This Document will give you the explanation of the code, that how it works:

The Flickr_8k_text folder that we downloaded contains file Flickr8k.token which is the main file of our dataset that contains image name and their respective captions separated by newline ("\n").

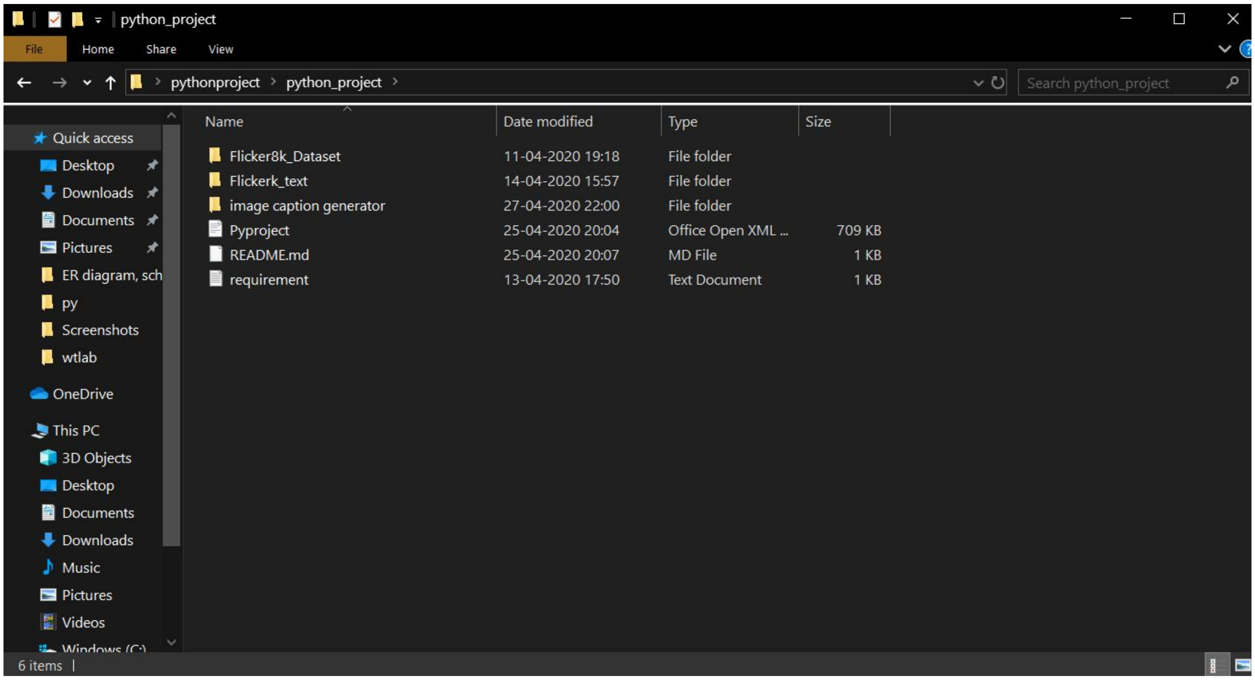
- **Flicker8k_Dataset –** Dataset folder which contains 8091 images.
- **Flickr_8k_text –** Dataset folder which contains text files and captions of images.

The below files will be created while making the project:-

- **Models –** It will contain our trained models.
- **Descriptions.txt –** This text file contains all image names and their captions after preprocessing.
- **Features.p –** Pickle object that contains an image and their feature vector extracted from the Xception pre-trained CNN model.
- **Tokenizer.p –** Contains tokens mapped with an index value.
- **Model.png –** Visual representation of dimensions of our project.
- **Testing_caption_generator.py –** Python file for generating a caption of any image.
- **Training_caption_generator.ipynb –** Jupyter notebook in which we train and build our image caption generator.

The Python programming language also uses the .p file extension. These P files store Python module files that have been converted into byte streams.

