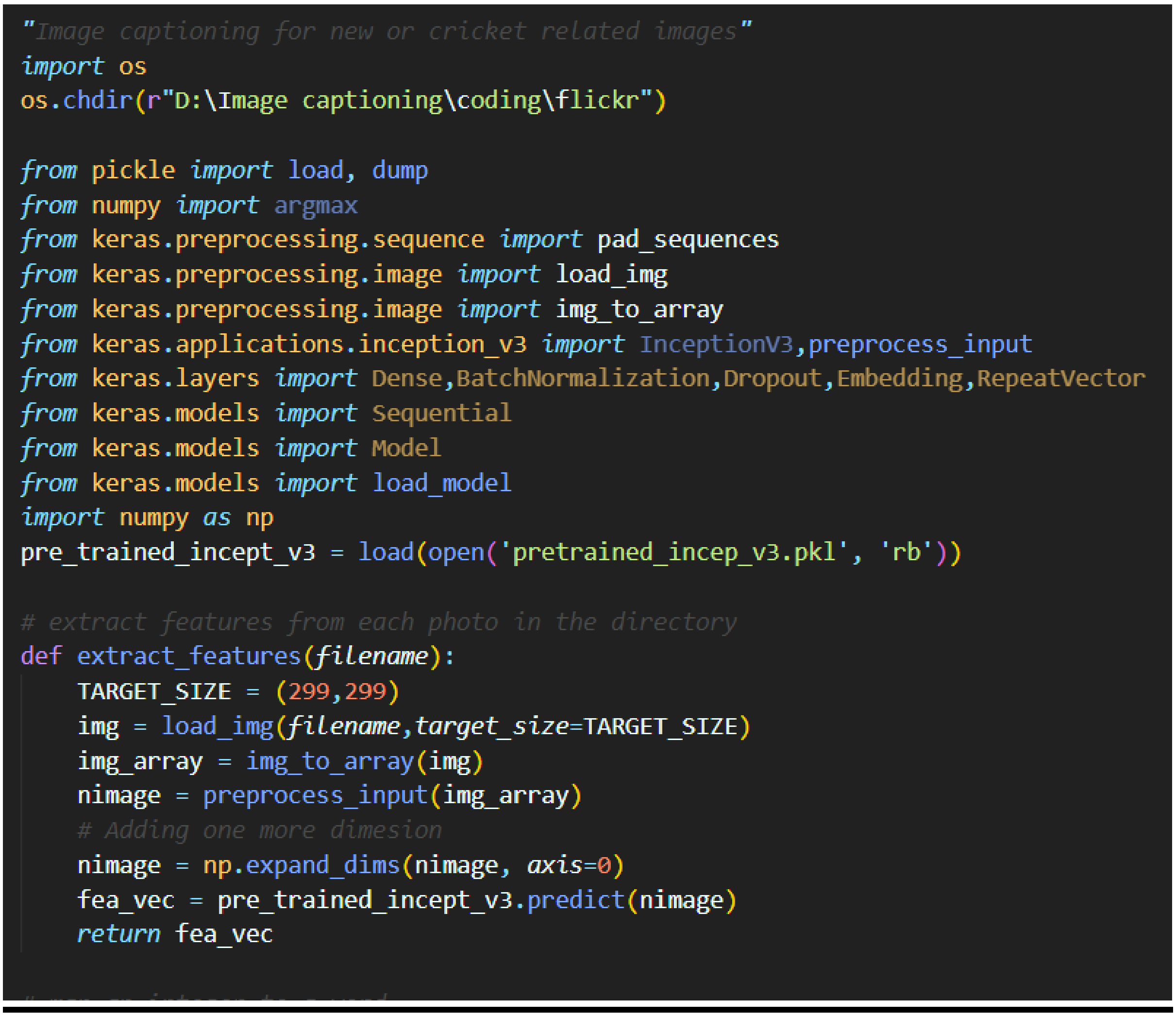
```python
    # calculate BLEU score
    print('BLEU-1: %f' % corpus_bleu(actual,
                                     predicted, weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu(actual,
                                      predicted, weights=(0.5, 0.5, 0, 0)))

    return actual, predicted, key_list


with open(
      r"D:\Image captioning\145129_343604_bundle_archive\
        Flickr_Data\Flickr_Data\Flickr_TextData\Flickr_8k.testImages.txt","r")as f:
    test_ids = f.read()


# prepare test image features
with open("image_features.pkl","rb") as f:
    image_extracted = load(f)


# prepare test image descriptions
with open("descriptions.pkl","rb") as f:
    descriptions = load(f)

test_image_extracted = dict()
test_descriptions = dict()

l = 0
a = 0
for tid in test_ids.split("\n"):
    a+=1
#     if a>10:
#         break
```



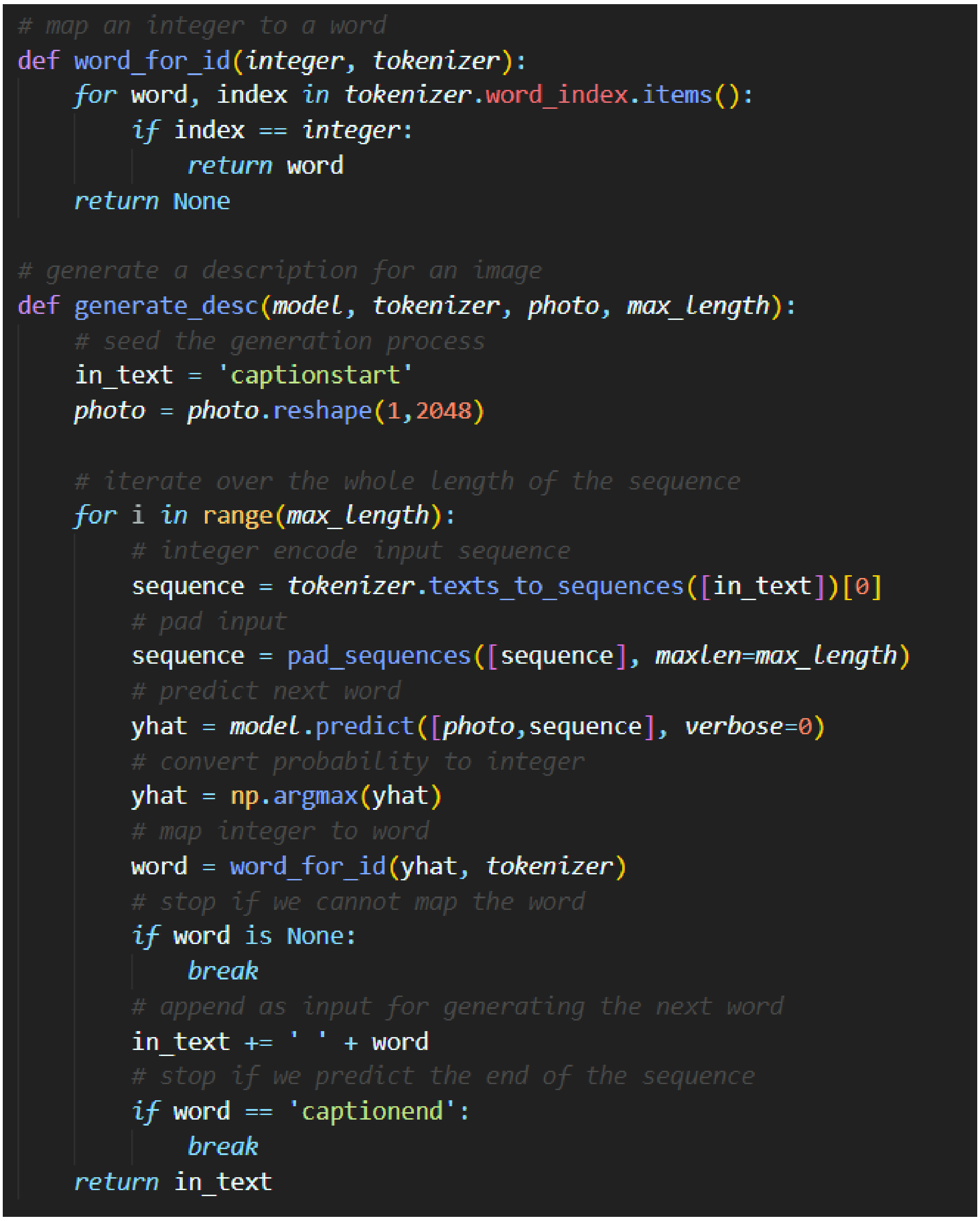
```
#    if a>10:
#        break
    test_id = tid.strip()
    if test_id in image_extracted:
        test_image_extracted[test_id] = image_extracted[test_id]
        test_descriptions[test_id] = descriptions[test_id]
    else:
        l+=1
print('Total test_ids not found in image_features/descriptions dictionary:',l)


# save the files
dump(test_image_extracted,open("test_image_features.pkl","wb"))
dump(test_descriptions,open("test_descriptions.pkl","wb"))


# load the tokenizer
tokenizer = load(open('token.pkl', 'rb'))

# pre-define the max sequence length (from training)
max_length = 34

bleu1_list = []
bleu2_list = []

# In last block, we got that:
best_epoch = 5 #(16th epoch)

final_model = load_model('model_demo'+str(best_epoch)+'.h5')
```



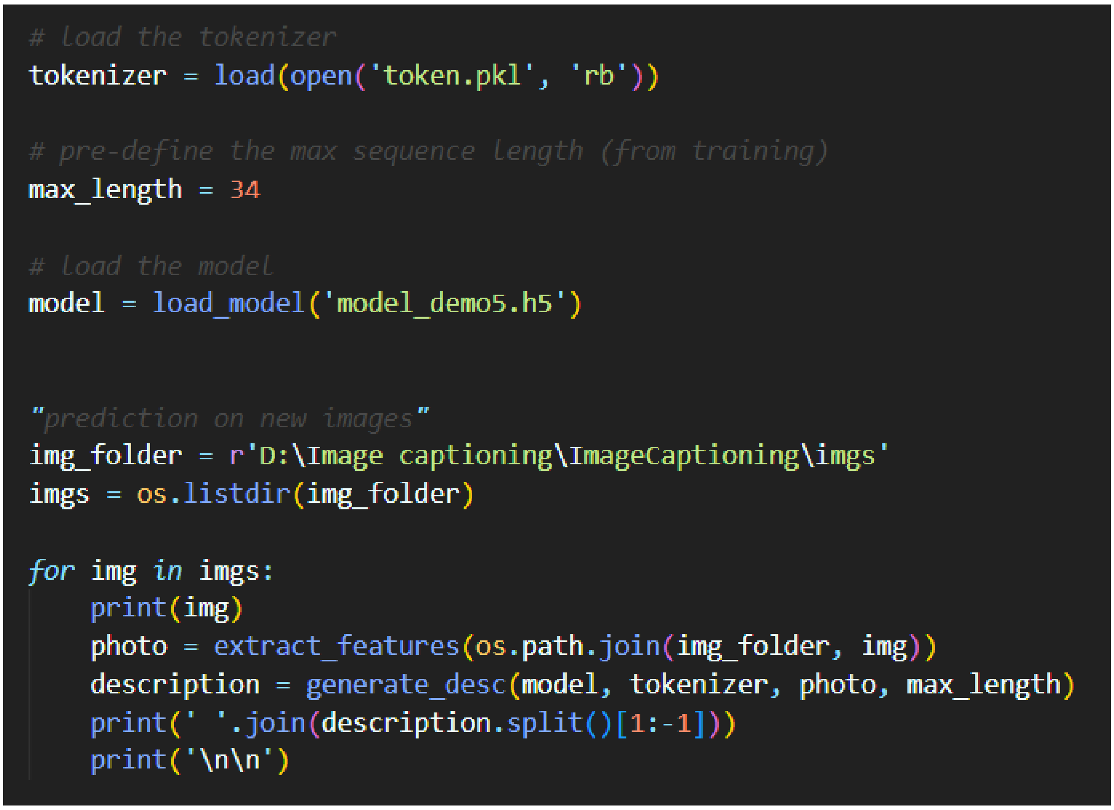
```
# evaluate model
ref_captions, pred_captions, key_list = evaluate_model(final_model, test_descriptions, test_image_extracted, tokenizer, max_length)

Run Cell | Run Above | Debug Cell
#%%
"Checking individual BLEU-1 & BLEU-2 scores for each image on random first 20 images of randomized test set"
import os
os.chdir(r"D:\Image captioning\coding\flickr")
import cv2
import warnings
warnings.filterwarnings("ignore")

for i in range(8,min(16,len(key_list))):
#    image_id = list(test_image_extracted.keys())[i]
    image_id = key_list[i]
    img = cv2.imread(r"D:\Image captioning\145129_343604_bundle_archive\Flickr_Data\Flickr_Data\Images\{}".format(image_id))
    cv2.imshow(image_id, img)
    print(image_id)
    print('Ref. captions::')
    print([' '.join(cap1) for cap1 in ref_captions[i]])
    print('Predicted caption::')
    print([' '.join(pred_captions[i])])

    print('BLEU scoring::')
    print('BLEU-1: %f' % corpus_bleu([ref_captions[i]], [pred_captions[i]], weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu([ref_captions[i]], [pred_captions[i]], weights=(0.5, 0.5, 0, 0)))
    print("\n\n")
```



```python
"Image captioning for new or cricket related images"
import os
os.chdir(r"D:\Image captioning\coding\flickr")

from pickle import load, dump
from numpy import argmax
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.layers import Dense,BatchNormalization,Dropout,Embedding,RepeatVector
from keras.models import Sequential
from keras.models import Model
from keras.models import load_model
import numpy as np
pre_trained_incept_v3 = load(open('pretrained_incep_v3.pkl', 'rb'))

# extract features from each photo in the directory
def extract_features(filename):
    TARGET_SIZE = (299,299)
    img = load_img(filename,target_size=TARGET_SIZE)
    img_array = img_to_array(img)
    nimage = preprocess_input(img_array)
    # Adding one more dimesion
    nimage = np.expand_dims(nimage, axis=0)
    fea_vec = pre_trained_incept_v3.predict(nimage)
    return fea_vec

# map an integer to a word
```

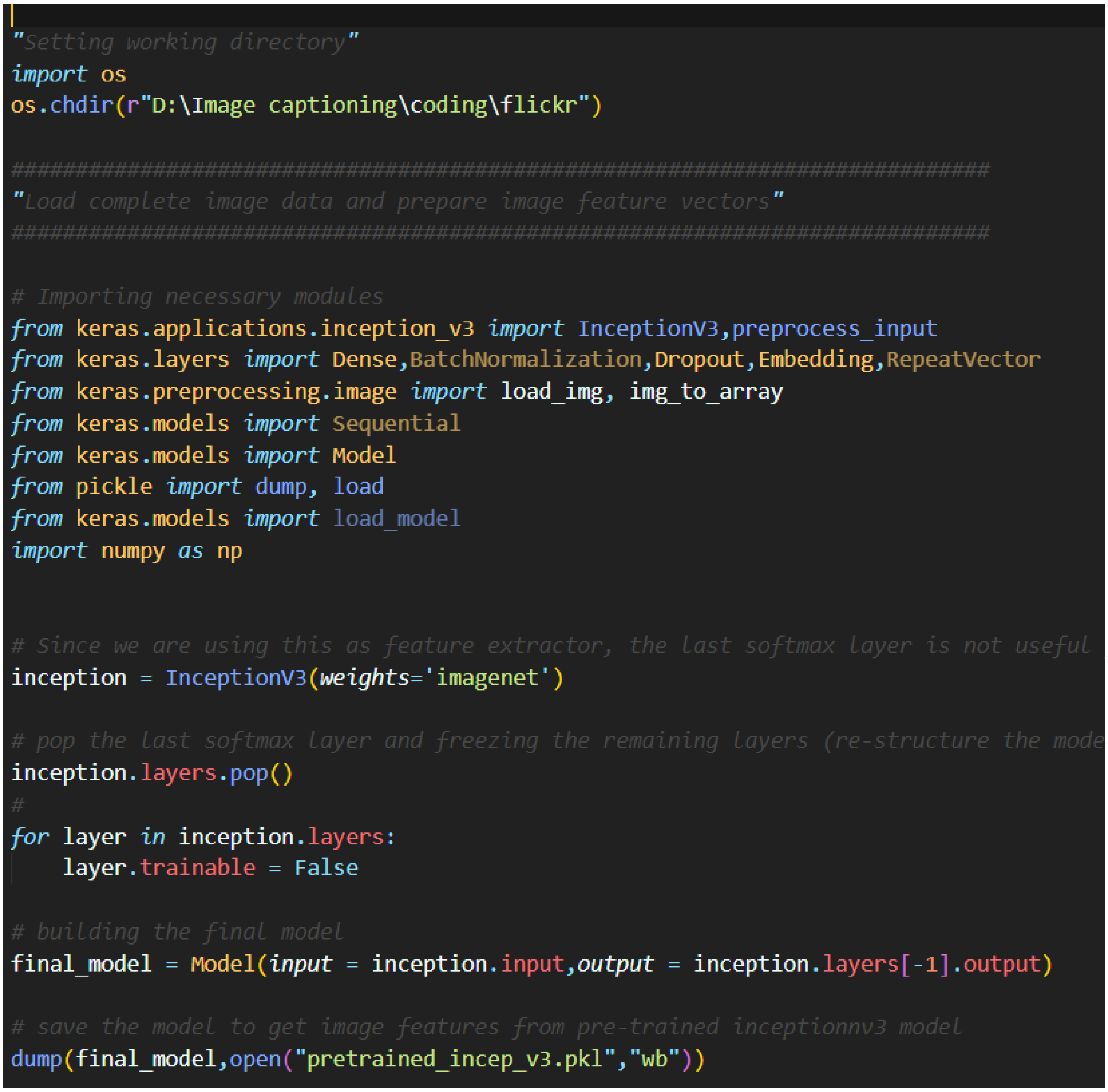


```python
# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'captionstart'
    photo = photo.reshape(1,2048)

    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
        # predict next word
        yhat = model.predict([photo,sequence], verbose=0)
        # convert probability to integer
        yhat = np.argmax(yhat)
        # map integer to word
        word = word_for_id(yhat, tokenizer)
        # stop if we cannot map the word
        if word is None:
            break
        # append as input for generating the next word
        in_text += ' ' + word
        # stop if we predict the end of the sequence
        if word == 'captionend':
            break
    return in_text
```

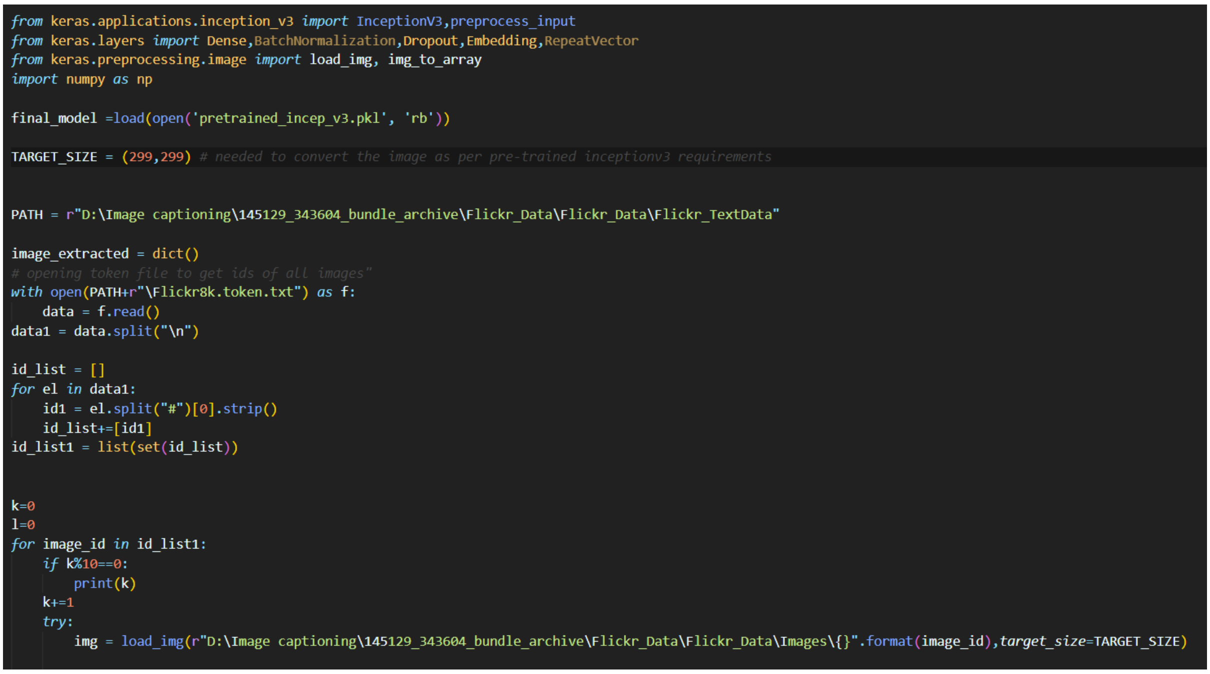


```python
# load the tokenizer
tokenizer = load(open('token.pkl', 'rb'))

# pre-define the max sequence length (from training)
max_length = 34

# load the model
model = load_model('model_demo5.h5')


"prediction on new images"
img_folder = r'D:\Image captioning\ImageCaptioning\imgs'
imgs = os.listdir(img_folder)

for img in imgs:
    print(img)
    photo = extract_features(os.path.join(img_folder, img))
    description = generate_desc(model, tokenizer, photo, max_length)
    print(' '.join(description.split()[1:-1]))
    print('\n\n')
```

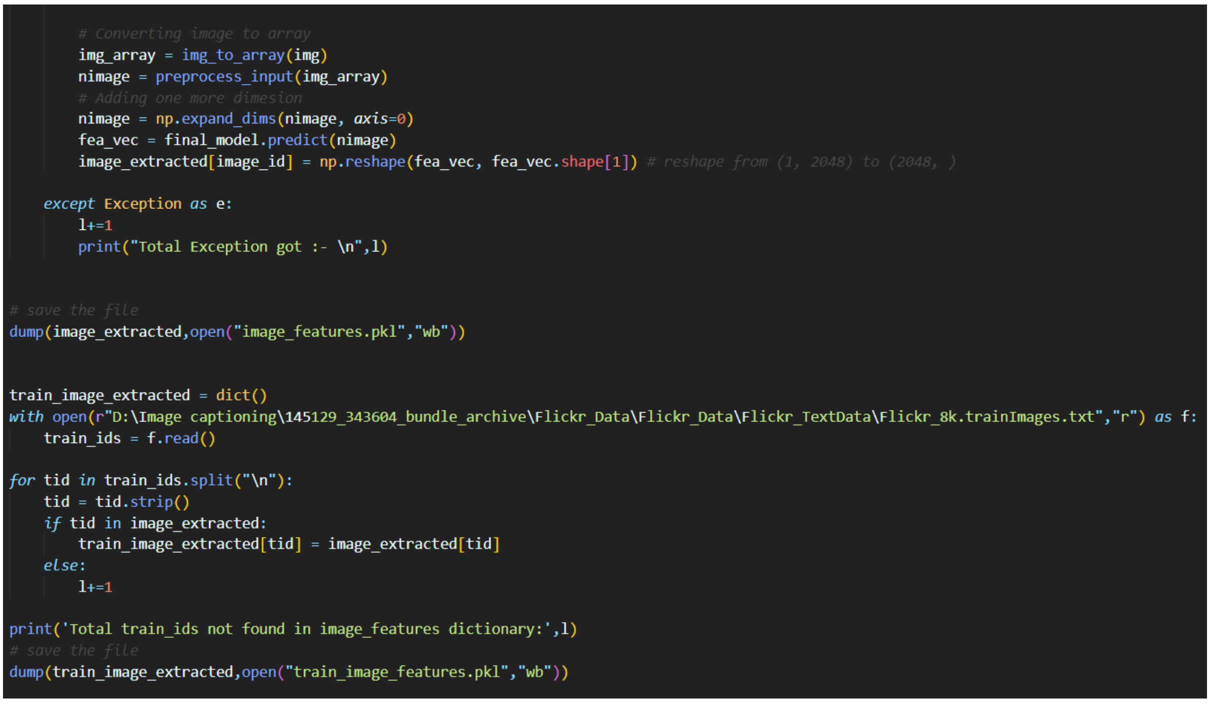
```python
# load the tokenizer
tokenizer = load(open('token.pkl', 'rb'))

# pre-define the max sequence length (from training)
max_length = 34

# load the model
model = load_model('model_demo5.h5')


"prediction on new images"
img_folder = r'D:\Image captioning\ImageCaptioning\imgs'
imgs = os.listdir(img_folder)

for img in imgs:
    print(img)
    photo = extract_features(os.path.join(img_folder, img))
    description = generate_desc(model, tokenizer, photo, max_length)
    print(' '.join(description.split()[1:-1]))
    print('\n\n')
```



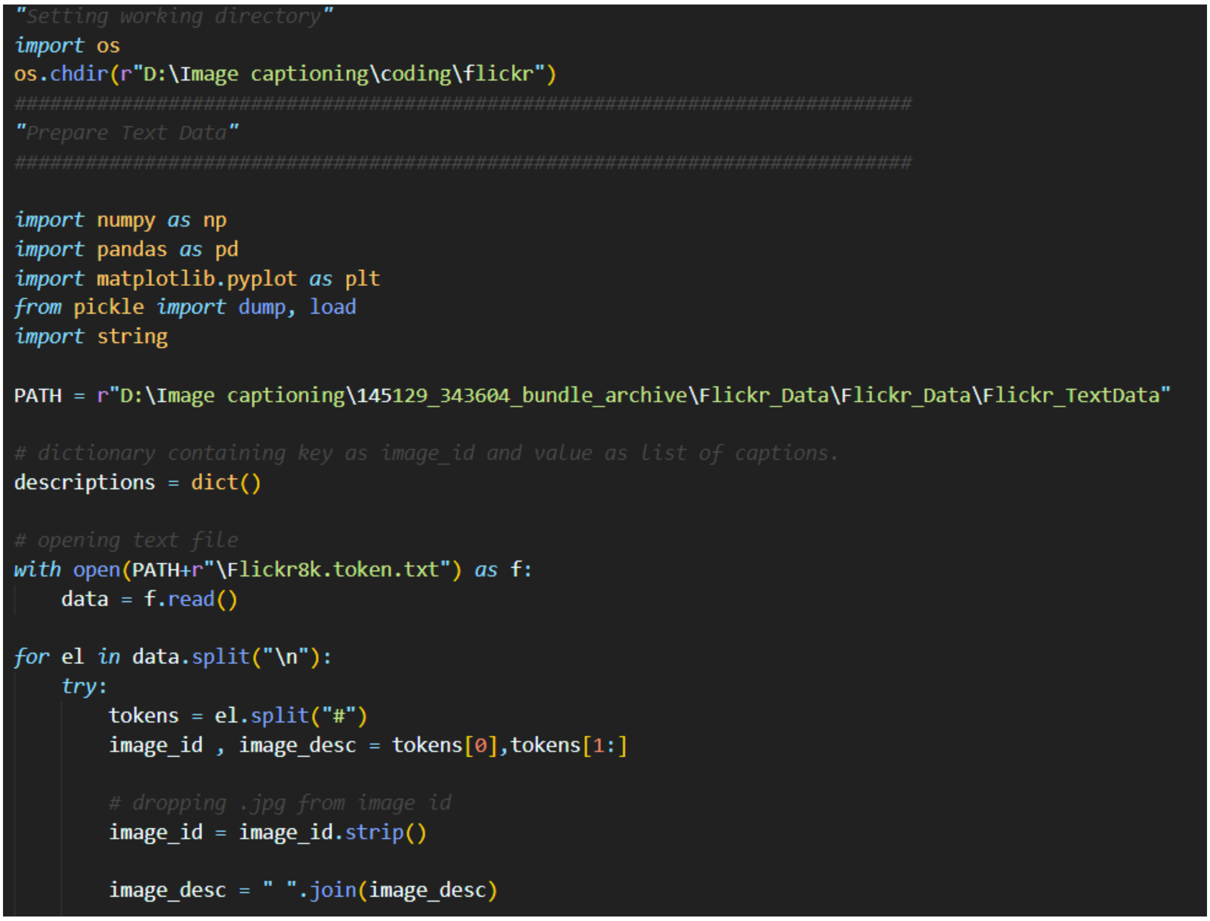
```
"Setting working directory"
import os
os.chdir(r"D:\Image captioning\coding\flickr")

##########################################################################
"Load complete image data and prepare image feature vectors"
##########################################################################

# Importing necessary modules
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.layers import Dense,BatchNormalization,Dropout,Embedding,RepeatVector
from keras.preprocessing.image import load_img, img_to_array
from keras.models import Sequential
from keras.models import Model
from pickle import dump, load
from keras.models import load_model
import numpy as np


# Since we are using this as feature extractor, the last softmax layer is not useful
inception = InceptionV3(weights='imagenet')

# pop the last softmax layer and freezing the remaining layers (re-structure the mode
inception.layers.pop()
#
for layer in inception.layers:
    layer.trainable = False

# building the final model
final_model = Model(input = inception.input,output = inception.layers[-1].output)

# save the model to get image features from pre-trained inceptionnv3 model
dump(final_model,open("pretrained_incep_v3.pkl","wb"))
```



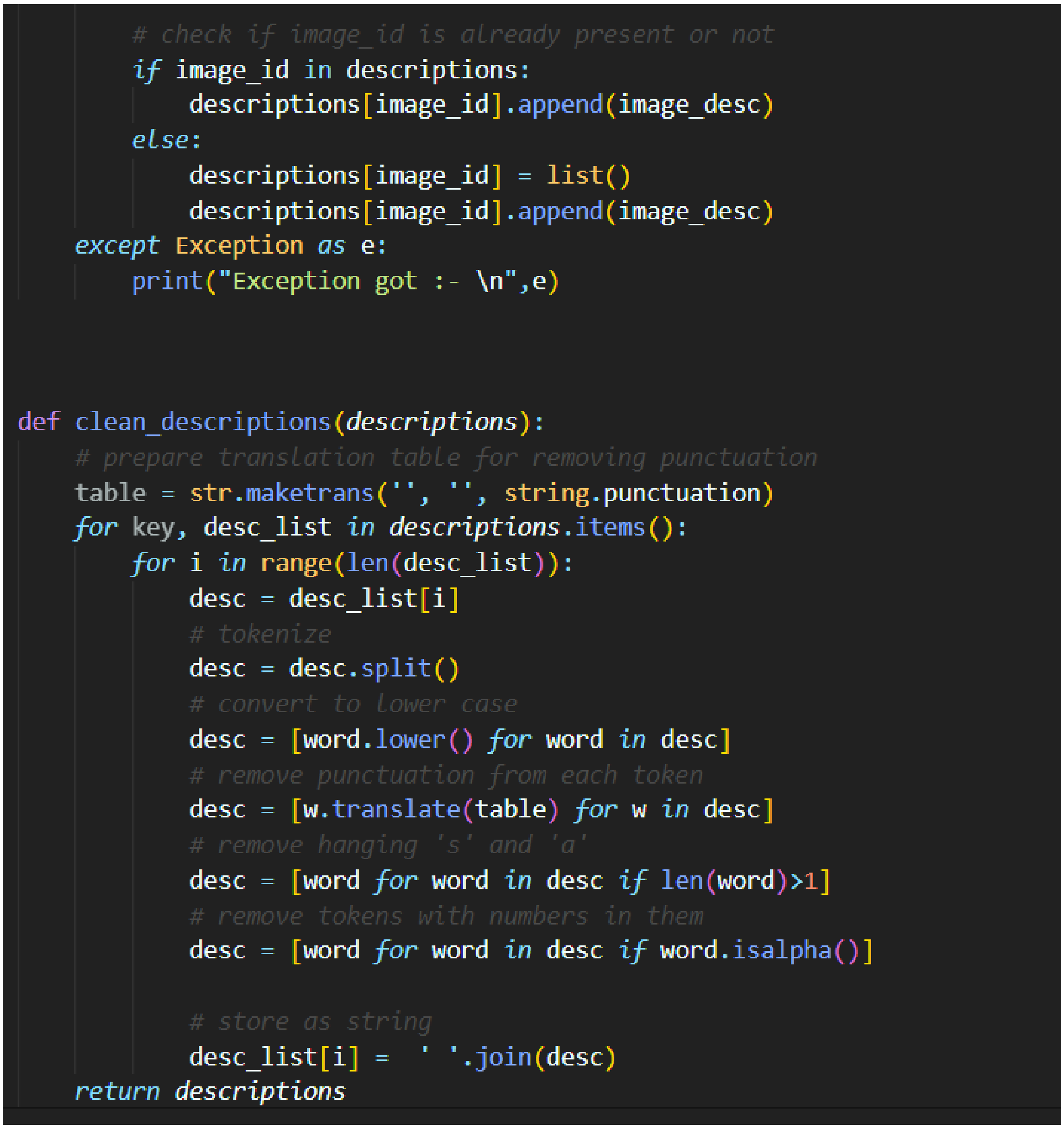
```
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.layers import Dense,BatchNormalization,Dropout,Embedding,RepeatVector
from keras.preprocessing.image import load_img, img_to_array
import numpy as np

final_model =load(open('pretrained_incep_v3.pkl', 'rb'))

TARGET_SIZE = (299,299) # needed to convert the image as per pre-trained inceptionv3 requirements


PATH = r"D:\Image captioning\145129_343604_bundle_archive\Flickr_Data\Flickr_Data\Flickr_TextData"

image_extracted = dict()
# opening token file to get ids of all images"
with open(PATH+r"\Flickr8k.token.txt") as f:
    data = f.read()
data1 = data.split("\n")

id_list = []
for el in data1:
    id1 = el.split("#")[0].strip()
    id_list+=[id1]
id_list1 = list(set(id_list))


k=0
l=0
for image_id in id_list1:
    if k%10==0:
        print(k)
    k+=1
    try:
        img = load_img(r"D:\Image captioning\145129_343604_bundle_archive\Flickr_Data\Flickr_Data\Images\{}".format(image_id),target_size=TARGET_SIZE)
```



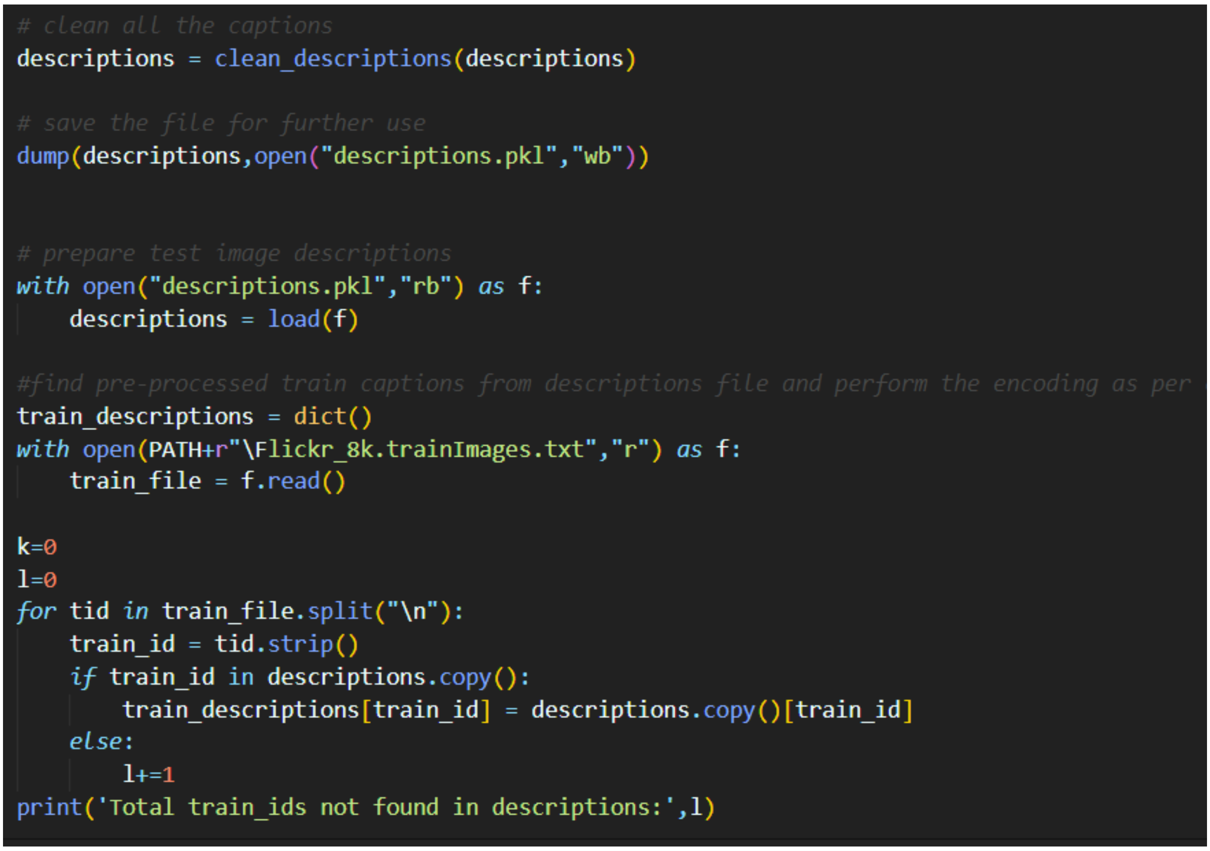
```
        # Converting image to array
        img_array = img_to_array(img)
        nimage = preprocess_input(img_array)
        # Adding one more dimesion
        nimage = np.expand_dims(nimage, axis=0)
        fea_vec = final_model.predict(nimage)
        image_extracted[image_id] = np.reshape(fea_vec, fea_vec.shape[1]) # reshape from (1, 2048) to (2048, )

    except Exception as e:
        l+=1
        print("Total Exception got :- \n",l)


# save the file
dump(image_extracted,open("image_features.pkl","wb"))


train_image_extracted = dict()
with open(r"D:\Image captioning\145129_343604_bundle_archive\Flickr_Data\Flickr_Data\Flickr_TextData\Flickr_8k.trainImages.txt","r") as f:
    train_ids = f.read()

for tid in train_ids.split("\n"):
    tid = tid.strip()
    if tid in image_extracted:
        train_image_extracted[tid] = image_extracted[tid]
    else:
        l+=1

print('Total train_ids not found in image_features dictionary:',l)
# save the file
dump(train_image_extracted,open("train_image_features.pkl","wb"))
```



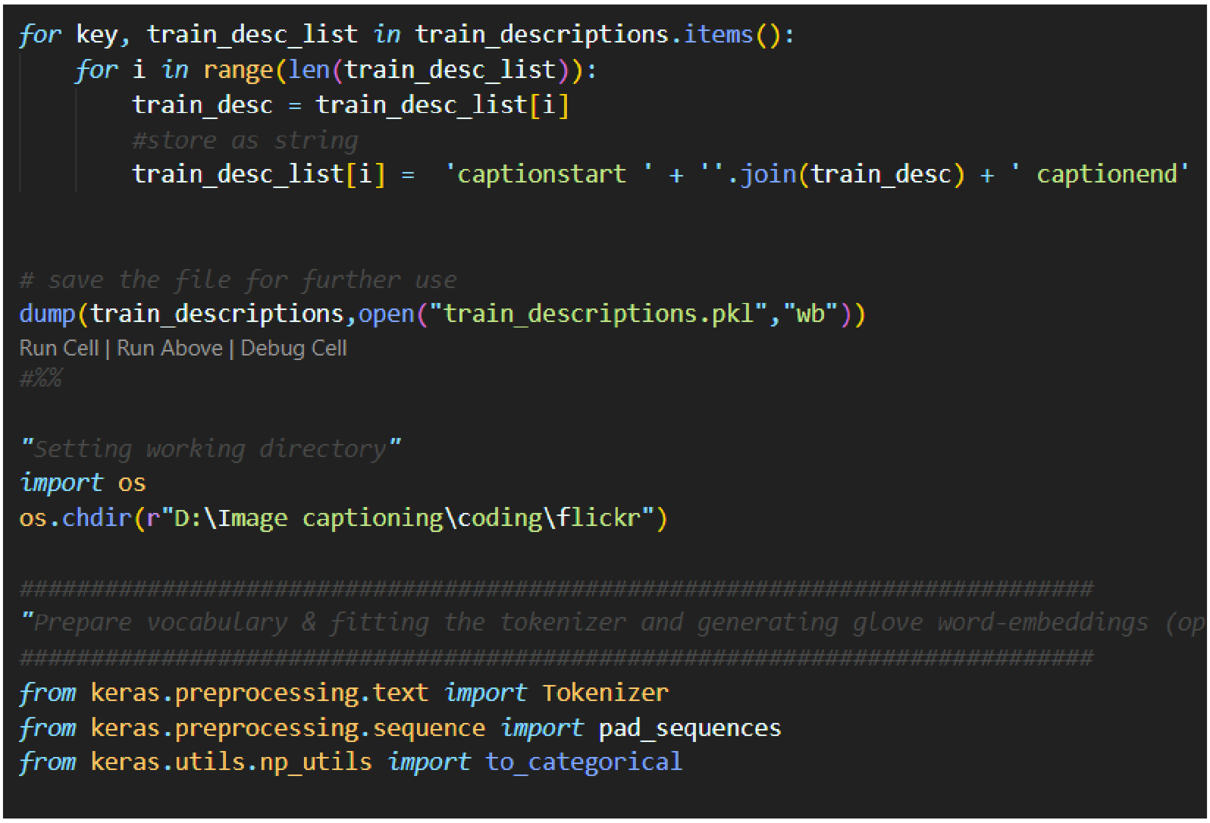
```
"Setting working directory"
import os
os.chdir(r"D:\Image captioning\coding\flickr")
##############################################################################
"Prepare Text Data"
##############################################################################


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pickle import dump, load
import string

PATH = r"D:\Image captioning\145129_343604_bundle_archive\Flickr_Data\Flickr_Data\Flickr_TextData"

# dictionary containing key as image_id and value as list of captions.
descriptions = dict()

# opening text file
with open(PATH+r"\Flickr8k.token.txt") as f:
    data = f.read()

for el in data.split("\n"):
    try:
        tokens = el.split("#")
        image_id , image_desc = tokens[0],tokens[1:]

        # dropping .jpg from image id
        image_id = image_id.strip()

        image_desc = " ".join(image_desc)
```



