https://colab.research.google.com/drive/1CMR6sfjV6Do8MGeJn9FxLkL7-hiYjW3R#scrollTo=3AC-BruL1YdG (https://colab.research.google.com/drive/1CMR6sfjV6Do8MGeJn9FxLkL7-hiYjW3R#scrollTo=3AC-BruL1YdG)

Image Captioning with Conditioned LSTM Generators

New Section

Follow the instructions in this notebook step-by step. Much of the code is provided, but some sections are marked with **todo**.

Specifically, you will build the following components:

- Create matrices of image representations using an off-the-shelf image encoder.
- Read and preprocess the image captions.
- Write a generator function that returns one training instance (input/output sequence pair) at a time.
- Train an LSTM language generator on the caption data.
- Write a decoder function for the language generator.
- Add the image input to write an LSTM caption generator.

Please submit a copy of this notebook only, including all outputs. Do not submit any of the data files. due to Oct 18 11.59 am There is a 24 hours grace time and no submission accept after grace time

Getting Started

First, run the following commands to make sure you have all required packages.

In [1]:
```
import os
from collections import defaultdict
import numpy as np
import PIL
from matplotlib import pyplot as plt
%matplotlib inline
import string

from keras import Sequential, Model
from keras.layers import Embedding, LSTM, Dense, Input, Bidirectional, RepeatVector, Concatenate, Activation
from keras.activations import softmax
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences

from keras.applications.inception_v3 import InceptionV3

from keras.optimizers import Adam

from google.colab import drive
```

Access to the flickr8k data

We will use the flickr8k data set, described here in more detail:

> M. Hodosh, P. Young and J. Hockenmaier (2013) "Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics", Journal of Artificial Intelligence Research, Volume 47, pages 853-899 http://www.jair.org/papers/paper3994.html (http://www.jair.org/papers/paper3994.html) when discussing our results

I have uploaded all the data and model files you'll need to my GDrive and you can access the folder here: https://drive.google.com/drive/folders/1i9Iun4h3EN1vSd1A1woez0mXJ9vRjFIT?usp=sharing (https://drive.google.com/drive/folders/1i9Iun4h3EN1vSd1A1woez0mXJ9vRjFIT?usp=sharing)

Google Drive does not allow to copy a folder, so you'll need to download the whole folder and then upload it again to your own drive. Please assign the name you chose for this folder to the variable `my_data_dir` in the next cell.

N.B.: Usage of this data is limited to this homework assignment. If you would like to experiment with the data set beyond this course, I suggest that you submit your owndownload request here: https://forms.illinois.edu/sec/1713398 (https://forms.illinois.edu/sec/1713398)

In [2]:
```
#this is where you put the name of your data folder.
#Please make sure it's correct because it'll be used in many places later.
my_data_dir="datahw"
```

Mounting your GDrive so you can access the files from Colab

In [3]:
```
#running this command will generate a message that will ask you to click on a link where you'll obtain your GDrive auth code.
#copy paste that code in the text box that will appear below
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [4]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Please look at the 'Files' tab on the left side and make sure you can see the 'hw5_data' folder that you have in your GDrive.

Part I: Image Encodings (2.5 pts)

The files Flickr_8k.trainImages.txt Flickr_8k.devImages.txt Flickr_8k.testImages.txt, contain a list of training, development, and test images, respectively. Let's load these lists.

In [5]:
```
def load_image_list(filename):
    with open(filename,'r') as image_list_f:
        return [line.strip() for line in image_list_f]
```

In [6]:
```
train_list = load_image_list('/content/drive/My Drive/'+my_data_dir+'/Flickr_8k.trainImages.txt')
dev_list = load_image_list('/content/drive/My Drive/'+my_data_dir+'/Flickr_8k.devImages.txt')
test_list = load_image_list('/content/drive/My Drive/'+my_data_dir+'/Flickr_8k.testImages.txt')
```

Let's see how many images there are

In [7]:
```
len(train_list), len(dev_list), len(test_list)
```

Out[7]:
```
(6000, 1000, 1000)
```

Each entry is an image filename.

In [8]:
```
dev_list[20]
```

Out[8]:
```
'3693961165_9d6c333d5b.jpg'
```

The images are located in a subdirectory.

In [9]:
```
IMG_PATH = '/content/drive/My Drive/'+my_data_dir+'/Flickr8k_Dataset'
```

We can use PIL to open the image and matplotlib to display it.

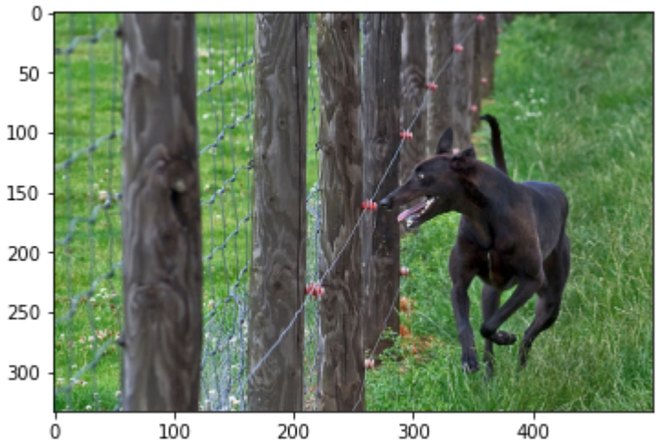
In [10]:
```
image = PIL.Image.open(os.path.join(IMG_PATH, dev_list[20]))
image
```

Out[10]:

if you can't see the image, try

In [11]:
```
plt.imshow(image)
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x7f0292c29748>
```



We are going to use an off-the-shelf pre-trained image encoder, the Inception V3 network. The model is a version of a convolution neural network for object detection. Here is more detail about this model (not required for this project):

> Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826). https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.html (https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.html)

The model requires that input images are presented as 299x299 pixels, with 3 color channels (RGB). The individual RGB values need to range between 0 and 1.0. The flickr images don't fit.

In [12]:
```
np.asarray(image).shape
```

Out[12]:
```
(333, 500, 3)
```

The values range from 0 to 255.

In [13]:
```
np.asarray(image)
```

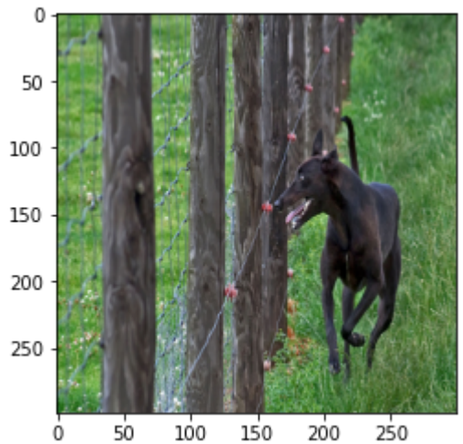
Out[13]:
```
array([[[118, 161,  89],
        [120, 164,  89],
        [111, 157,  82],
        ...,
        [ 68, 106,  65],
        [ 64, 102,  61],
        [ 65, 104,  60]],