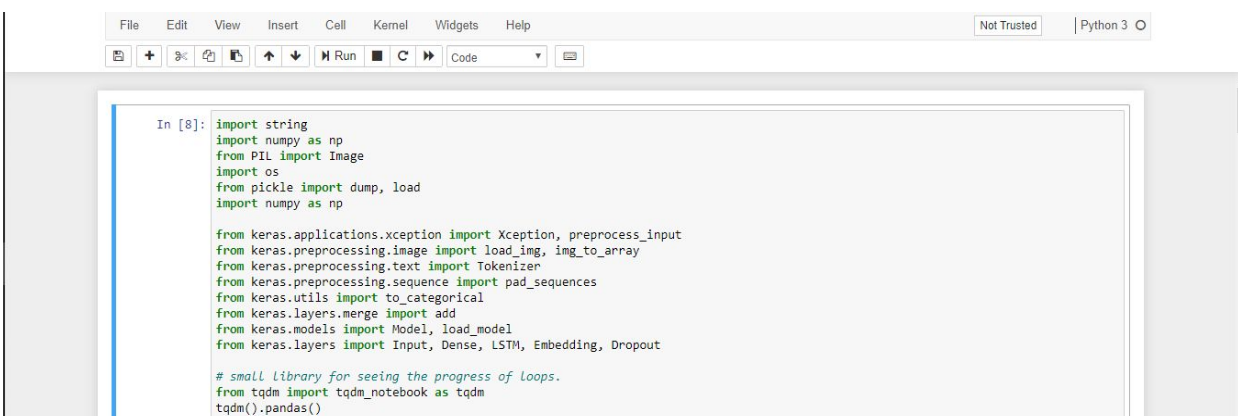
After downloading the folder would look like this:



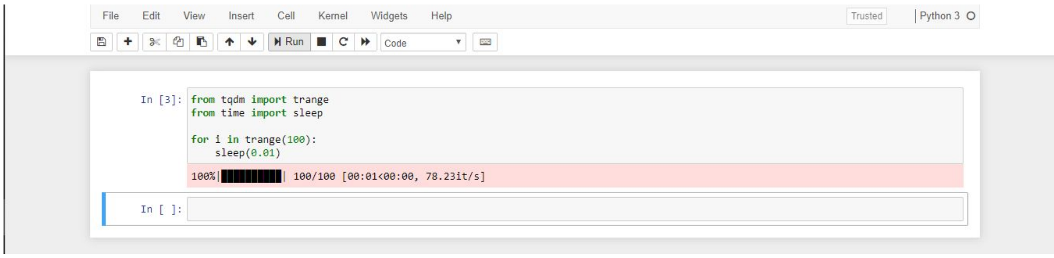
Open training_caption_generator.ipynb in the jupyter notebook:

ABOUT THE CODE:

1. First, we import all the necessary packages



Here the tdqm is the small library that shows the progress bar or say progress of the code you run.

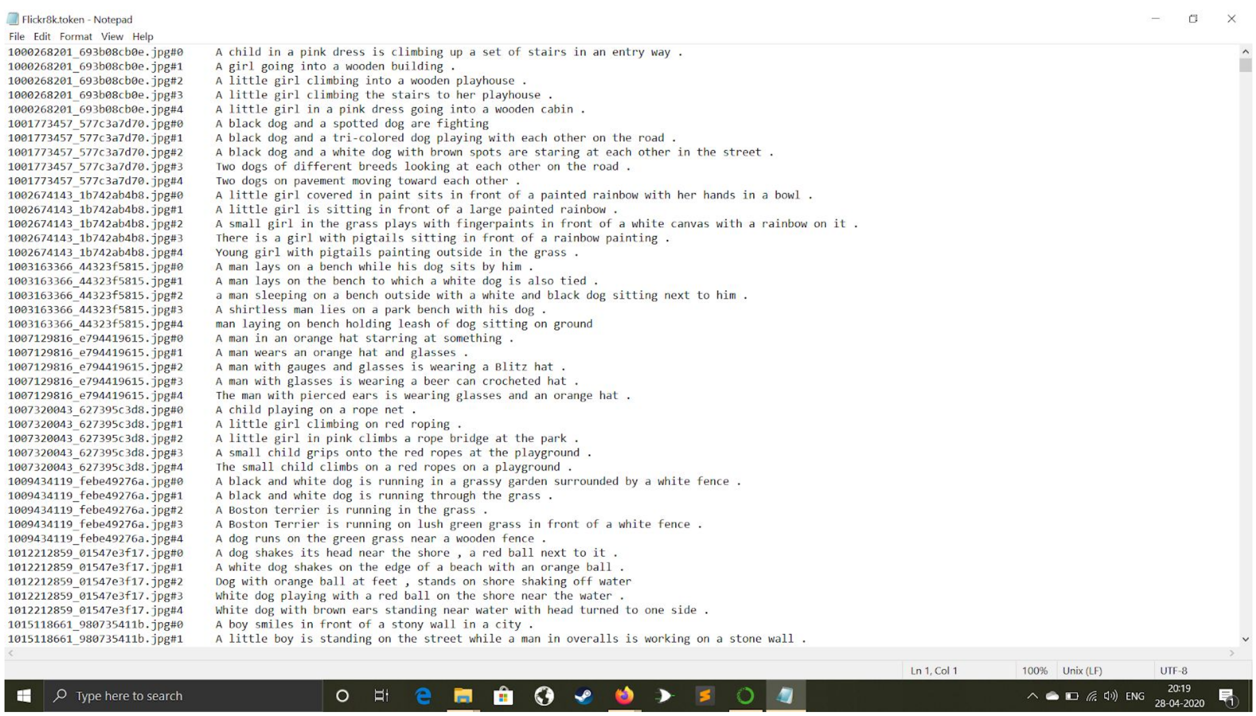
For example: 

2. Getting and performing data cleaning

The main text file which contains all image captions is **Flickr8k.token** in our **Flickr_8k_text** folder.