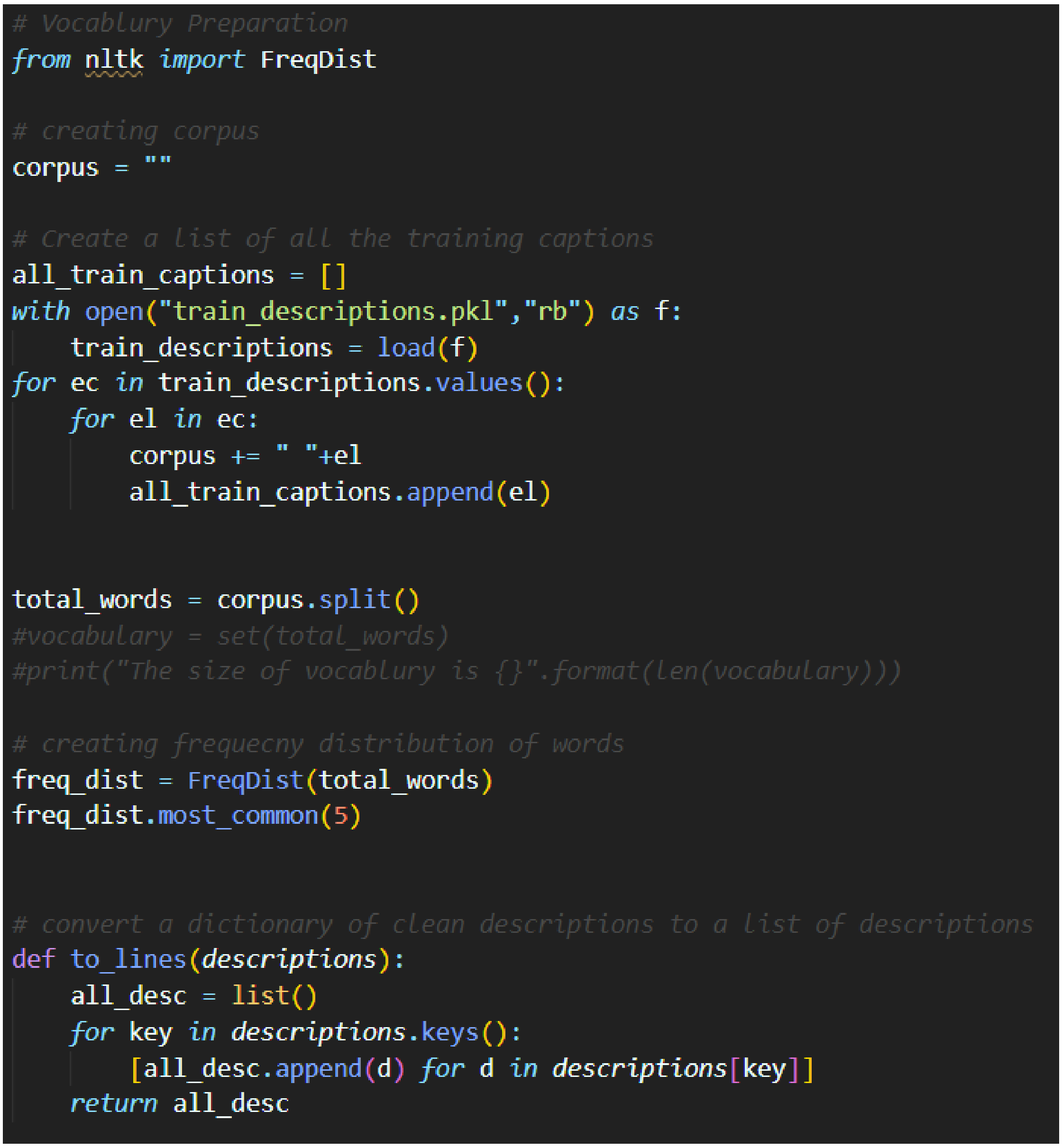
```python
        # check if image_id is already present or not
        if image_id in descriptions:
            descriptions[image_id].append(image_desc)
        else:
            descriptions[image_id] = list()
            descriptions[image_id].append(image_desc)
    except Exception as e:
        print("Exception got :- \n",e)



def clean_descriptions(descriptions):
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            # tokenize
            desc = desc.split()
            # convert to lower case
            desc = [word.lower() for word in desc]
            # remove punctuation from each token
            desc = [w.translate(table) for w 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()]

            # store as string
            desc_list[i] =  ' '.join(desc)
    return descriptions
```



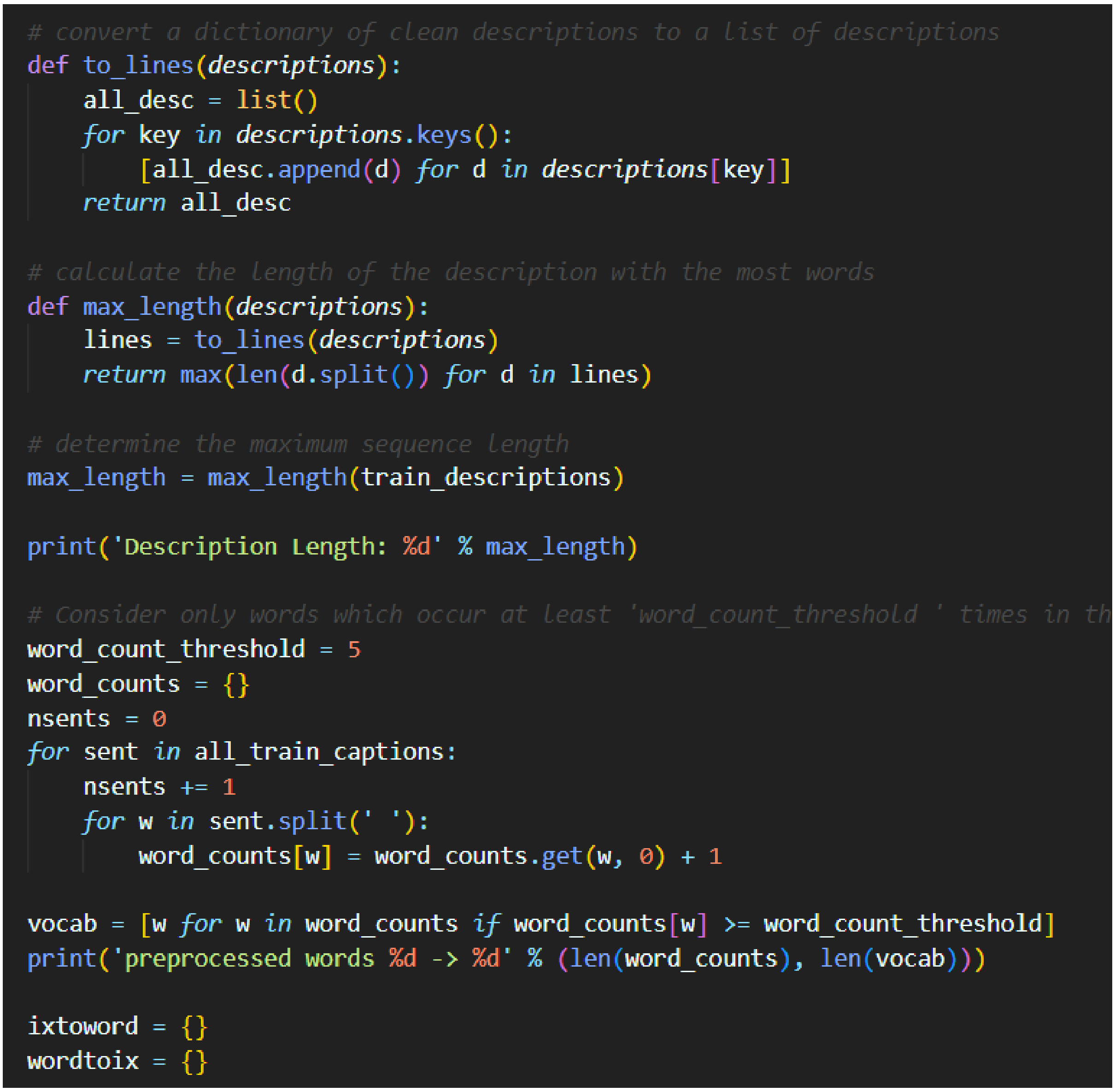
```
# clean all the captions
descriptions = clean_descriptions(descriptions)

# save the file for further use
dump(descriptions,open("descriptions.pkl","wb"))


# prepare test image descriptions
with open("descriptions.pkl","rb") as f:
    descriptions = load(f)

#find pre-processed train captions from descriptions file and perform the encoding as per
train_descriptions = dict()
with open(PATH+r"\Flickr_8k.trainImages.txt","r") as f:
    train_file = f.read()

k=0
l=0
for tid in train_file.split("\n"):
    train_id = tid.strip()
    if train_id in descriptions.copy():
        train_descriptions[train_id] = descriptions.copy()[train_id]
    else:
        l+=1
print('Total train_ids not found in descriptions:',l)
```



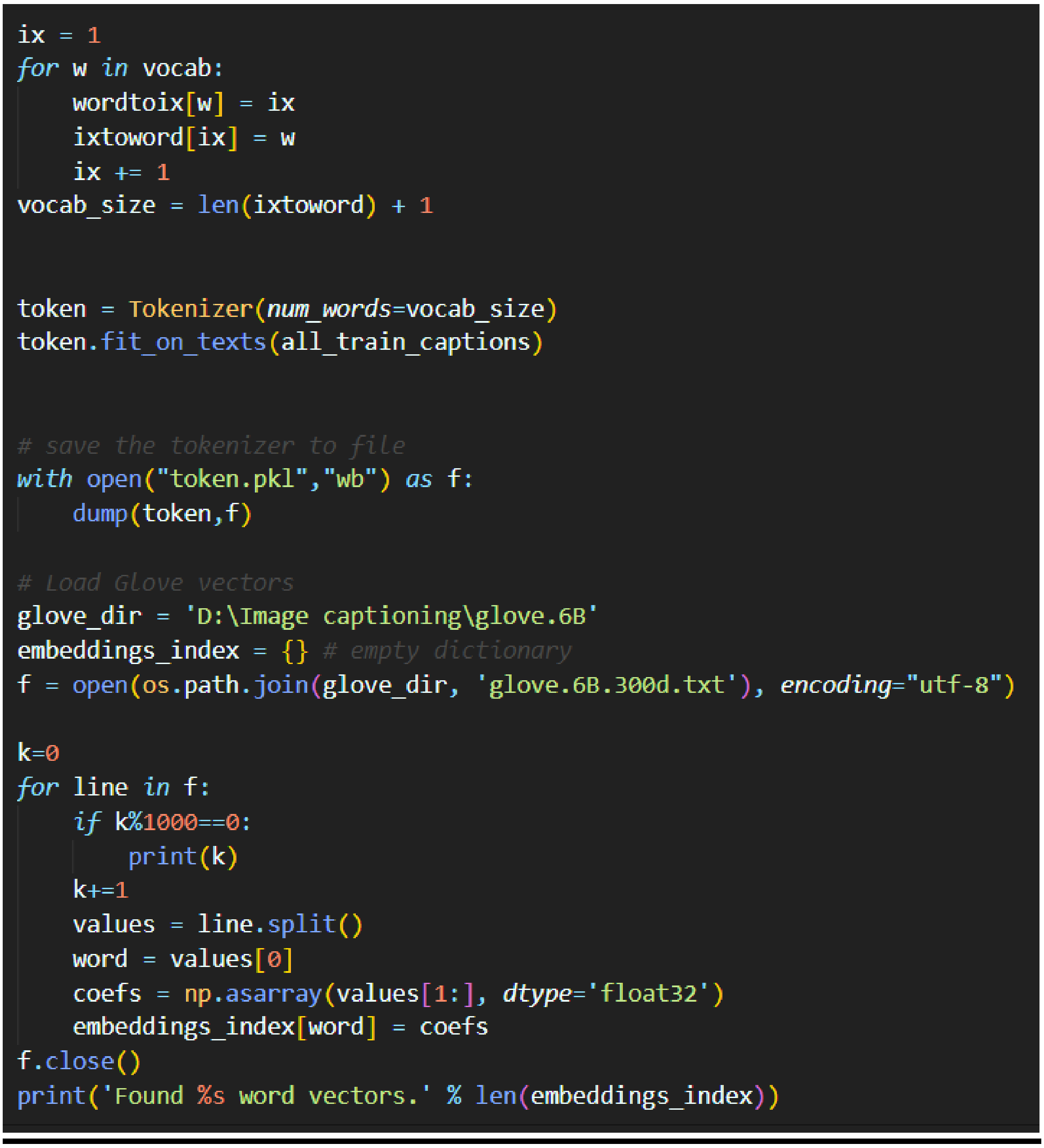
```
for key, train_desc_list in train_descriptions.items():
    for i in range(len(train_desc_list)):
        train_desc = train_desc_list[i]
        #store as string
        train_desc_list[i] =  'captionstart ' + ''.join(train_desc) + ' captionend'


# save the file for further use
dump(train_descriptions,open("train_descriptions.pkl","wb"))
Run Cell | Run Above | Debug Cell
#%%

"Setting working directory"
import os
os.chdir(r"D:\Image captioning\coding\flickr")

#################################################################################
"Prepare vocabulary & fitting the tokenizer and generating glove word-embeddings (op
#################################################################################
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
```



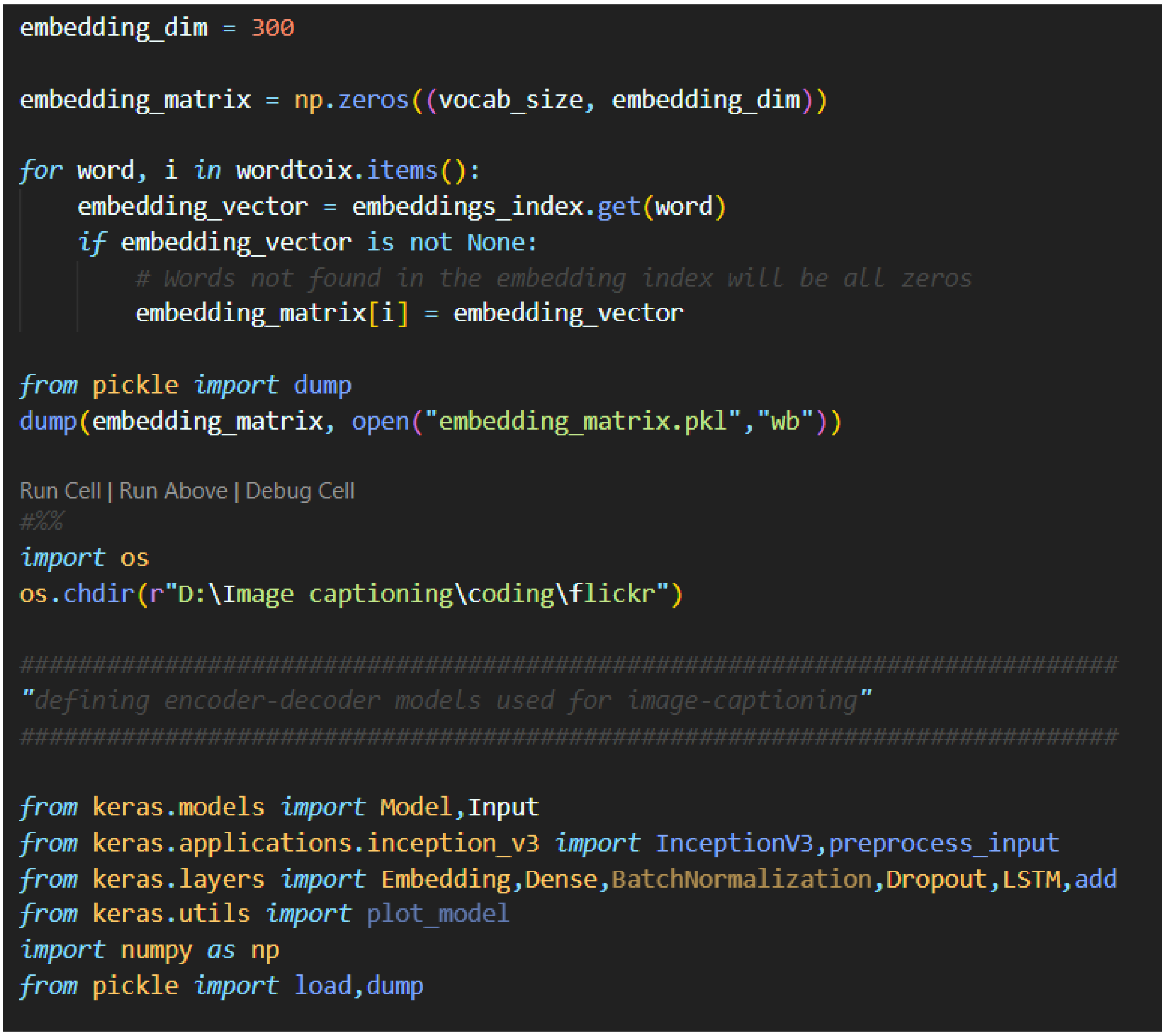
```
# Vocablury Preparation
from nltk import FreqDist

# creating corpus
corpus = ""

# Create a list of all the training captions
all_train_captions = []
with open("train_descriptions.pkl","rb") as f:
    train_descriptions = load(f)
for ec in train_descriptions.values():
    for el in ec:
        corpus += " "+el
        all_train_captions.append(el)


total_words = corpus.split()
#vocabulary = set(total_words)
#print("The size of vocablury is {}".format(len(vocabulary)))

# creating frequecny distribution of words
freq_dist = FreqDist(total_words)
freq_dist.most_common(5)


# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc
```



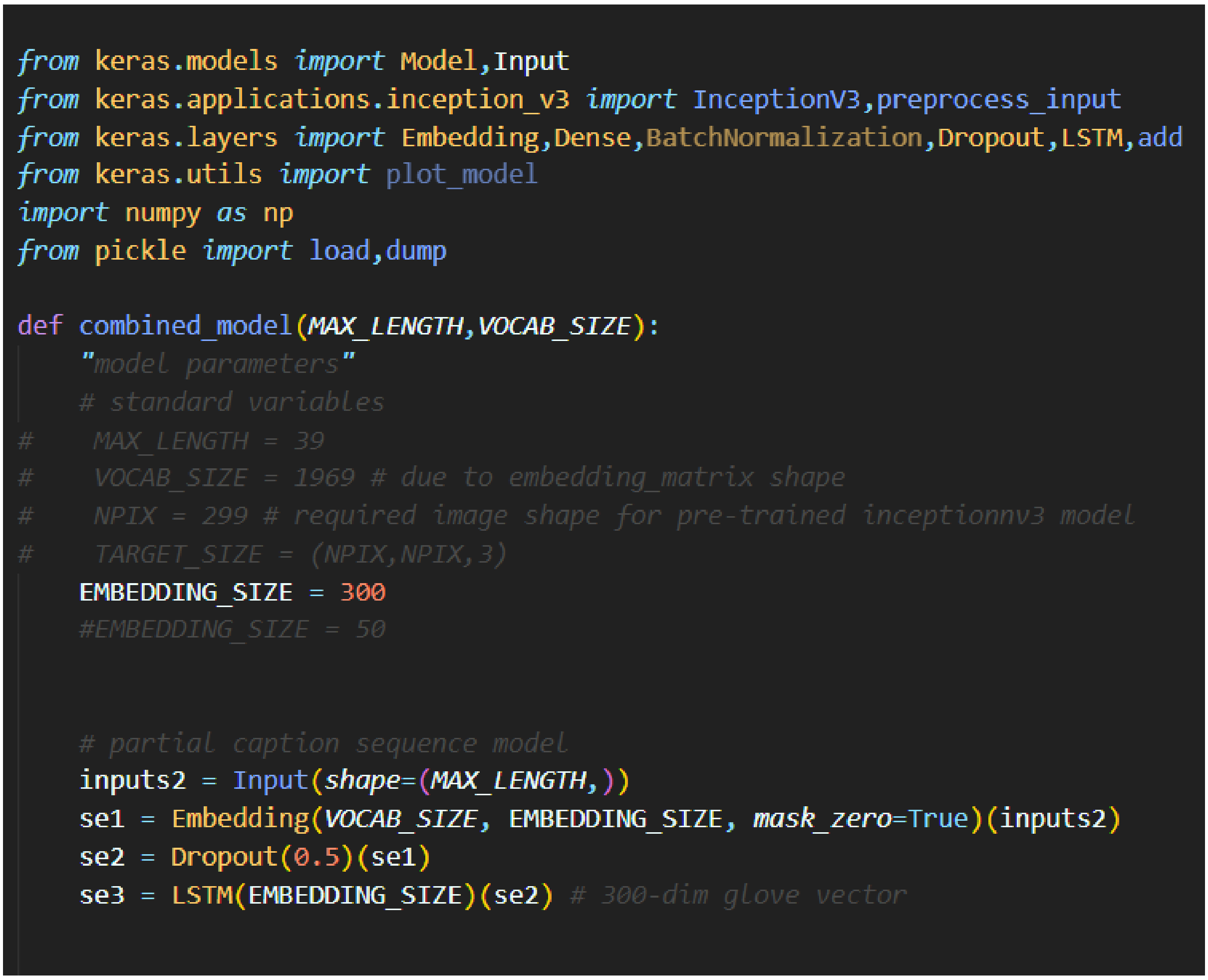
```python
# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)

# determine the maximum sequence length
max_length = max_length(train_descriptions)

print('Description Length: %d' % max_length)

# Consider only words which occur at least 'word_count_threshold ' times in th
word_count_threshold = 5
word_counts = {}
nsents = 0
for sent in all_train_captions:
    nsents += 1
    for w in sent.split(' '):
        word_counts[w] = word_counts.get(w, 0) + 1

vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
print('preprocessed words %d -> %d' % (len(word_counts), len(vocab)))

ixtoword = {}
wordtoix = {}
```



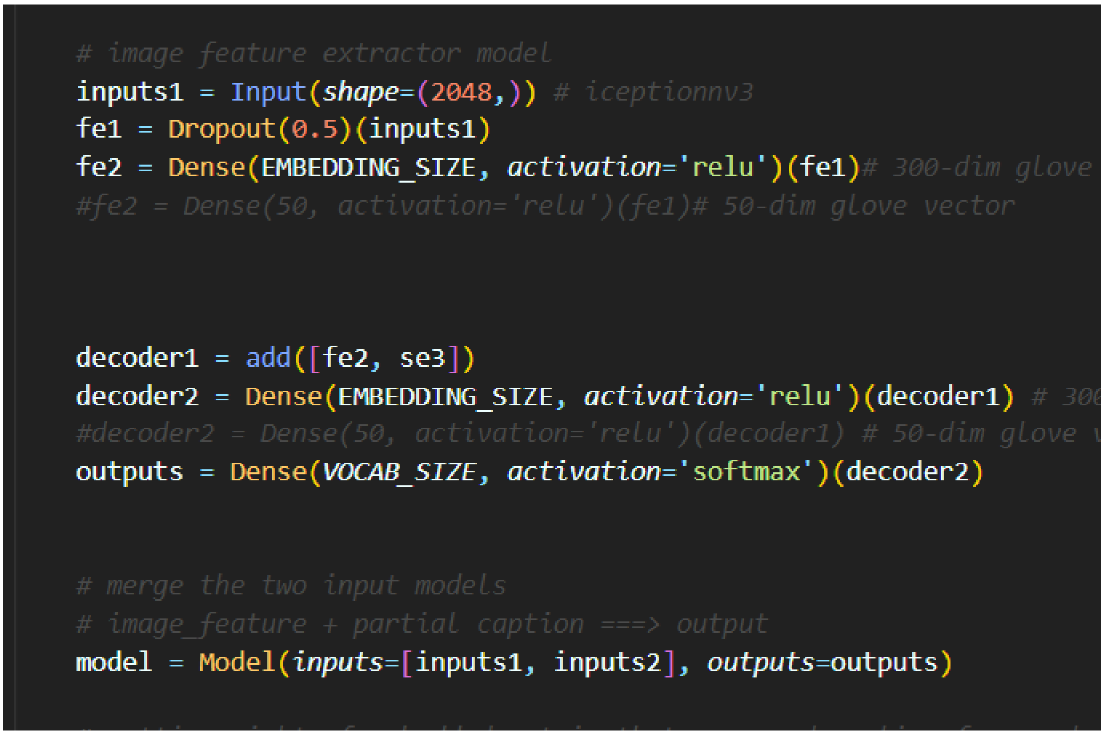
```
ix = 1
for w in vocab:
    wordtoix[w] = ix
    ixtoword[ix] = w
    ix += 1
vocab_size = len(ixtoword) + 1


token = Tokenizer(num_words=vocab_size)
token.fit_on_texts(all_train_captions)


# save the tokenizer to file
with open("token.pkl","wb") as f:
    dump(token,f)

# Load Glove vectors
glove_dir = 'D:\Image captioning\glove.6B'
embeddings_index = {} # empty dictionary
f = open(os.path.join(glove_dir, 'glove.6B.300d.txt'), encoding="utf-8")

k=0
for line in f:
    if k%1000==0:
        print(k)
    k+=1
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Found %s word vectors.' % len(embeddings_index))
```



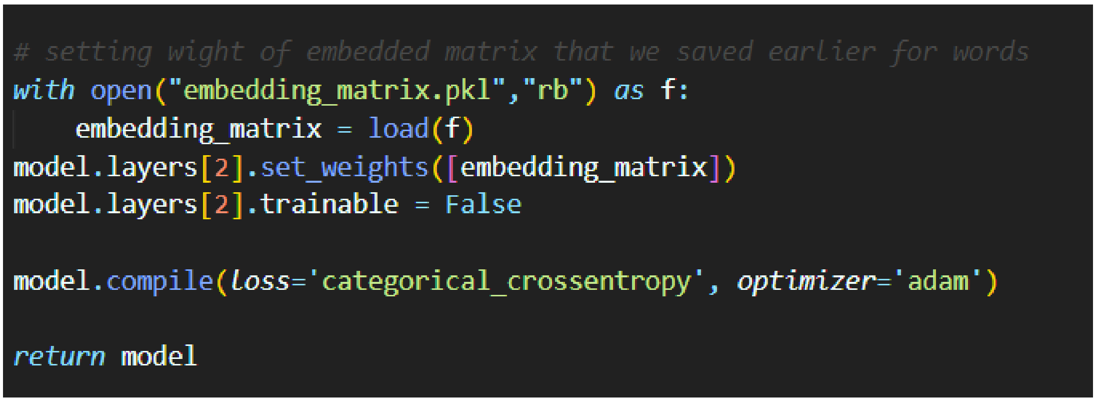
```
embedding_dim = 300

embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in wordtoix.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector

from pickle import dump
dump(embedding_matrix, open("embedding_matrix.pkl","wb"))

Run Cell | Run Above | Debug Cell
#%%
import os
os.chdir(r"D:\Image captioning\coding\flickr")

############################################################################
"defining encoder-decoder models used for image-captioning"
############################################################################

from keras.models import Model,Input
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.layers import Embedding,Dense,BatchNormalization,Dropout,LSTM,add
from keras.utils import plot_model
import numpy as np
from pickle import load,dump
```



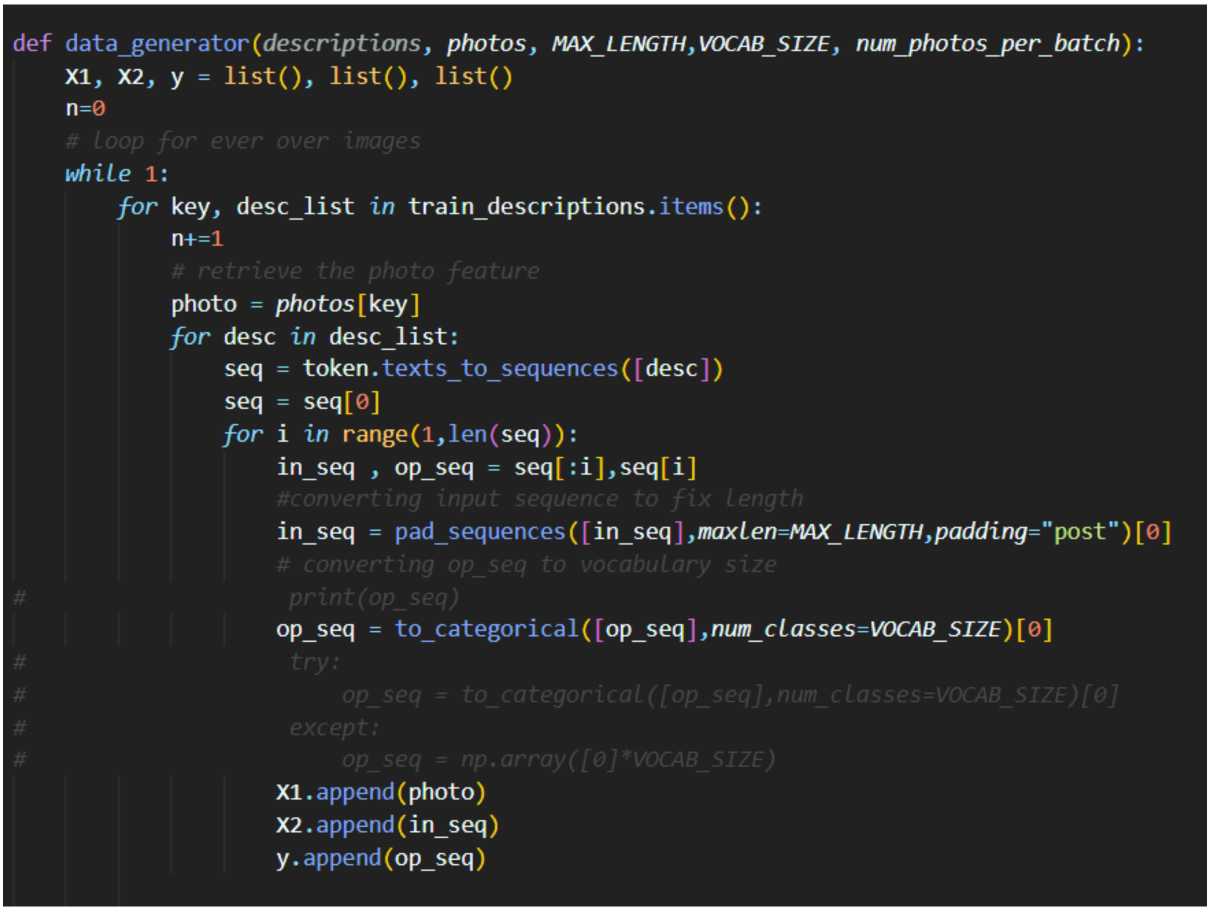
```python
from keras.models import Model,Input
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.layers import Embedding,Dense,BatchNormalization,Dropout,LSTM,add
from keras.utils import plot_model
import numpy as np
from pickle import load,dump

def combined_model(MAX_LENGTH,VOCAB_SIZE):
    "model parameters"
    # standard variables
#    MAX_LENGTH = 39
#    VOCAB_SIZE = 1969 # due to embedding_matrix shape
#    NPIX = 299 # required image shape for pre-trained inceptionnv3 model
#    TARGET_SIZE = (NPIX,NPIX,3)
    EMBEDDING_SIZE = 300
    #EMBEDDING_SIZE = 50


    # partial caption sequence model
    inputs2 = Input(shape=(MAX_LENGTH,))
    se1 = Embedding(VOCAB_SIZE, EMBEDDING_SIZE, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(EMBEDDING_SIZE)(se2) # 300-dim glove vector
```



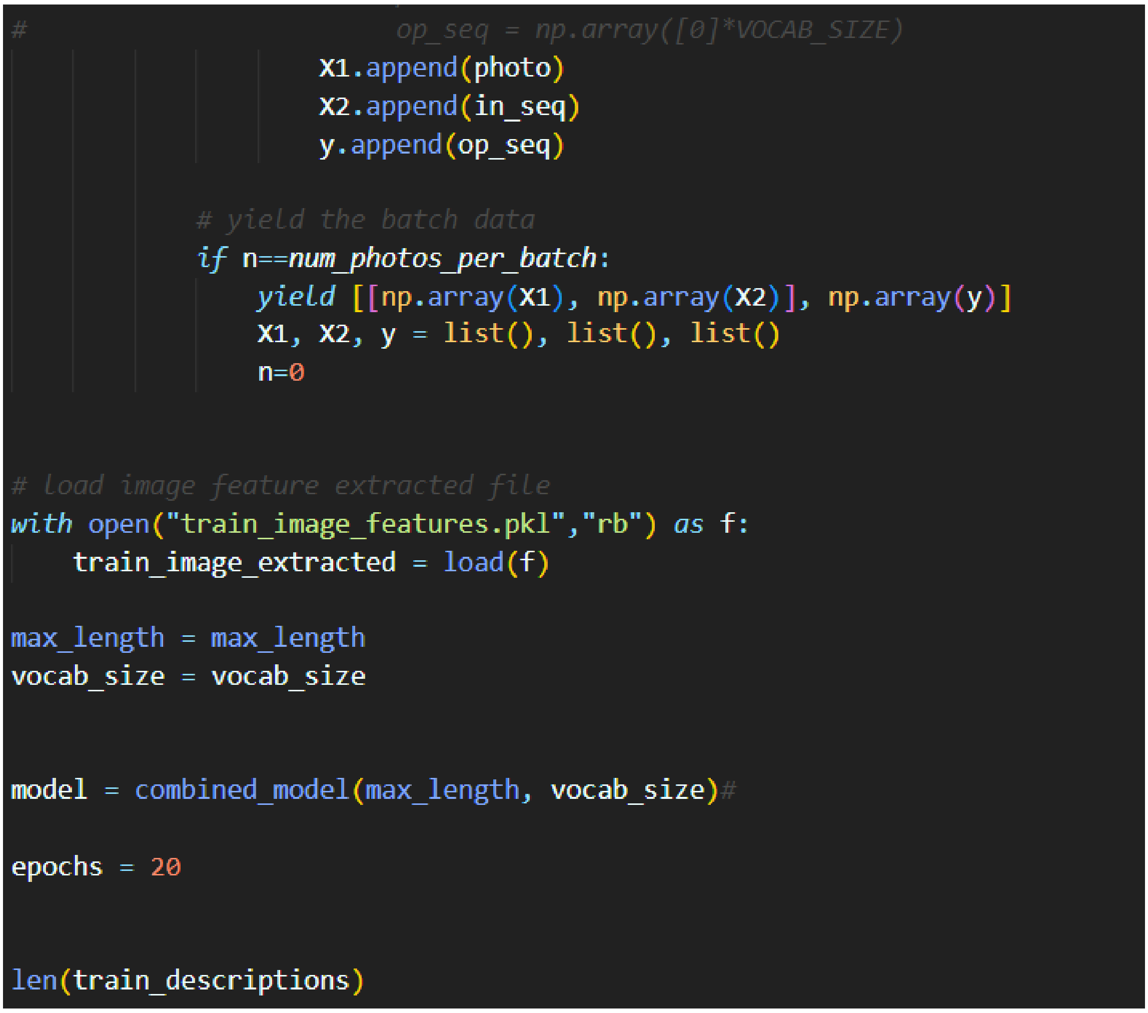
```python
    # image feature extractor model
    inputs1 = Input(shape=(2048,)) # iceptionnv3
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(EMBEDDING_SIZE, activation='relu')(fe1)# 300-dim glove
    #fe2 = Dense(50, activation='relu')(fe1)# 50-dim glove vector



    decoder1 = add([fe2, se3])
    decoder2 = Dense(EMBEDDING_SIZE, activation='relu')(decoder1) # 300
    #decoder2 = Dense(50, activation='relu')(decoder1) # 50-dim glove v
    outputs = Dense(VOCAB_SIZE, activation='softmax')(decoder2)


    # merge the two input models
    # image_feature + partial caption ===> output
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
```



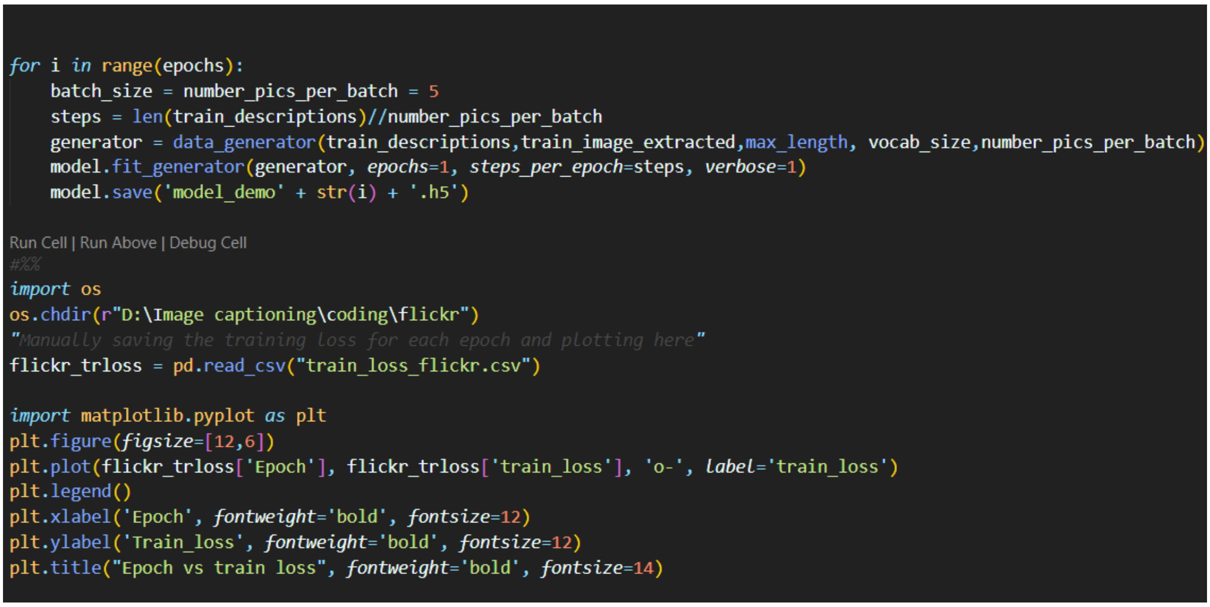
```
# setting wight of embedded matrix that we saved earlier for words
with open("embedding_matrix.pkl","rb") as f:
    embedding_matrix = load(f)
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False

model.compile(loss='categorical_crossentropy', optimizer='adam')

return model
```



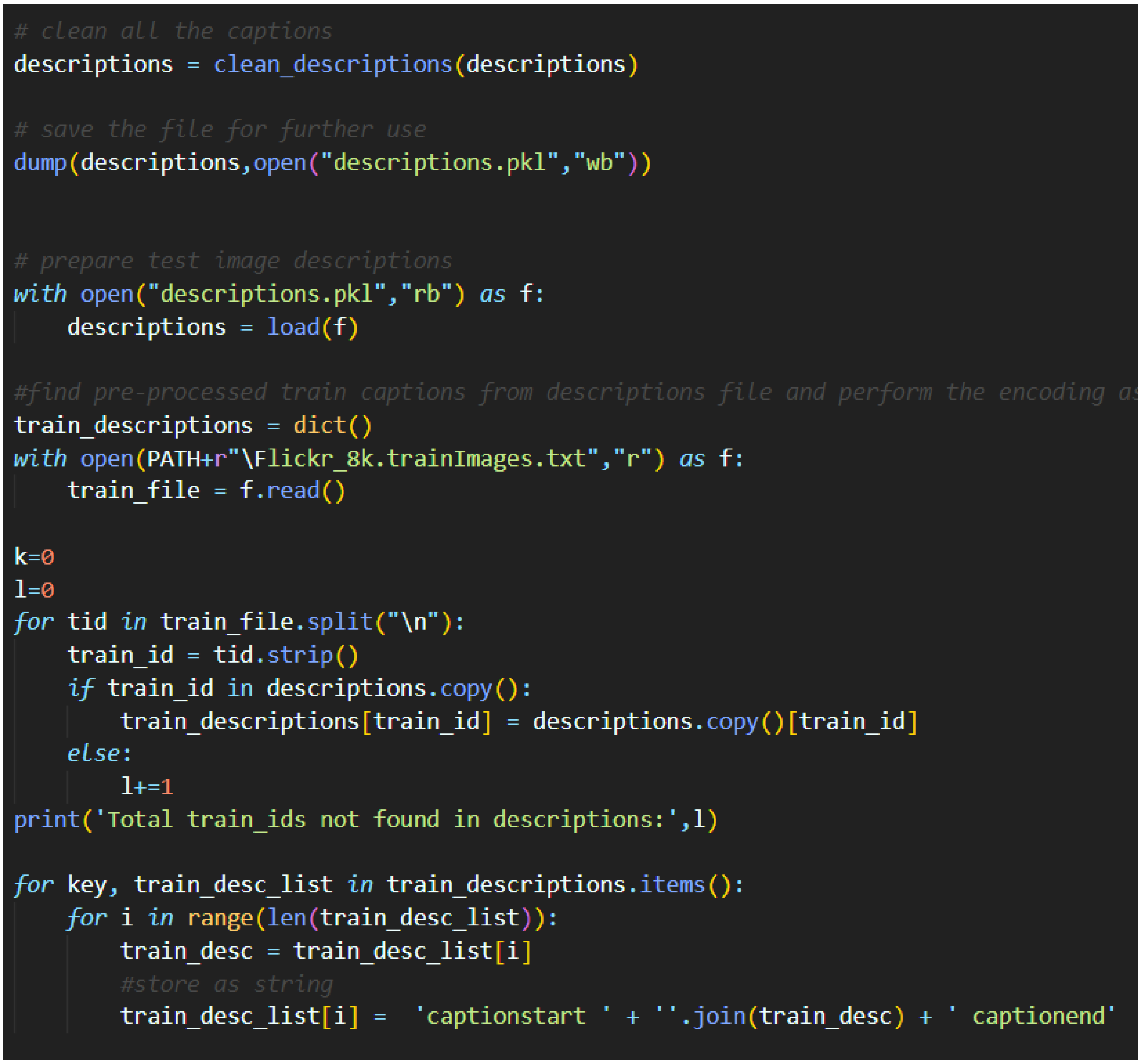
```
def data_generator(descriptions, photos, MAX_LENGTH,VOCAB_SIZE, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    # loop for ever over images
    while 1:
        for key, desc_list in train_descriptions.items():
            n+=1
            # retrieve the photo feature
            photo = photos[key]
            for desc in desc_list:
                seq = token.texts_to_sequences([desc])
                seq = seq[0]
                for i in range(1,len(seq)):
                    in_seq , op_seq = seq[:i],seq[i]
                    #converting input sequence to fix length
                    in_seq = pad_sequences([in_seq],maxlen=MAX_LENGTH,padding="post")[0]
                    # converting op_seq to vocabulary size
#                    print(op_seq)
                    op_seq = to_categorical([op_seq],num_classes=VOCAB_SIZE)[0]
#                    try:
#                        op_seq = to_categorical([op_seq],num_classes=VOCAB_SIZE)[0]
#                    except:
#                        op_seq = np.array([0]*VOCAB_SIZE)
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(op_seq)
```