       [[125, 168,  96],
        [121, 164,  92],
        [119, 165,  90],
        ...,
        [ 72, 115,  72],
        [ 65, 108,  65],
        [ 72, 115,  70]],

       [[129, 175, 102],
        [123, 169,  96],
        [115, 161,  88],
        ...,
        [ 88, 129,  87],
        [ 75, 116,  72],
        [ 75, 116,  72]],

       ...,

       [[ 41, 118,  46],
        [ 36, 113,  41],
        [ 45, 111,  49],
        ...,
        [ 23,  77,  15],
        [ 60, 114,  62],
        [ 19,  59,   0]],

       [[100, 158,  97],
        [ 38, 100,  37],
        [ 46, 117,  51],
        ...,
        [ 25,  54,   8],
        [ 88, 112,  76],
        [ 65, 106,  48]],

       [[ 89, 148,  84],
```

```
        [ 44, 112,  35],
        [ 71, 130,  72],
        ...,
        [152, 188, 142],
        [113, 151, 110],
        [ 94, 138,  75]]], dtype=uint8)
```

We can use PIL to resize the image and then divide every value by 255.

In [14]:
```
new_image = np.asarray(image.resize((299,299))) / 255.0
plt.imshow(new_image)
```

Out[14]:
```
<matplotlib.image.AxesImage at 0x7f0292702240>
```



In [15]:
```
new_image.shape
```

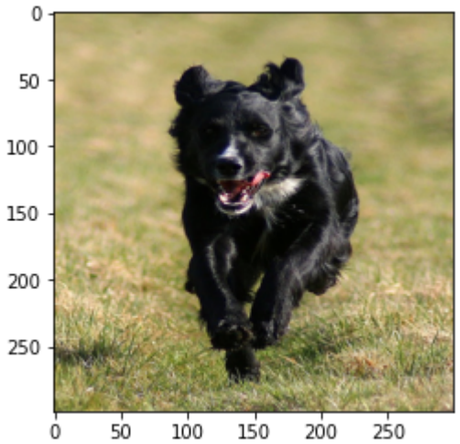
Out[15]:
```
(299, 299, 3)
```

Let's put this all in a function for convenience.

In [16]:
```
def get_image(image_name):
    image = PIL.Image.open(os.path.join(IMG_PATH, image_name))
    return np.asarray(image.resize((299,299))) / 255.0
```

In [17]:
```
plt.imshow(get_image(dev_list[25]))
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x7f02926e77b8>
```



Next, we load the pre-trained Inception model.

In [18]:
```
img_model = InceptionV3(weights='imagenet') # This will download the weight files for you and might take a while.
```

In [19]:
```
img_model.summary() # this is quite a complex model.
```

Model: "inception_v3"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 299, 299, 3)	0	
conv2d (Conv2D)	(None, 149, 149, 32)	864	input_1[0][0]
batch_normalization (BatchNorma	(None, 149, 149, 32)	96	conv2d[0][0]
activation (Activation)	(None, 149, 149, 32)	0	batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 147, 147, 32)	9216	activation[0][0]
batch_normalization_1 (BatchNor	(None, 147, 147, 32)	96	conv2d_1[0][0]
activation_1 (Activation)	(None, 147, 147, 32)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 147, 147, 64)	18432	activation_1[0][0]
batch_normalization_2 (BatchNor	(None, 147, 147, 64)	192	conv2d_2[0][0]
activation_2 (Activation)	(None, 147, 147, 64)	0	batch_normalization_2[0][0]
max_pooling2d (MaxPooling2D)	(None, 73, 73, 64)	0	activation_2[0][0]
conv2d_3 (Conv2D)	(None, 73, 73, 80)	5120	max_pooling2d[0][0]
batch_normalization_3 (BatchNor	(None, 73, 73, 80)	240	conv2d_3[0][0]
activation_3 (Activation)	(None, 73, 73, 80)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 71, 71, 192)	138240	activation_3[0][0]
batch_normalization_4 (BatchNor	(None, 71, 71, 192)	576	conv2d_4[0][0]
activation_4 (Activation)	(None, 71, 71, 192)	0	batch_normalization_4[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 35, 35, 192)	0	activation_4[0][0]
conv2d_8 (Conv2D)	(None, 35, 35, 64)	12288	max_pooling2d_1[0][0]
batch_normalization_8 (BatchNor	(None, 35, 35, 64)	192	conv2d_8[0][0]

activation_8 (Activation)	(None, 35, 35, 64)	0	batch_normalization_8[0][0]
conv2d_6 (Conv2D)	(None, 35, 35, 48)	9216	max_pooling2d_1[0][0]
conv2d_9 (Conv2D)	(None, 35, 35, 96)	55296	activation_8[0][0]
batch_normalization_6 (BatchNor	(None, 35, 35, 48)	144	conv2d_6[0][0]
batch_normalization_9 (BatchNor	(None, 35, 35, 96)	288	conv2d_9[0][0]
activation_6 (Activation)	(None, 35, 35, 48)	0	batch_normalization_6[0][0]
activation_9 (Activation)	(None, 35, 35, 96)	0	batch_normalization_9[0][0]
average_pooling2d (AveragePooli	(None, 35, 35, 192)	0	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 35, 35, 64)	12288	max_pooling2d_1[0][0]
conv2d_7 (Conv2D)	(None, 35, 35, 64)	76800	activation_6[0][0]
conv2d_10 (Conv2D)	(None, 35, 35, 96)	82944	activation_9[0][0]
conv2d_11 (Conv2D)	(None, 35, 35, 32)	6144	average_pooling2d[0][0]
batch_normalization_5 (BatchNor	(None, 35, 35, 64)	192	conv2d_5[0][0]
batch_normalization_7 (BatchNor	(None, 35, 35, 64)	192	conv2d_7[0][0]
batch_normalization_10 (BatchNo	(None, 35, 35, 96)	288	conv2d_10[0][0]
batch_normalization_11 (BatchNo	(None, 35, 35, 32)	96	conv2d_11[0][0]
activation_5 (Activation)	(None, 35, 35, 64)	0	batch_normalization_5[0][0]
activation_7 (Activation)	(None, 35, 35, 64)	0	batch_normalization_7[0][0]
activation_10 (Activation)	(None, 35, 35, 96)	0	batch_normalization_10[0][0]
activation_11 (Activation)	(None, 35, 35, 32)	0	batch_normalization_11[0][0]
mixed0 (Concatenate)	(None, 35, 35, 256)	0	activation_5[0][0] activation_7[0][0]

			activation_10[0][0] activation_11[0][0]
conv2d_15 (Conv2D)	(None, 35, 35, 64)	16384	mixed0[0][0]
batch_normalization_15 (BatchNo	(None, 35, 35, 64)	192	conv2d_15[0][0]
activation_15 (Activation)	(None, 35, 35, 64)	0	batch_normalization_15[0][0]
conv2d_13 (Conv2D)	(None, 35, 35, 48)	12288	mixed0[0][0]
conv2d_16 (Conv2D)	(None, 35, 35, 96)	55296	activation_15[0][0]
batch_normalization_13 (BatchNo	(None, 35, 35, 48)	144	conv2d_13[0][0]
batch_normalization_16 (BatchNo	(None, 35, 35, 96)	288	conv2d_16[0][0]
activation_13 (Activation)	(None, 35, 35, 48)	0	batch_normalization_13[0][0]
activation_16 (Activation)	(None, 35, 35, 96)	0	batch_normalization_16[0][0]
average_pooling2d_1 (AveragePoo	(None, 35, 35, 256)	0	mixed0[0][0]
conv2d_12 (Conv2D)	(None, 35, 35, 64)	16384	mixed0[0][0]
conv2d_14 (Conv2D)	(None, 35, 35, 64)	76800	activation_13[0][0]
conv2d_17 (Conv2D)	(None, 35, 35, 96)	82944	activation_16[0][0]
conv2d_18 (Conv2D)	(None, 35, 35, 64)	16384	average_pooling2d_1[0][0]
batch_normalization_12 (BatchNo	(None, 35, 35, 64)	192	conv2d_12[0][0]
batch_normalization_14 (BatchNo	(None, 35, 35, 64)	192	conv2d_14[0][0]
batch_normalization_17 (BatchNo	(None, 35, 35, 96)	288	conv2d_17[0][0]
batch_normalization_18 (BatchNo	(None, 35, 35, 64)	192	conv2d_18[0][0]
activation_12 (Activation)	(None, 35, 35, 64)	0	batch_normalization_12[0][0]
activation_14 (Activation)	(None, 35, 35, 64)	0	batch_normalization_14[0][0]

```
activation_17 (Activation)      (None, 35, 35, 96)   0           batch_normalization_17[0][0]
__________________________________________________________________________________________
activation_18 (Activation)      (None, 35, 35, 64)   0           batch_normalization_18[0][0]
__________________________________________________________________________________________
mixed1 (Concatenate)            (None, 35, 35, 288)  0           activation_12[0][0]
                                                                 activation_14[0][0]
                                                                 activation_17[0][0]
                                                                 activation_18[0][0]
__________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 35, 35, 64)   18432       mixed1[0][0]
__________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, 35, 35, 64)   192         conv2d_22[0][0]
__________________________________________________________________________________________
activation_22 (Activation)      (None, 35, 35, 64)   0           batch_normalization_22[0][0]
__________________________________________________________________________________________
conv2d_20 (Conv2D)              (None, 35, 35, 48)   13824       mixed1[0][0]
__________________________________________________________________________________________
conv2d_23 (Conv2D)              (None, 35, 35, 96)   55296       activation_22[0][0]
__________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 35, 35, 48)   144         conv2d_20[0][0]
__________________________________________________________________________________________
batch_normalization_23 (BatchNo (None, 35, 35, 96)   288         conv2d_23[0][0]
__________________________________________________________________________________________
activation_20 (Activation)      (None, 35, 35, 48)   0           batch_normalization_20[0][0]
__________________________________________________________________________________________
activation_23 (Activation)      (None, 35, 35, 96)   0           batch_normalization_23[0][0]
__________________________________________________________________________________________
average_pooling2d_2 (AveragePoo (None, 35, 35, 288)  0           mixed1[0][0]
__________________________________________________________________________________________
conv2d_19 (Conv2D)              (None, 35, 35, 64)   18432       mixed1[0][0]
__________________________________________________________________________________________
conv2d_21 (Conv2D)              (None, 35, 35, 64)   76800       activation_20[0][0]
__________________________________________________________________________________________
conv2d_24 (Conv2D)              (None, 35, 35, 96)   82944       activation_23[0][0]
__________________________________________________________________________________________
conv2d_25 (Conv2D)              (None, 35, 35, 64)   18432       average_pooling2d_2[0][0]
__________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, 35, 35, 64)   192         conv2d_19[0][0]
__________________________________________________________________________________________
batch_normalization_21 (BatchNo (None, 35, 35, 64)   192         conv2d_21[0][0]
__________________________________________________________________________________________
batch_normalization_24 (BatchNo (None, 35, 35, 96)   288         conv2d_24[0][0]
__________________________________________________________________________________________
```