Have a look at the file –

Here for each image 5 captions(#0 to #4) have been written as shown in the screenshot below:



```
1000268201_693b08cb0e.jpg#0	A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg#1	A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2	A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3	A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4	A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg#0	A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg#1	A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg#2	A black dog and a white dog with brown spots are staring at each other in the street .
1001773457_577c3a7d70.jpg#3	Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg#4	Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg#0	A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .
1002674143_1b742ab4b8.jpg#1	A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg#2	A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
1002674143_1b742ab4b8.jpg#3	There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg#4	Young girl with pigtails painting outside in the grass .
1003163366_44323f5815.jpg#0	A man lays on a bench while his dog sits by him .
1003163366_44323f5815.jpg#1	A man lays on the bench to which a white dog is also tied .
1003163366_44323f5815.jpg#2	a man sleeping on a bench outside with a white and black dog sitting next to him .
1003163366_44323f5815.jpg#3	A shirtless man lies on a park bench with his dog .
1003163366_44323f5815.jpg#4	man laying on bench holding leash of dog sitting on ground
1007129816_e794419615.jpg#0	A man in an orange hat starring at something .
1007129816_e794419615.jpg#1	A man wears an orange hat and glasses .
1007129816_e794419615.jpg#2	A man with gauges and glasses is wearing a Blitz hat .
1007129816_e794419615.jpg#3	A man with glasses is wearing a beer can crocheted hat .
1007129816_e794419615.jpg#4	The man with pierced ears is wearing glasses and an orange hat .
1007320043_627395c3d8.jpg#0	A child playing on a rope net .
1007320043_627395c3d8.jpg#1	A little girl climbing on red roping .
1007320043_627395c3d8.jpg#2	A little girl in pink climbs a rope bridge at the park .
1007320043_627395c3d8.jpg#3	A small child grips onto the red ropes at the playground .
1007320043_627395c3d8.jpg#4	The small child climbs on a red ropes on a playground .
1009434119_febe49276a.jpg#0	A black and white dog is running in a grassy garden surrounded by a white fence .
1009434119_febe49276a.jpg#1	A black and white dog is running through the grass .
1009434119_febe49276a.jpg#2	A Boston terrier is running in the grass .
1009434119_febe49276a.jpg#3	A Boston Terrier is running on lush green grass in front of a white fence .
1009434119_febe49276a.jpg#4	A dog runs on the green grass near a wooden fence .
1012212859_01547e3f17.jpg#0	A dog shakes its head near the shore , a red ball next to it .
1012212859_01547e3f17.jpg#1	A white dog shakes on the edge of a beach with an orange ball .
1012212859_01547e3f17.jpg#2	Dog with orange ball at feet , stands on shore shaking off water
1012212859_01547e3f17.jpg#3	White dog playing with a red ball on the shore near the water .
1012212859_01547e3f17.jpg#4	White dog with brown ears standing near water with head turned to one side .
1015118661_980735411b.jpg#0	A boy smiles in front of a stony wall in a city .
1015118661_980735411b.jpg#1	A little boy is standing on the street while a man in overalls is working on a stone wall .
```

We will define 5 functions:

- **load_doc(filename) –** For loading the document file and reading the contents inside the file into a string.

```
In [1]: # Loading a text file into memory
def load_doc(filename):
    # Opening the file as read only
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text
```

- **all_img_captions(filename)** – This function will create a **descriptions** dictionary that maps images with a list of 5 captions.

```
In [ ]: def all_img_captions(filename):
    file = load_doc(filename)
    captions = file.split('\n')
    descriptions ={}
    for caption in captions[:-1]:
        img, caption = caption.split('\t')
        if img[:-2] not in descriptions:
            descriptions[img[:-2]] = [caption]
        else:
            descriptions[img[:-2]].append(caption)
    return descriptions
```

The descriptions dictionary will look something like this:

- **cleaning_text(descriptions)** – This function takes all descriptions and performs data cleaning. This is an important step when we work with textual data, according to our goal, we decide what type of cleaning we want to perform on the text. In our case, we will be removing punctuations, converting all text to lowercase and removing words that contain numbers. So, a caption like "A man riding on a three-wheeled wheelchair" will be transformed into "man riding on three wheeled wheelchair"

Here in this code:

table = str.maketrans('','',string.punctuation)
is used for removing the punctuation marks in the string of our descriptions.

for i,img_caption in enumerate(caps):
A lot of times when dealing with iterators, we also get a need to keep a count of iterations. Python eases the programmers' task by providing a built-in function enumerate() for this task.
Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.