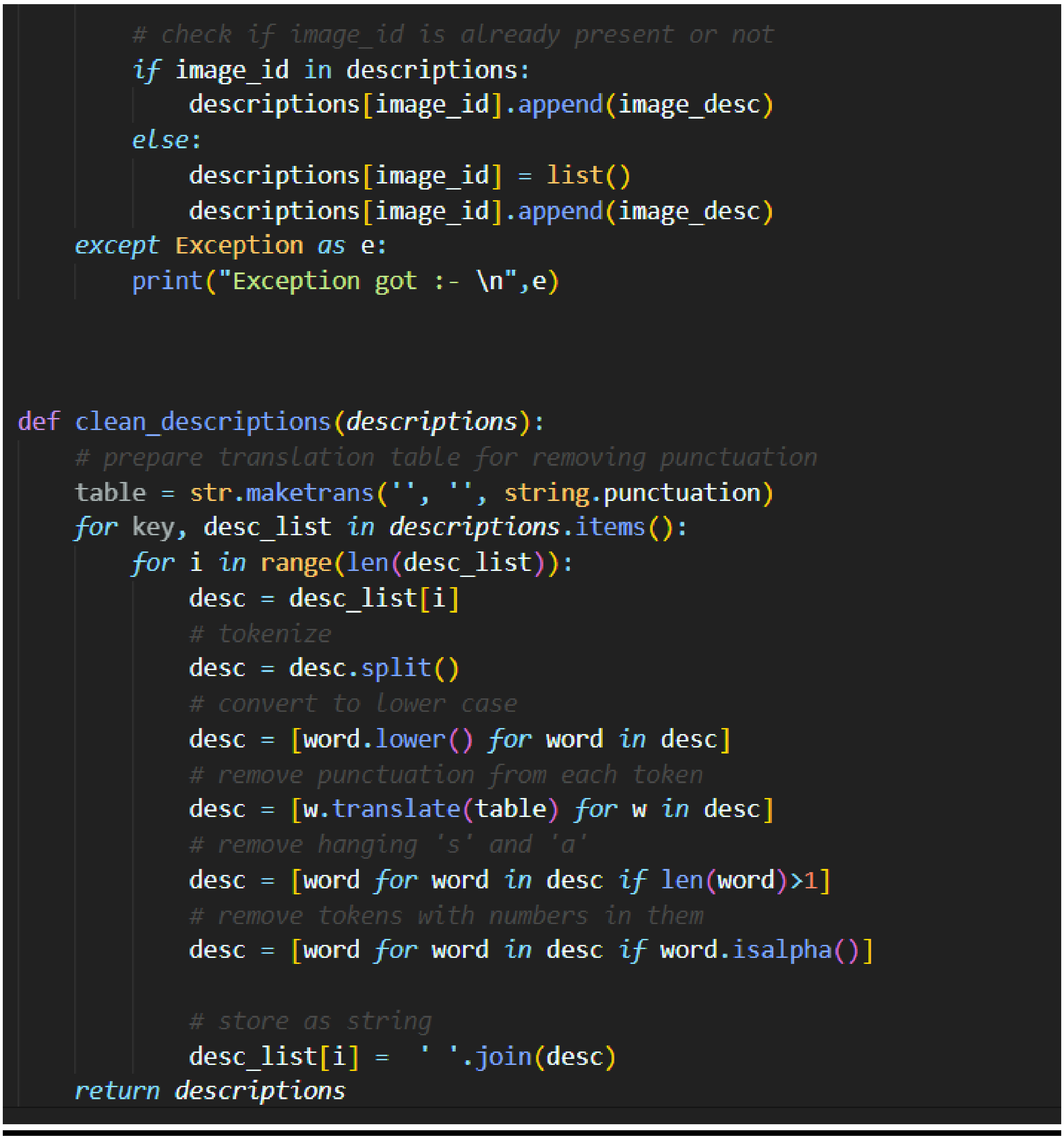
```
#                        op_seq = np.array([0]*VOCAB_SIZE)
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(op_seq)

            # yield the batch data
            if n==num_photos_per_batch:
                yield [[np.array(X1), np.array(X2)], np.array(y)]
                X1, X2, y = list(), list(), list()
                n=0


# load image feature extracted file
with open("train_image_features.pkl","rb") as f:
    train_image_extracted = load(f)

max_length = max_length
vocab_size = vocab_size


model = combined_model(max_length, vocab_size)#

epochs = 20


len(train_descriptions)
```



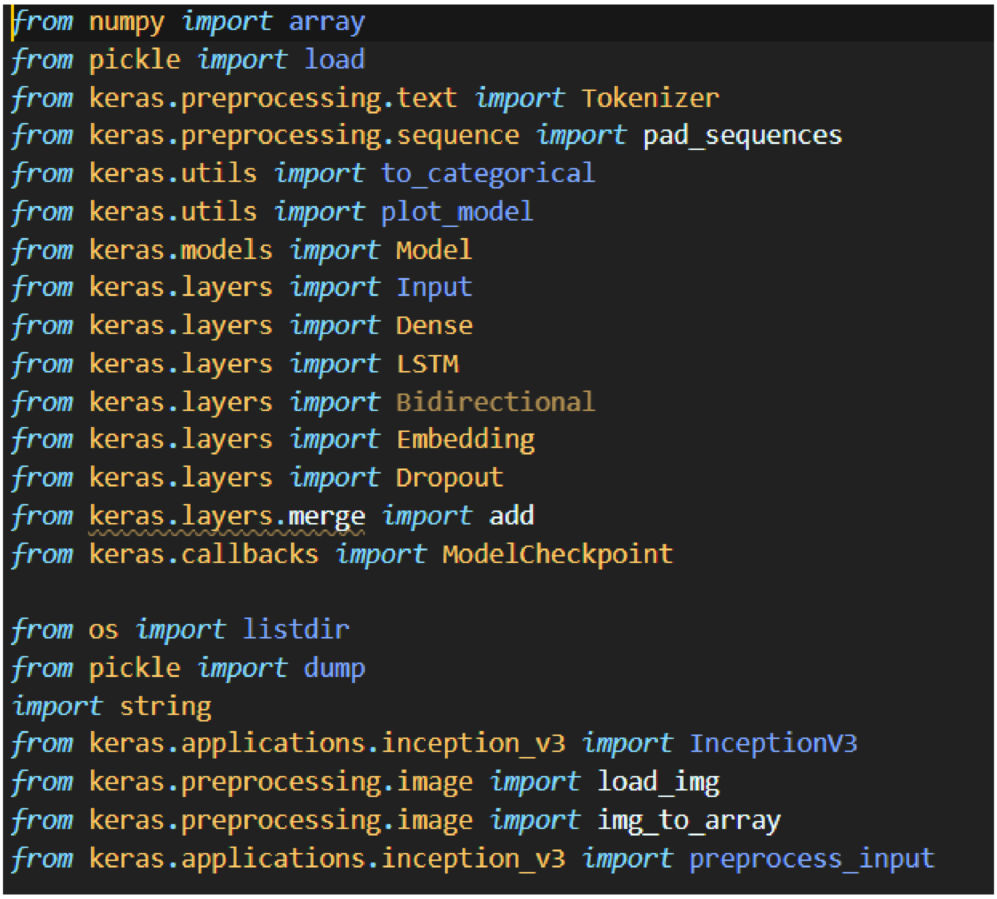
```
for i in range(epochs):
    batch_size = number_pics_per_batch = 5
    steps = len(train_descriptions)//number_pics_per_batch
    generator = data_generator(train_descriptions,train_image_extracted,max_length, vocab_size,number_pics_per_batch)
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    model.save('model_demo' + str(i) + '.h5')

Run Cell | Run Above | Debug Cell
#%%
import os
os.chdir(r"D:\Image captioning\coding\flickr")
"Manually saving the training loss for each epoch and plotting here"
flickr_trloss = pd.read_csv("train_loss_flickr.csv")

import matplotlib.pyplot as plt
plt.figure(figsize=[12,6])
plt.plot(flickr_trloss['Epoch'], flickr_trloss['train_loss'], 'o-', label='train_loss')
plt.legend()
plt.xlabel('Epoch', fontweight='bold', fontsize=12)
plt.ylabel('Train_loss', fontweight='bold', fontsize=12)
plt.title("Epoch vs train loss", fontweight='bold', fontsize=14)
```



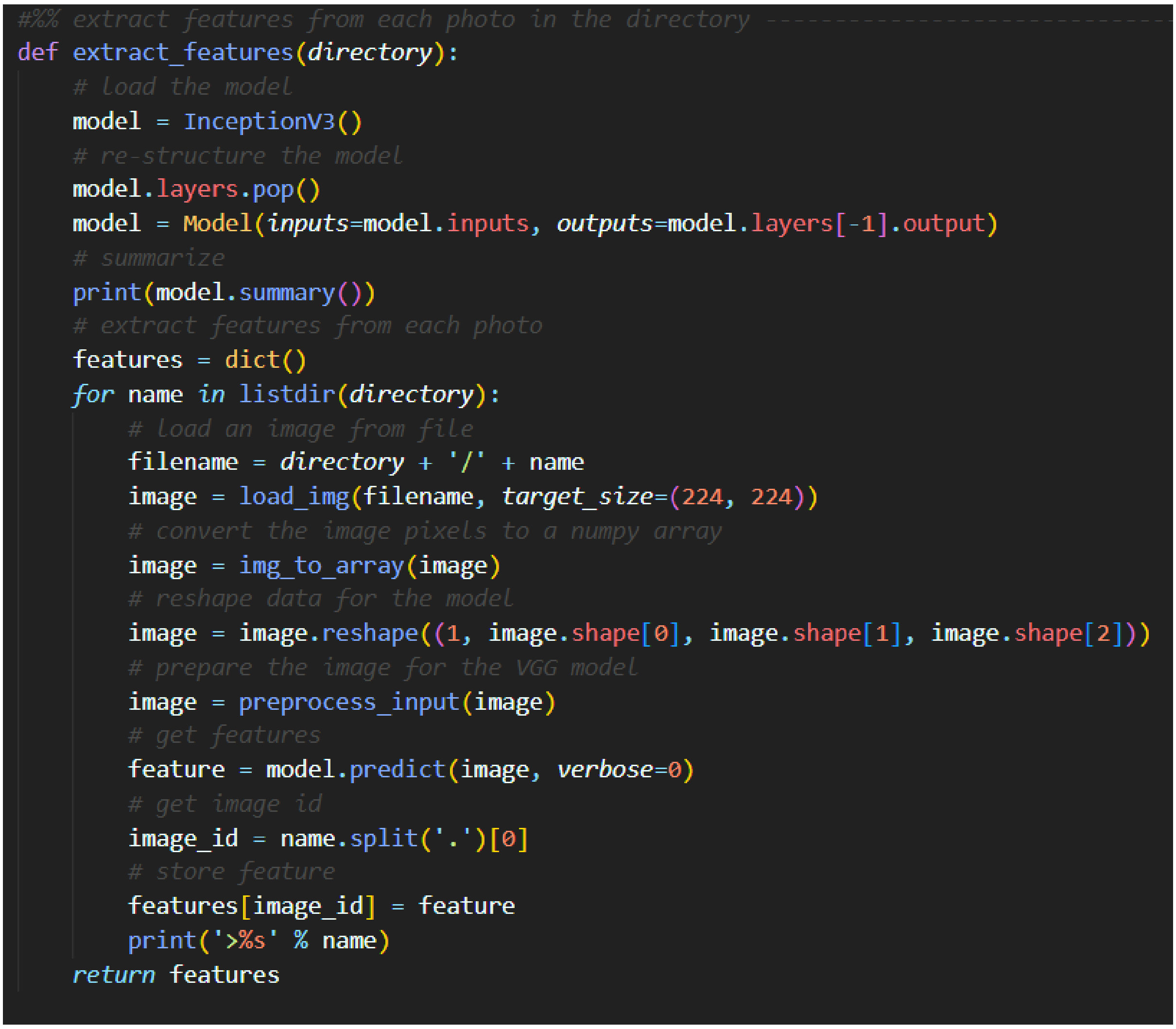
```python
# clean all the captions
descriptions = clean_descriptions(descriptions)

# save the file for further use
dump(descriptions,open("descriptions.pkl","wb"))


# prepare test image descriptions
with open("descriptions.pkl","rb") as f:
    descriptions = load(f)

#find pre-processed train captions from descriptions file and perform the encoding as
train_descriptions = dict()
with open(PATH+r"\Flickr_8k.trainImages.txt","r") as f:
    train_file = f.read()

k=0
l=0
for tid in train_file.split("\n"):
    train_id = tid.strip()
    if train_id in descriptions.copy():
        train_descriptions[train_id] = descriptions.copy()[train_id]
    else:
        l+=1
print('Total train_ids not found in descriptions:',l)

for key, train_desc_list in train_descriptions.items():
    for i in range(len(train_desc_list)):
        train_desc = train_desc_list[i]
        #store as string
        train_desc_list[i] =  'captionstart ' + ''.join(train_desc) + ' captionend'
```



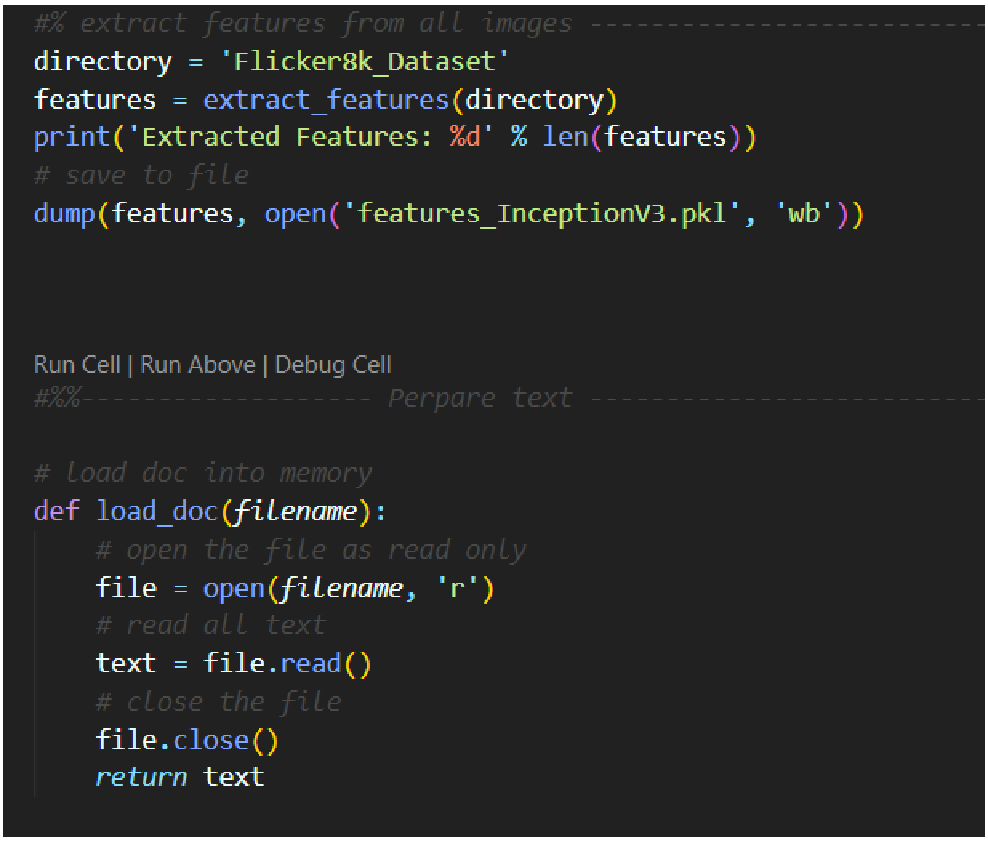
```
        # check if image_id is already present or not
        if image_id in descriptions:
            descriptions[image_id].append(image_desc)
        else:
            descriptions[image_id] = list()
            descriptions[image_id].append(image_desc)
    except Exception as e:
        print("Exception got :- \n",e)



def clean_descriptions(descriptions):
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            # tokenize
            desc = desc.split()
            # convert to lower case
            desc = [word.lower() for word in desc]
            # remove punctuation from each token
            desc = [w.translate(table) for w 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()]

            # store as string
            desc_list[i] =  ' '.join(desc)
    return descriptions
```



```python
from numpy import array
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Bidirectional
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint

from os import listdir
from pickle import dump
import string
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.inception_v3 import preprocess_input
```

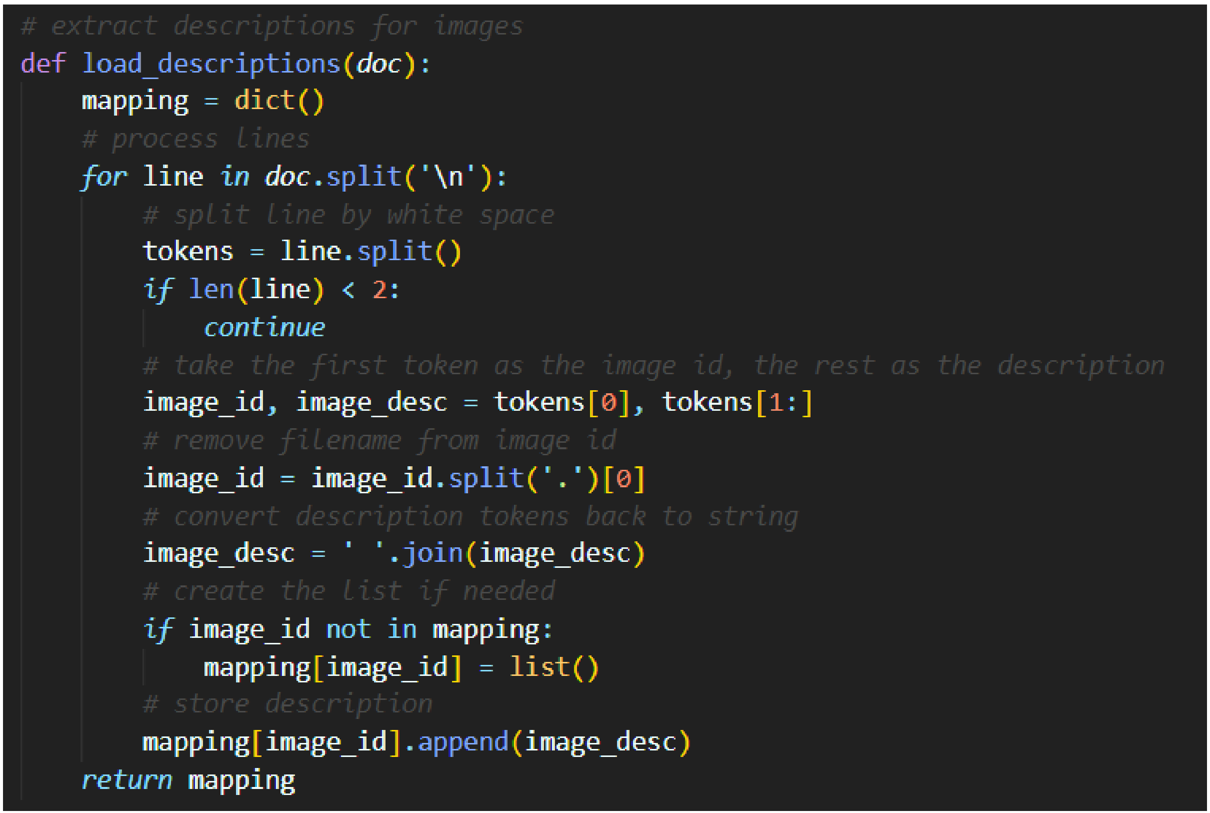


```python
#%% extract features from each photo in the directory ------------------------------
def extract_features(directory):
    # load the model
    model = InceptionV3()
    # re-structure the model
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    # summarize
    print(model.summary())
    # extract features from each photo
    features = dict()
    for name in listdir(directory):
        # load an image from file
        filename = directory + '/' + name
        image = load_img(filename, target_size=(224, 224))
        # convert the image pixels to a numpy array
        image = img_to_array(image)
        # reshape data for the model
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        # prepare the image for the VGG model
        image = preprocess_input(image)
        # get features
        feature = model.predict(image, verbose=0)
        # get image id
        image_id = name.split('.')[0]
        # store feature
        features[image_id] = feature
        print('>%s' % name)
    return features
```

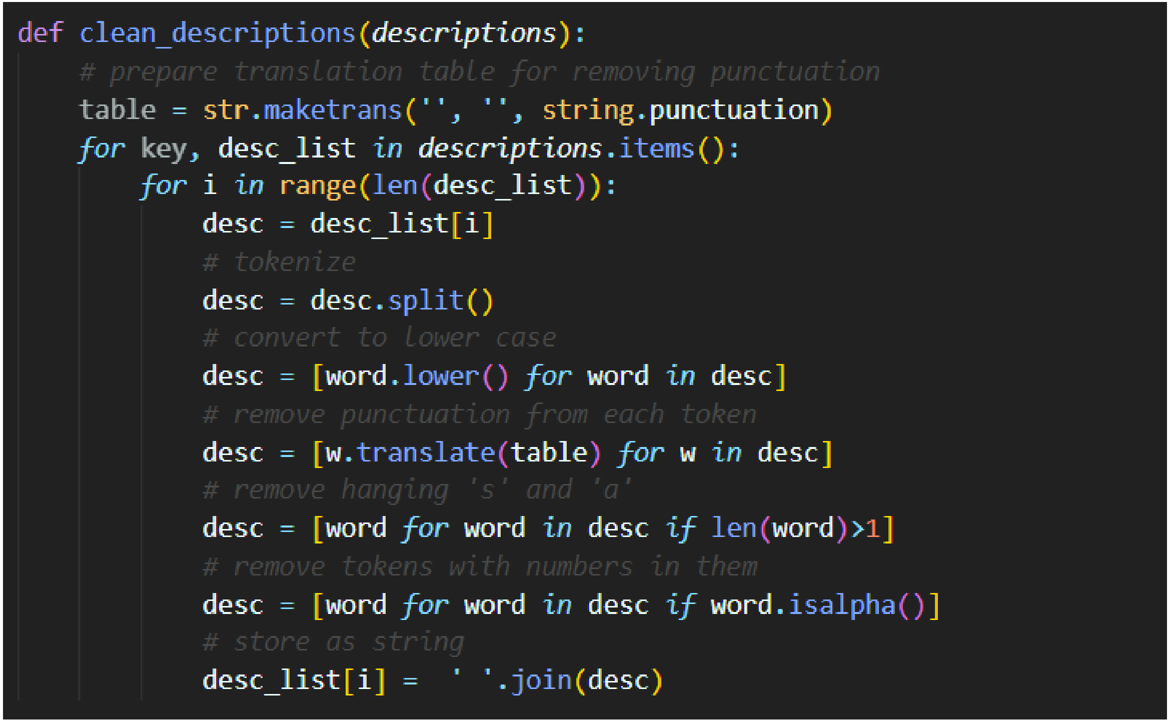


```
#% extract features from all images -------------------------
directory = 'Flicker8k_Dataset'
features = extract_features(directory)
print('Extracted Features: %d' % len(features))
# save to file
dump(features, open('features_InceptionV3.pkl', 'wb'))
```

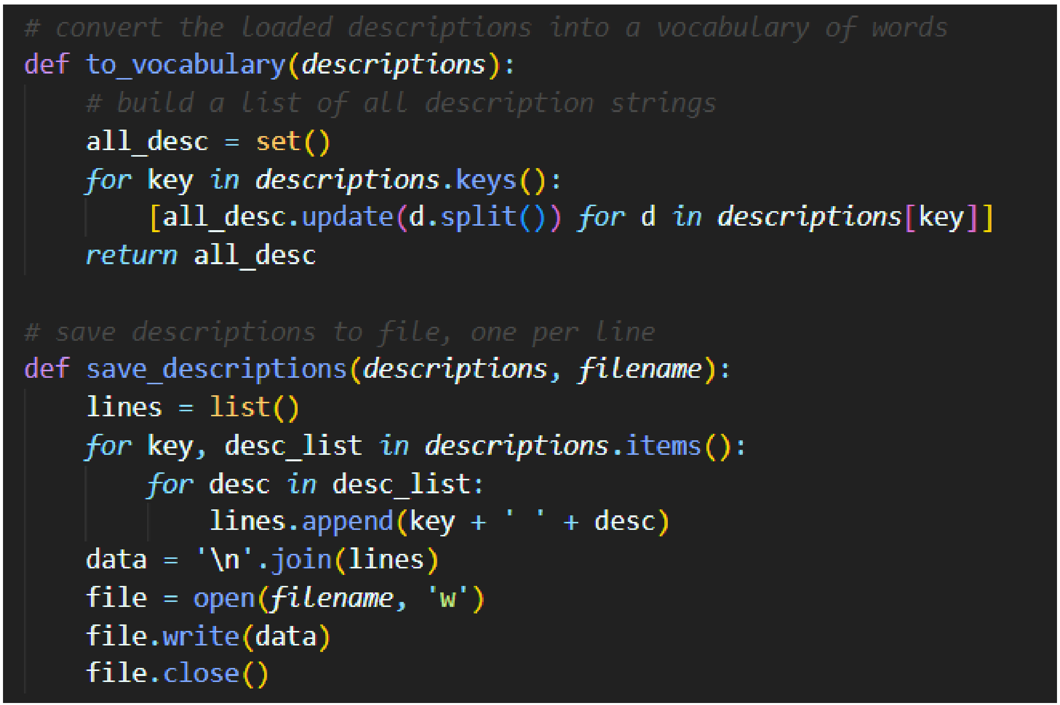
```
#%%------------------- Perpare text -------------------------

# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text
```



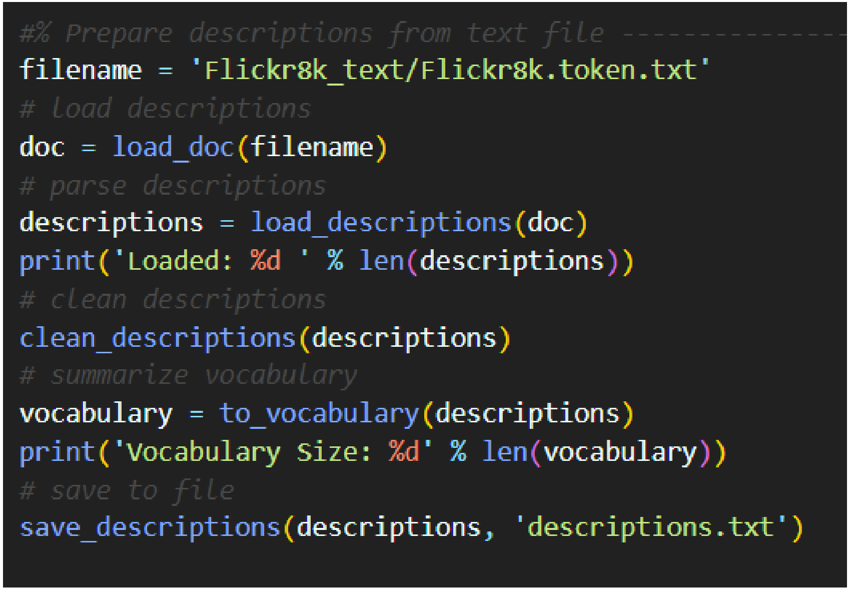
```python
# extract descriptions for images
def load_descriptions(doc):
    mapping = dict()
    # process lines
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        if len(line) < 2:
            continue
        # take the first token as the image id, the rest as the description
        image_id, image_desc = tokens[0], tokens[1:]
        # remove filename from image id
        image_id = image_id.split('.')[0]
        # convert description tokens back to string
        image_desc = ' '.join(image_desc)
        # create the list if needed
        if image_id not in mapping:
            mapping[image_id] = list()
        # store description
        mapping[image_id].append(image_desc)
    return mapping
```



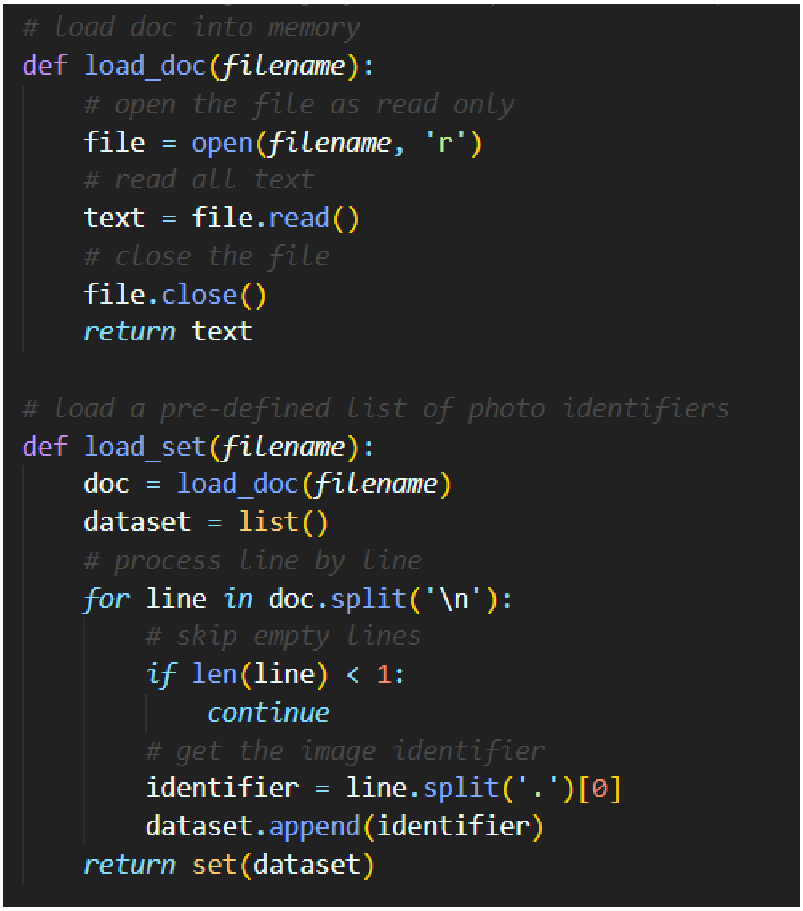
```python
def clean_descriptions(descriptions):
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            # tokenize
            desc = desc.split()
            # convert to lower case
            desc = [word.lower() for word in desc]
            # remove punctuation from each token
            desc = [w.translate(table) for w 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()]
            # store as string
            desc_list[i] =  ' '.join(desc)
```



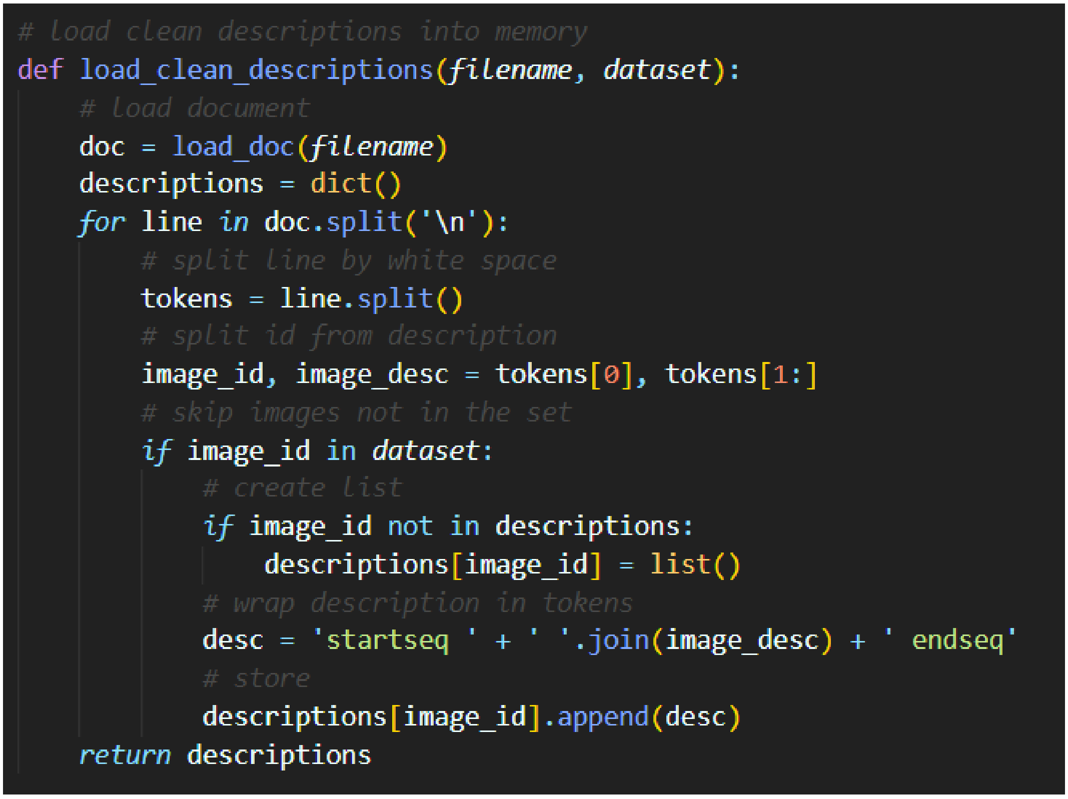
```
# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    # build a list of all description strings
    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc

# save descriptions to file, one per line
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + ' ' + desc)
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()
```



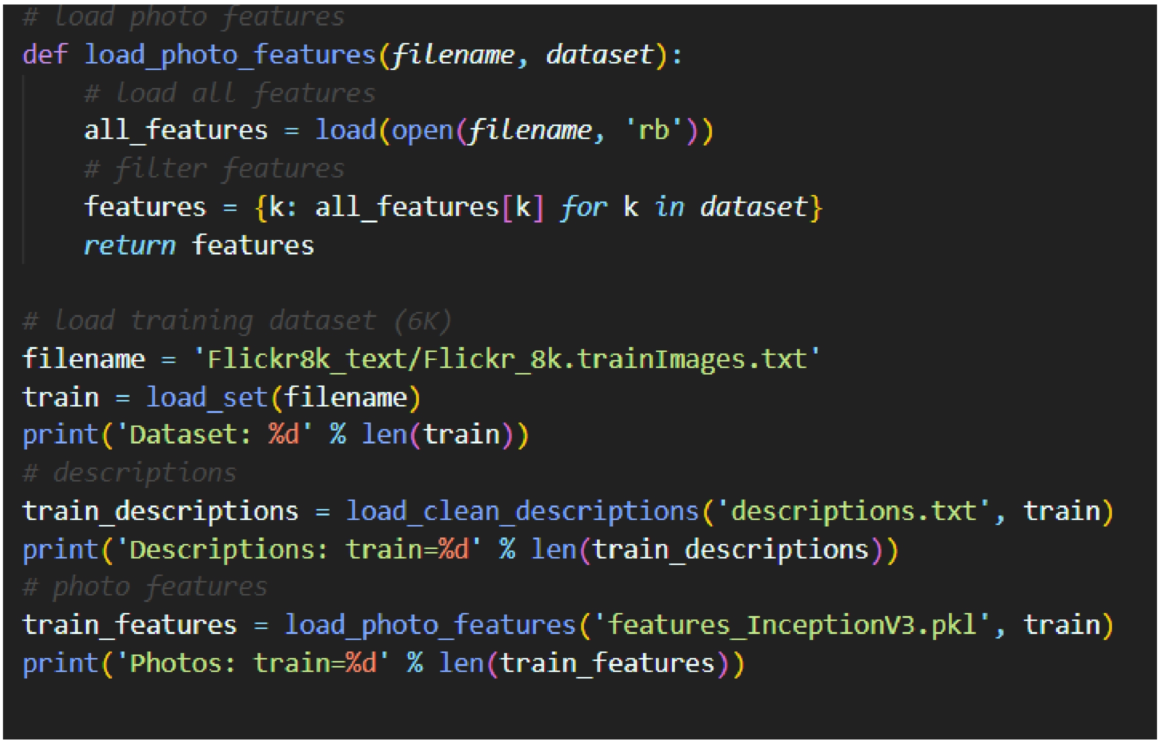
```
#% Prepare descriptions from text file ---------------
filename = 'Flickr8k_text/Flickr8k.token.txt'
# load descriptions
doc = load_doc(filename)
# parse descriptions
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))
# clean descriptions
clean_descriptions(descriptions)
# summarize vocabulary
vocabulary = to_vocabulary(descriptions)
print('Vocabulary Size: %d' % len(vocabulary))
# save to file
save_descriptions(descriptions, 'descriptions.txt')
```



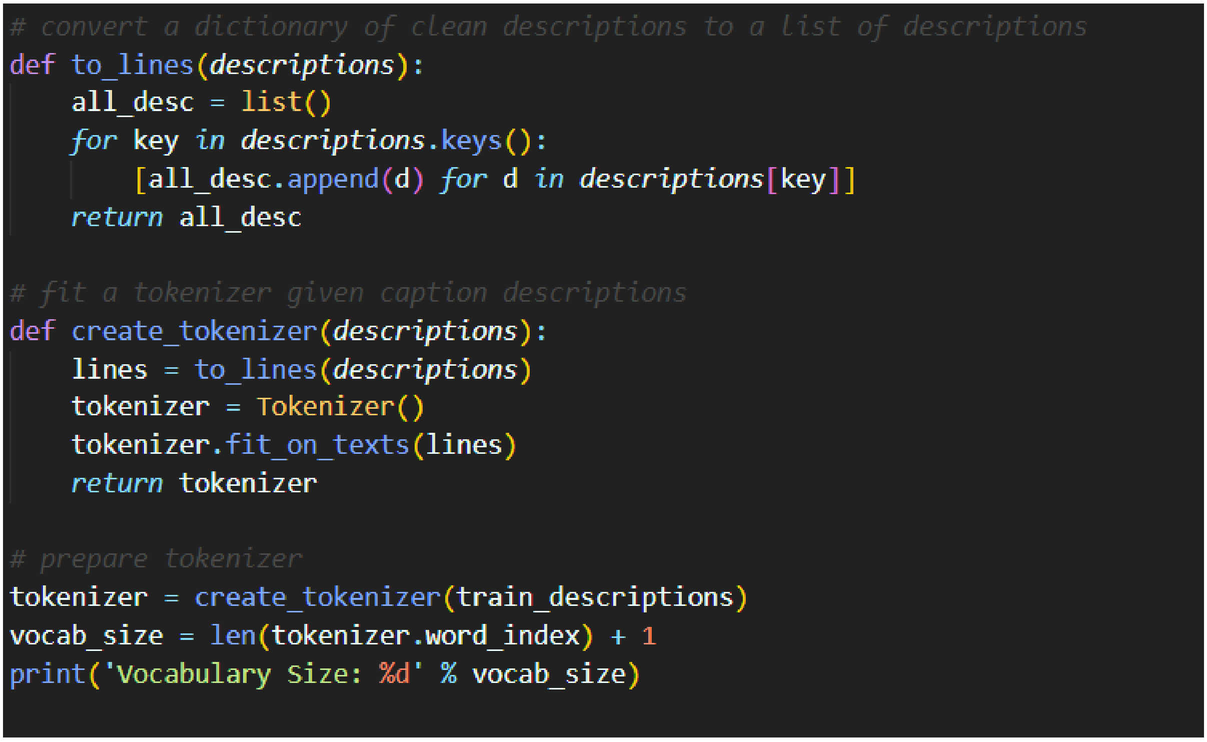
```python
# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text

# load a pre-defined list of photo identifiers
def load_set(filename):
    doc = load_doc(filename)
    dataset = list()
    # process line by line
    for line in doc.split('\n'):
        # skip empty lines
        if len(line) < 1:
            continue
        # get the image identifier
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)
```