```
batch_normalization_25 (BatchNo (None, 35, 35, 64)   192         conv2d_25[0][0]
__________________________________________________________________________________________
activation_19 (Activation)      (None, 35, 35, 64)   0           batch_normalization_19[0][0]
__________________________________________________________________________________________
activation_21 (Activation)      (None, 35, 35, 64)   0           batch_normalization_21[0][0]
__________________________________________________________________________________________
activation_24 (Activation)      (None, 35, 35, 96)   0           batch_normalization_24[0][0]
__________________________________________________________________________________________
activation_25 (Activation)      (None, 35, 35, 64)   0           batch_normalization_25[0][0]
__________________________________________________________________________________________
mixed2 (Concatenate)            (None, 35, 35, 288)  0           activation_19[0][0]
                                                                 activation_21[0][0]
                                                                 activation_24[0][0]
                                                                 activation_25[0][0]
__________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, 35, 35, 64)   18432       mixed2[0][0]
__________________________________________________________________________________________
batch_normalization_27 (BatchNo (None, 35, 35, 64)   192         conv2d_27[0][0]
__________________________________________________________________________________________
activation_27 (Activation)      (None, 35, 35, 64)   0           batch_normalization_27[0][0]
__________________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 35, 35, 96)   55296       activation_27[0][0]
__________________________________________________________________________________________
batch_normalization_28 (BatchNo (None, 35, 35, 96)   288         conv2d_28[0][0]
__________________________________________________________________________________________
activation_28 (Activation)      (None, 35, 35, 96)   0           batch_normalization_28[0][0]
__________________________________________________________________________________________
conv2d_26 (Conv2D)              (None, 17, 17, 384)  995328      mixed2[0][0]
__________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 17, 17, 96)   82944       activation_28[0][0]
__________________________________________________________________________________________
batch_normalization_26 (BatchNo (None, 17, 17, 384)  1152        conv2d_26[0][0]
__________________________________________________________________________________________
batch_normalization_29 (BatchNo (None, 17, 17, 96)   288         conv2d_29[0][0]
__________________________________________________________________________________________
activation_26 (Activation)      (None, 17, 17, 384)  0           batch_normalization_26[0][0]
__________________________________________________________________________________________
activation_29 (Activation)      (None, 17, 17, 96)   0           batch_normalization_29[0][0]
__________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 17, 17, 288)  0           mixed2[0][0]
__________________________________________________________________________________________
mixed3 (Concatenate)            (None, 17, 17, 768)  0           activation_26[0][0]
                                                                 activation_29[0][0]
```

			max_pooling2d_2[0][0]
conv2d_34 (Conv2D)	(None, 17, 17, 128)	98304	mixed3[0][0]
batch_normalization_34 (BatchNo	(None, 17, 17, 128)	384	conv2d_34[0][0]
activation_34 (Activation)	(None, 17, 17, 128)	0	batch_normalization_34[0][0]
conv2d_35 (Conv2D)	(None, 17, 17, 128)	114688	activation_34[0][0]
batch_normalization_35 (BatchNo	(None, 17, 17, 128)	384	conv2d_35[0][0]
activation_35 (Activation)	(None, 17, 17, 128)	0	batch_normalization_35[0][0]
conv2d_31 (Conv2D)	(None, 17, 17, 128)	98304	mixed3[0][0]
conv2d_36 (Conv2D)	(None, 17, 17, 128)	114688	activation_35[0][0]
batch_normalization_31 (BatchNo	(None, 17, 17, 128)	384	conv2d_31[0][0]
batch_normalization_36 (BatchNo	(None, 17, 17, 128)	384	conv2d_36[0][0]
activation_31 (Activation)	(None, 17, 17, 128)	0	batch_normalization_31[0][0]
activation_36 (Activation)	(None, 17, 17, 128)	0	batch_normalization_36[0][0]
conv2d_32 (Conv2D)	(None, 17, 17, 128)	114688	activation_31[0][0]
conv2d_37 (Conv2D)	(None, 17, 17, 128)	114688	activation_36[0][0]
batch_normalization_32 (BatchNo	(None, 17, 17, 128)	384	conv2d_32[0][0]
batch_normalization_37 (BatchNo	(None, 17, 17, 128)	384	conv2d_37[0][0]
activation_32 (Activation)	(None, 17, 17, 128)	0	batch_normalization_32[0][0]
activation_37 (Activation)	(None, 17, 17, 128)	0	batch_normalization_37[0][0]
average_pooling2d_3 (AveragePoo	(None, 17, 17, 768)	0	mixed3[0][0]
conv2d_30 (Conv2D)	(None, 17, 17, 192)	147456	mixed3[0][0]
conv2d_33 (Conv2D)	(None, 17, 17, 192)	172032	activation_32[0][0]

conv2d_38 (Conv2D)	(None, 17, 17, 192)	172032	activation_37[0][0]
conv2d_39 (Conv2D)	(None, 17, 17, 192)	147456	average_pooling2d_3[0][0]
batch_normalization_30 (BatchNo	(None, 17, 17, 192)	576	conv2d_30[0][0]
batch_normalization_33 (BatchNo	(None, 17, 17, 192)	576	conv2d_33[0][0]
batch_normalization_38 (BatchNo	(None, 17, 17, 192)	576	conv2d_38[0][0]
batch_normalization_39 (BatchNo	(None, 17, 17, 192)	576	conv2d_39[0][0]
activation_30 (Activation)	(None, 17, 17, 192)	0	batch_normalization_30[0][0]
activation_33 (Activation)	(None, 17, 17, 192)	0	batch_normalization_33[0][0]
activation_38 (Activation)	(None, 17, 17, 192)	0	batch_normalization_38[0][0]
activation_39 (Activation)	(None, 17, 17, 192)	0	batch_normalization_39[0][0]
mixed4 (Concatenate)	(None, 17, 17, 768)	0	activation_30[0][0] activation_33[0][0] activation_38[0][0] activation_39[0][0]
conv2d_44 (Conv2D)	(None, 17, 17, 160)	122880	mixed4[0][0]
batch_normalization_44 (BatchNo	(None, 17, 17, 160)	480	conv2d_44[0][0]
activation_44 (Activation)	(None, 17, 17, 160)	0	batch_normalization_44[0][0]
conv2d_45 (Conv2D)	(None, 17, 17, 160)	179200	activation_44[0][0]
batch_normalization_45 (BatchNo	(None, 17, 17, 160)	480	conv2d_45[0][0]
activation_45 (Activation)	(None, 17, 17, 160)	0	batch_normalization_45[0][0]
conv2d_41 (Conv2D)	(None, 17, 17, 160)	122880	mixed4[0][0]
conv2d_46 (Conv2D)	(None, 17, 17, 160)	179200	activation_45[0][0]
batch_normalization_41 (BatchNo	(None, 17, 17, 160)	480	conv2d_41[0][0]

batch_normalization_46 (BatchNo	(None, 17, 17, 160)	480	conv2d_46[0][0]
activation_41 (Activation)	(None, 17, 17, 160)	0	batch_normalization_41[0][0]
activation_46 (Activation)	(None, 17, 17, 160)	0	batch_normalization_46[0][0]
conv2d_42 (Conv2D)	(None, 17, 17, 160)	179200	activation_41[0][0]
conv2d_47 (Conv2D)	(None, 17, 17, 160)	179200	activation_46[0][0]
batch_normalization_42 (BatchNo	(None, 17, 17, 160)	480	conv2d_42[0][0]
batch_normalization_47 (BatchNo	(None, 17, 17, 160)	480	conv2d_47[0][0]
activation_42 (Activation)	(None, 17, 17, 160)	0	batch_normalization_42[0][0]
activation_47 (Activation)	(None, 17, 17, 160)	0	batch_normalization_47[0][0]
average_pooling2d_4 (AveragePoo	(None, 17, 17, 768)	0	mixed4[0][0]
conv2d_40 (Conv2D)	(None, 17, 17, 192)	147456	mixed4[0][0]
conv2d_43 (Conv2D)	(None, 17, 17, 192)	215040	activation_42[0][0]
conv2d_48 (Conv2D)	(None, 17, 17, 192)	215040	activation_47[0][0]
conv2d_49 (Conv2D)	(None, 17, 17, 192)	147456	average_pooling2d_4[0][0]
batch_normalization_40 (BatchNo	(None, 17, 17, 192)	576	conv2d_40[0][0]
batch_normalization_43 (BatchNo	(None, 17, 17, 192)	576	conv2d_43[0][0]
batch_normalization_48 (BatchNo	(None, 17, 17, 192)	576	conv2d_48[0][0]
batch_normalization_49 (BatchNo	(None, 17, 17, 192)	576	conv2d_49[0][0]
activation_40 (Activation)	(None, 17, 17, 192)	0	batch_normalization_40[0][0]
activation_43 (Activation)	(None, 17, 17, 192)	0	batch_normalization_43[0][0]
activation_48 (Activation)	(None, 17, 17, 192)	0	batch_normalization_48[0][0]