For Example:

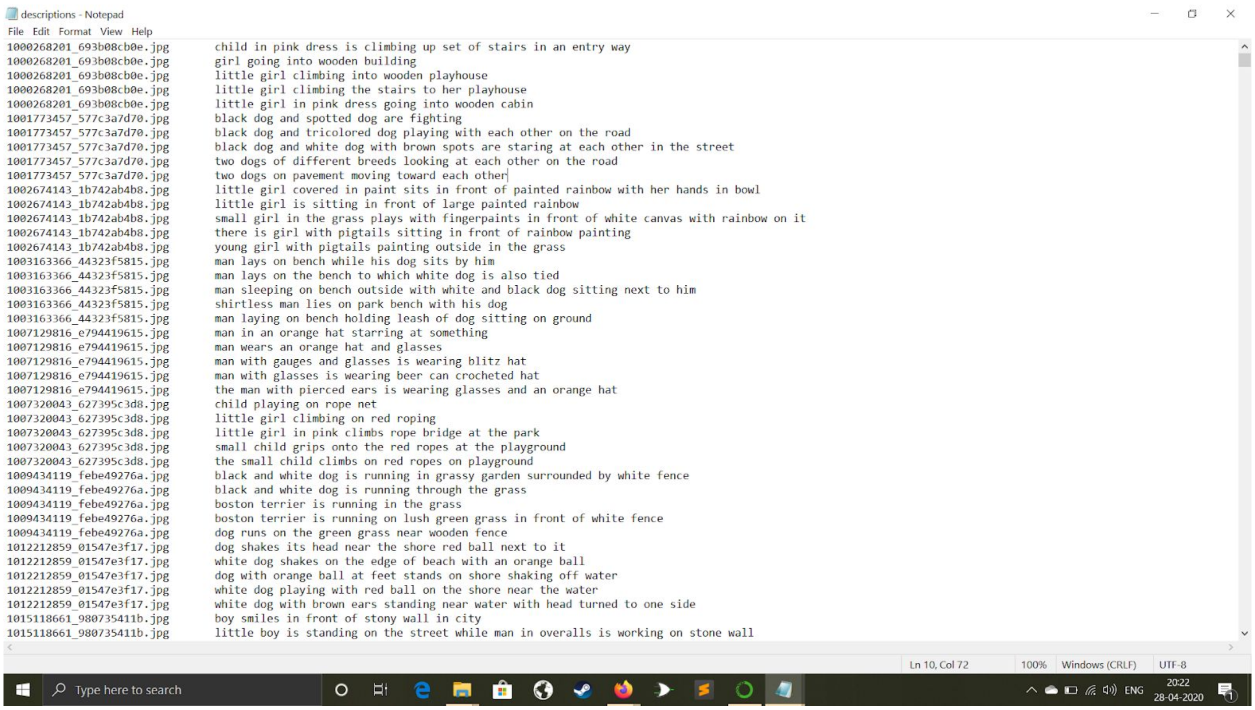
```
In [8]: l1 = ["eat","sleep","repeat"]
        s1 = "abhi"

        obj1 = enumerate(l1)
        obj2 = enumerate(s1)

        print (list(enumerate(l1)))
        print (list(enumerate(s1,2)))

        [(0, 'eat'), (1, 'sleep'), (2, 'repeat')]
        [(2, 'a'), (3, 'b'), (4, 'h'), (5, 'i')]
```

And here in the for loop the following work is done:

- converts to lower case
- remove punctuation from each token
- remove hanging 's and a
- remove tokens with numbers in them
- convert back to string

- **text_vocabulary(descriptions) –** This is a simple function that will separate all the unique words and create the vocabulary from all the descriptions.
- **save_descriptions(descriptions, filename) –** This function will create a list of all the descriptions that have been preprocessed and store them into a file. We will create a descriptions.txt file to store all the captions.