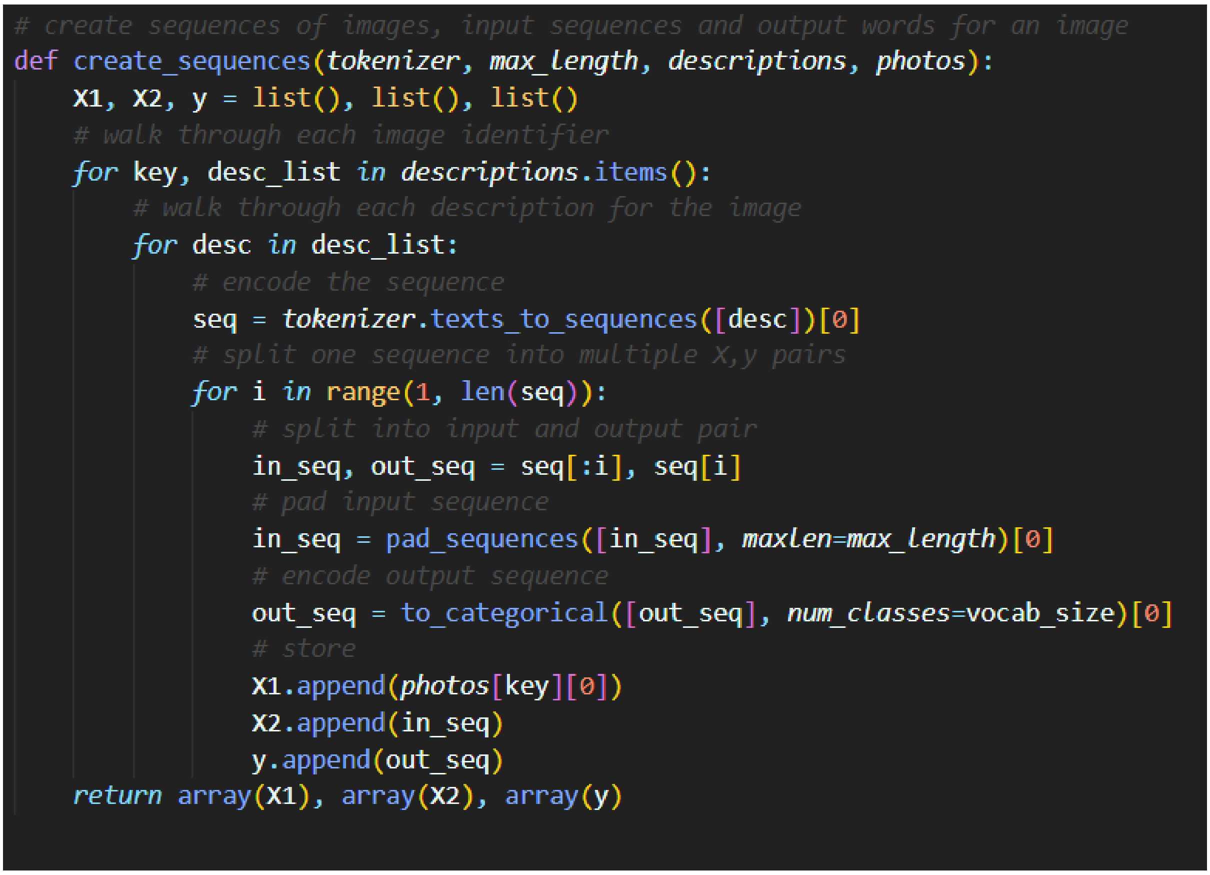
```python
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    # load document
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        # split id from description
        image_id, image_desc = tokens[0], tokens[1:]
        # skip images not in the set
        if image_id in dataset:
            # create list
            if image_id not in descriptions:
                descriptions[image_id] = list()
            # wrap description in tokens
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            # store
            descriptions[image_id].append(desc)
    return descriptions
```



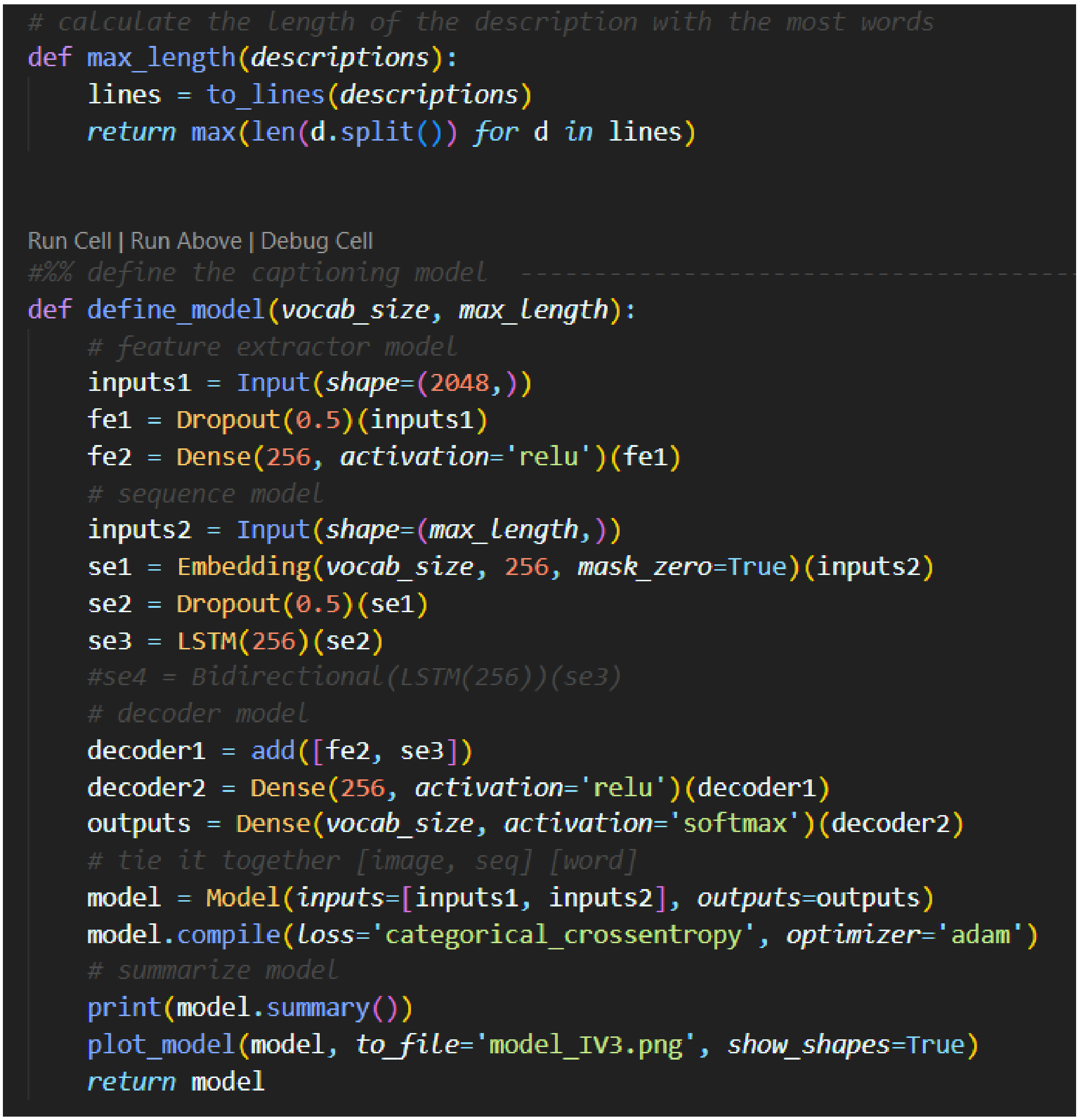
```
# load photo features
def load_photo_features(filename, dataset):
    # load all features
    all_features = load(open(filename, 'rb'))
    # filter features
    features = {k: all_features[k] for k in dataset}
    return features

# load training dataset (6K)
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# photo features
train_features = load_photo_features('features_InceptionV3.pkl', train)
print('Photos: train=%d' % len(train_features))
```



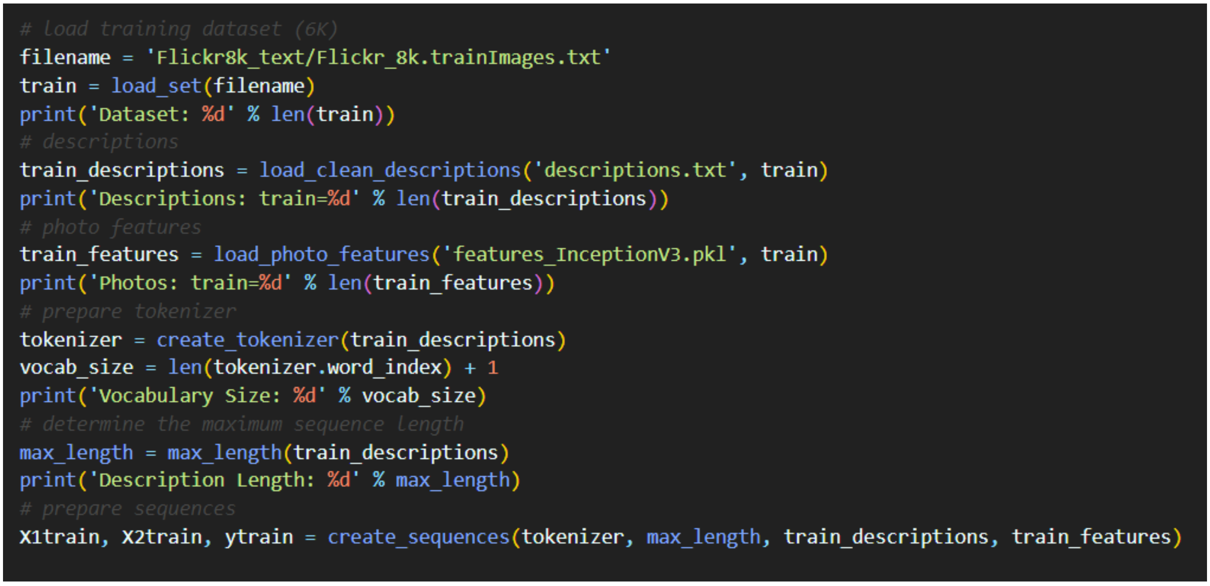
```
# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
```



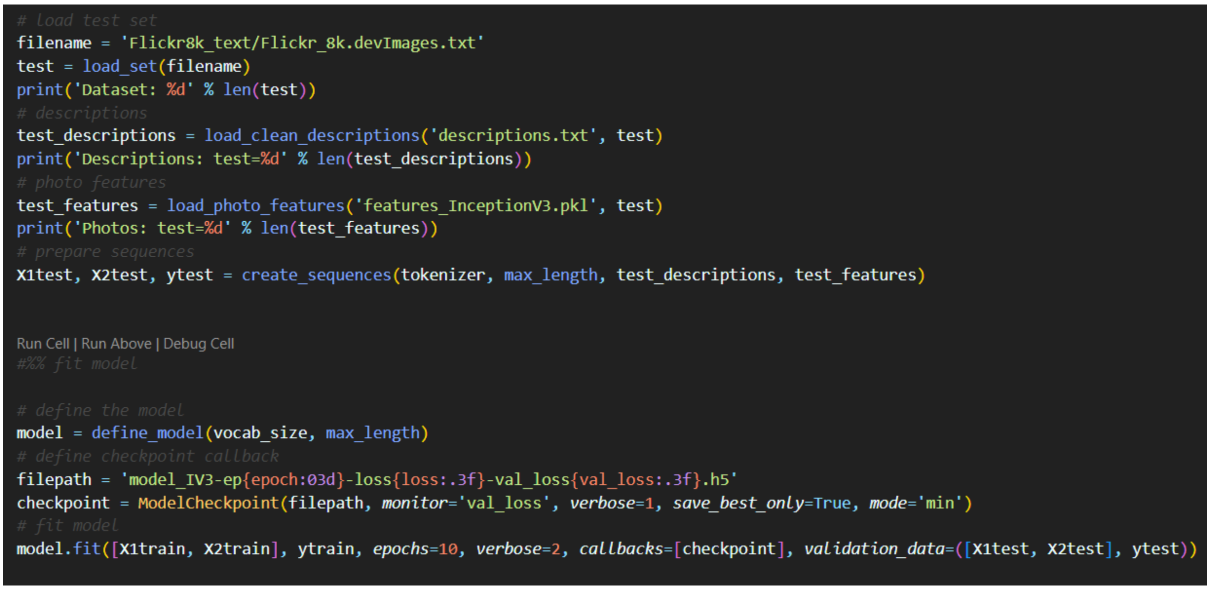
```python
# create sequences of images, input sequences and output words for an image
def create_sequences(tokenizer, max_length, descriptions, photos):
    X1, X2, y = list(), list(), list()
    # walk through each image identifier
    for key, desc_list in descriptions.items():
        # walk through 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(photos[key][0])
                X2.append(in_seq)
                y.append(out_seq)
    return array(X1), array(X2), array(y)
```



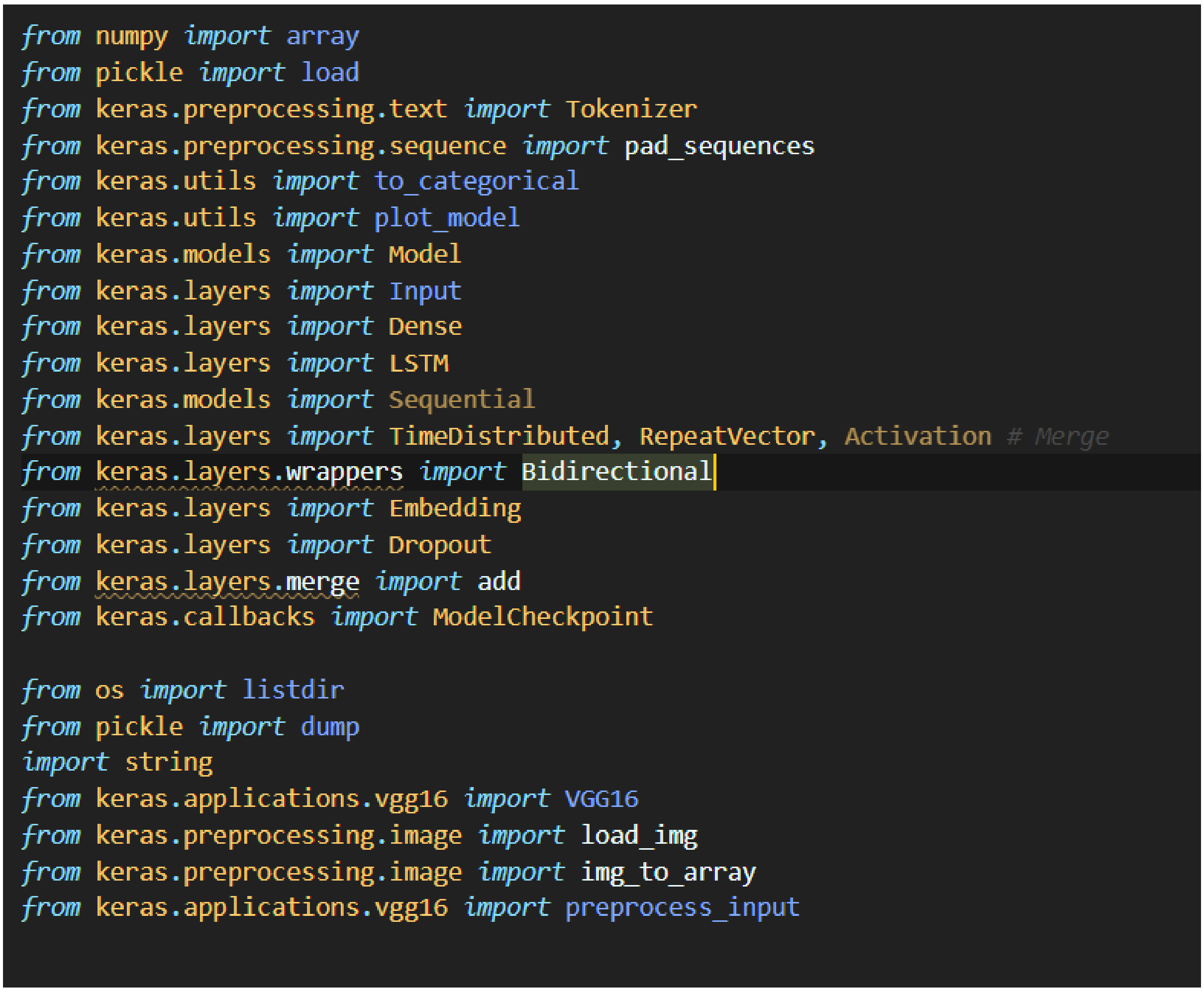
```python
# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)


Run Cell | Run Above | Debug Cell
#%% define the captioning model  -----------------------------------
def define_model(vocab_size, max_length):
    # feature extractor model
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    # 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)
    #se4 = Bidirectional(LSTM(256))(se3)
    # decoder model
    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_IV3.png', show_shapes=True)
    return model
```



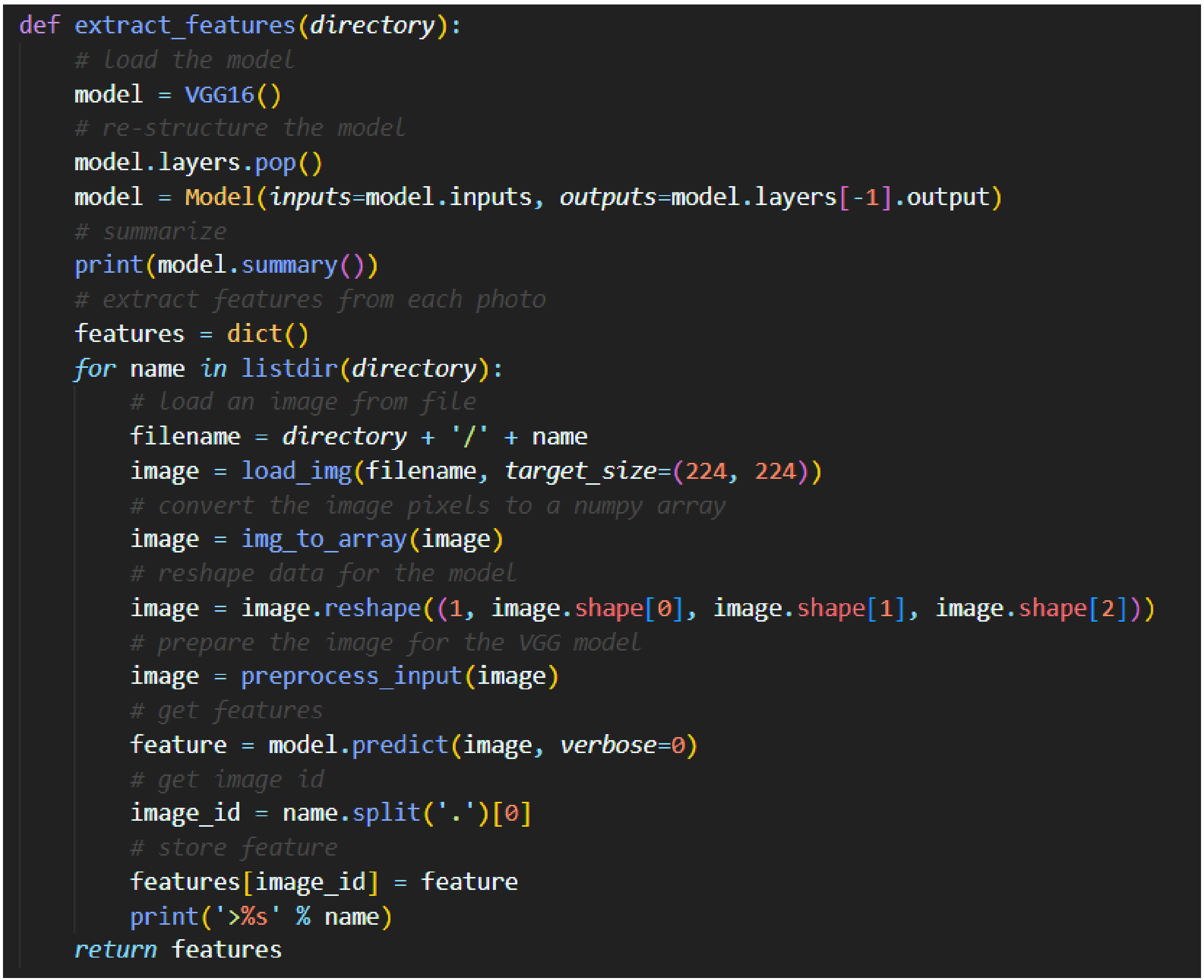
```python
# load training dataset (6K)
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# photo features
train_features = load_photo_features('features_InceptionV3.pkl', train)
print('Photos: train=%d' % len(train_features))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
# prepare sequences
X1train, X2train, ytrain = create_sequences(tokenizer, max_length, train_descriptions, train_features)
```



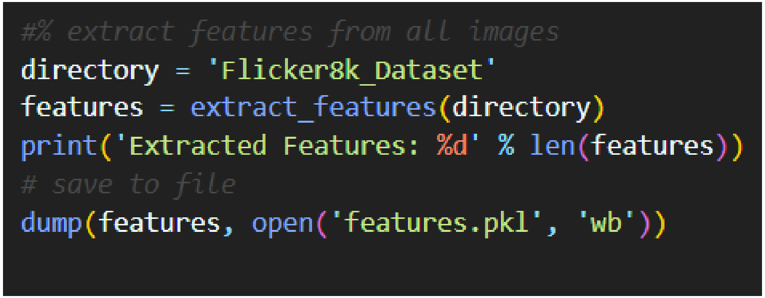
```
# load test set
filename = 'Flickr8k_text/Flickr_8k.devImages.txt'
test = load_set(filename)
print('Dataset: %d' % len(test))
# descriptions
test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))
# photo features
test_features = load_photo_features('features_InceptionV3.pkl', test)
print('Photos: test=%d' % len(test_features))
# prepare sequences
X1test, X2test, ytest = create_sequences(tokenizer, max_length, test_descriptions, test_features)


Run Cell | Run Above | Debug Cell
#%% fit model

# define the model
model = define_model(vocab_size, max_length)
# define checkpoint callback
filepath = 'model_IV3-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
# fit model
model.fit([X1train, X2train], ytrain, epochs=10, verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest))
```



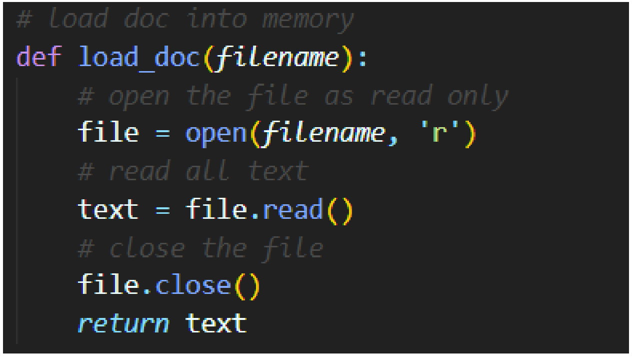
```
from numpy import array
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
from keras.models import Sequential
from keras.layers import TimeDistributed, RepeatVector, Activation # Merge
from keras.layers.wrappers import Bidirectional
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint

from os import listdir
from pickle import dump
import string
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
```



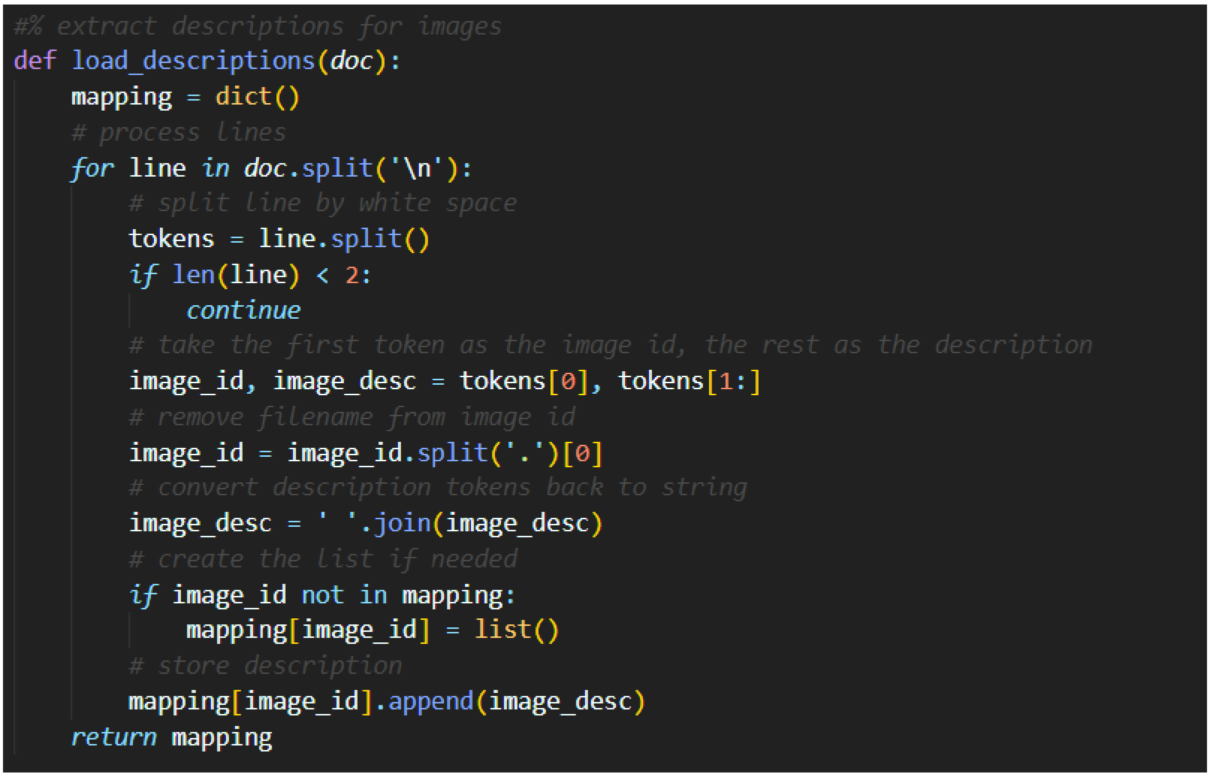
```python
def extract_features(directory):
    # load the model
    model = VGG16()
    # re-structure the model
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    # summarize
    print(model.summary())
    # extract features from each photo
    features = dict()
    for name in listdir(directory):
        # load an image from file
        filename = directory + '/' + name
        image = load_img(filename, target_size=(224, 224))
        # convert the image pixels to a numpy array
        image = img_to_array(image)
        # reshape data for the model
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        # prepare the image for the VGG model
        image = preprocess_input(image)
        # get features
        feature = model.predict(image, verbose=0)
        # get image id
        image_id = name.split('.')[0]
        # store feature
        features[image_id] = feature
        print('>%s' % name)
    return features
```



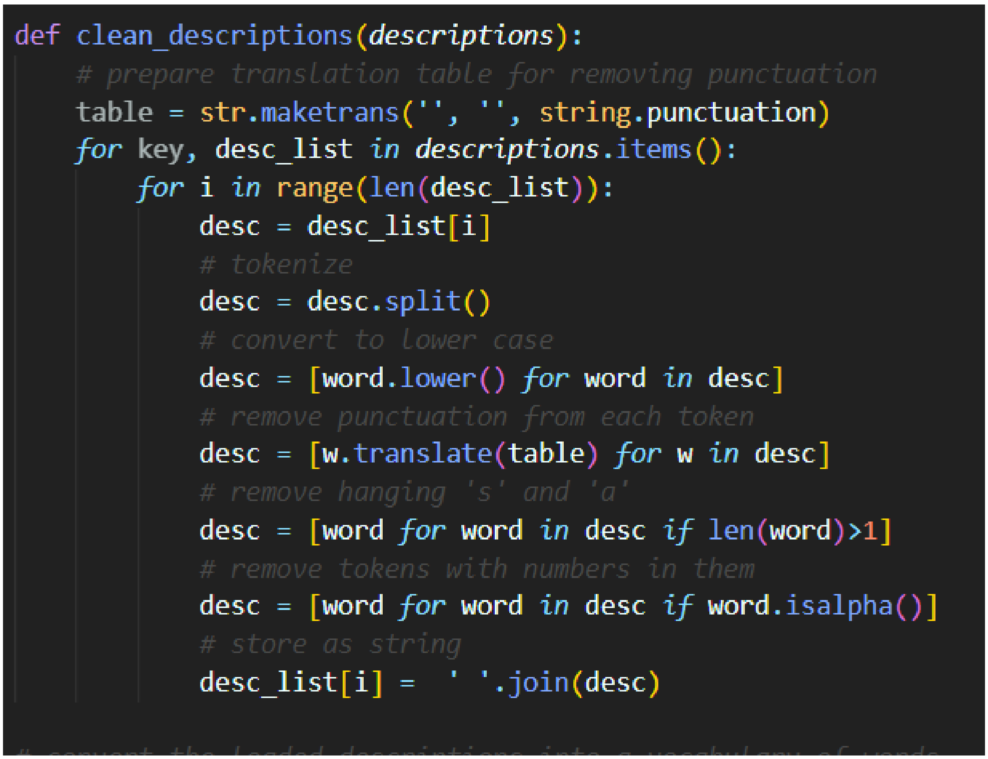
```python
#% extract features from all images
directory = 'Flicker8k_Dataset'
features = extract_features(directory)
print('Extracted Features: %d' % len(features))
# save to file
dump(features, open('features.pkl', 'wb'))
```



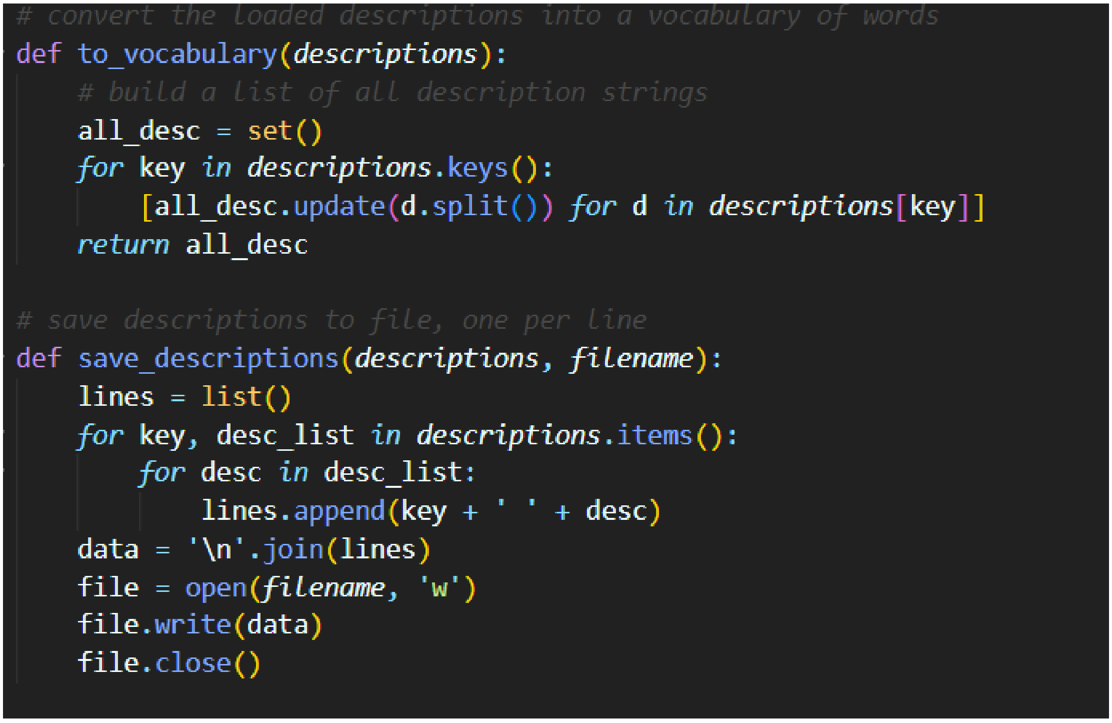
```python
# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text
```



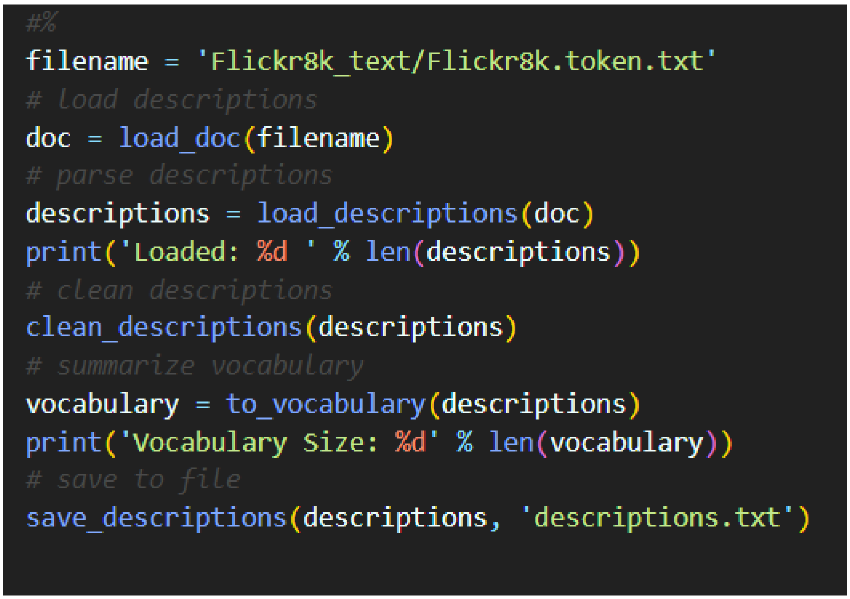
```python
#% extract descriptions for images
def load_descriptions(doc):
    mapping = dict()
    # process lines
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        if len(line) < 2:
            continue
        # take the first token as the image id, the rest as the description
        image_id, image_desc = tokens[0], tokens[1:]
        # remove filename from image id
        image_id = image_id.split('.')[0]
        # convert description tokens back to string
        image_desc = ' '.join(image_desc)
        # create the list if needed
        if image_id not in mapping:
            mapping[image_id] = list()
        # store description
        mapping[image_id].append(image_desc)
    return mapping
```



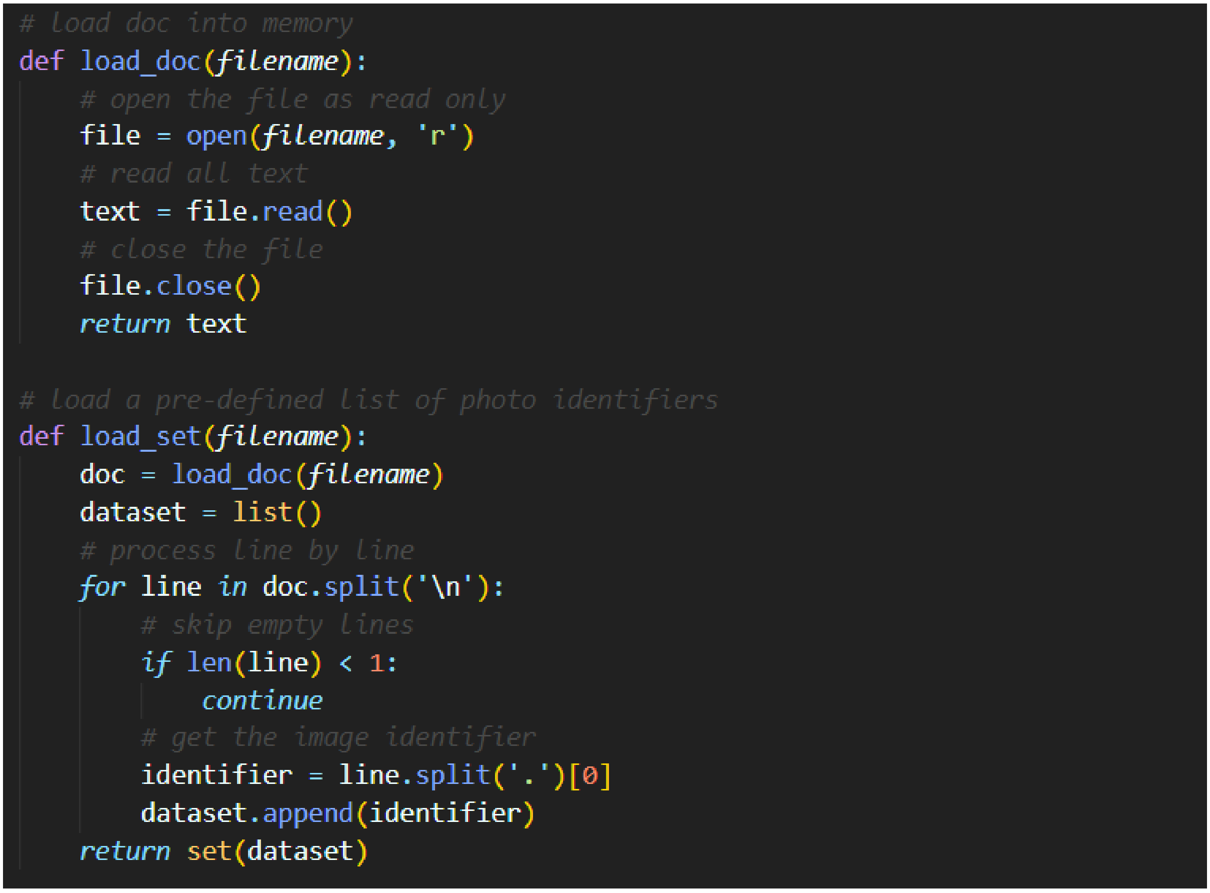
```python
def clean_descriptions(descriptions):
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            # tokenize
            desc = desc.split()
            # convert to lower case
            desc = [word.lower() for word in desc]
            # remove punctuation from each token
            desc = [w.translate(table) for w 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()]
            # store as string
            desc_list[i] =  ' '.join(desc)
```



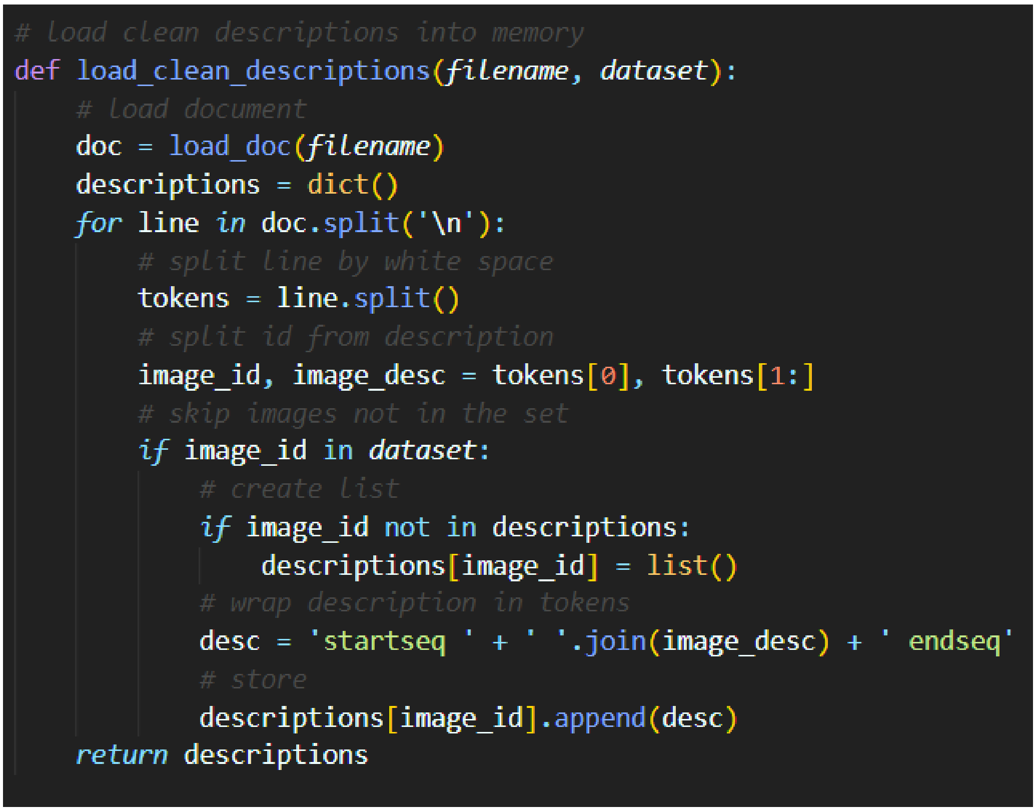
```python
# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    # build a list of all description strings
    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc

# save descriptions to file, one per line
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + ' ' + desc)
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()
```



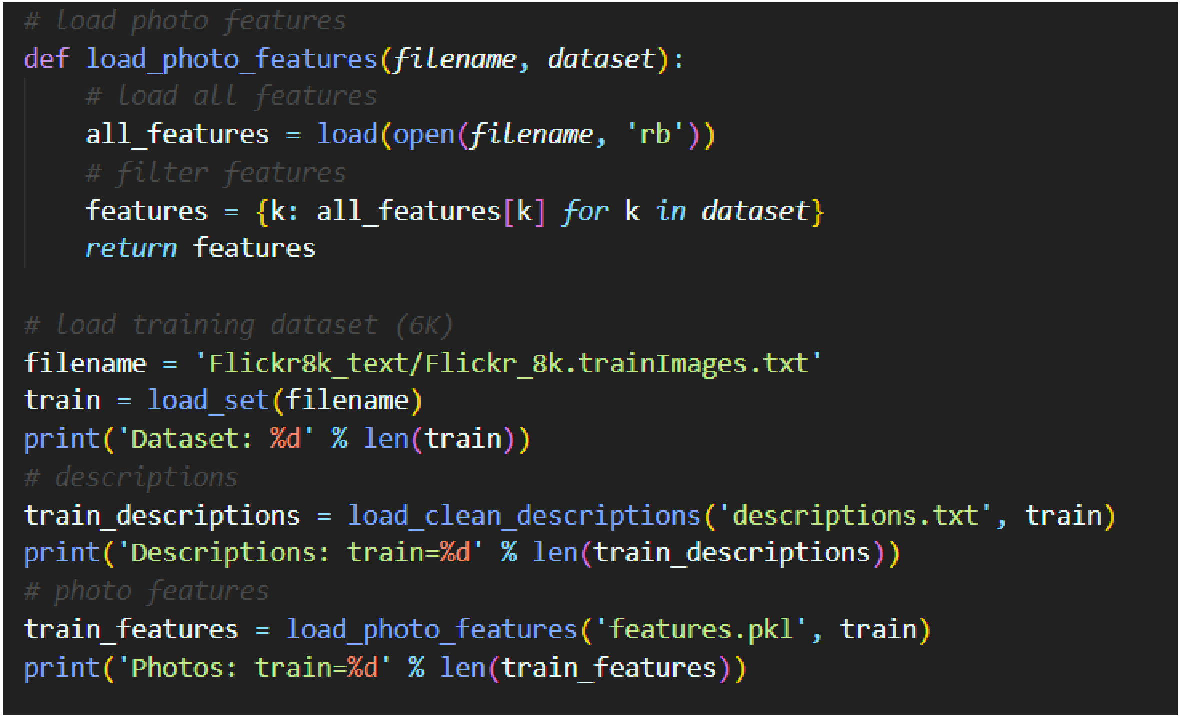
```
#%
filename = 'Flickr8k_text/Flickr8k.token.txt'
# load descriptions
doc = load_doc(filename)
# parse descriptions
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))
# clean descriptions
clean_descriptions(descriptions)
# summarize vocabulary
vocabulary = to_vocabulary(descriptions)
print('Vocabulary Size: %d' % len(vocabulary))
# save to file
save_descriptions(descriptions, 'descriptions.txt')
```