```
activation_49 (Activation)      (None, 17, 17, 192)  0           batch_normalization_49[0][0]
__________________________________________________________________________________________________
mixed5 (Concatenate)            (None, 17, 17, 768)  0           activation_40[0][0]
                                                                 activation_43[0][0]
                                                                 activation_48[0][0]
                                                                 activation_49[0][0]
__________________________________________________________________________________________________
conv2d_54 (Conv2D)              (None, 17, 17, 160)  122880      mixed5[0][0]
__________________________________________________________________________________________________
batch_normalization_54 (BatchNo (None, 17, 17, 160)  480         conv2d_54[0][0]
__________________________________________________________________________________________________
activation_54 (Activation)      (None, 17, 17, 160)  0           batch_normalization_54[0][0]
__________________________________________________________________________________________________
conv2d_55 (Conv2D)              (None, 17, 17, 160)  179200      activation_54[0][0]
__________________________________________________________________________________________________
batch_normalization_55 (BatchNo (None, 17, 17, 160)  480         conv2d_55[0][0]
__________________________________________________________________________________________________
activation_55 (Activation)      (None, 17, 17, 160)  0           batch_normalization_55[0][0]
__________________________________________________________________________________________________
conv2d_51 (Conv2D)              (None, 17, 17, 160)  122880      mixed5[0][0]
__________________________________________________________________________________________________
conv2d_56 (Conv2D)              (None, 17, 17, 160)  179200      activation_55[0][0]
__________________________________________________________________________________________________
batch_normalization_51 (BatchNo (None, 17, 17, 160)  480         conv2d_51[0][0]
__________________________________________________________________________________________________
batch_normalization_56 (BatchNo (None, 17, 17, 160)  480         conv2d_56[0][0]
__________________________________________________________________________________________________
activation_51 (Activation)      (None, 17, 17, 160)  0           batch_normalization_51[0][0]
__________________________________________________________________________________________________
activation_56 (Activation)      (None, 17, 17, 160)  0           batch_normalization_56[0][0]
__________________________________________________________________________________________________
conv2d_52 (Conv2D)              (None, 17, 17, 160)  179200      activation_51[0][0]
__________________________________________________________________________________________________
conv2d_57 (Conv2D)              (None, 17, 17, 160)  179200      activation_56[0][0]
__________________________________________________________________________________________________
batch_normalization_52 (BatchNo (None, 17, 17, 160)  480         conv2d_52[0][0]
__________________________________________________________________________________________________
batch_normalization_57 (BatchNo (None, 17, 17, 160)  480         conv2d_57[0][0]
__________________________________________________________________________________________________
activation_52 (Activation)      (None, 17, 17, 160)  0           batch_normalization_52[0][0]
__________________________________________________________________________________________________
activation_57 (Activation)      (None, 17, 17, 160)  0           batch_normalization_57[0][0]
__________________________________________________________________________________________________
```