  It will look something like this:


descriptions - Notepad

File Edit Format View Help

```
1000268201_693b08cb0e.jpg	child in pink dress is climbing up set of stairs in an entry way
1000268201_693b08cb0e.jpg	girl going into wooden building
1000268201_693b08cb0e.jpg	little girl climbing into wooden playhouse
1000268201_693b08cb0e.jpg	little girl climbing the stairs to her playhouse
1000268201_693b08cb0e.jpg	little girl in pink dress going into wooden cabin
1001773457_577c3a7d70.jpg	black dog and spotted dog are fighting
1001773457_577c3a7d70.jpg	black dog and tricolored dog playing with each other on the road
1001773457_577c3a7d70.jpg	black dog and white dog with brown spots are staring at each other in the street
1001773457_577c3a7d70.jpg	two dogs of different breeds looking at each other on the road
1001773457_577c3a7d70.jpg	two dogs on pavement moving toward each other
1002674143_1b742ab4b8.jpg	little girl covered in paint sits in front of painted rainbow with her hands in bowl
1002674143_1b742ab4b8.jpg	little girl is sitting in front of large painted rainbow
1002674143_1b742ab4b8.jpg	small girl in the grass plays with fingerpaints in front of white canvas with rainbow on it
1002674143_1b742ab4b8.jpg	there is girl with pigtails sitting in front of rainbow painting
1002674143_1b742ab4b8.jpg	young girl with pigtails painting outside in the grass
1003163366_44323f5815.jpg	man lays on bench while his dog sits by him
1003163366_44323f5815.jpg	man lays on the bench to which white dog is also tied
1003163366_44323f5815.jpg	man sleeping on bench outside with white and black dog sitting next to him
1003163366_44323f5815.jpg	shirtless man lies on park bench with his dog
1003163366_44323f5815.jpg	man laying on bench holding leash of dog sitting on ground
1007129816_e794419615.jpg	man in an orange hat starring at something
1007129816_e794419615.jpg	man wears an orange hat and glasses
1007129816_e794419615.jpg	man with gauges and glasses is wearing blitz hat
1007129816_e794419615.jpg	man with glasses is wearing beer can crocheted hat
1007129816_e794419615.jpg	the man with pierced ears is wearing glasses and an orange hat
1007320043_627395c3d8.jpg	child playing on rope net
1007320043_627395c3d8.jpg	little girl climbing on red roping
1007320043_627395c3d8.jpg	little girl in pink climbs rope bridge at the park
1007320043_627395c3d8.jpg	small child grips onto the red ropes at the playground
1007320043_627395c3d8.jpg	the small child climbs on red ropes on playground
1009434119_febe49276a.jpg	black and white dog is running in grassy garden surrounded by white fence
1009434119_febe49276a.jpg	black and white dog is running through the grass
1009434119_febe49276a.jpg	boston terrier is running in the grass
1009434119_febe49276a.jpg	boston terrier is running on lush green grass in front of white fence
1009434119_febe49276a.jpg	dog runs on the green grass near wooden fence
1012212859_01547e3f17.jpg	dog shakes its head near the shore red ball next to it
1012212859_01547e3f17.jpg	white dog shakes on the edge of beach with an orange ball
1012212859_01547e3f17.jpg	dog with orange ball at feet stands on shore shaking off water
1012212859_01547e3f17.jpg	white dog playing with red ball on the shore near the water
1012212859_01547e3f17.jpg	white dog with brown ears standing near water with head turned to one side
1015118661_980735411b.jpg	boy smiles in front of stony wall in city
1015118661_980735411b.jpg	little boy is standing on the street while man in overalls is working on stone wall
```

Ln 10, Col 72 100% Windows (CRLF) UTF-8

3. Extracting the feature vector from all images

This technique is also called transfer learning, we don't have to do everything on our own, we use the pre-trained model that have been already trained on large datasets and extract the features from these models and use them for our tasks. We are using the Xception model which has been trained on imagenet dataset that had 1000 different classes to classify. We can directly import this model from the keras.applications . Make sure you are connected to the internet as the weights get automatically downloaded. Since the Xception model was originally built for imagenet, we will do little changes for integrating with our model. One thing to notice is that the Xception model takes 299*299*3 image size as input. We will remove the last classification layer and get the 2048 feature vector.

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

The function **extract_features()** will extract features for all images and we will map image names with their respective feature array. Then we will dump the features dictionary into a "features.p" pickle file.

We already discussed in starting about the pickle file.

```
In [28]:
def extract_features(directory):
        model = Xception( include_top=False, pooling='avg' )
        features = {}
        for img in tqdm(os.listdir(directory)):
            filename = directory + "/" + img
            image = Image.open(filename)
            image = image.resize((299,299))
            image = np.expand_dims(image, axis=0)
            #image = preprocess_input(image)
            image = image/127.5
            image = image - 1.0

            feature = model.predict(image)
            features[img] = feature
        return features
```

```
In [29]: #2048 feature vector
features = extract_features(dataset_images)
dump(features, open("features.p","wb"))
```

```
In [19]: features = load(open("features.p","rb"))
```

Here dumping the data in the file you will see the progress bar which is due to use of TDQM.