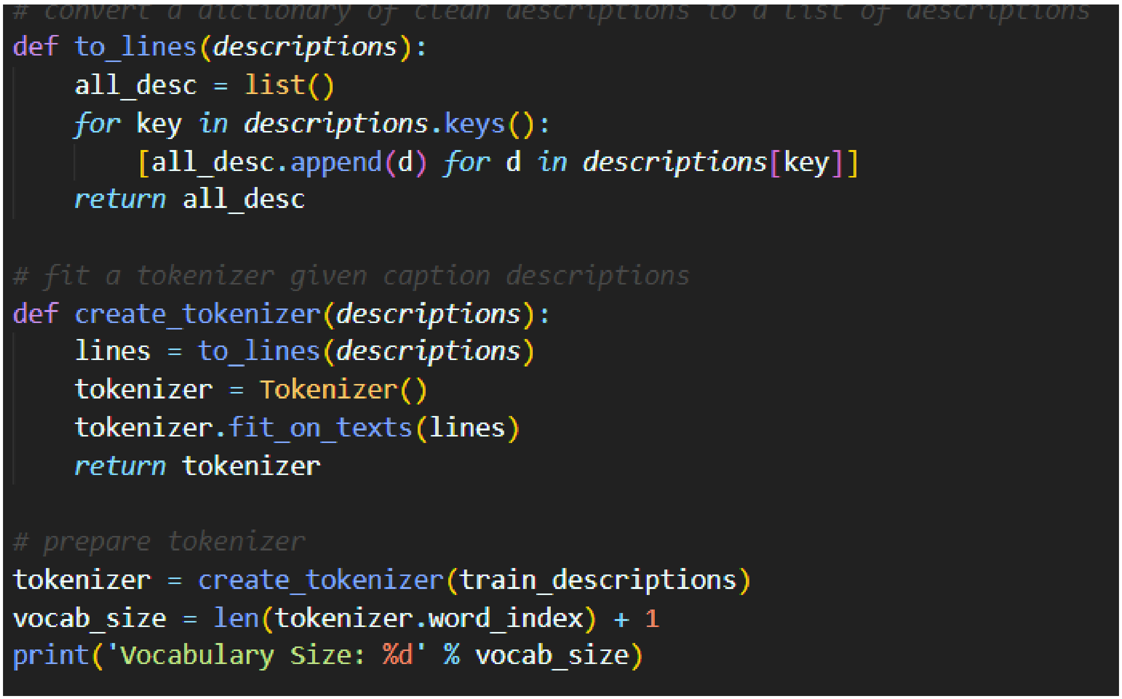
```
# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text

# load a pre-defined list of photo identifiers
def load_set(filename):
    doc = load_doc(filename)
    dataset = list()
    # process line by line
    for line in doc.split('\n'):
        # skip empty lines
        if len(line) < 1:
            continue
        # get the image identifier
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)
```



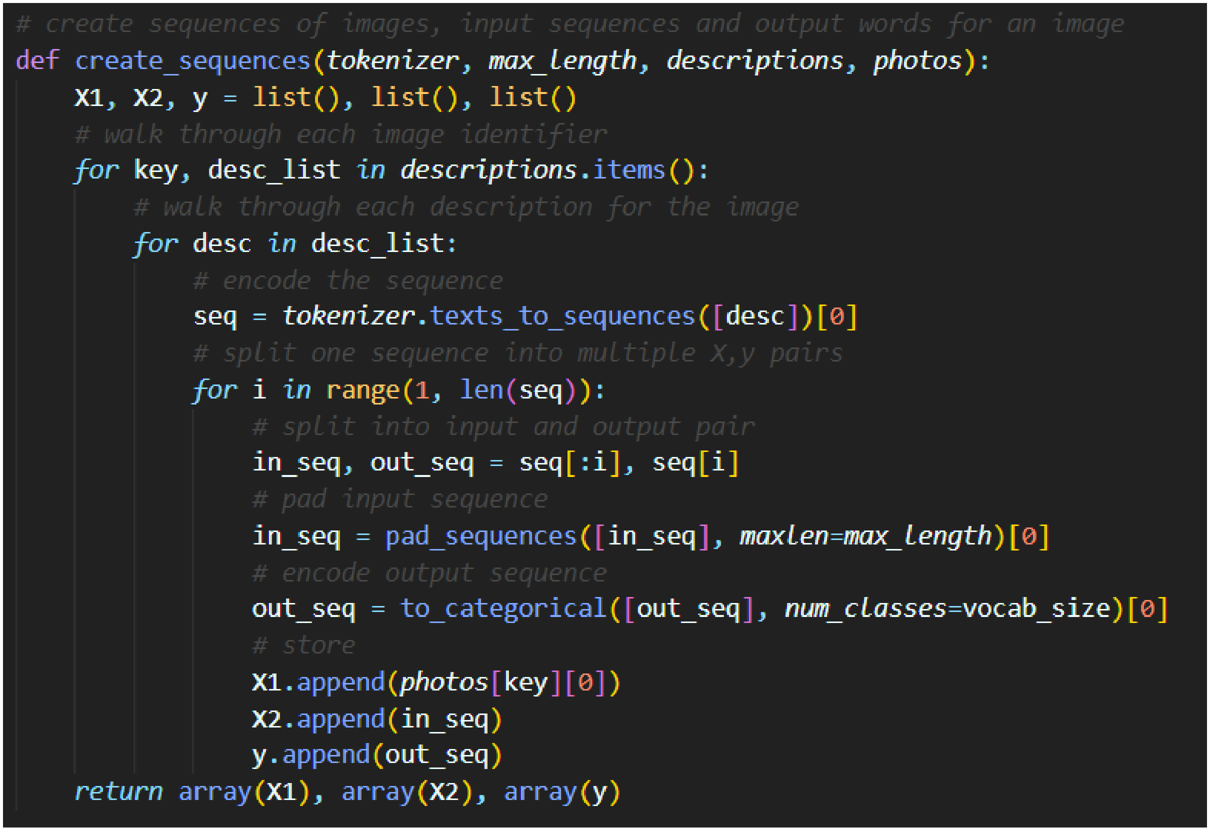
```python
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    # load document
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        # split id from description
        image_id, image_desc = tokens[0], tokens[1:]
        # skip images not in the set
        if image_id in dataset:
            # create list
            if image_id not in descriptions:
                descriptions[image_id] = list()
            # wrap description in tokens
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            # store
            descriptions[image_id].append(desc)
    return descriptions
```



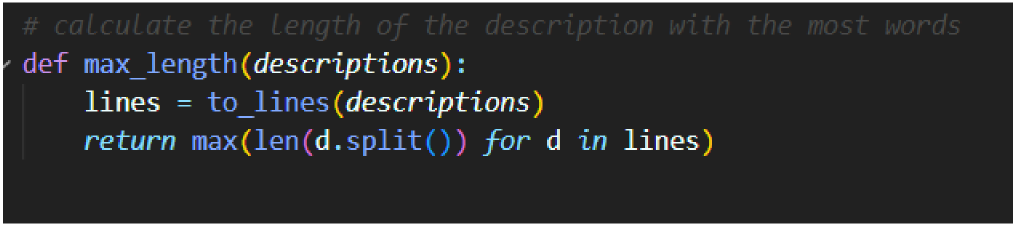
```python
# load photo features
def load_photo_features(filename, dataset):
    # load all features
    all_features = load(open(filename, 'rb'))
    # filter features
    features = {k: all_features[k] for k in dataset}
    return features

# load training dataset (6K)
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# photo features
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
```



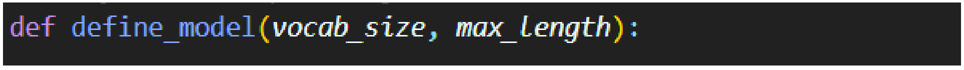
```python
# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
```



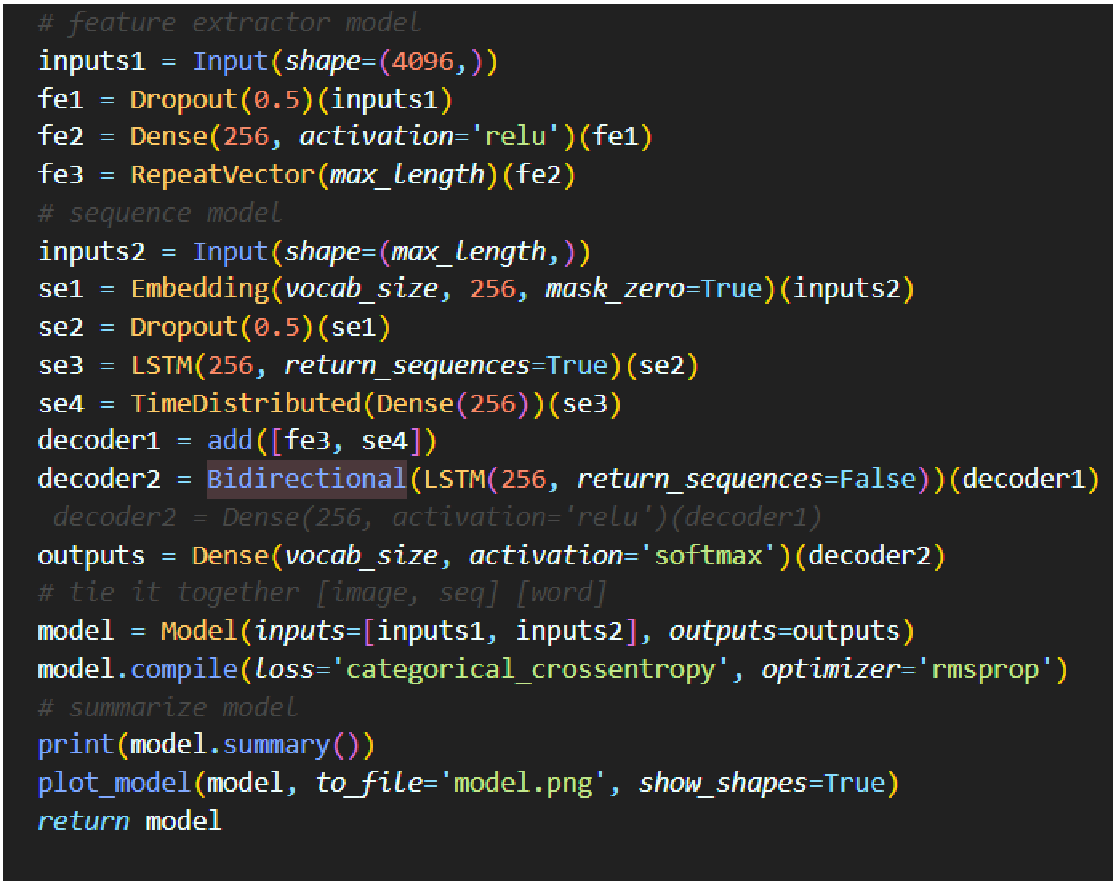
```python
# create sequences of images, input sequences and output words for an image
def create_sequences(tokenizer, max_length, descriptions, photos):
    X1, X2, y = list(), list(), list()
    # walk through each image identifier
    for key, desc_list in descriptions.items():
        # walk through 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(photos[key][0])
                X2.append(in_seq)
                y.append(out_seq)
    return array(X1), array(X2), array(y)
```



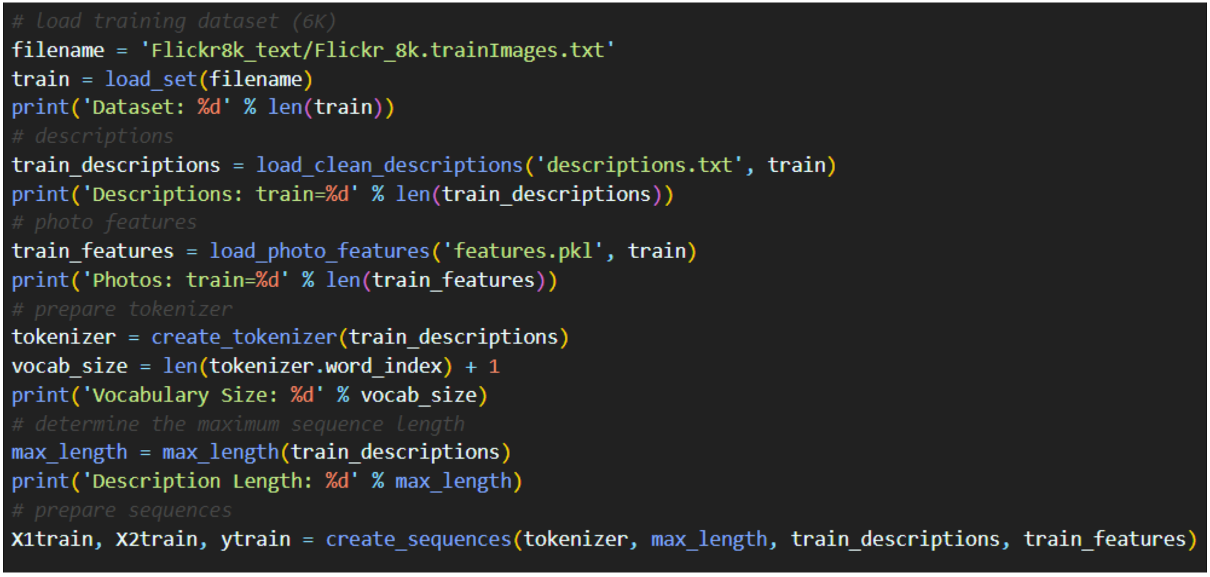
```python
# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)
```



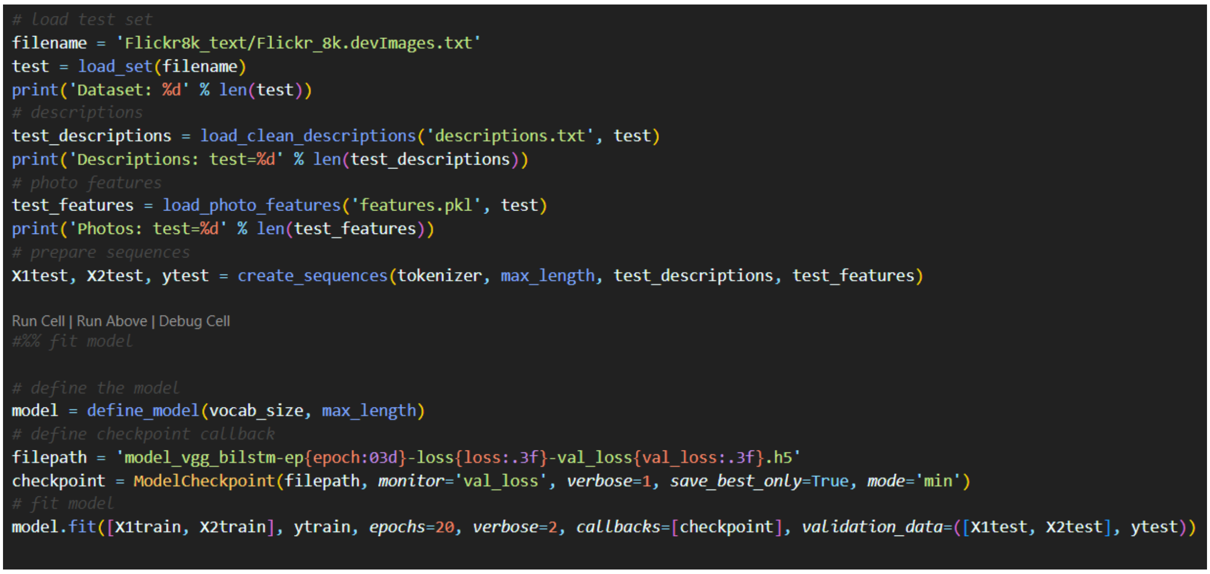
```python
def define_model(vocab_size, max_length):
```



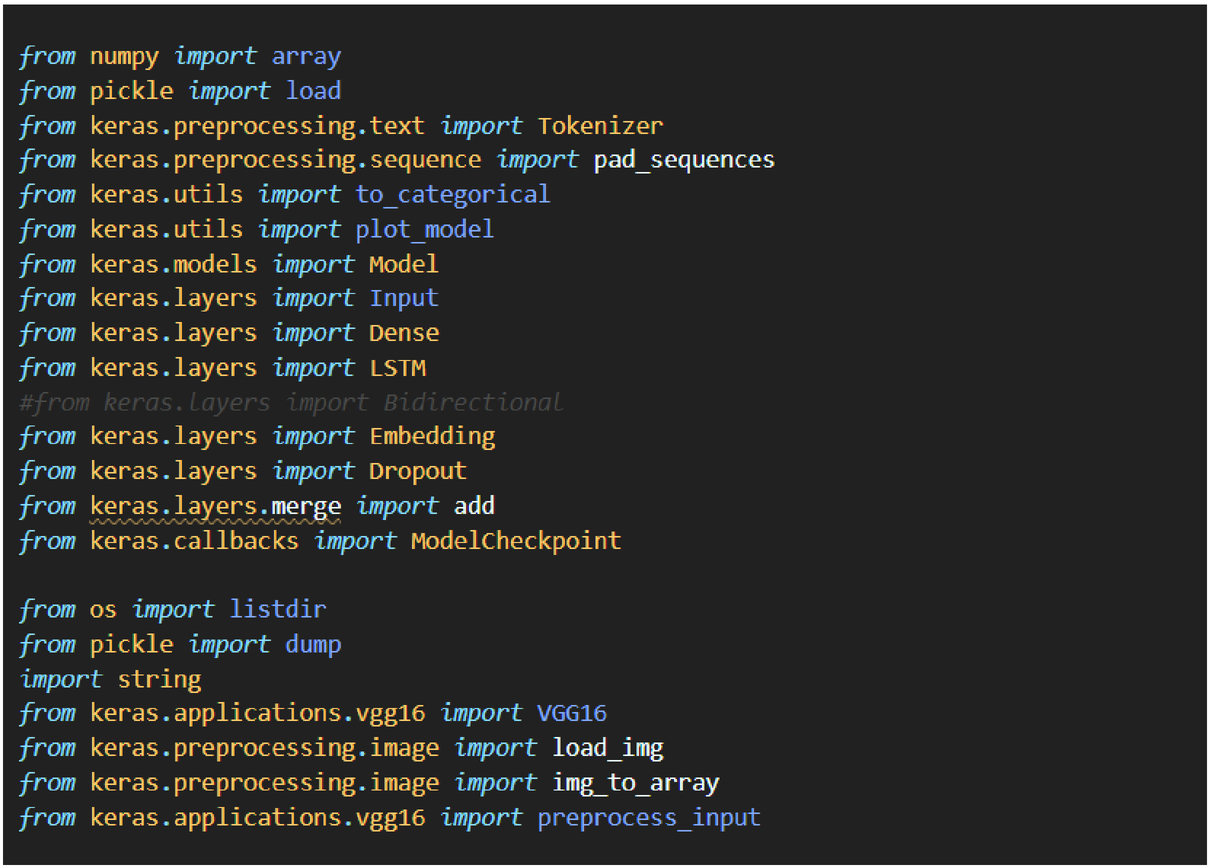
```
# feature extractor model
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
fe3 = RepeatVector(max_length)(fe2)
# sequence model
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256, return_sequences=True)(se2)
se4 = TimeDistributed(Dense(256))(se3)
decoder1 = add([fe3, se4])
decoder2 = Bidirectional(LSTM(256, return_sequences=False))(decoder1)
 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='rmsprop')
# summarize model
print(model.summary())
plot_model(model, to_file='model.png', show_shapes=True)
return model
```



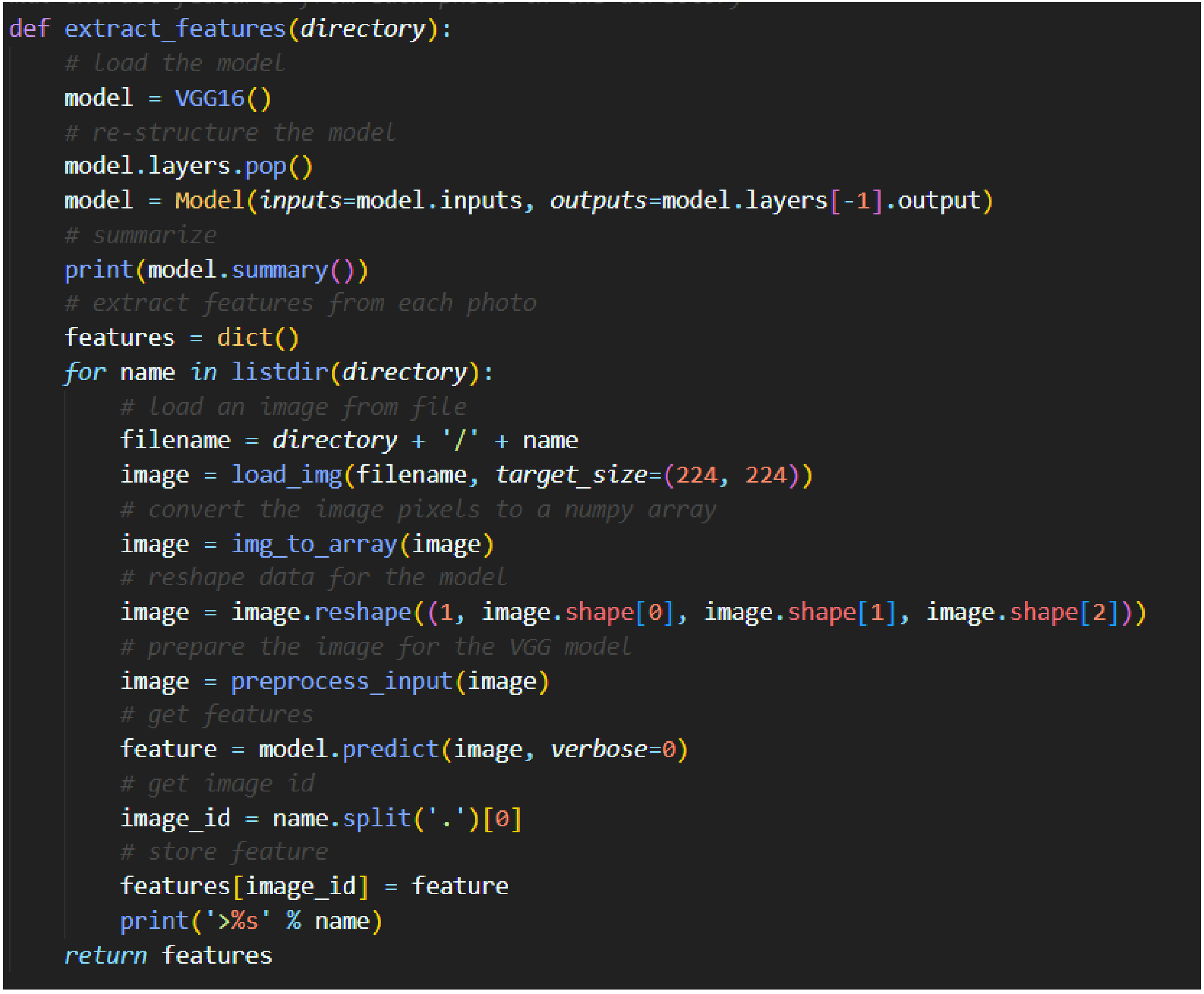
```
# load training dataset (6K)
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# photo features
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
# prepare sequences
X1train, X2train, ytrain = create_sequences(tokenizer, max_length, train_descriptions, train_features)
```



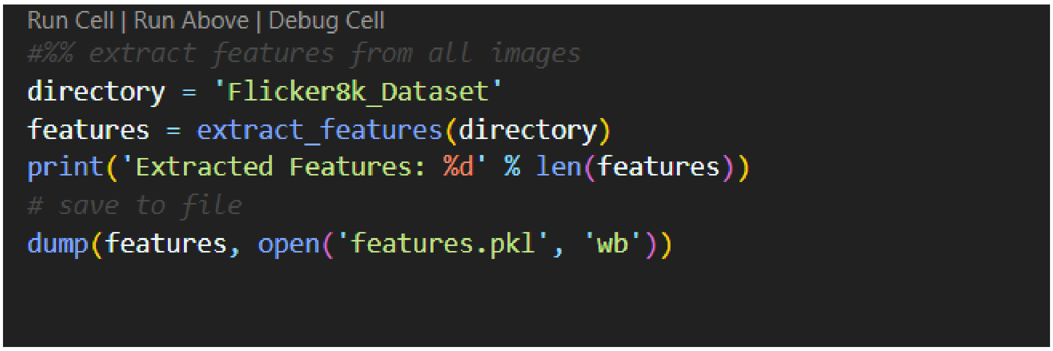
```
# load test set
filename = 'Flickr8k_text/Flickr_8k.devImages.txt'
test = load_set(filename)
print('Dataset: %d' % len(test))
# descriptions
test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))
# photo features
test_features = load_photo_features('features.pkl', test)
print('Photos: test=%d' % len(test_features))
# prepare sequences
X1test, X2test, ytest = create_sequences(tokenizer, max_length, test_descriptions, test_features)

Run Cell | Run Above | Debug Cell
#%% fit model

# define the model
model = define_model(vocab_size, max_length)
# define checkpoint callback
filepath = 'model_vgg_bilstm-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
# fit model
model.fit([X1train, X2train], ytrain, epochs=20, verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest))
```



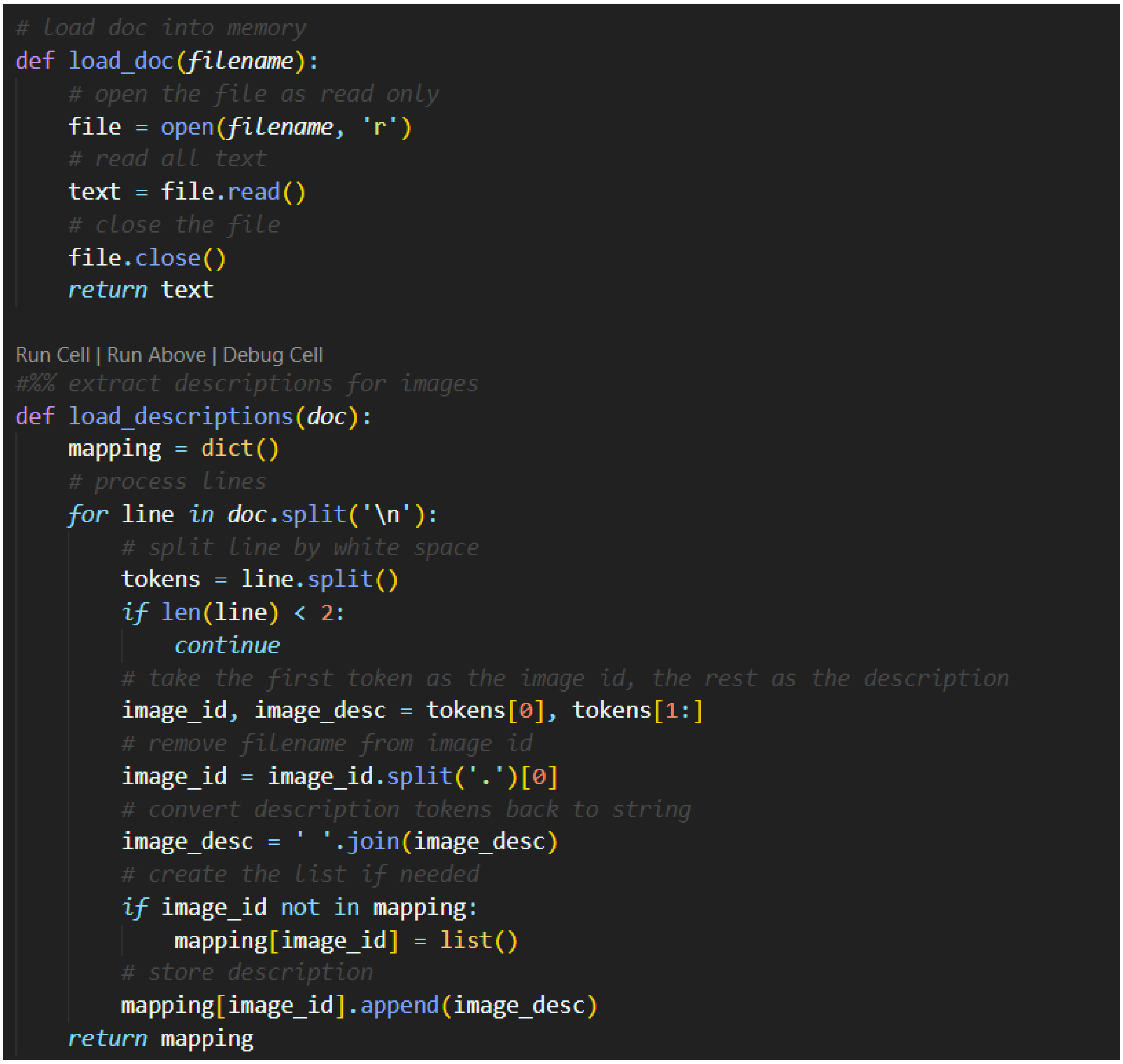
```python
from numpy import array
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
#from keras.layers import Bidirectional
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint

from os import listdir
from pickle import dump
import string
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
```



```python
def extract_features(directory):
    # load the model
    model = VGG16()
    # re-structure the model
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    # summarize
    print(model.summary())
    # extract features from each photo
    features = dict()
    for name in listdir(directory):
        # load an image from file
        filename = directory + '/' + name
        image = load_img(filename, target_size=(224, 224))
        # convert the image pixels to a numpy array
        image = img_to_array(image)
        # reshape data for the model
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        # prepare the image for the VGG model
        image = preprocess_input(image)
        # get features
        feature = model.predict(image, verbose=0)
        # get image id
        image_id = name.split('.')[0]
        # store feature
        features[image_id] = feature
        print('>%s' % name)
    return features
```

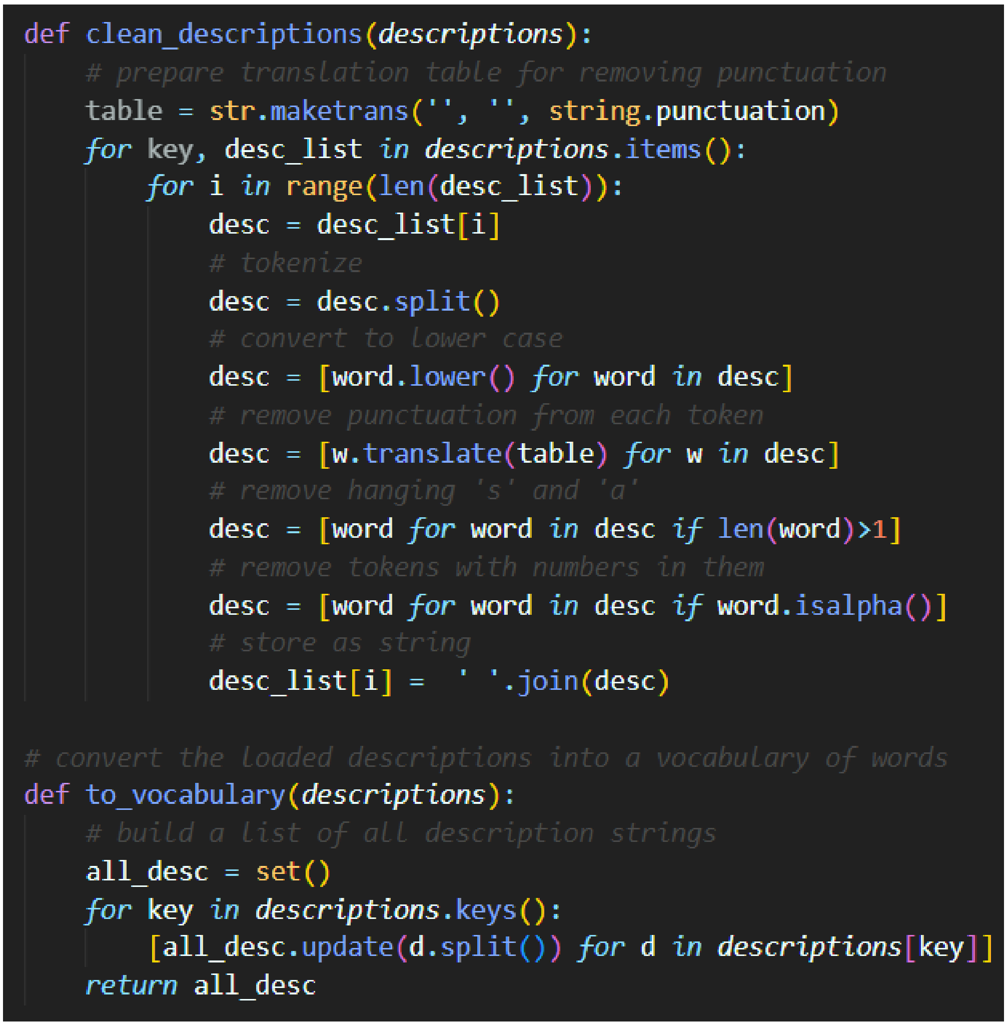


```
#%% extract features from all images
directory = 'Flicker8k_Dataset'
features = extract_features(directory)
print('Extracted Features: %d' % len(features))
# save to file
dump(features, open('features.pkl', 'wb'))
```

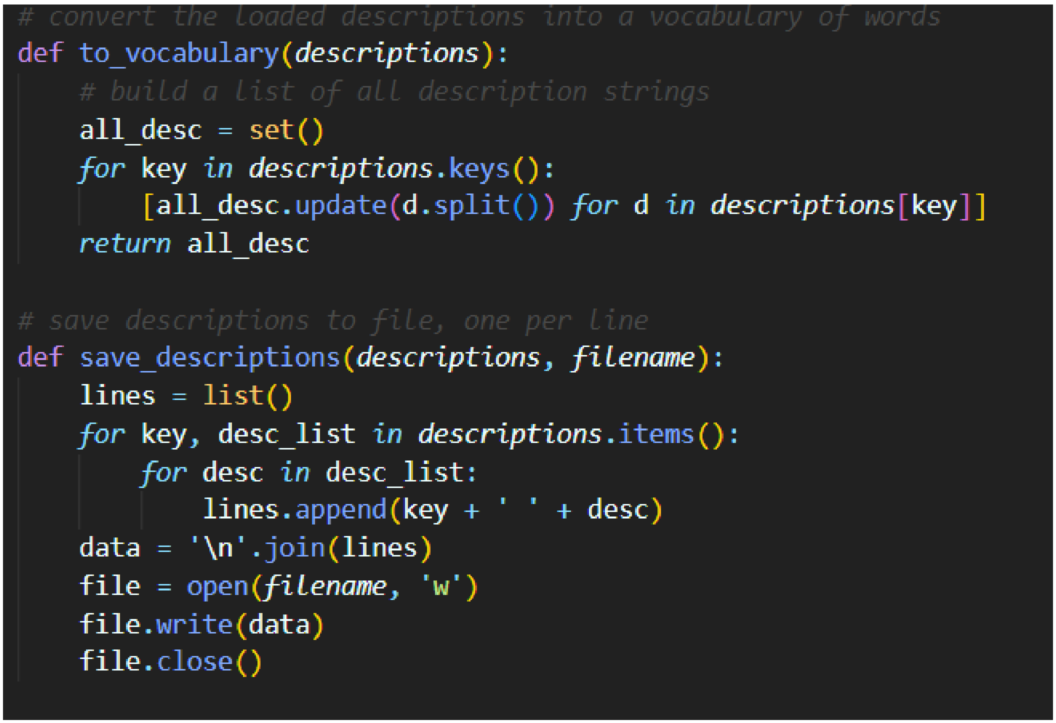


```
# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text
```

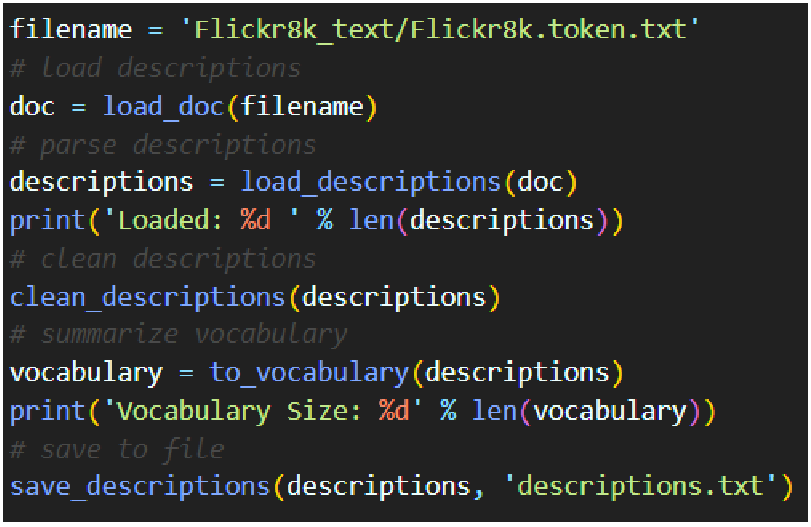
```
#%% extract descriptions for images
def load_descriptions(doc):
    mapping = dict()
    # process lines
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        if len(line) < 2:
            continue
        # take the first token as the image id, the rest as the description
        image_id, image_desc = tokens[0], tokens[1:]
        # remove filename from image id
        image_id = image_id.split('.')[0]
        # convert description tokens back to string
        image_desc = ' '.join(image_desc)
        # create the list if needed
        if image_id not in mapping:
            mapping[image_id] = list()
        # store description
        mapping[image_id].append(image_desc)
    return mapping
```



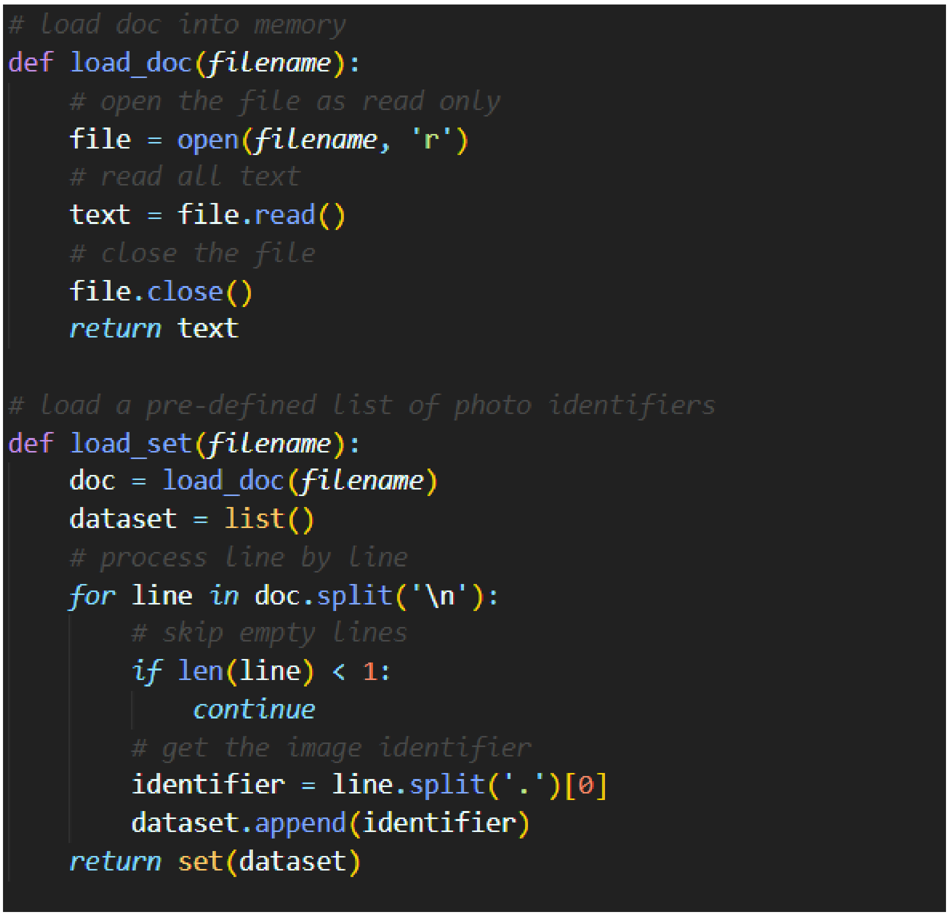
```python
def clean_descriptions(descriptions):
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            # tokenize
            desc = desc.split()
            # convert to lower case
            desc = [word.lower() for word in desc]
            # remove punctuation from each token
            desc = [w.translate(table) for w 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()]
            # store as string
            desc_list[i] =  ' '.join(desc)

# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    # build a list of all description strings
    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc
```



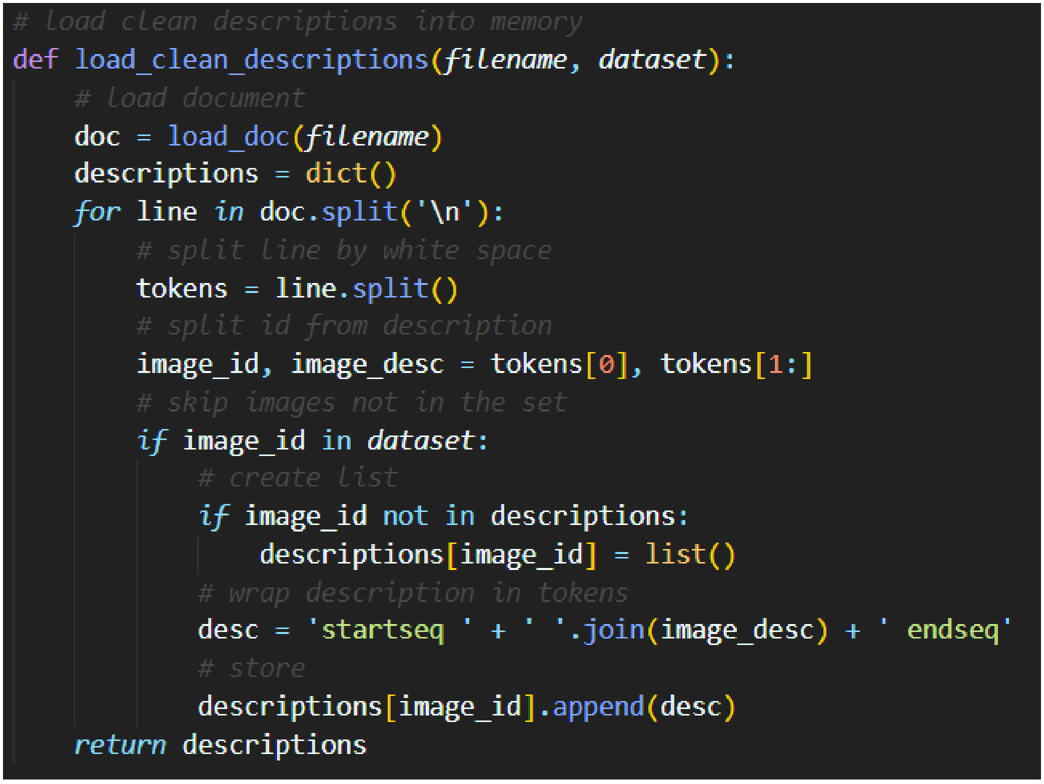
```python
# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    # build a list of all description strings
    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc

# save descriptions to file, one per line
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + ' ' + desc)
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()
```