```
average_pooling2d_5 (AveragePoo (None, 17, 17, 768)  0           mixed5[0][0]
__________________________________________________________________________________________________
conv2d_50 (Conv2D)              (None, 17, 17, 192)  147456      mixed5[0][0]
__________________________________________________________________________________________________
conv2d_53 (Conv2D)              (None, 17, 17, 192)  215040      activation_52[0][0]
__________________________________________________________________________________________________
conv2d_58 (Conv2D)              (None, 17, 17, 192)  215040      activation_57[0][0]
__________________________________________________________________________________________________
conv2d_59 (Conv2D)              (None, 17, 17, 192)  147456      average_pooling2d_5[0][0]
__________________________________________________________________________________________________
batch_normalization_50 (BatchNo (None, 17, 17, 192)  576         conv2d_50[0][0]
__________________________________________________________________________________________________
batch_normalization_53 (BatchNo (None, 17, 17, 192)  576         conv2d_53[0][0]
__________________________________________________________________________________________________
batch_normalization_58 (BatchNo (None, 17, 17, 192)  576         conv2d_58[0][0]
__________________________________________________________________________________________________
batch_normalization_59 (BatchNo (None, 17, 17, 192)  576         conv2d_59[0][0]
__________________________________________________________________________________________________
activation_50 (Activation)      (None, 17, 17, 192)  0           batch_normalization_50[0][0]
__________________________________________________________________________________________________
activation_53 (Activation)      (None, 17, 17, 192)  0           batch_normalization_53[0][0]
__________________________________________________________________________________________________
activation_58 (Activation)      (None, 17, 17, 192)  0           batch_normalization_58[0][0]
__________________________________________________________________________________________________
activation_59 (Activation)      (None, 17, 17, 192)  0           batch_normalization_59[0][0]
__________________________________________________________________________________________________
mixed6 (Concatenate)            (None, 17, 17, 768)  0           activation_50[0][0]
                                                                 activation_53[0][0]
                                                                 activation_58[0][0]
                                                                 activation_59[0][0]
__________________________________________________________________________________________________
conv2d_64 (Conv2D)              (None, 17, 17, 192)  147456      mixed6[0][0]
__________________________________________________________________________________________________
batch_normalization_64 (BatchNo (None, 17, 17, 192)  576         conv2d_64[0][0]
__________________________________________________________________________________________________
activation_64 (Activation)      (None, 17, 17, 192)  0           batch_normalization_64[0][0]
__________________________________________________________________________________________________
conv2d_65 (Conv2D)              (None, 17, 17, 192)  258048      activation_64[0][0]
__________________________________________________________________________________________________
batch_normalization_65 (BatchNo (None, 17, 17, 192)  576         conv2d_65[0][0]
__________________________________________________________________________________________________
activation_65 (Activation)      (None, 17, 17, 192)  0           batch_normalization_65[0][0]
__________________________________________________________________________________________________
```