NOTE:- dumping may take a time(may be 15 to 20 mins) as it is a huge dataset of 8000 pictures. CPU might take upto an hour.

4. Loading dataset for Training the model

In our **Flickr_8k_test** folder, we have **Flickr_8k.trainImages.txt** file that contains a list of 6000 image names that we will use for training.
For loading the training dataset, we need more functions:

- **load_photos(filename) –** This will load the text file in a string and will return the list of image names.
- **load_clean_descriptions(filename, photos) –** This function will create a dictionary that contains captions for each photo from the list of photos. We also append the <start> and <end> identifier for each caption. We need this so that our LSTM model can identify the starting and ending of the caption.
- **load_features(photos) –** This function will give us the dictionary for image names and their feature vector which we have previously extracted from the Xception model.

In [20]:

```python
#Load the data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos


def load_clean_descriptions(filename, photos):
    #loading clean_descriptions
    file = load_doc(filename)
    descriptions = {}
    for line in file.split("\n"):

        words = line.split()
        if len(words)<1 :
            continue

        image, image_caption = words[0], words[1:]

        if image in photos:
            if image not in descriptions:
                descriptions[image] = []
            desc = '<start> ' + " ".join(image_caption) + ' <end>'
            descriptions[image].append(desc)

    return descriptions


def load_features(photos):
    #loading all features
    all_features = load(open("features.p","rb"))
    #selecting only needed features
    features = {k:all_features[k] for k in photos}
    return features
```

In [21]:

```python
filename = dataset_text + "/" + "Flickr_8k.trainImages.txt"

#train = loading_data(filename)
train_imgs = load_photos(filename)
train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)
train_features = load_features(train_imgs)
```

5. Tokenizing the vocabulary

Computers don’t understand English words, for computers, we will have to represent them with numbers. So, we will map each word of the vocabulary with a unique index value. Keras library provides us with the tokenizer function that we will use to create tokens from our vocabulary and save them to a **“tokenizer.p”** pickle file.

Well let me give a better understanding of tokenizer by a sample code runned here:

```
from keras.preprocessing.text import Tokenizer
tok = Tokenizer()
tok.fit_on_texts(["this comment is not toxic"])
print(tok.texts_to_sequences(["this comment is not toxic"]))
print(tok.texts_to_sequences(["this very long comment is not toxic"]))
```

This gives the following output

```
Using TensorFlow backend.
[[1, 2, 3, 4, 5]]
[[1, 2, 3, 4, 5]]
```

In other words unknown words are skipped.

CODE:

```
In [22]: #converting dictionary to clean list of descriptions
         def dict_to_list(descriptions):
             all_desc = []
             for key in descriptions.keys():
                 [all_desc.append(d) for d in descriptions[key]]
             return all_desc

         #creating tokenizer class
         #this will vectorise text corpus
         #each integer will represent token in dictionary

         from keras.preprocessing.text import Tokenizer

         def create_tokenizer(descriptions):
             desc_list = dict_to_list(descriptions)
             tokenizer = Tokenizer()
             tokenizer.fit_on_texts(desc_list)
             return tokenizer
```

```
In [23]: # give each word a index, and store that into tokenizer.p pickle file
         tokenizer = create_tokenizer(train_descriptions)
         dump(tokenizer, open('tokenizer.p', 'wb'))
         vocab_size = len(tokenizer.word_index) + 1
         vocab_size

Out[23]: 7577
```

Our vocabulary contains 7577 words.

We calculate the maximum length of the descriptions. This is important for deciding the model structure parameters. Max_length of description is 32.

6. Create Data generator

Let us first see how the input and output of our model will look like. To make this task into a supervised learning task, we have to provide input and output to the model for training. We have to train our model on 6000 images and each image will contain 2048 length feature vector and caption is also represented as numbers. This amount of data for 6000 images is not possible to hold into memory so we will be using a generator method that will yield batches.

The generator will yield the input and output sequence.

For example:

The input to our model is [x1, x2] and the output will be y, where x1 is the 2048 feature vector of that image, x2 is the input text sequence and y is the output text sequence that the model has to predict.