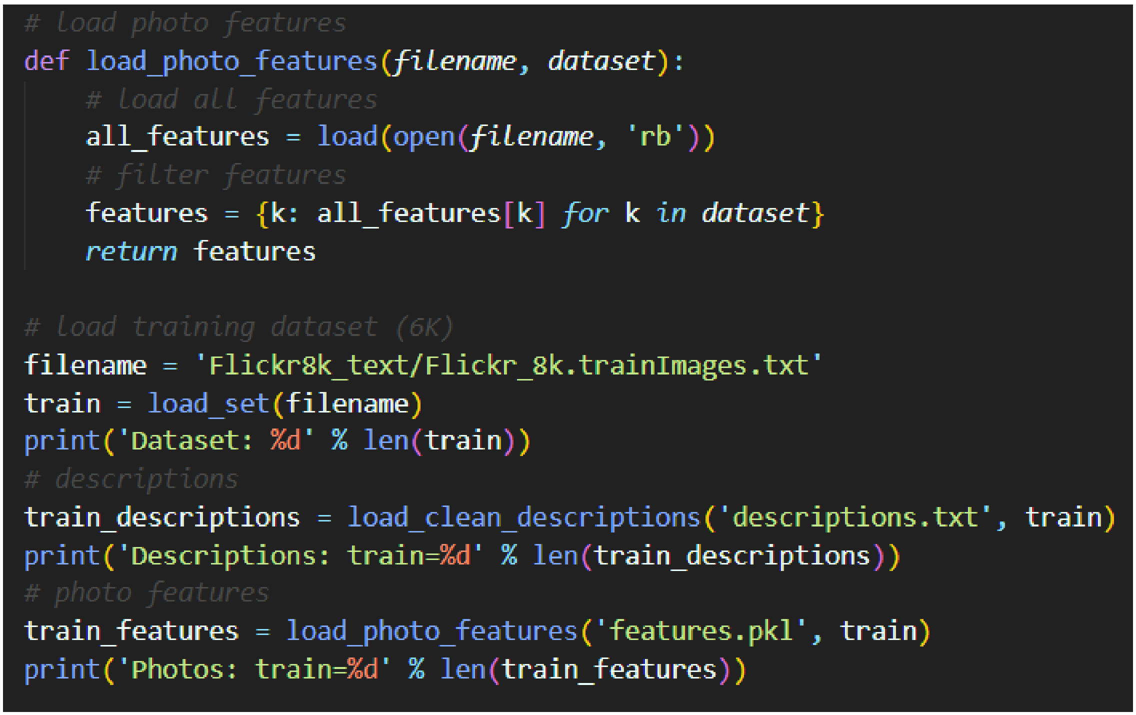
```python
filename = 'Flickr8k_text/Flickr8k.token.txt'
# load descriptions
doc = load_doc(filename)
# parse descriptions
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))
# clean descriptions
clean_descriptions(descriptions)
# summarize vocabulary
vocabulary = to_vocabulary(descriptions)
print('Vocabulary Size: %d' % len(vocabulary))
# save to file
save_descriptions(descriptions, 'descriptions.txt')
```



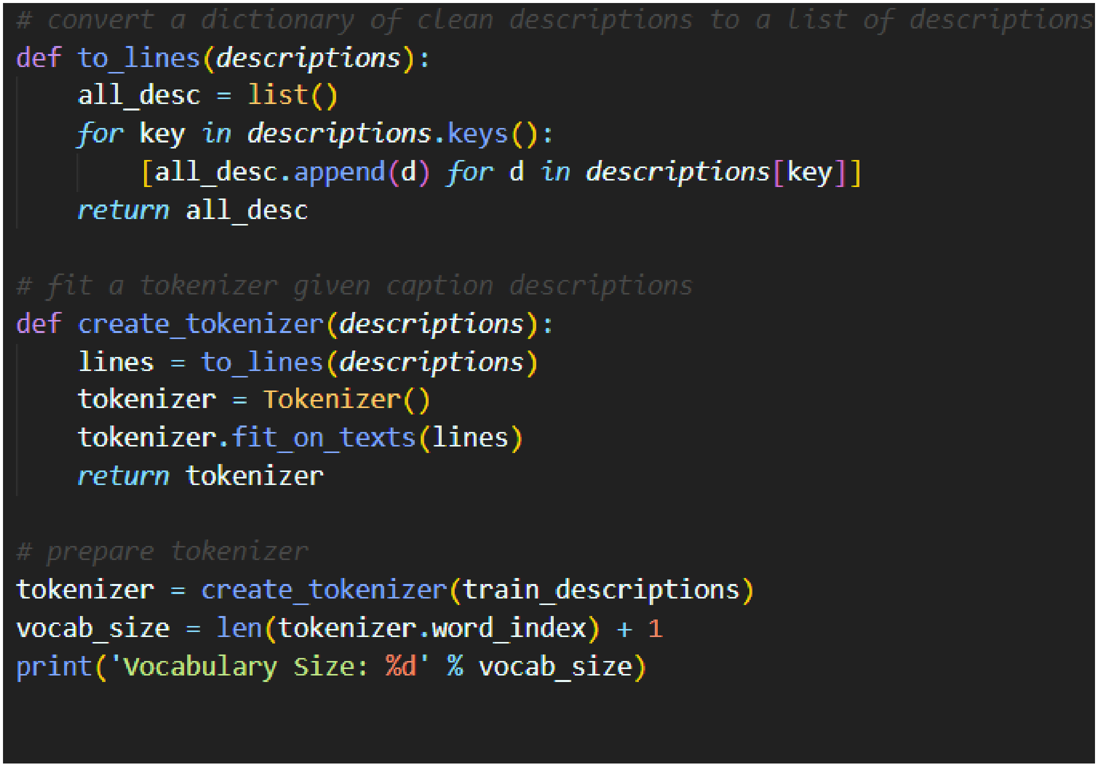
```python
# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text

# load a pre-defined list of photo identifiers
def load_set(filename):
    doc = load_doc(filename)
    dataset = list()
    # process line by line
    for line in doc.split('\n'):
        # skip empty lines
        if len(line) < 1:
            continue
        # get the image identifier
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)
```



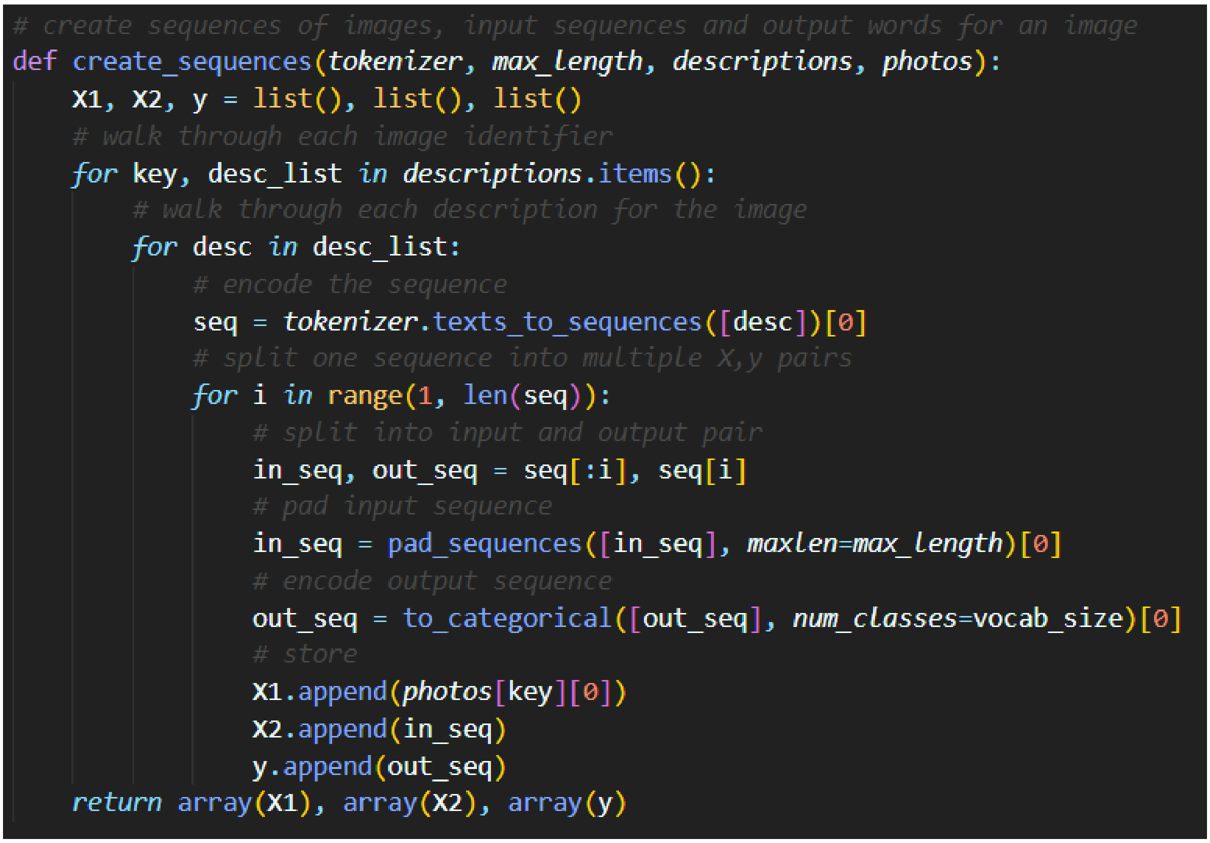
```python
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    # load document
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        # split id from description
        image_id, image_desc = tokens[0], tokens[1:]
        # skip images not in the set
        if image_id in dataset:
            # create list
            if image_id not in descriptions:
                descriptions[image_id] = list()
            # wrap description in tokens
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            # store
            descriptions[image_id].append(desc)
    return descriptions
```



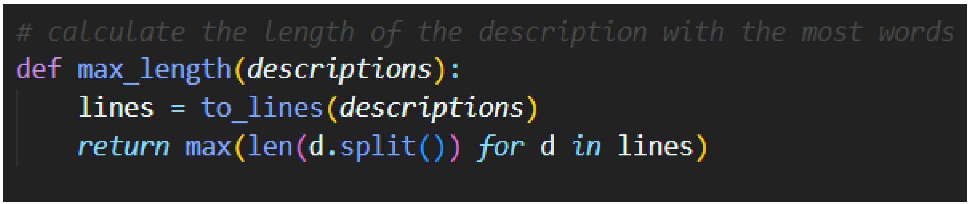
```python
# load photo features
def load_photo_features(filename, dataset):
    # load all features
    all_features = load(open(filename, 'rb'))
    # filter features
    features = {k: all_features[k] for k in dataset}
    return features

# load training dataset (6K)
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# photo features
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
```



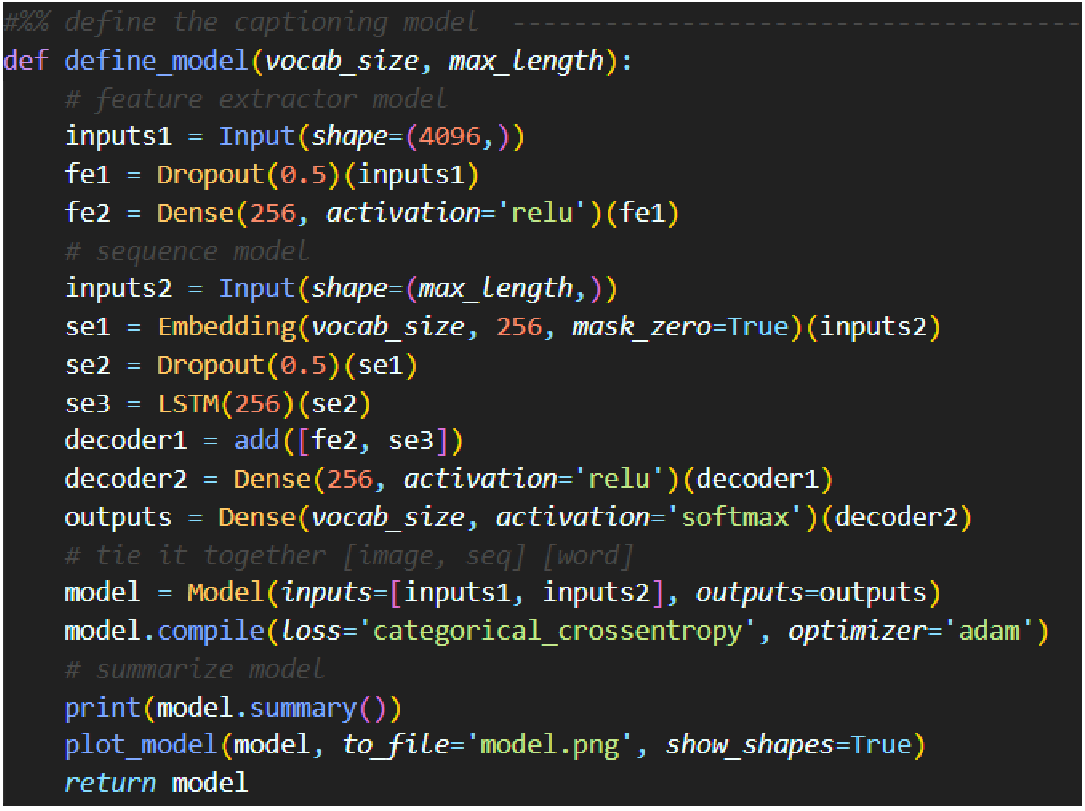
```python
# convert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
```



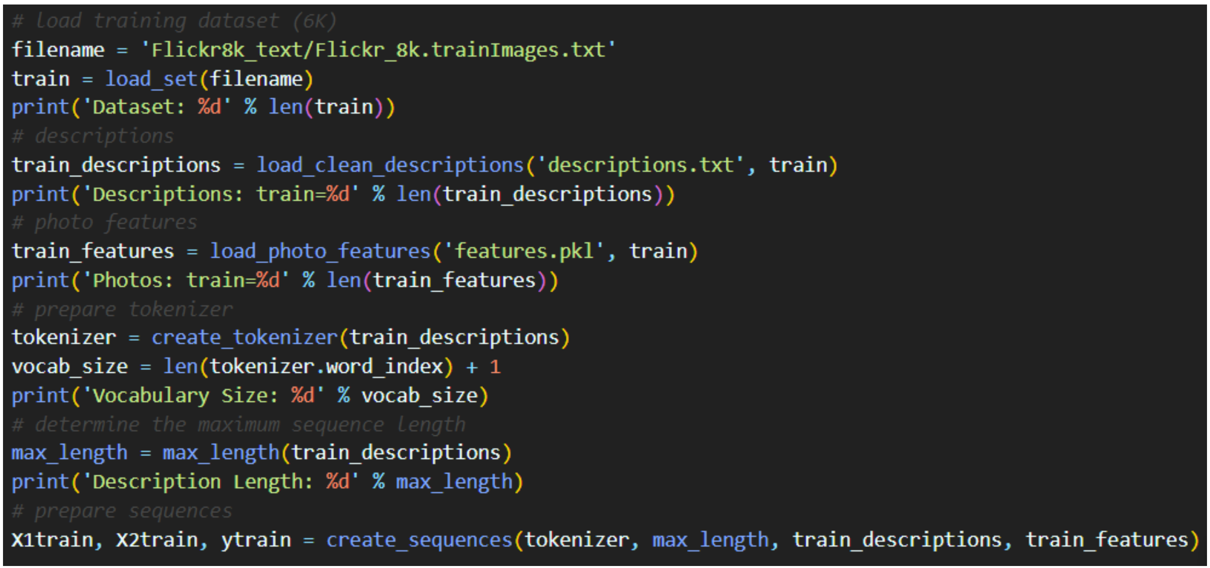
```python
# create sequences of images, input sequences and output words for an image
def create_sequences(tokenizer, max_length, descriptions, photos):
    X1, X2, y = list(), list(), list()
    # walk through each image identifier
    for key, desc_list in descriptions.items():
        # walk through 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(photos[key][0])
                X2.append(in_seq)
                y.append(out_seq)
    return array(X1), array(X2), array(y)
```



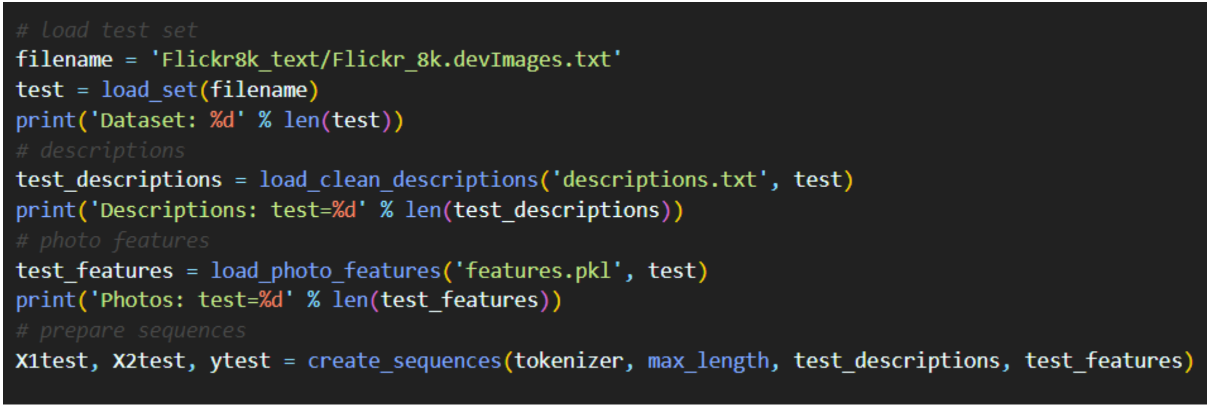
```python
# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)
```



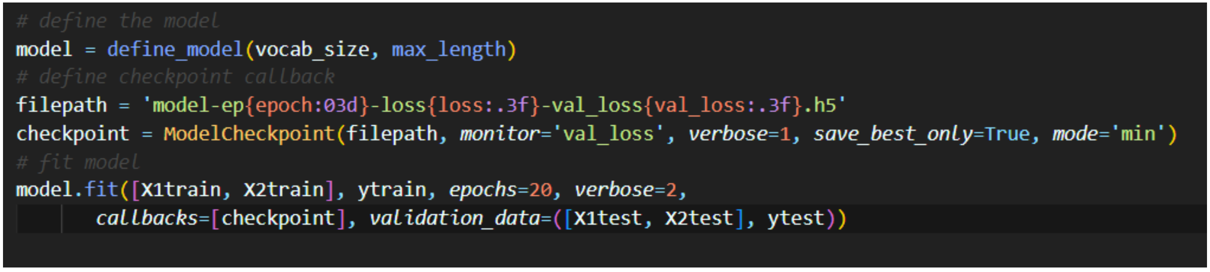
```python
#%% define the captioning model  ------------------------------------
def define_model(vocab_size, max_length):
    # feature extractor model
    inputs1 = Input(shape=(4096,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    # sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)
    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
```



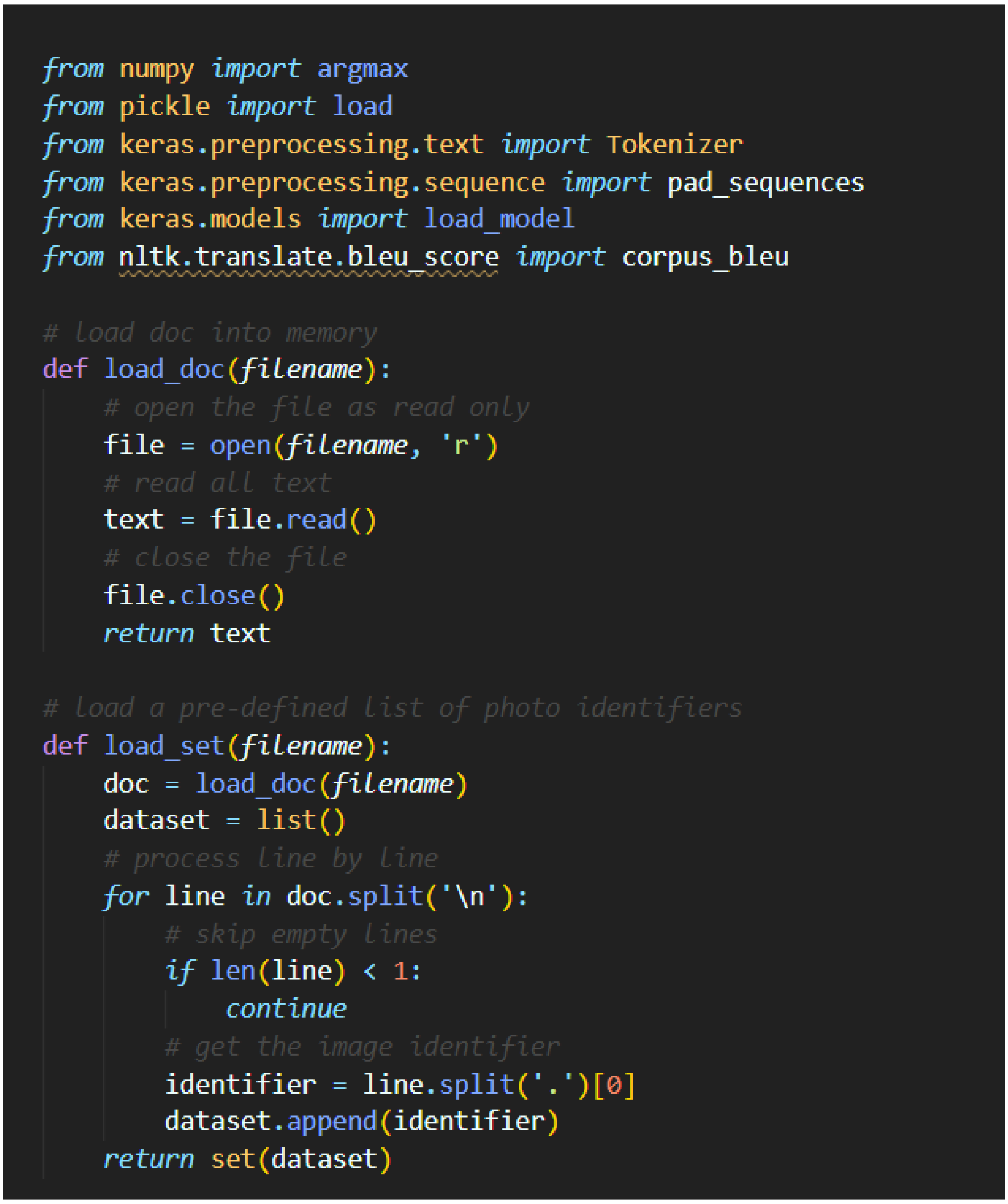
```
# load training dataset (6K)
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# photo features
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
# prepare sequences
X1train, X2train, ytrain = create_sequences(tokenizer, max_length, train_descriptions, train_features)
```



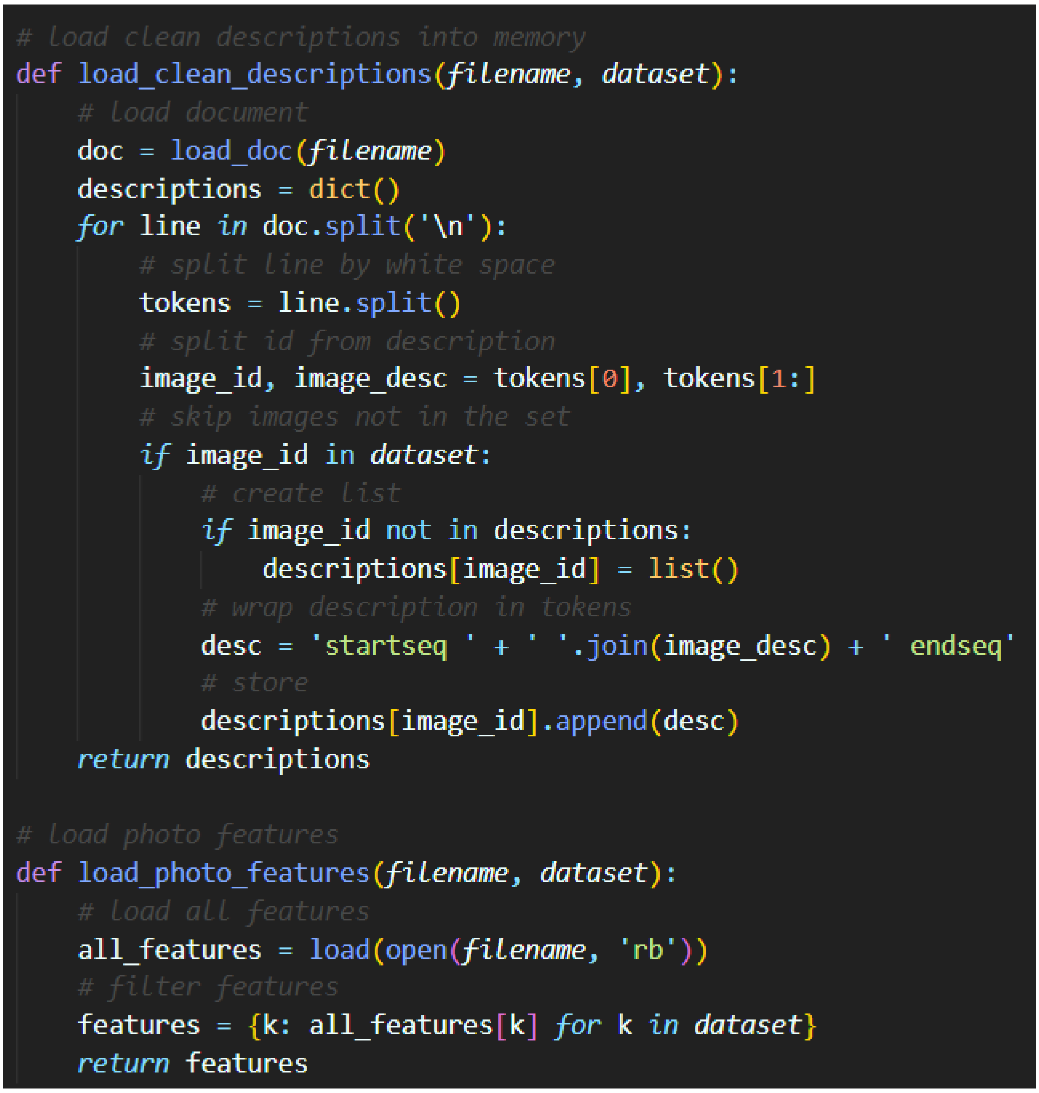
```
# load test set
filename = 'Flickr8k_text/Flickr_8k.devImages.txt'
test = load_set(filename)
print('Dataset: %d' % len(test))
# descriptions
test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))
# photo features
test_features = load_photo_features('features.pkl', test)
print('Photos: test=%d' % len(test_features))
# prepare sequences
X1test, X2test, ytest = create_sequences(tokenizer, max_length, test_descriptions, test_features)
```



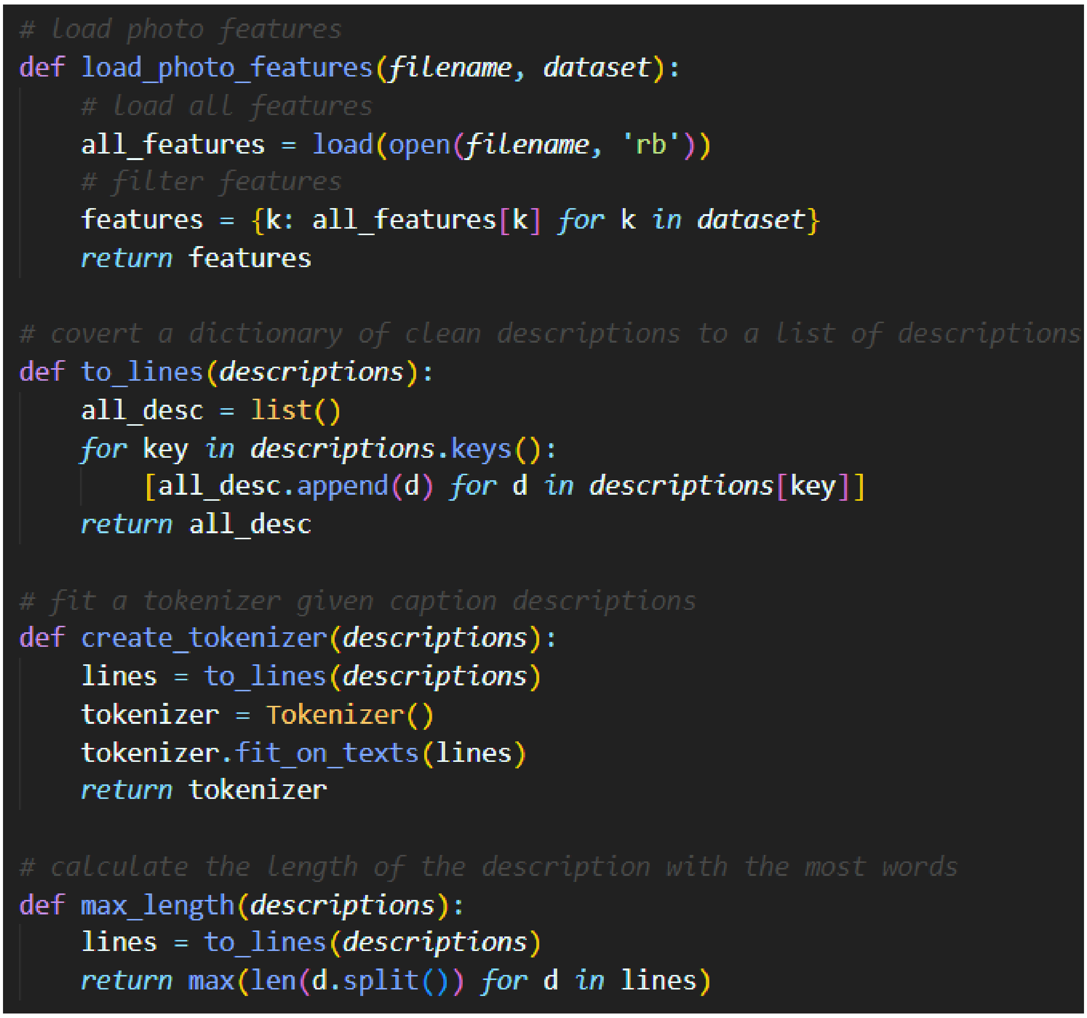
```
# define the model
model = define_model(vocab_size, max_length)
# define checkpoint callback
filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
# fit model
model.fit([X1train, X2train], ytrain, epochs=20, verbose=2,
        callbacks=[checkpoint], validation_data=([X1test, X2test], ytest))
```



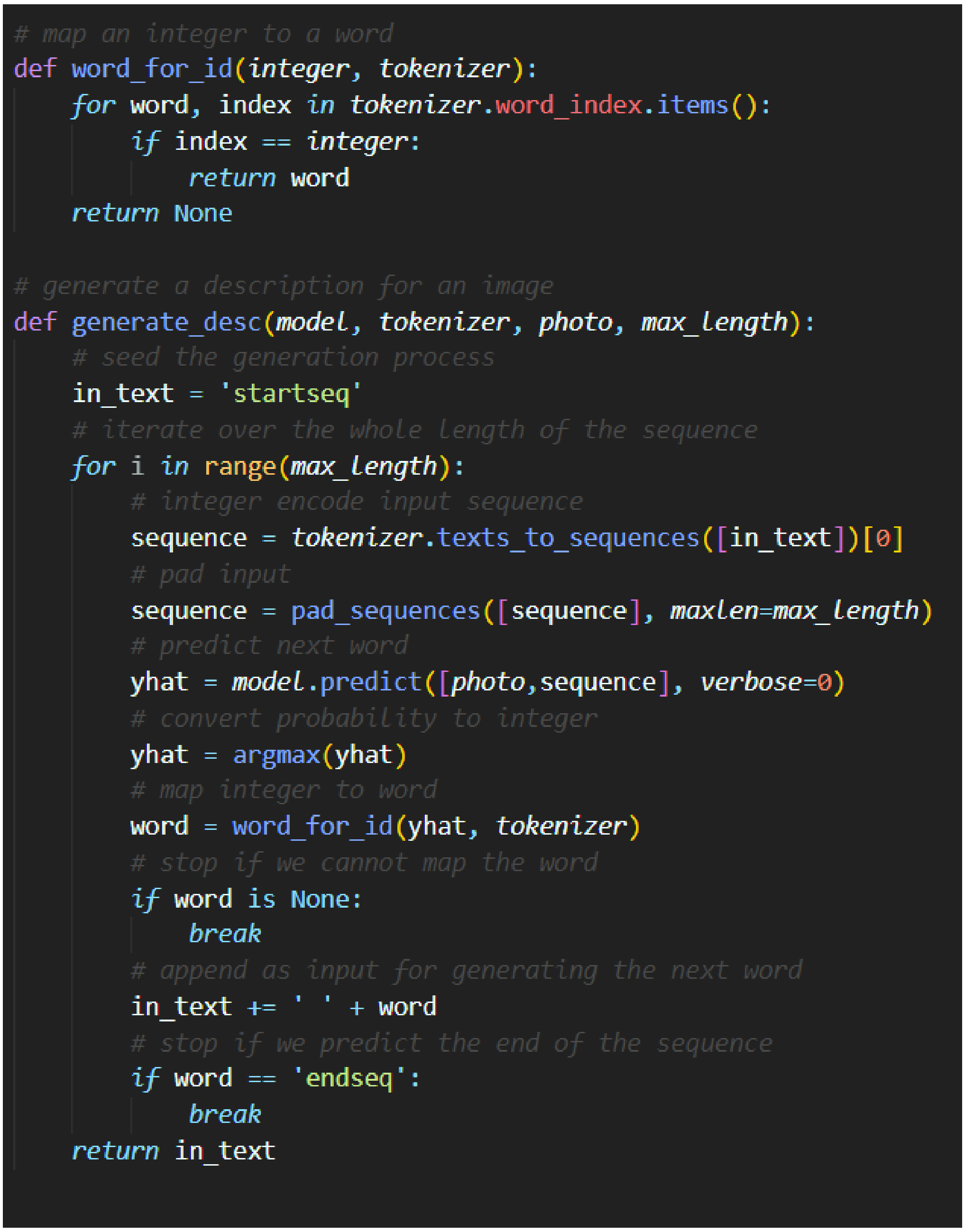
```python
from numpy import argmax
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu

# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text

# load a pre-defined list of photo identifiers
def load_set(filename):
    doc = load_doc(filename)
    dataset = list()
    # process line by line
    for line in doc.split('\n'):
        # skip empty lines
        if len(line) < 1:
            continue
        # get the image identifier
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)
```



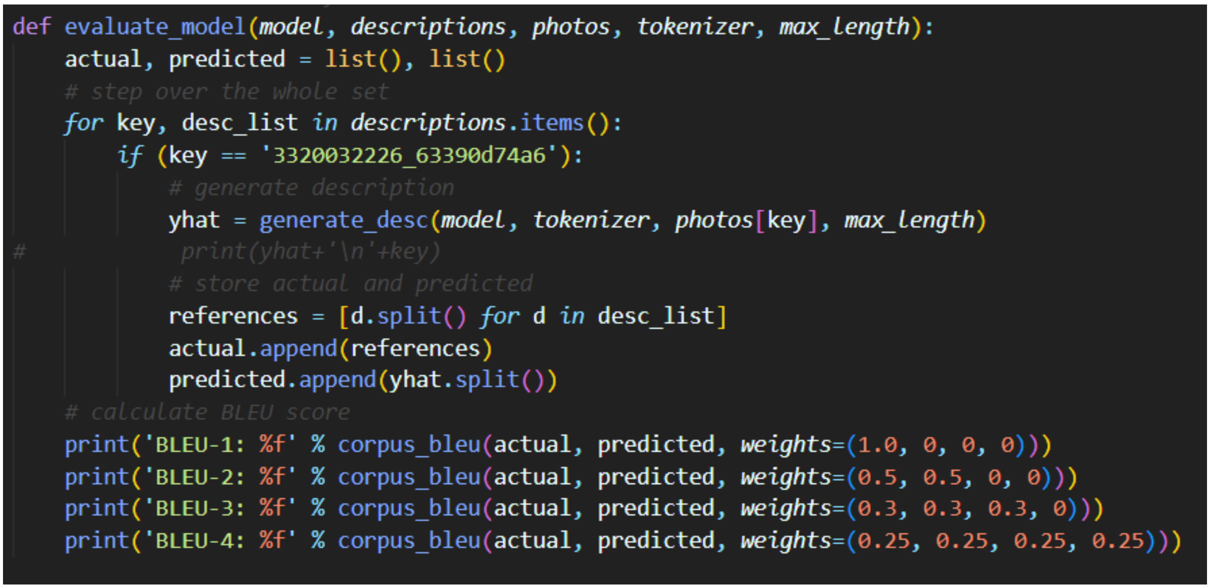
```python
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    # load document
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        # split id from description
        image_id, image_desc = tokens[0], tokens[1:]
        # skip images not in the set
        if image_id in dataset:
            # create list
            if image_id not in descriptions:
                descriptions[image_id] = list()
            # wrap description in tokens
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            # store
            descriptions[image_id].append(desc)
    return descriptions

# load photo features
def load_photo_features(filename, dataset):
    # load all features
    all_features = load(open(filename, 'rb'))
    # filter features
    features = {k: all_features[k] for k in dataset}
    return features
```



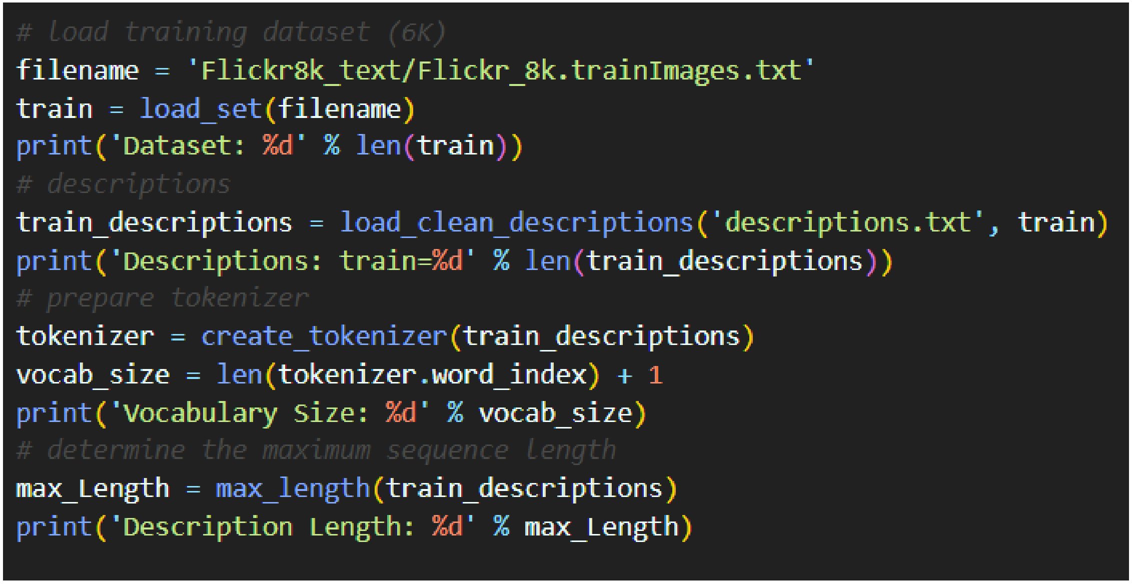
```python
# load photo features
def load_photo_features(filename, dataset):
    # load all features
    all_features = load(open(filename, 'rb'))
    # filter features
    features = {k: all_features[k] for k in dataset}
    return features

# covert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)
```