```
conv2d_61 (Conv2D)              (None, 17, 17, 192)  147456      mixed6[0][0]
__________________________________________________________________________________________________
conv2d_66 (Conv2D)              (None, 17, 17, 192)  258048      activation_65[0][0]
__________________________________________________________________________________________________
batch_normalization_61 (BatchNo (None, 17, 17, 192)  576         conv2d_61[0][0]
__________________________________________________________________________________________________
batch_normalization_66 (BatchNo (None, 17, 17, 192)  576         conv2d_66[0][0]
__________________________________________________________________________________________________
activation_61 (Activation)      (None, 17, 17, 192)  0           batch_normalization_61[0][0]
__________________________________________________________________________________________________
activation_66 (Activation)      (None, 17, 17, 192)  0           batch_normalization_66[0][0]
__________________________________________________________________________________________________
conv2d_62 (Conv2D)              (None, 17, 17, 192)  258048      activation_61[0][0]
__________________________________________________________________________________________________
conv2d_67 (Conv2D)              (None, 17, 17, 192)  258048      activation_66[0][0]
__________________________________________________________________________________________________
batch_normalization_62 (BatchNo (None, 17, 17, 192)  576         conv2d_62[0][0]
__________________________________________________________________________________________________
batch_normalization_67 (BatchNo (None, 17, 17, 192)  576         conv2d_67[0][0]
__________________________________________________________________________________________________
activation_62 (Activation)      (None, 17, 17, 192)  0           batch_normalization_62[0][0]
__________________________________________________________________________________________________
activation_67 (Activation)      (None, 17, 17, 192)  0           batch_normalization_67[0][0]
__________________________________________________________________________________________________
average_pooling2d_6 (AveragePoo (None, 17, 17, 768)  0           mixed6[0][0]
__________________________________________________________________________________________________
conv2d_60 (Conv2D)              (None, 17, 17, 192)  147456      mixed6[0][0]
__________________________________________________________________________________________________
conv2d_63 (Conv2D)              (None, 17, 17, 192)  258048      activation_62[0][0]
__________________________________________________________________________________________________
conv2d_68 (Conv2D)              (None, 17, 17, 192)  258048      activation_67[0][0]
__________________________________________________________________________________________________
conv2d_69 (Conv2D)              (None, 17, 17, 192)  147456      average_pooling2d_6[0][0]
__________________________________________________________________________________________________
batch_normalization_60 (BatchNo (None, 17, 17, 192)  576         conv2d_60[0][0]
__________________________________________________________________________________________________
batch_normalization_63 (BatchNo (None, 17, 17, 192)  576         conv2d_63[0][0]
__________________________________________________________________________________________________
batch_normalization_68 (BatchNo (None, 17, 17, 192)  576         conv2d_68[0][0]
__________________________________________________________________________________________________
batch_normalization_69 (BatchNo (None, 17, 17, 192)  576         conv2d_69[0][0]
__________________________________________________________________________________________________
activation_60 (Activation)      (None, 17, 17, 192)  0           batch_normalization_60[0][0]
```

activation_63 (Activation)	(None, 17, 17, 192)	0	batch_normalization_63[0][0]
activation_68 (Activation)	(None, 17, 17, 192)	0	batch_normalization_68[0][0]
activation_69 (Activation)	(None, 17, 17, 192)	0	batch_normalization_69[0][0]
mixed7 (Concatenate)	(None, 17, 17, 768)	0	activation_60[0][0] activation_63[0][0] activation_68[0][0] activation_69[0][0]
conv2d_72 (Conv2D)	(None, 17, 17, 192)	147456	mixed7[0][0]
batch_normalization_72 (BatchNo	(None, 17, 17, 192)	576	conv2d_72[0][0]
activation_72 (Activation)	(None, 17, 17, 192)	0	batch_normalization_72[0][0]
conv2d_73 (Conv2D)	(None, 17, 17, 192)	258048	activation_72[0][0]
batch_normalization_73 (BatchNo	(None, 17, 17, 192)	576	conv2d_73[0][0]
activation_73 (Activation)	(None, 17, 17, 192)	0	batch_normalization_73[0][0]
conv2d_70 (Conv2D)	(None, 17, 17, 192)	147456	mixed7[0][0]
conv2d_74 (Conv2D)	(None, 17, 17, 192)	258048	activation_73[0][0]