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

Lets see the vector of the image first:

```
In [25]: features['1000268201_693b08cb0e.jpg'][0]

Out[25]: array([0.36452794, 0.12713662, 0.0013574 , ..., 0.221817  , 0.01178991,
                0.24176797], dtype=float32)
```

Define the model

1 Photo feature extractor - we extracted features from pretrained model Xception.

2 Sequence processor - word embedding layer that handles text, followed by LSTM

3 Decoder - Both 1 and 2 model produce fixed length vector. They are merged together and processed by dense layer to make final prediction

CODE:

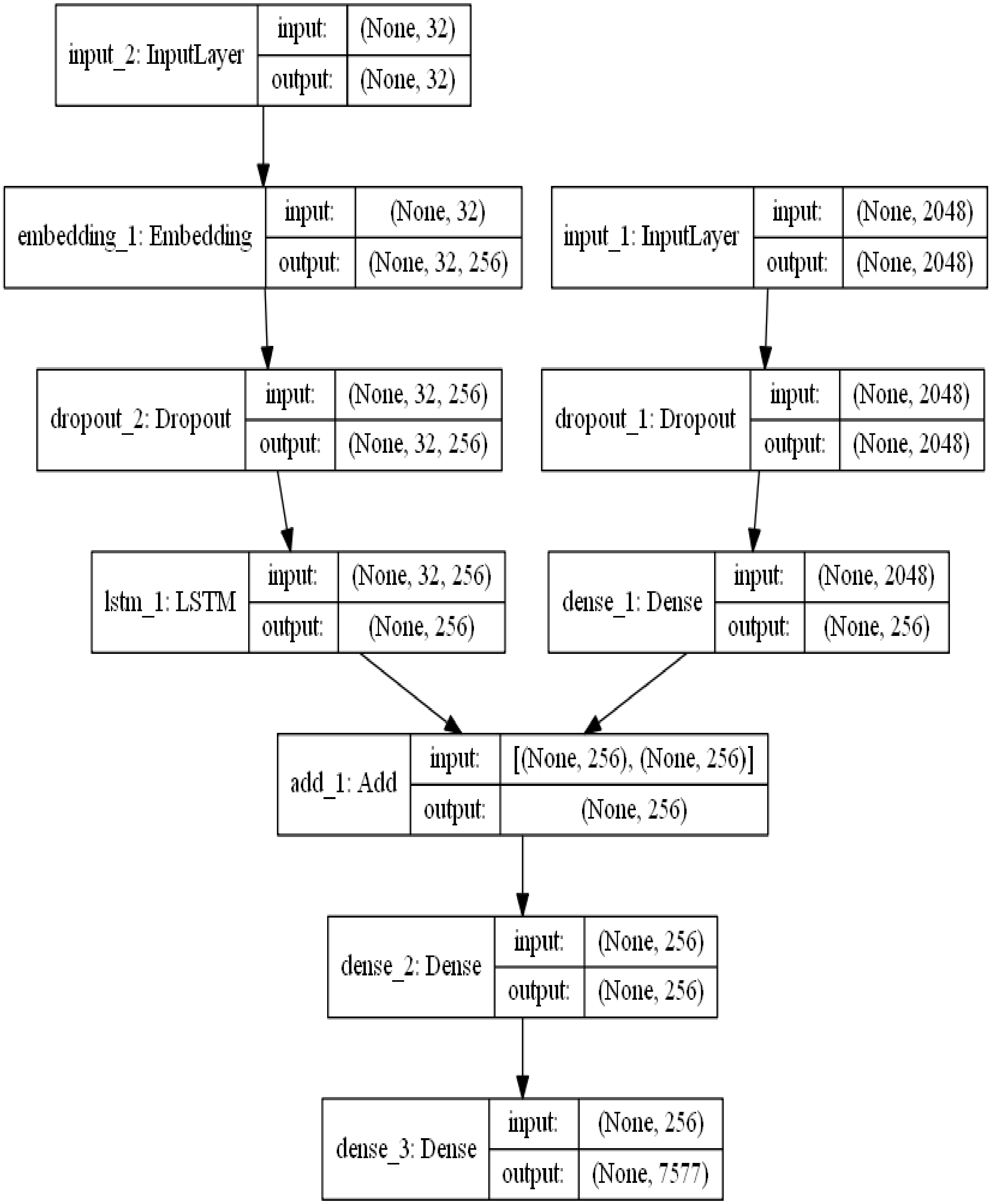
```
In [27]: #create input-output sequence pairs from the image description.

         #data generator, used by model.fit_generator()
         def data_generator(descriptions, features, tokenizer, max_length):
             while 1:
                 for key, description_list in descriptions.items():
                     #retrieve photo features
                     feature = features[key][0]
                     input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, feature)
                     yield [[input_image, input_sequence], output_word]

         def create_sequences(tokenizer, max_length, desc_list, feature):
             X1, X2, y = list(), list(), list()
             # walk through each description for the image
             for desc in desc_list:
                 # encode the sequence
                 seq = tokenizer.texts_to_sequences([desc])[0]
                 # split one sequence into multiple X,y pairs
                 for i in range(1, len(seq)):
                     # split into input and output pair
                     in_seq, out_seq = seq[:i], seq[i]
                     # pad input sequence
                     in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                     # encode output sequence
                     out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                     # store
                     X1.append(feature)
                     X2.append(in_seq)
                     y.append(out_seq)
             return np.array(X1), np.array(X2), np.array(y)

In [28]: [a,b],c = next(data_generator(train_descriptions, features, tokenizer, max_length))
         a.shape, b.shape, c.shape

Out[28]: ((47, 2048), (47, 32), (47, 7577))
```