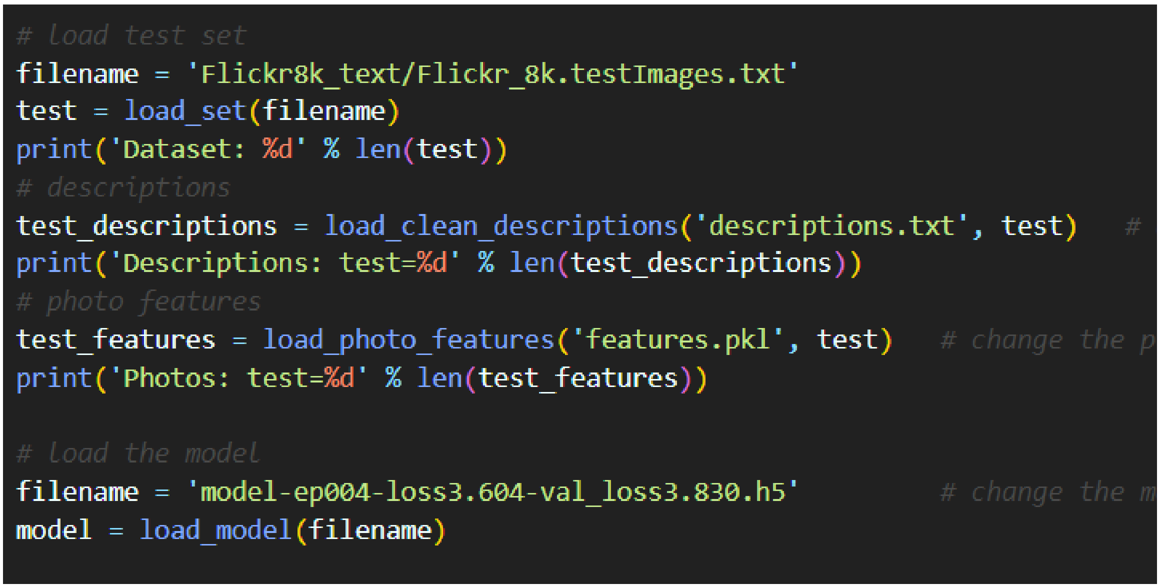
```
# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'startseq'
    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
        # predict next word
        yhat = model.predict([photo,sequence], verbose=0)
        # convert probability to integer
        yhat = argmax(yhat)
        # map integer to word
        word = word_for_id(yhat, tokenizer)
        # stop if we cannot map the word
        if word is None:
            break
        # append as input for generating the next word
        in_text += ' ' + word
        # stop if we predict the end of the sequence
        if word == 'endseq':
            break
    return in_text
```



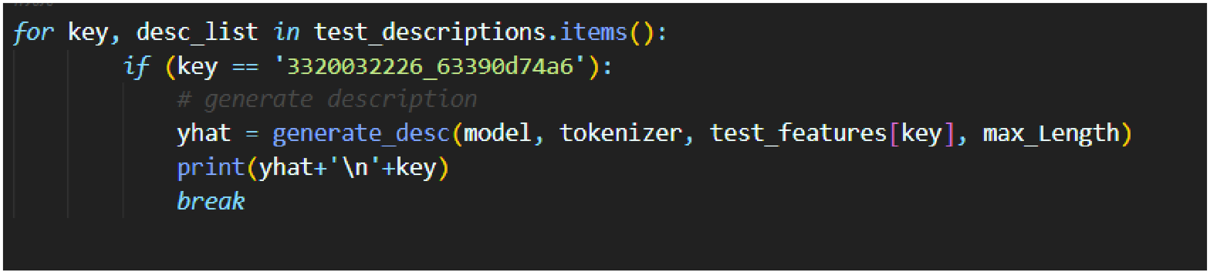
```
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
    actual, predicted = list(), list()
    # step over the whole set
    for key, desc_list in descriptions.items():
        if (key == '3320032226_63390d74a6'):
            # generate description
            yhat = generate_desc(model, tokenizer, photos[key], max_length)
#            print(yhat+'\n'+key)
            # store actual and predicted
            references = [d.split() for d in desc_list]
            actual.append(references)
            predicted.append(yhat.split())
    # calculate BLEU score
    print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
    print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
    print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
```



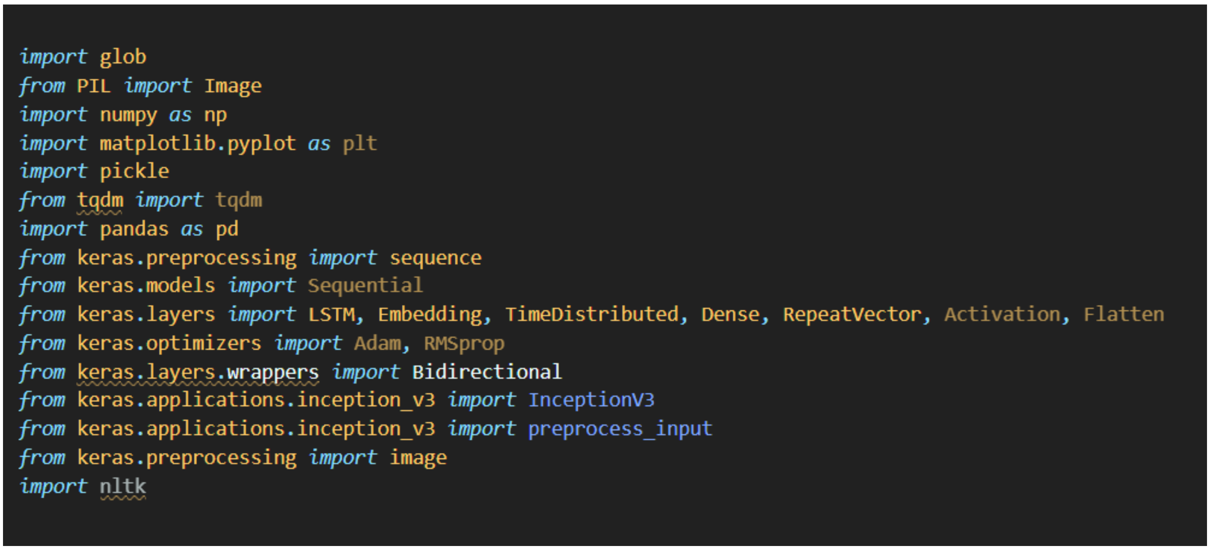
```
# load training dataset (6K)
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
# determine the maximum sequence length
max_Length = max_length(train_descriptions)
print('Description Length: %d' % max_Length)
```



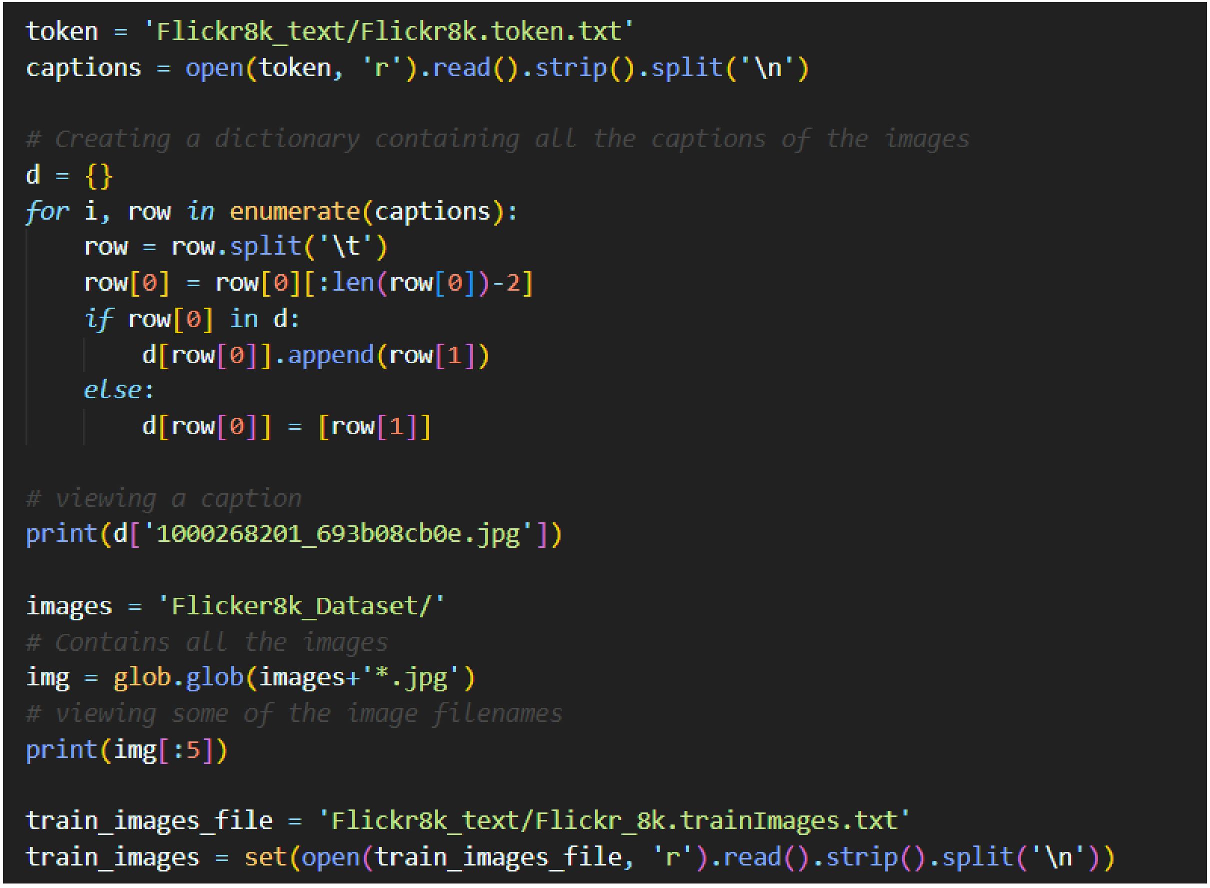
```python
# load test set
filename = 'Flickr8k_text/Flickr_8k.testImages.txt'
test = load_set(filename)
print('Dataset: %d' % len(test))
# descriptions
test_descriptions = load_clean_descriptions('descriptions.txt', test)   #
print('Descriptions: test=%d' % len(test_descriptions))
# photo features
test_features = load_photo_features('features.pkl', test)   # change the p
print('Photos: test=%d' % len(test_features))

# load the model
filename = 'model-ep004-loss3.604-val_loss3.830.h5'         # change the m
model = load_model(filename)
```



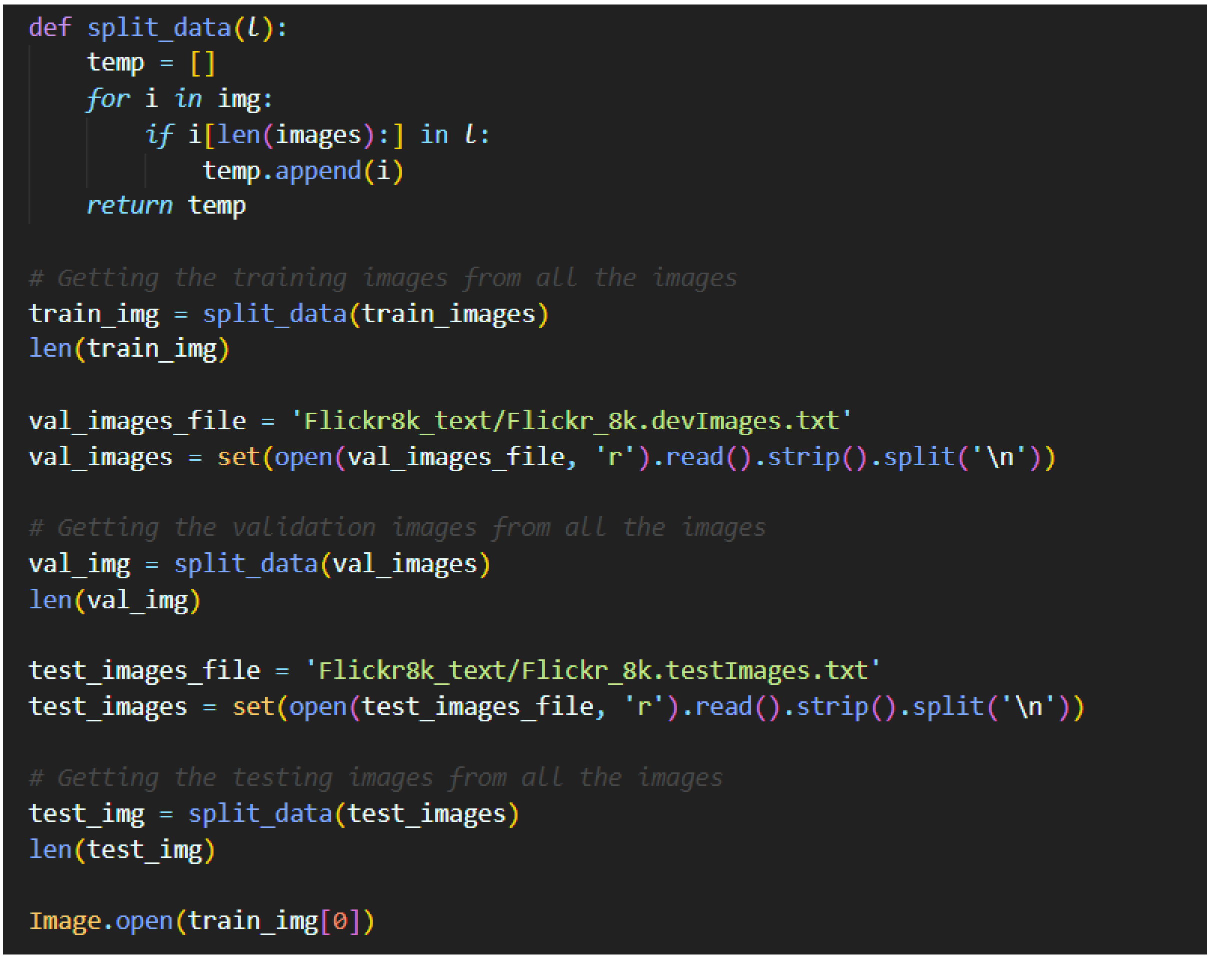
```python
for key, desc_list in test_descriptions.items():
        if (key == '3320032226_63390d74a6'):
            # generate description
            yhat = generate_desc(model, tokenizer, test_features[key], max_Length)
            print(yhat+'\n'+key)
            break
```



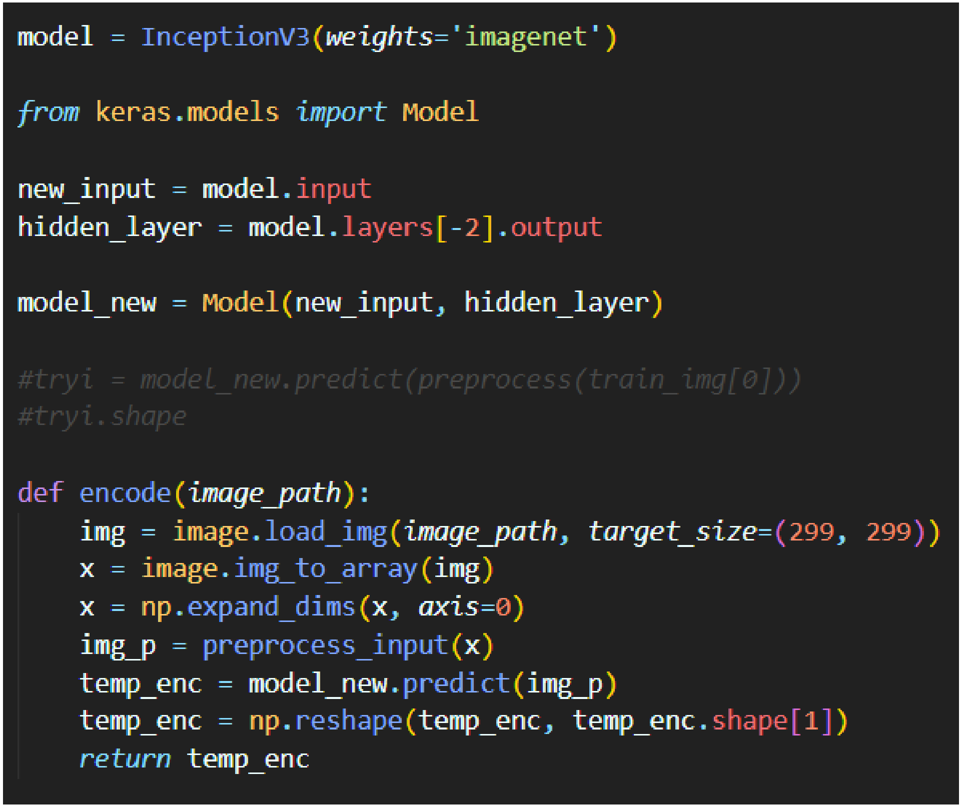
```
import glob
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import pickle
from tqdm import tqdm
import pandas as pd
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector, Activation, Flatten
from keras.optimizers import Adam, RMSprop
from keras.layers.wrappers import Bidirectional
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input
from keras.preprocessing import image
import nltk
```



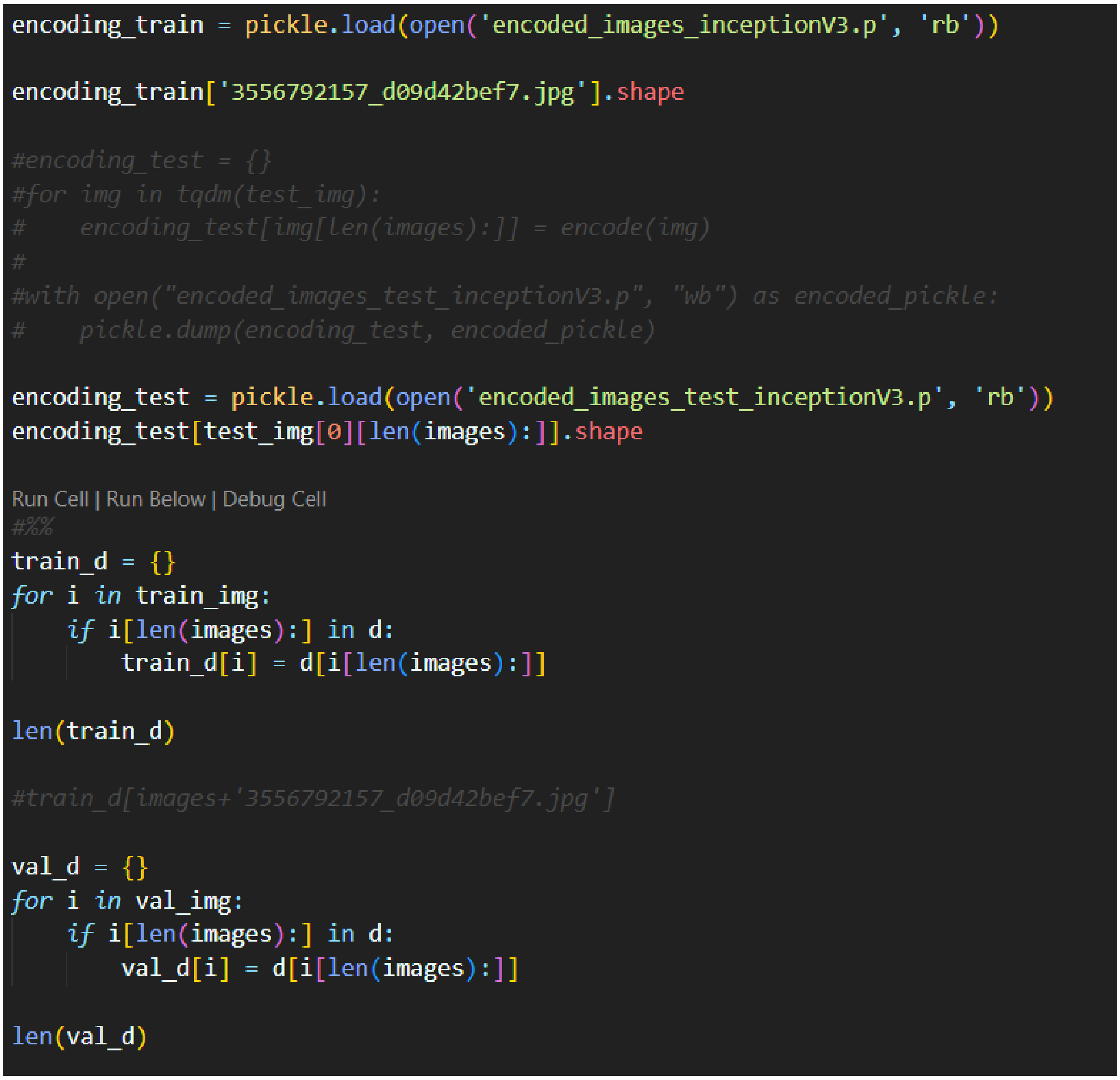
```
token = 'Flickr8k_text/Flickr8k.token.txt'
captions = open(token, 'r').read().strip().split('\n')

# Creating a dictionary containing all the captions of the images
d = {}
for i, row in enumerate(captions):
    row = row.split('\t')
    row[0] = row[0][:len(row[0])-2]
    if row[0] in d:
        d[row[0]].append(row[1])
    else:
        d[row[0]] = [row[1]]

# viewing a caption
print(d['1000268201_693b08cb0e.jpg'])

images = 'Flicker8k_Dataset/'
# Contains all the images
img = glob.glob(images+'*.jpg')
# viewing some of the image filenames
print(img[:5])

train_images_file = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train_images = set(open(train_images_file, 'r').read().strip().split('\n'))
```



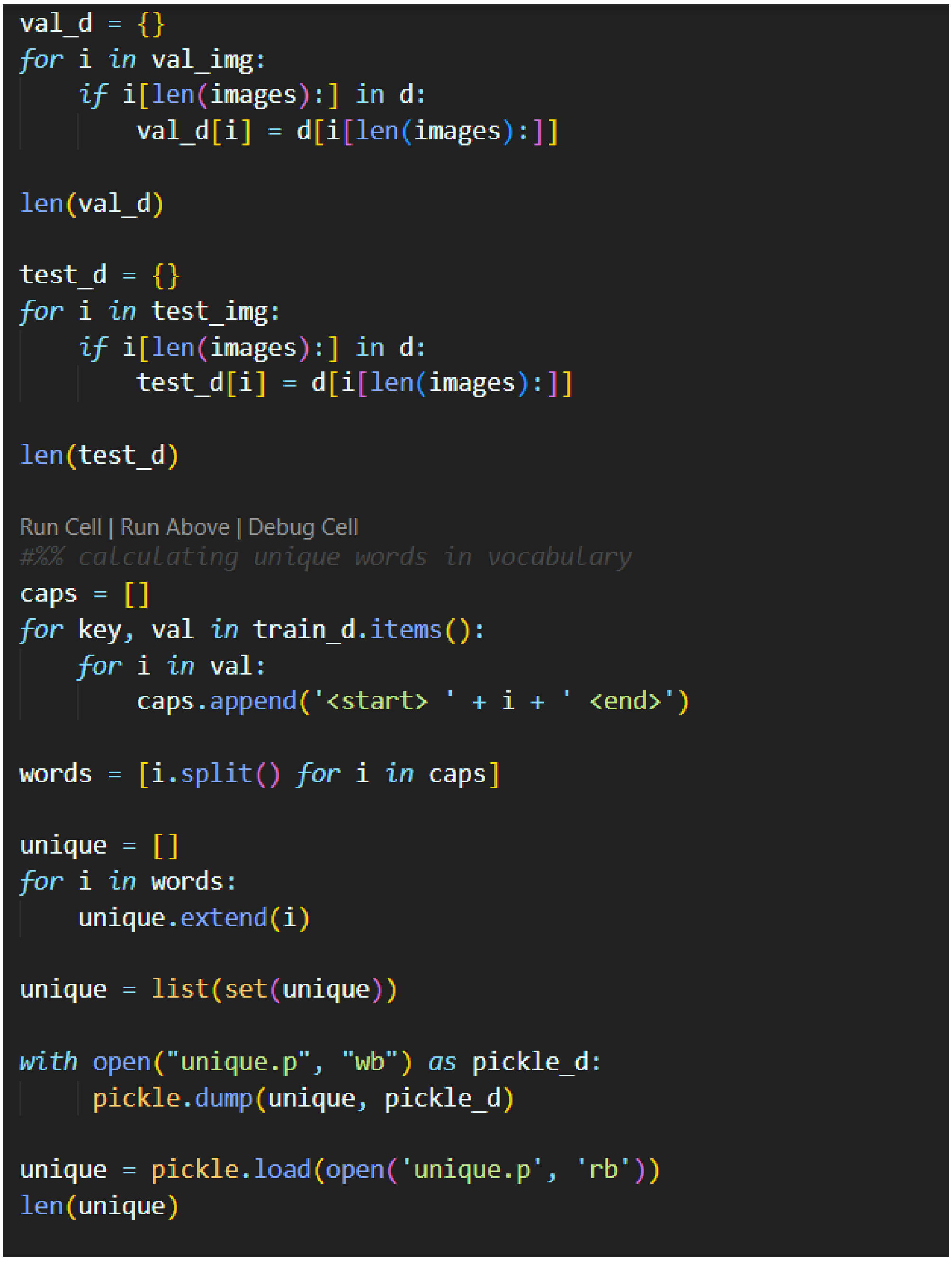
```python
def split_data(l):
    temp = []
    for i in img:
        if i[len(images):] in l:
            temp.append(i)
    return temp

# Getting the training images from all the images
train_img = split_data(train_images)
len(train_img)

val_images_file = 'Flickr8k_text/Flickr_8k.devImages.txt'
val_images = set(open(val_images_file, 'r').read().strip().split('\n'))

# Getting the validation images from all the images
val_img = split_data(val_images)
len(val_img)

test_images_file = 'Flickr8k_text/Flickr_8k.testImages.txt'
test_images = set(open(test_images_file, 'r').read().strip().split('\n'))

# Getting the testing images from all the images
test_img = split_data(test_images)
len(test_img)

Image.open(train_img[0])
```



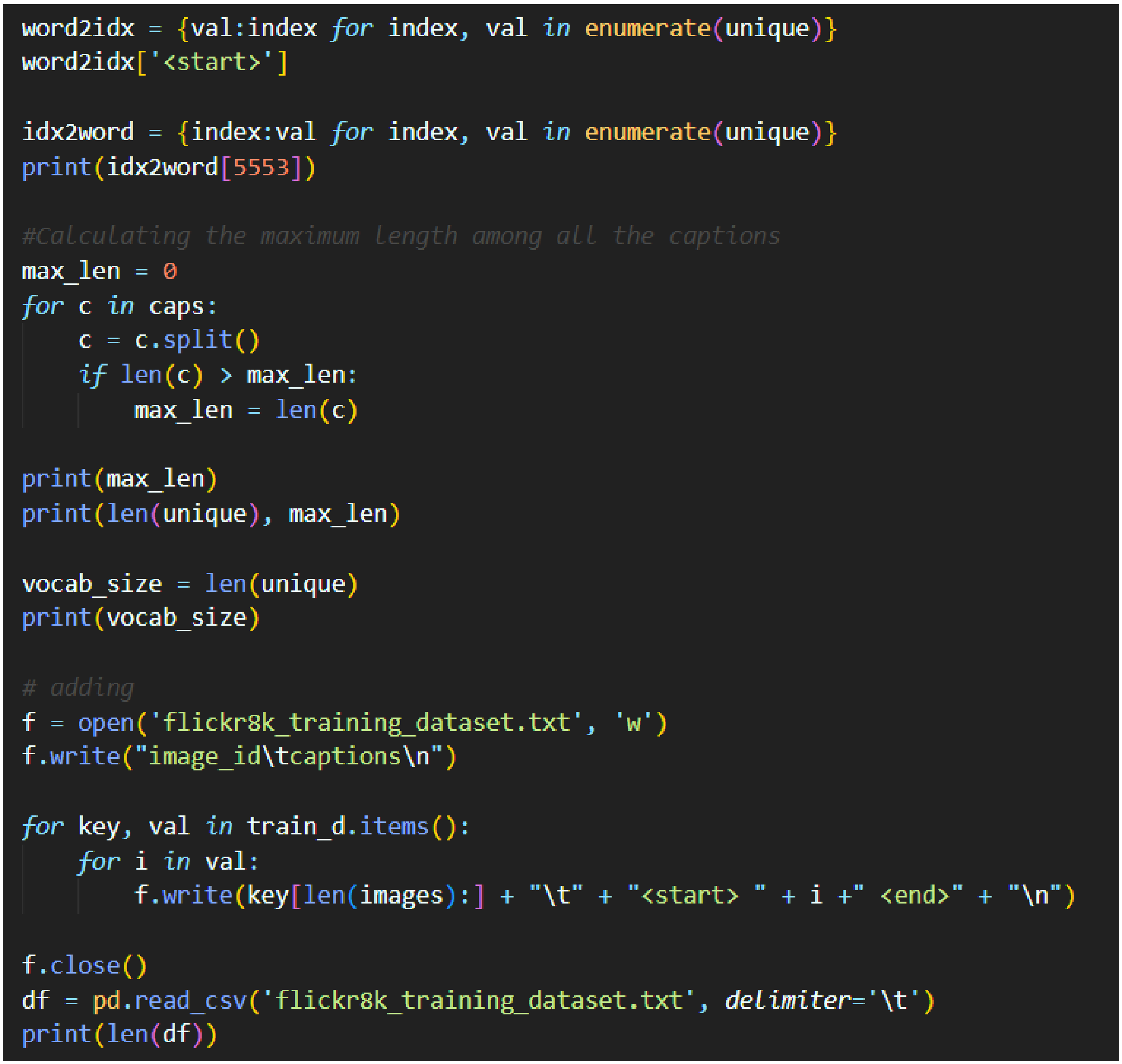
```python
model = InceptionV3(weights='imagenet')

from keras.models import Model

new_input = model.input
hidden_layer = model.layers[-2].output

model_new = Model(new_input, hidden_layer)

#tryi = model_new.predict(preprocess(train_img[0]))
#tryi.shape

def encode(image_path):
    img = image.load_img(image_path, target_size=(299, 299))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    img_p = preprocess_input(x)
    temp_enc = model_new.predict(img_p)
    temp_enc = np.reshape(temp_enc, temp_enc.shape[1])
    return temp_enc
```



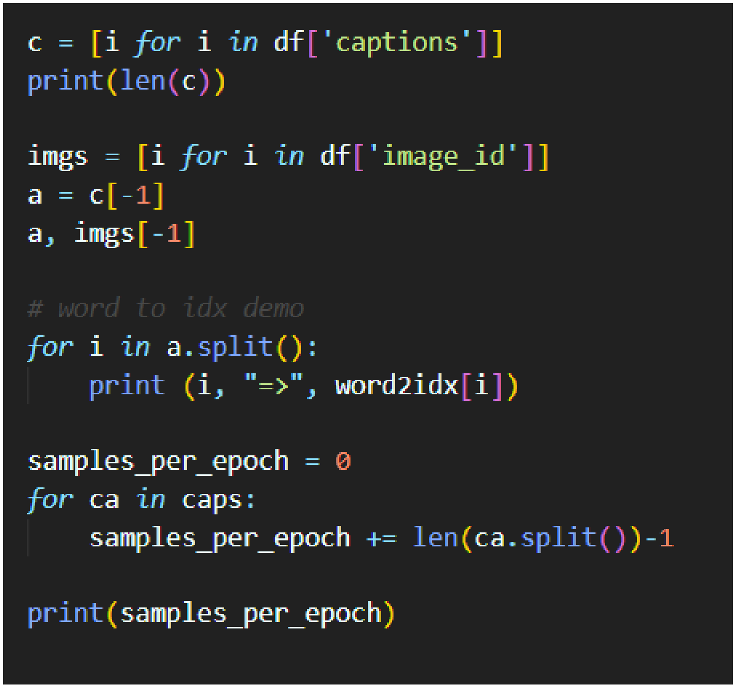
```
encoding_train = pickle.load(open('encoded_images_inceptionV3.p', 'rb'))

encoding_train['3556792157_d09d42bef7.jpg'].shape

#encoding_test = {}
#for img in tqdm(test_img):
#    encoding_test[img[len(images):]] = encode(img)
#
#with open("encoded_images_test_inceptionV3.p", "wb") as encoded_pickle:
#    pickle.dump(encoding_test, encoded_pickle)

encoding_test = pickle.load(open('encoded_images_test_inceptionV3.p', 'rb'))
encoding_test[test_img[0][len(images):]].shape

#%%
train_d = {}
for i in train_img:
    if i[len(images):] in d:
        train_d[i] = d[i[len(images):]]

len(train_d)

#train_d[images+'3556792157_d09d42bef7.jpg']

val_d = {}
for i in val_img:
    if i[len(images):] in d:
        val_d[i] = d[i[len(images):]]

len(val_d)
```



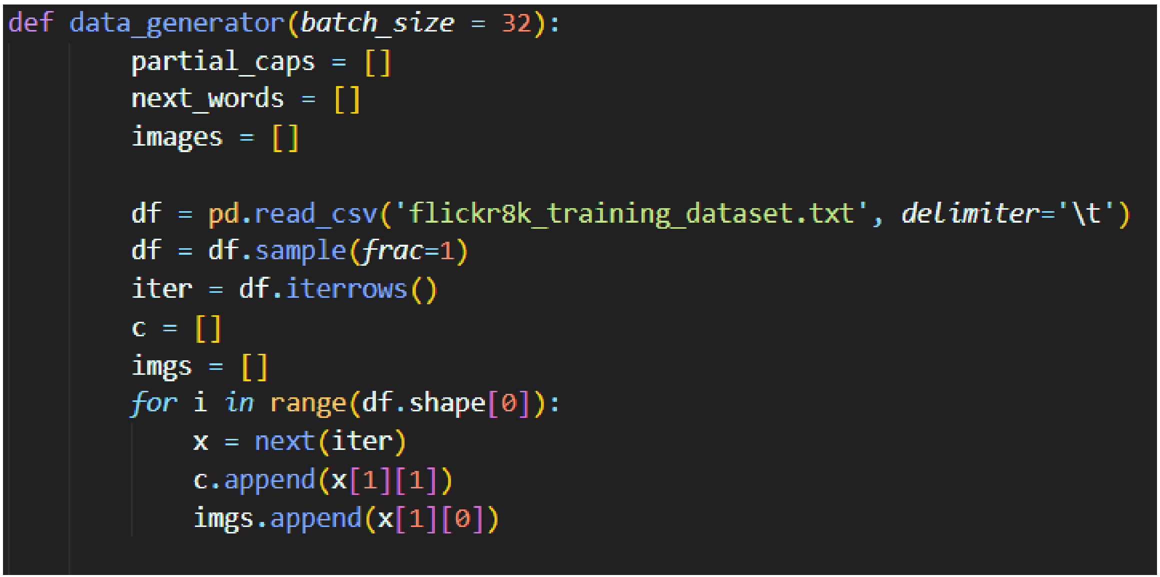
```
val_d = {}
for i in val_img:
    if i[len(images):] in d:
        val_d[i] = d[i[len(images):]]

len(val_d)

test_d = {}
for i in test_img:
    if i[len(images):] in d:
        test_d[i] = d[i[len(images):]]

len(test_d)

#%% calculating unique words in vocabulary
caps = []
for key, val in train_d.items():
    for i in val:
        caps.append('<start> ' + i + ' <end>')

words = [i.split() for i in caps]

unique = []
for i in words:
    unique.extend(i)

unique = list(set(unique))

with open("unique.p", "wb") as pickle_d:
    pickle.dump(unique, pickle_d)

unique = pickle.load(open('unique.p', 'rb'))
len(unique)
```



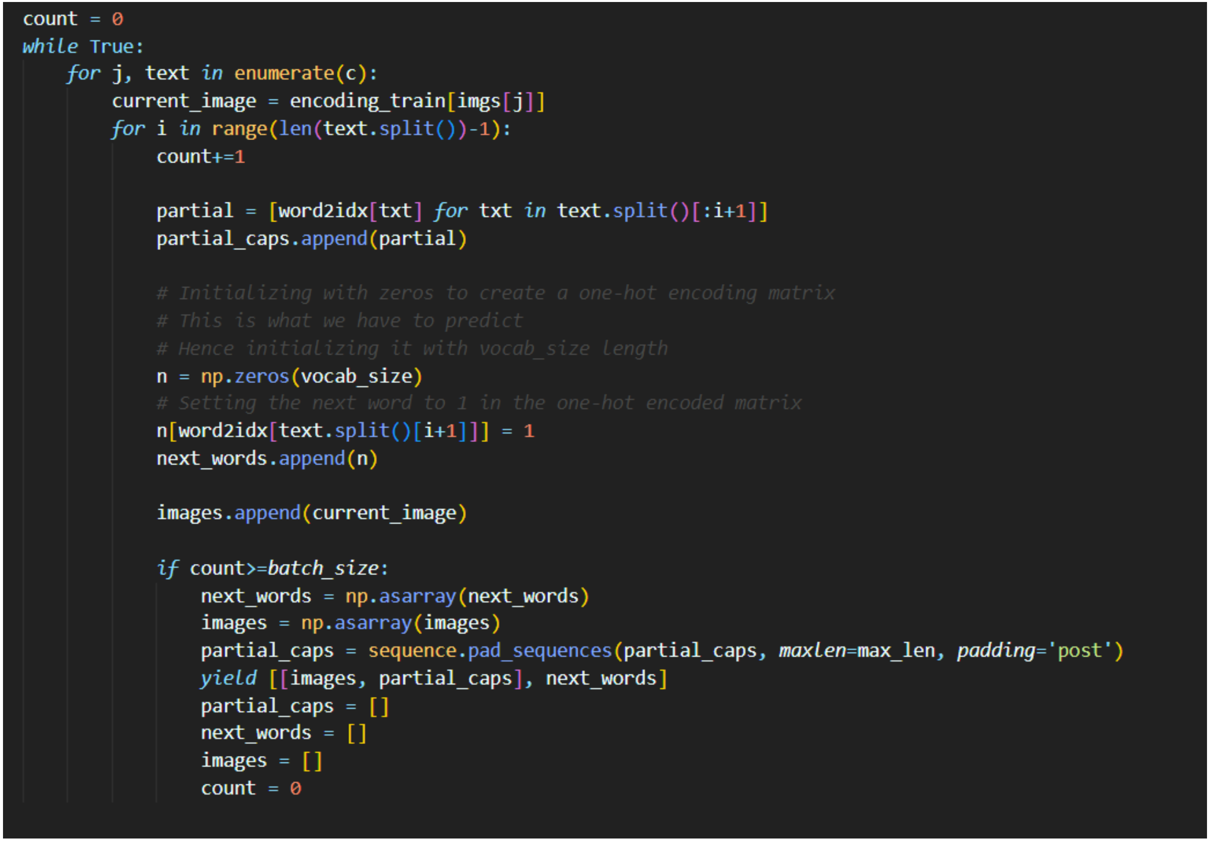
```
word2idx = {val:index for index, val in enumerate(unique)}
word2idx['<start>']

idx2word = {index:val for index, val in enumerate(unique)}
print(idx2word[5553])

#Calculating the maximum length among all the captions
max_len = 0
for c in caps:
    c = c.split()
    if len(c) > max_len:
        max_len = len(c)

print(max_len)
print(len(unique), max_len)

vocab_size = len(unique)
print(vocab_size)

# adding
f = open('flickr8k_training_dataset.txt', 'w')
f.write("image_id\tcaptions\n")

for key, val in train_d.items():
    for i in val:
        f.write(key[len(images):] + "\t" + "<start> " + i +" <end>" + "\n")

f.close()
df = pd.read_csv('flickr8k_training_dataset.txt', delimiter='\t')
print(len(df))
```



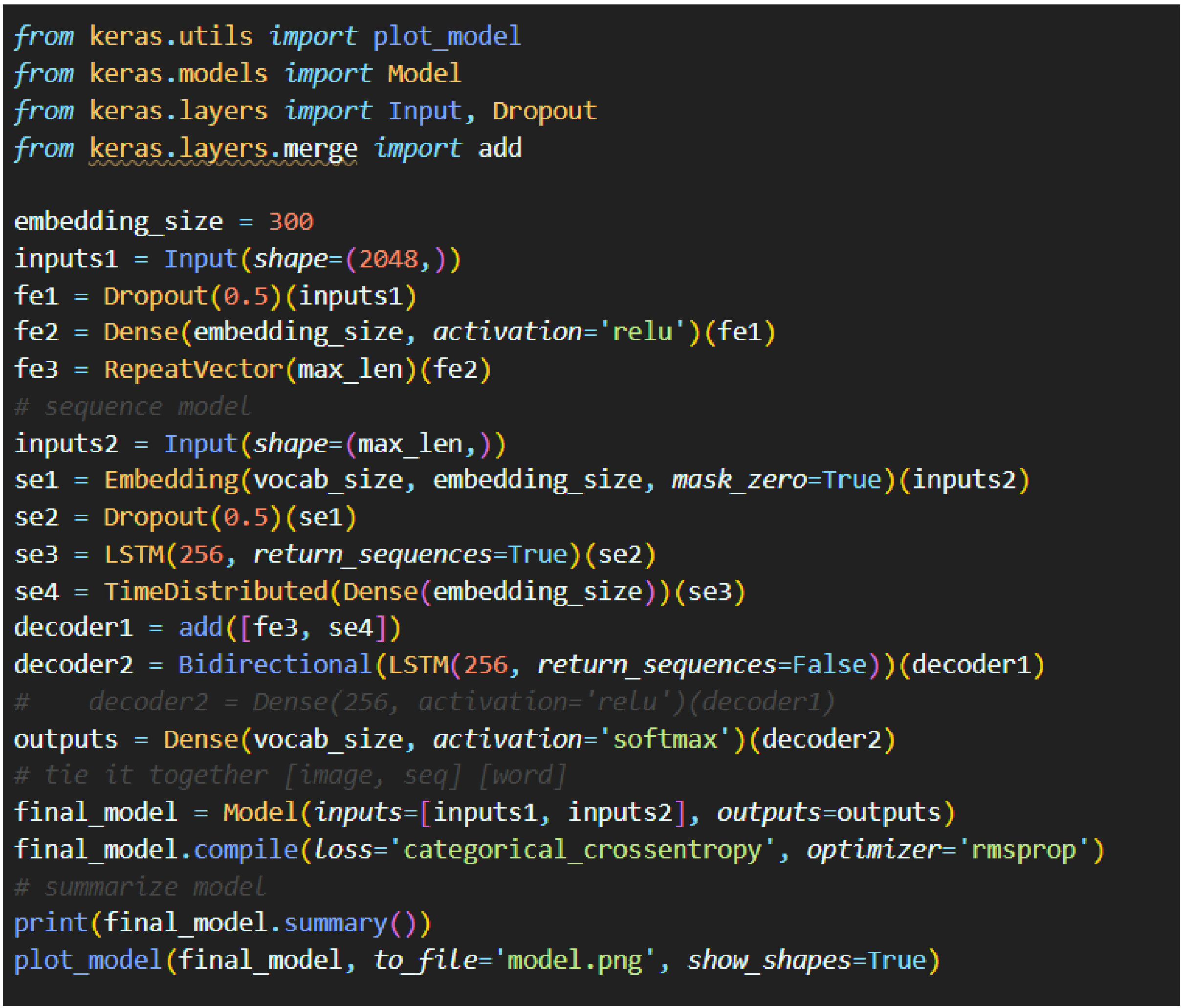
```
c = [i for i in df['captions']]
print(len(c))

imgs = [i for i in df['image_id']]
a = c[-1]
a, imgs[-1]

# word to idx demo
for i in a.split():
    print (i, "=>", word2idx[i])

samples_per_epoch = 0
for ca in caps:
    samples_per_epoch += len(ca.split())-1

print(samples_per_epoch)
```