batch_normalization_70 (BatchNo	(None, 17, 17, 192)	576	conv2d_70[0][0]
batch_normalization_74 (BatchNo	(None, 17, 17, 192)	576	conv2d_74[0][0]
activation_70 (Activation)	(None, 17, 17, 192)	0	batch_normalization_70[0][0]
activation_74 (Activation)	(None, 17, 17, 192)	0	batch_normalization_74[0][0]
conv2d_71 (Conv2D)	(None, 8, 8, 320)	552960	activation_70[0][0]
conv2d_75 (Conv2D)	(None, 8, 8, 192)	331776	activation_74[0][0]
batch_normalization_71 (BatchNo	(None, 8, 8, 320)	960	conv2d_71[0][0]
batch_normalization_75 (BatchNo	(None, 8, 8, 192)	576	conv2d_75[0][0]

activation_71 (Activation)	(None, 8, 8, 320)	0	batch_normalization_71[0][0]
activation_75 (Activation)	(None, 8, 8, 192)	0	batch_normalization_75[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 768)	0	mixed7[0][0]
mixed8 (Concatenate)	(None, 8, 8, 1280)	0	activation_71[0][0] activation_75[0][0] max_pooling2d_3[0][0]
conv2d_80 (Conv2D)	(None, 8, 8, 448)	573440	mixed8[0][0]
batch_normalization_80 (BatchNo	(None, 8, 8, 448)	1344	conv2d_80[0][0]
activation_80 (Activation)	(None, 8, 8, 448)	0	batch_normalization_80[0][0]
conv2d_77 (Conv2D)	(None, 8, 8, 384)	491520	mixed8[0][0]
conv2d_81 (Conv2D)	(None, 8, 8, 384)	1548288	activation_80[0][0]
batch_normalization_77 (BatchNo	(None, 8, 8, 384)	1152	conv2d_77[0][0]
batch_normalization_81 (BatchNo	(None, 8, 8, 384)	1152	conv2d_81[0][0]
activation_77 (Activation)	(None, 8, 8, 384)	0	batch_normalization_77[0][0]
activation_81 (Activation)	(None, 8, 8, 384)	0	batch_normalization_81[0][0]
conv2d_78 (Conv2D)	(None, 8, 8, 384)	442368	activation_77[0][0]
conv2d_79 (Conv2D)	(None, 8, 8, 384)	442368	activation_77[0][0]
conv2d_82 (Conv2D)	(None, 8, 8, 384)	442368	activation_81[0][0]
conv2d_83 (Conv2D)	(None, 8, 8, 384)	442368	activation_81[0][0]
average_pooling2d_7 (AveragePoo	(None, 8, 8, 1280)	0	mixed8[0][0]
conv2d_76 (Conv2D)	(None, 8, 8, 320)	409600	mixed8[0][0]
batch_normalization_78 (BatchNo	(None, 8, 8, 384)	1152	conv2d_78[0][0]

batch_normalization_79 (BatchNo	(None, 8, 8, 384)	1152	conv2d_79[0][0]
batch_normalization_82 (BatchNo	(None, 8, 8, 384)	1152	conv2d_82[0][0]
batch_normalization_83 (BatchNo	(None, 8, 8, 384)	1152	conv2d_83[0][0]
conv2d_84 (Conv2D)	(None, 8, 8, 192)	245760	average_pooling2d_7[0][0]
batch_normalization_76 (BatchNo	(None, 8, 8, 320)	960	conv2d_76[0][0]
activation_78 (Activation)	(None, 8, 8, 384)	0	batch_normalization_78[0][0]
activation_79 (Activation)	(None, 8, 8, 384)	0	batch_normalization_79[0][0]
activation_82 (Activation)	(None, 8, 8, 384)	0	batch_normalization_82[0][0]
activation_83 (Activation)	(None, 8, 8, 384)	0	batch_normalization_83[0][0]
batch_normalization_84 (BatchNo	(None, 8, 8, 192)	576	conv2d_84[0][0]
activation_76 (Activation)	(None, 8, 8, 320)	0	batch_normalization_76[0][0]
mixed9_0 (Concatenate)	(None, 8, 8, 768)	0	activation_78[0][0] activation_79[0][0]
concatenate (Concatenate)	(None, 8, 8, 768)	0	activation_82[0][0] activation_83[0][0]
activation_84 (Activation)	(None, 8, 8, 192)	0	batch_normalization_84[0][0]
mixed9 (Concatenate)	(None, 8, 8, 2048)	0	activation_76[0][0] mixed9_0[0][0] concatenate[0][0] activation_84[0][0]
conv2d_89 (Conv2D)	(None, 8, 8, 448)	917504	mixed9[0][0]
batch_normalization_89 (BatchNo	(None, 8, 8, 448)	1344	conv2d_89[0][0]
activation_89 (Activation)	(None, 8, 8, 448)	0	batch_normalization_89[0][0]
conv2d_86 (Conv2D)	(None, 8, 8, 384)	786432	mixed9[0][0]

conv2d_90 (Conv2D)	(None, 8, 8, 384)	1548288	activation_89[0][0]
batch_normalization_86 (BatchNo	(None, 8, 8, 384)	1152	conv2d_86[0][0]
batch_normalization_90 (BatchNo	(None, 8, 8, 384)	1152	conv2d_90[0][0]
activation_86 (Activation)	(None, 8, 8, 384)	0	batch_normalization_86[0][0]
activation_90 (Activation)	(None, 8, 8, 384)	0	batch_normalization_90[0][0]
conv2d_87 (Conv2D)	(None, 8, 8, 384)	442368	activation_86[0][0]
conv2d_88 (Conv2D)	(None, 8, 8, 384)	442368	activation_86[0][0]
conv2d_91 (Conv2D)	(None, 8, 8, 384)	442368	activation_90[0][0]
conv2d_92 (Conv2D)	(None, 8, 8, 384)	442368	activation_90[0][0]
average_pooling2d_8 (AveragePoo	(None, 8, 8, 2048)	0	mixed9[0][0]
conv2d_85 (Conv2D)	(None, 8, 8, 320)	655360	mixed9[0][0]
batch_normalization_87 (BatchNo	(None, 8, 8, 384)	1152	conv2d_87[0][0]
batch_normalization_88 (BatchNo	(None, 8, 8, 384)	1152	conv2d_88[0][0]
batch_normalization_91 (BatchNo	(None, 8, 8, 384)	1152	conv2d_91[0][0]
batch_normalization_92 (BatchNo	(None, 8, 8, 384)	1152	conv2d_92[0][0]
conv2d_93 (Conv2D)	(None, 8, 8, 192)	393216	average_pooling2d_8[0][0]
batch_normalization_85 (BatchNo	(None, 8, 8, 320)	960	conv2d_85[0][0]
activation_87 (Activation)	(None, 8, 8, 384)	0	batch_normalization_87[0][0]
activation_88 (Activation)	(None, 8, 8, 384)	0	batch_normalization_88[0][0]
activation_91 (Activation)	(None, 8, 8, 384)	0	batch_normalization_91[0][0]
activation_92 (Activation)	(None, 8, 8, 384)	0	batch_normalization_92[0][0]
batch_normalization_93 (BatchNo	(None, 8, 8, 192)	576	conv2d_93[0][0]