7. Defining the CNN-RNN model

To define the structure of the model, we will be using the Keras Model from Functional API. It will consist of three major parts:

- **Feature Extractor –** The feature extracted from the image has a size of 2048, with a dense layer, we will reduce the dimensions to 256 nodes.
- **Sequence Processor –** An embedding layer will handle the textual input, followed by the LSTM layer.
- **Decoder –** By merging the output from the above two layers, we will process by the dense layer to make the final prediction. The final layer will contain the number of nodes equal to our vocabulary size.

Virtual representation of the model is given below:-



```python
from keras.utils import plot_model

# define the captioning model
def define_model(vocab_size, max_length):

    # features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    # LSTM sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

    # Merging both models
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)

    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    # summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)

    return model
```

8. Training the model

To train the model, we will be using the 6000 training images by generating the input and output sequences in batches and fitting them to the model using model.fit_generator() method. We also save the model to our models folder. This will take some time depending on your system capability.

```
In [30]: # train our model
print('Dataset: ', len(train_imgs))
print('Descriptions: train=', len(train_descriptions))
print('Photos: train=', len(train_features))
print('Vocabulary Size:', vocab_size)
print('Description Length: ', max_length)

model = define_model(vocab_size, max_length)
epochs = 10
steps = len(train_descriptions)
# making a directory models to save our models
os.mkdir("models")
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
    model.fit_generator(generator, epochs=1, steps_per_epoch= steps, verbose=1)
    model.save("models/model_" + str(i) + ".h5")
```

```
Dataset:  6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length:  32
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_2 (InputLayer)             (None, 32)            0
____________________________________________________________________________________________________
input_1 (InputLayer)             (None, 2048)          0
____________________________________________________________________________________________________
embedding_1 (Embedding)          (None, 32, 256)       1939712     input_2[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 2048)          0           input_1[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 32, 256)       0           embedding_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 256)           524544      dropout_1[0][0]
```

9. Testing the model

The model has been trained, now, we will make a **separate file testing_caption_generator.py** which will load the model and generate predictions. The predictions contain the max length of index values so we will use the same tokenizer.p pickle file to get the words from their index values.

```python
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.applications.xception import Xception
from keras.models import load_model
from pickle import load
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import argparse


ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required=True, help="Image Path")
args = vars(ap.parse_args())
img_path = args['image']

def extract_features(filename, model):
        try:
            image = Image.open(filename)

        except:
            print("ERROR: Couldn't open image! Make sure the image path and extension is correct")
        image = image.resize((299,299))
        image = np.array(image)
        # for images that has 4 channels, we convert them into 3 channels
        if image.shape[2] == 4:
            image = image[..., :3]
        image = np.expand_dims(image, axis=0)
        image = image/127.5
        image = image - 1.0
        feature = model.predict(image)
        return feature

def word_for_id(integer, tokenizer):
 for word, index in tokenizer.word_index.items():
     if index == integer:
         return word
 return None


def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'start'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        pred = model.predict([photo,sequence], verbose=0)
        pred = np.argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'end':
            break
    return in_text


#path = 'Flicker8k_Dataset/111537222_07e56d5a30.jpg'
max_length = 32
tokenizer = load(open("tokenizer.p","rb"))
model = load_model('models/model_9.h5')
xception_model = Xception(include_top=False, pooling="avg")

photo = extract_features(img_path, xception_model)
img = Image.open(img_path)

description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

Sorry for some blur images as I have taken screenshot of my screen and cropped them so.

Hope u enjoy this as it has approx 70% accuracy for generating a perfect caption and just think that if processing just 6000 images takes such huge amount of time then google that actually process billions of such databases ,how hard and still precise they are.