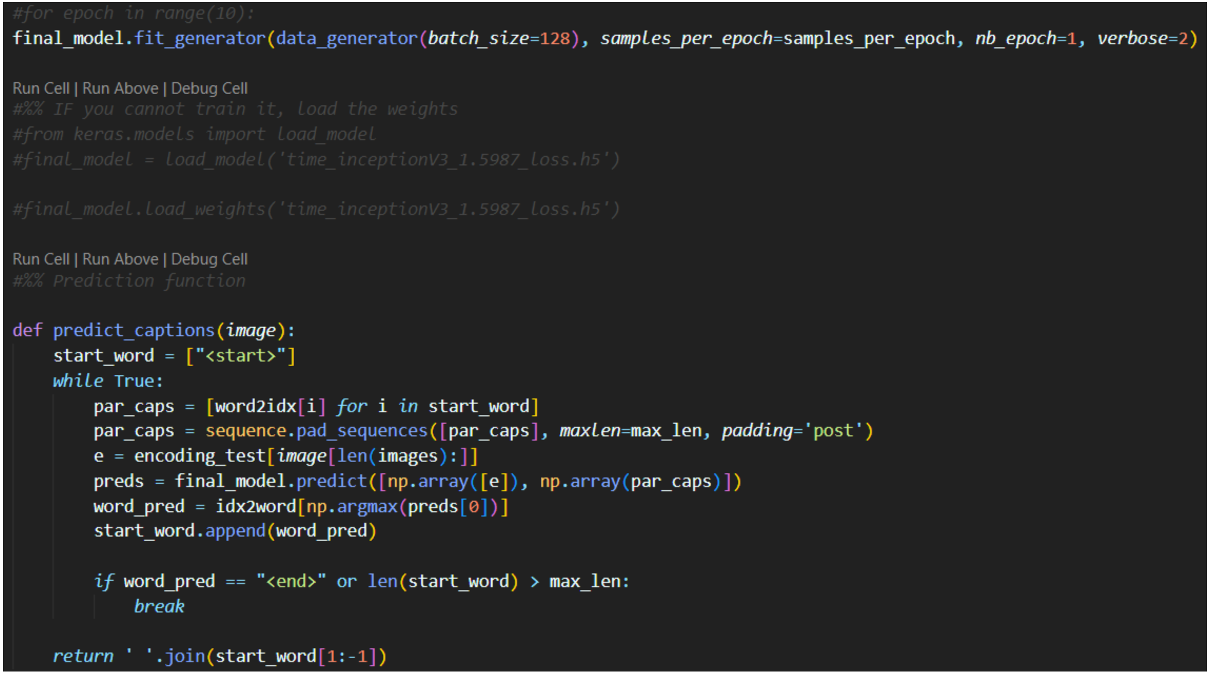
```python
def data_generator(batch_size = 32):
        partial_caps = []
        next_words = []
        images = []

        df = pd.read_csv('flickr8k_training_dataset.txt', delimiter='\t')
        df = df.sample(frac=1)
        iter = df.iterrows()
        c = []
        imgs = []
        for i in range(df.shape[0]):
            x = next(iter)
            c.append(x[1][1])
            imgs.append(x[1][0])
```



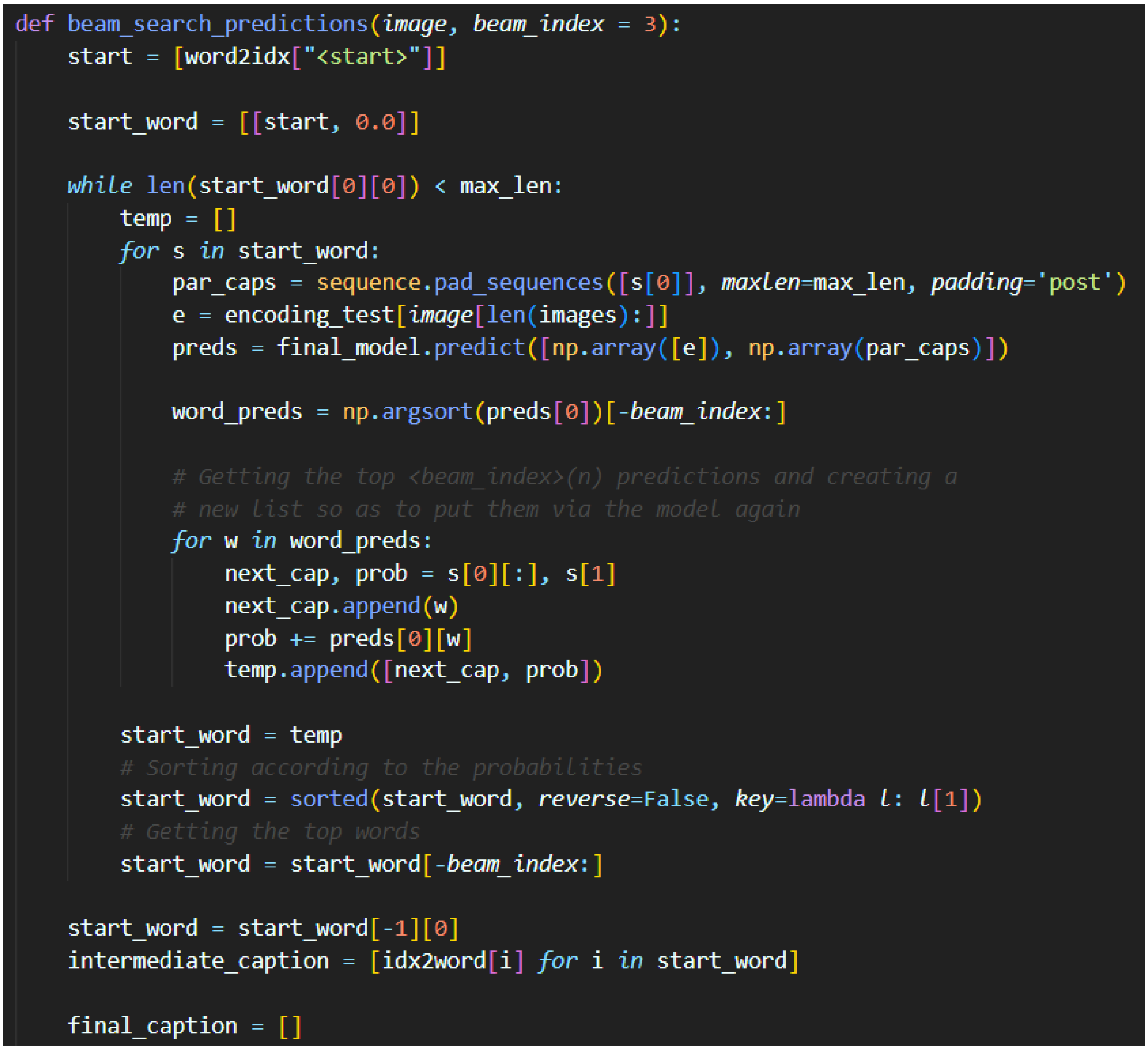
```python
count = 0
while True:
    for j, text in enumerate(c):
        current_image = encoding_train[imgs[j]]
        for i in range(len(text.split())-1):
            count+=1

            partial = [word2idx[txt] for txt in text.split()[:i+1]]
            partial_caps.append(partial)

            # Initializing with zeros to create a one-hot encoding matrix
            # This is what we have to predict
            # Hence initializing it with vocab_size length
            n = np.zeros(vocab_size)
            # Setting the next word to 1 in the one-hot encoded matrix
            n[word2idx[text.split()[i+1]]] = 1
            next_words.append(n)

            images.append(current_image)

            if count>=batch_size:
                next_words = np.asarray(next_words)
                images = np.asarray(images)
                partial_caps = sequence.pad_sequences(partial_caps, maxlen=max_len, padding='post')
                yield [[images, partial_caps], next_words]
                partial_caps = []
                next_words = []
                images = []
                count = 0
```



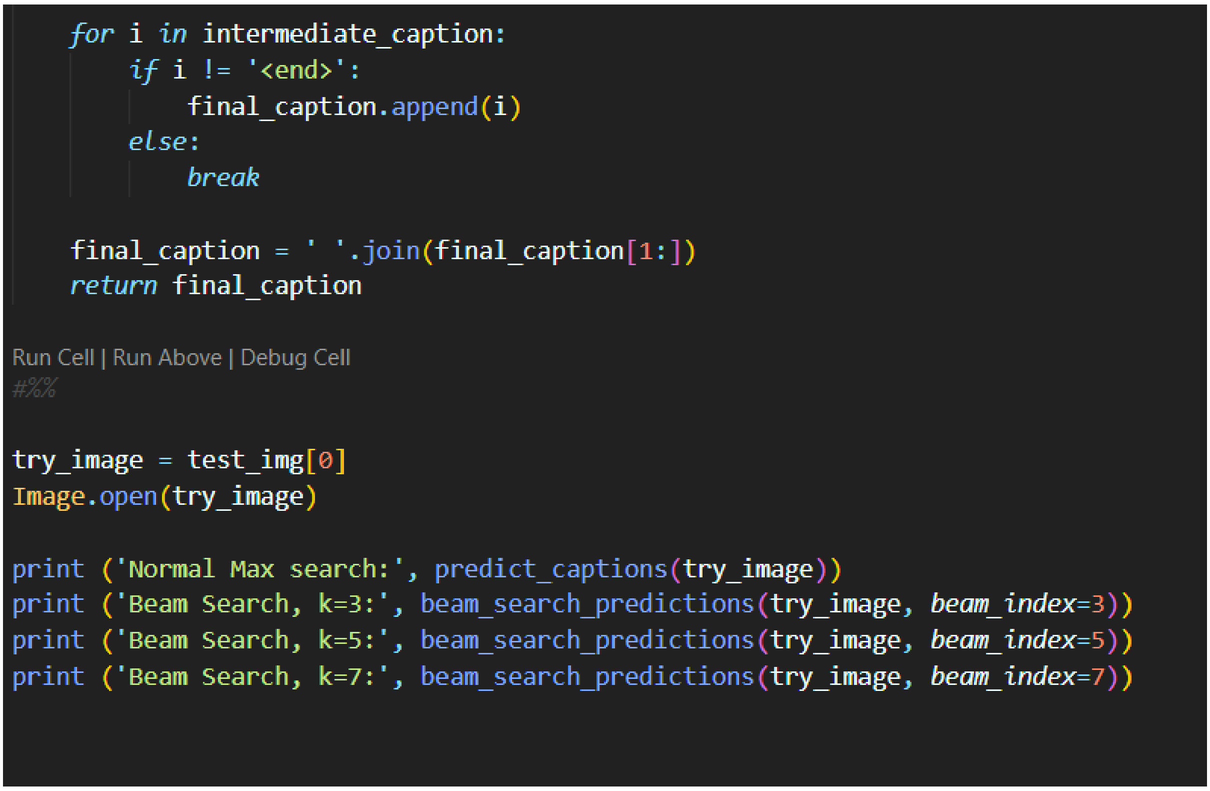
```python
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input, Dropout
from keras.layers.merge import add

embedding_size = 300
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(embedding_size, activation='relu')(fe1)
fe3 = RepeatVector(max_len)(fe2)
# sequence model
inputs2 = Input(shape=(max_len,))
se1 = Embedding(vocab_size, embedding_size, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256, return_sequences=True)(se2)
se4 = TimeDistributed(Dense(embedding_size))(se3)
decoder1 = add([fe3, se4])
decoder2 = Bidirectional(LSTM(256, return_sequences=False))(decoder1)
#    decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
# tie it together [image, seq] [word]
final_model = Model(inputs=[inputs1, inputs2], outputs=outputs)
final_model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
# summarize model
print(final_model.summary())
plot_model(final_model, to_file='model.png', show_shapes=True)
```



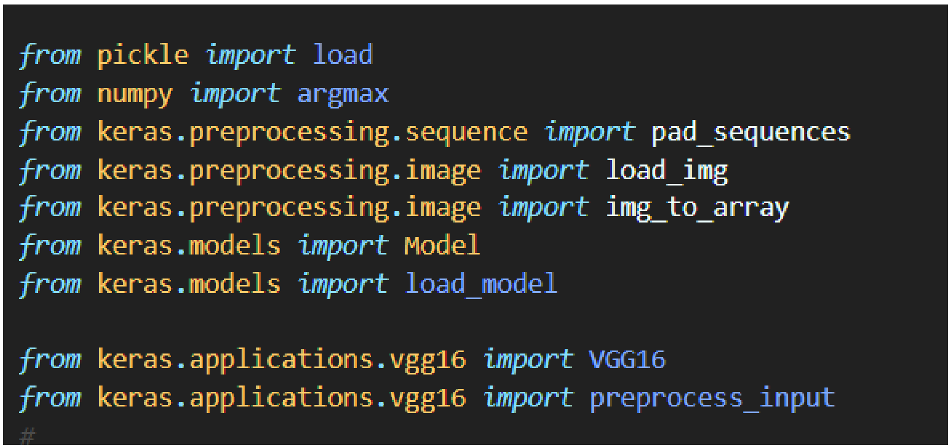
```python
#for epoch in range(10):
final_model.fit_generator(data_generator(batch_size=128), samples_per_epoch=samples_per_epoch, nb_epoch=1, verbose=2)

Run Cell | Run Above | Debug Cell
#%% IF you cannot train it, load the weights
#from keras.models import load_model
#final_model = load_model('time_inceptionV3_1.5987_loss.h5')

#final_model.load_weights('time_inceptionV3_1.5987_loss.h5')

Run Cell | Run Above | Debug Cell
#%% Prediction function

def predict_captions(image):
    start_word = ["<start>"]
    while True:
        par_caps = [word2idx[i] for i in start_word]
        par_caps = sequence.pad_sequences([par_caps], maxlen=max_len, padding='post')
        e = encoding_test[image[len(images):]]
        preds = final_model.predict([np.array([e]), np.array(par_caps)])
        word_pred = idx2word[np.argmax(preds[0])]
        start_word.append(word_pred)

        if word_pred == "<end>" or len(start_word) > max_len:
            break

    return ' '.join(start_word[1:-1])
```



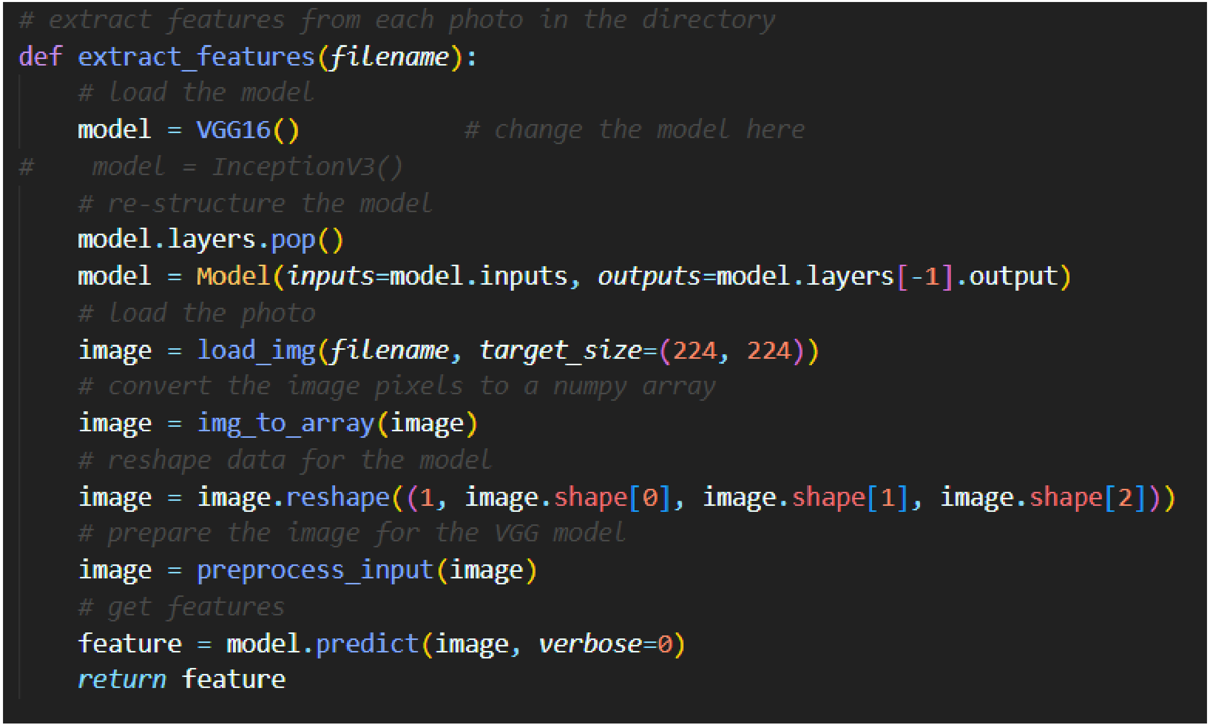
```
def beam_search_predictions(image, beam_index = 3):
    start = [word2idx["<start>"]]

    start_word = [[start, 0.0]]

    while len(start_word[0][0]) < max_len:
        temp = []
        for s in start_word:
            par_caps = sequence.pad_sequences([s[0]], maxlen=max_len, padding='post')
            e = encoding_test[image[len(images):]]
            preds = final_model.predict([np.array([e]), np.array(par_caps)])

            word_preds = np.argsort(preds[0])[-beam_index:]

            # Getting the top <beam_index>(n) predictions and creating a
            # new list so as to put them via the model again
            for w in word_preds:
                next_cap, prob = s[0][:], s[1]
                next_cap.append(w)
                prob += preds[0][w]
                temp.append([next_cap, prob])

        start_word = temp
        # Sorting according to the probabilities
        start_word = sorted(start_word, reverse=False, key=lambda l: l[1])
        # Getting the top words
        start_word = start_word[-beam_index:]

    start_word = start_word[-1][0]
    intermediate_caption = [idx2word[i] for i in start_word]

    final_caption = []
```



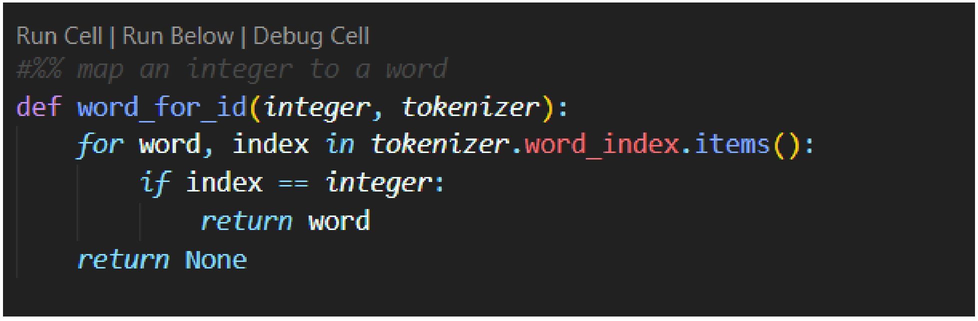
```
    for i in intermediate_caption:
        if i != '<end>':
            final_caption.append(i)
        else:
            break

    final_caption = ' '.join(final_caption[1:])
    return final_caption

Run Cell | Run Above | Debug Cell
#%%

try_image = test_img[0]
Image.open(try_image)

print ('Normal Max search:', predict_captions(try_image))
print ('Beam Search, k=3:', beam_search_predictions(try_image, beam_index=3))
print ('Beam Search, k=5:', beam_search_predictions(try_image, beam_index=5))
print ('Beam Search, k=7:', beam_search_predictions(try_image, beam_index=7))
```



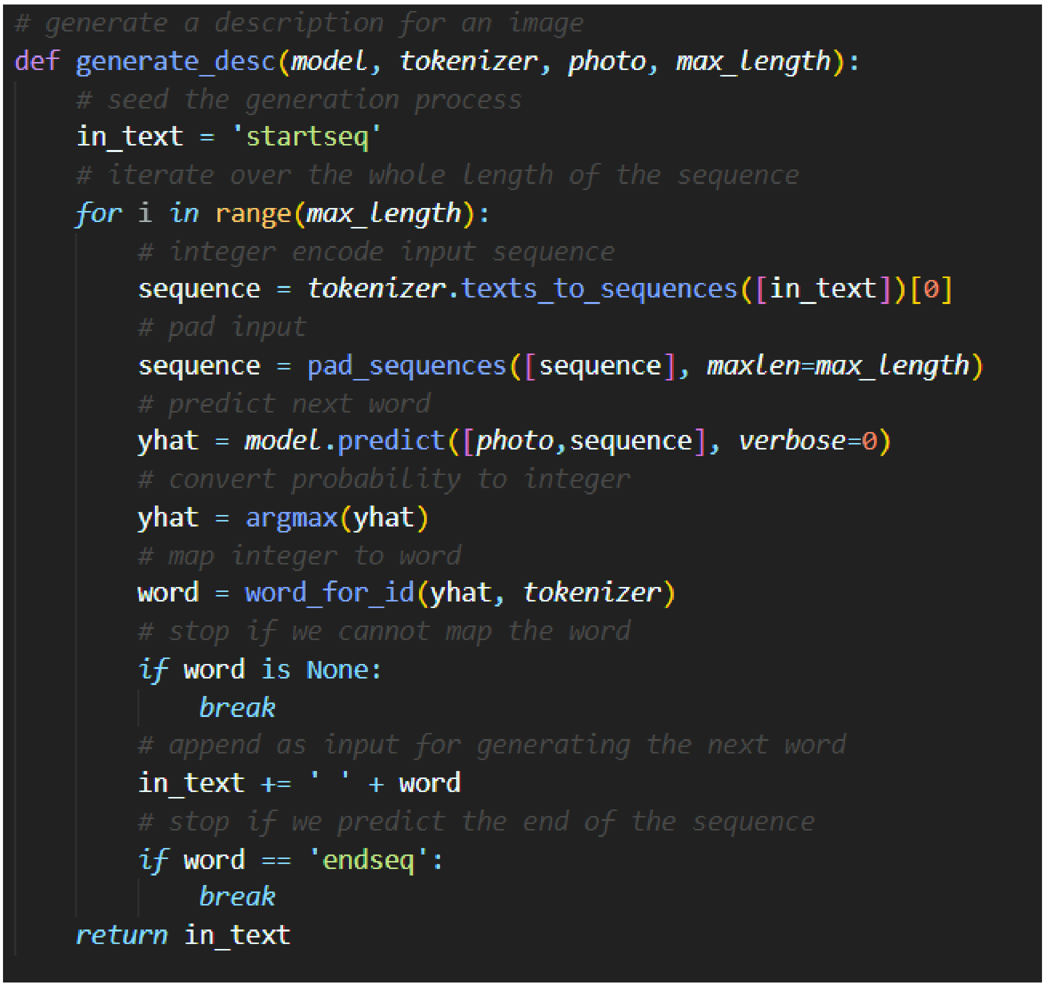
```
from pickle import load
from numpy import argmax
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import Model
from keras.models import load_model

from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
#
```



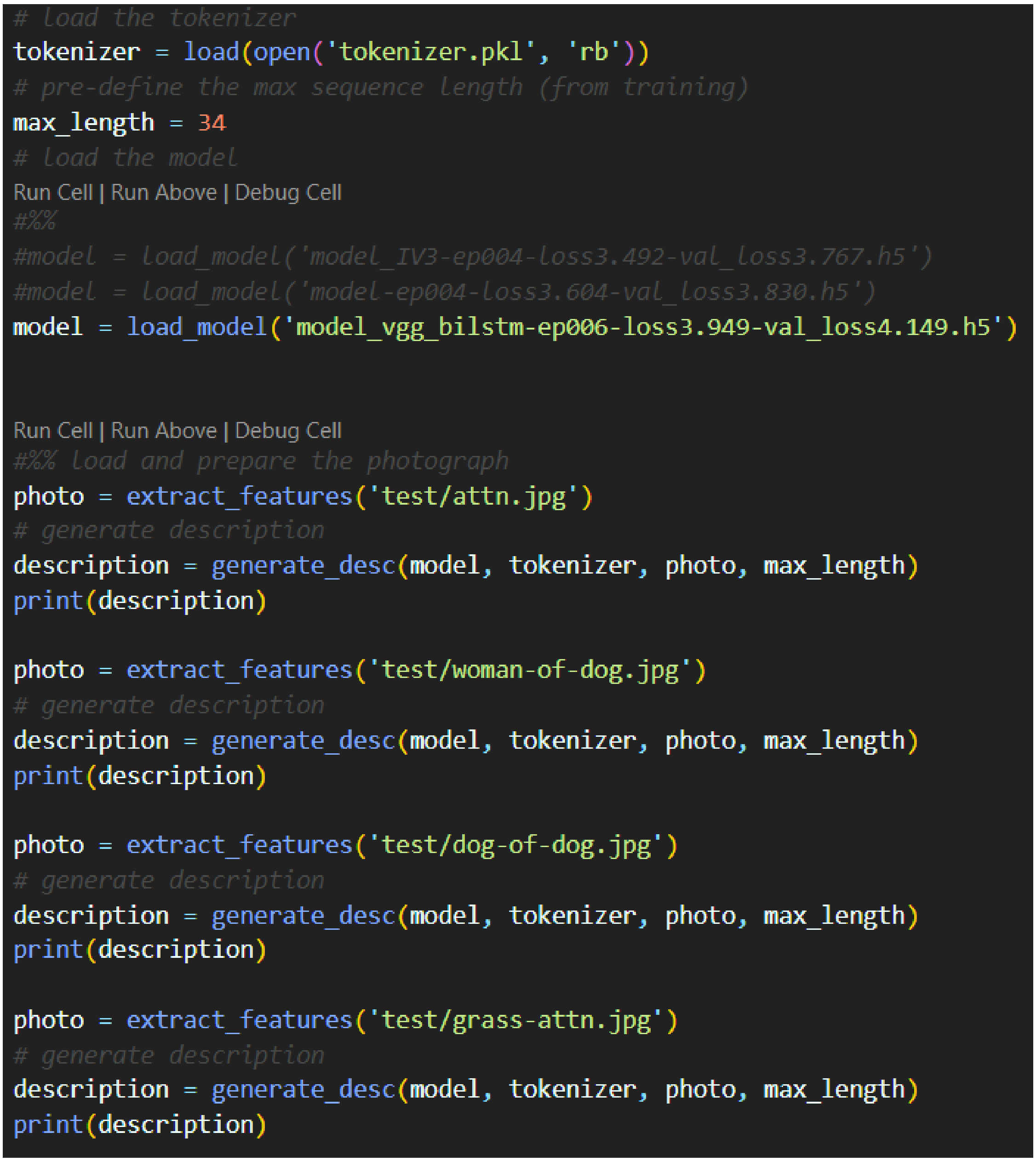
```
# extract features from each photo in the directory
def extract_features(filename):
    # load the model
    model = VGG16()          # change the model here
#    model = InceptionV3()
    # re-structure the model
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    # load the photo
    image = load_img(filename, target_size=(224, 224))
    # convert the image pixels to a numpy array
    image = img_to_array(image)
    # reshape data for the model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # prepare the image for the VGG model
    image = preprocess_input(image)
    # get features
    feature = model.predict(image, verbose=0)
    return feature
```



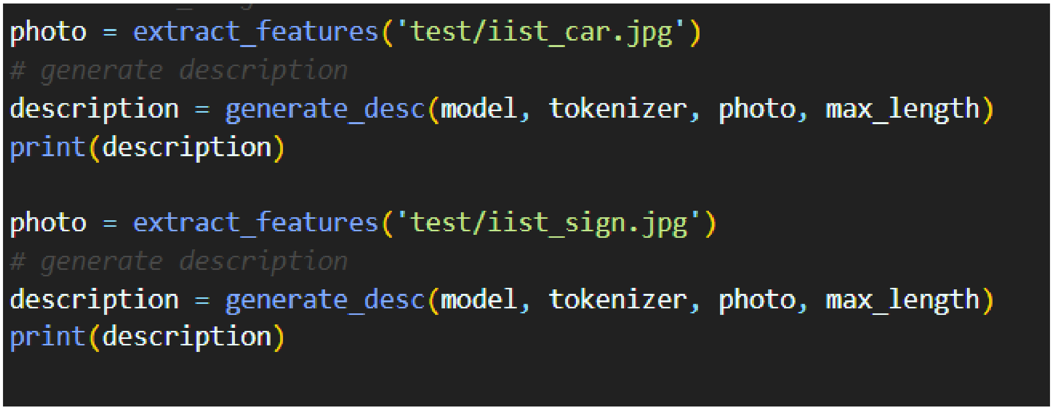
```
#%% map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```



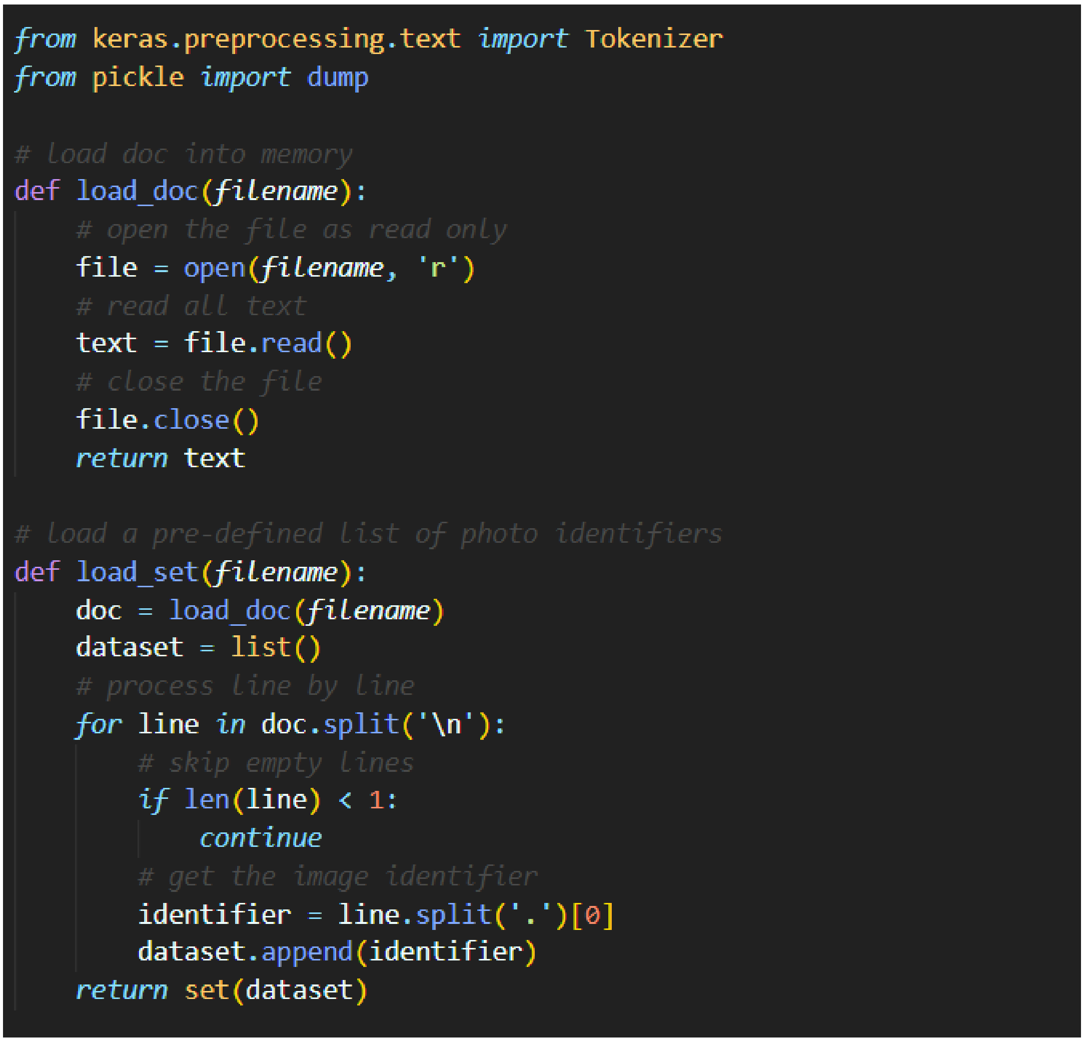
```python
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'startseq'
    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
        # predict next word
        yhat = model.predict([photo,sequence], verbose=0)
        # convert probability to integer
        yhat = argmax(yhat)
        # map integer to word
        word = word_for_id(yhat, tokenizer)
        # stop if we cannot map the word
        if word is None:
            break
        # append as input for generating the next word
        in_text += ' ' + word
        # stop if we predict the end of the sequence
        if word == 'endseq':
            break
    return in_text
```



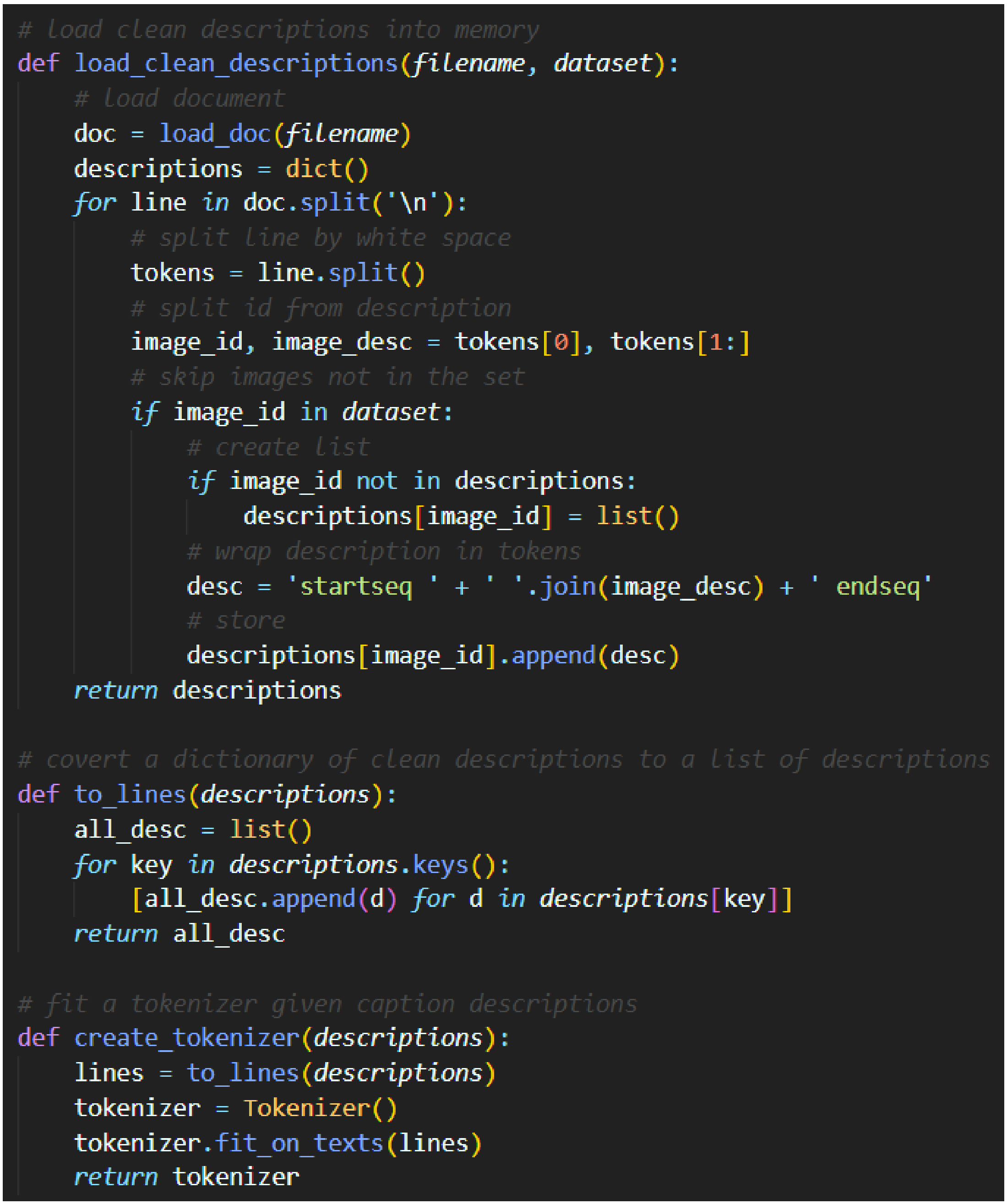
```python
# load the tokenizer
tokenizer = load(open('tokenizer.pkl', 'rb'))
# pre-define the max sequence length (from training)
max_length = 34
# load the model
Run Cell | Run Above | Debug Cell
#%%
#model = load_model('model_IV3-ep004-loss3.492-val_loss3.767.h5')
#model = load_model('model-ep004-loss3.604-val_loss3.830.h5')
model = load_model('model_vgg_bilstm-ep006-loss3.949-val_loss4.149.h5')


Run Cell | Run Above | Debug Cell
#%% load and prepare the photograph
photo = extract_features('test/attn.jpg')
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)

photo = extract_features('test/woman-of-dog.jpg')
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)

photo = extract_features('test/dog-of-dog.jpg')
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)

photo = extract_features('test/grass-attn.jpg')
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```



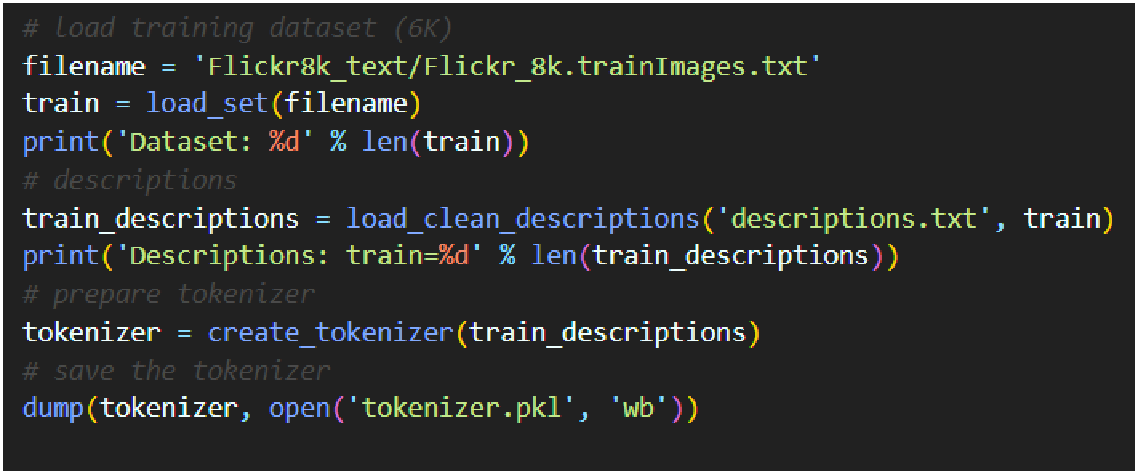
```python
photo = extract_features('test/iist_car.jpg')
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)

photo = extract_features('test/iist_sign.jpg')
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```



```python
from keras.preprocessing.text import Tokenizer
from pickle import dump

# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text

# load a pre-defined list of photo identifiers
def load_set(filename):
    doc = load_doc(filename)
    dataset = list()
    # process line by line
    for line in doc.split('\n'):
        # skip empty lines
        if len(line) < 1:
            continue
        # get the image identifier
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)
```



```python
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    # load document
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        # split id from description
        image_id, image_desc = tokens[0], tokens[1:]
        # skip images not in the set
        if image_id in dataset:
            # create list
            if image_id not in descriptions:
                descriptions[image_id] = list()
            # wrap description in tokens
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            # store
            descriptions[image_id].append(desc)
    return descriptions

# covert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer
```



```python
# load training dataset (6K)
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
# save the tokenizer
dump(tokenizer, open('tokenizer.pkl', 'wb'))
```

4. References

Agrawal, H.; Desai, K.; Chen, X.; Jain, R.; Batra, D.; Parikh, D.; Lee, S.; Anderson, P. Nocaps: Novel object captioning at scale. arXiv 2018, arXiv:1812.08658.

Aneja, J., Deshpande, A., & Schwing, A. G. (2018). Convolutional image captioning. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 5561-5570).

Barnard, K., Duygulu, P., Forsyth, D., De Freitas, N., Blei, D.M. and Jordan, M.I., 2003. Matching words and pictures. The Journal of Machine Learning Research, 3, pp.1107-1135

Chen, X., & Lawrence Zitnick, C. (2015). Mind's eye: A recurrent visual representation for image caption generation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2422-2431).

Chen, Y.C., Li, L., Yu, L., El Kholy, A., Ahmed, F., Gan, Z., Cheng, Y. and Liu, J., 2020, August. Uniter: Universal image-text representation learning. In European conference on computer vision (pp. 104-120). Cham: Springer International Publishing

Elliott, D. and Kádár, A., 2017. Imagination improves multimodal translation. arXiv preprint arXiv:1705.04350

Elliott, D. and Kádár, A., 2017. Imagination improves multimodal translation. arXiv preprint arXiv:1705.04350

Elliott, D., & Keller, F. (2013, October). Image description using visual dependency representations. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (pp. 1292-1302).

Elliott, D., Frank, S., Barrault, L., Bougares, F. and Specia, L., 2017. Findings of the second shared task on multimodal machine translation and multilingual image description. arXiv preprint arXiv:1710.07177

Elliott, D., Frank, S., Sima'an, K. and Specia, L., 2016. Multi30k: Multilingual english-german image descriptions. arXiv preprint arXiv:1605.00459

Fan, C.; Crandall, D.J. Deep Diary: Automatic caption generation for lifelogging image streams. In Proceedings of the European Conference on Computer Vision, Amsterdam, the Netherlands, 11–14 October 2016; pp. 459–473. [CrossRef]