```
activation_85 (Activation)      (None, 8, 8, 320)    0           batch_normalization_85[0][0]
__________________________________________________________________________________________________
mixed9_1 (Concatenate)          (None, 8, 8, 768)    0           activation_87[0][0]
                                                                 activation_88[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 8, 8, 768)    0           activation_91[0][0]
                                                                 activation_92[0][0]
__________________________________________________________________________________________________
activation_93 (Activation)      (None, 8, 8, 192)    0           batch_normalization_93[0][0]
__________________________________________________________________________________________________
mixed10 (Concatenate)           (None, 8, 8, 2048)   0           activation_85[0][0]
                                                                 mixed9_1[0][0]
                                                                 concatenate_1[0][0]
                                                                 activation_93[0][0]
__________________________________________________________________________________________________
avg_pool (GlobalAveragePooling2 (None, 2048)         0           mixed10[0][0]
__________________________________________________________________________________________________
predictions (Dense)             (None, 1000)         2049000     avg_pool[0][0]
==================================================================================================
Total params: 23,851,784
Trainable params: 23,817,352
Non-trainable params: 34,432
__________________________________________________________________________________________________
```

This is a prediction model,so the output is typically a softmax-activated vector representing 1000 possible object types. Because we are interested in an encoded representation of the image we are just going to use the second-to-last layer as a source of image encodings. Each image will be encoded as a vector of size 2048.

We will use the following hack: hook up the input into a new Keras model and use the penultimate layer of the existing model as output.

In [20]:
```
new_input = img_model.input
new_output = img_model.layers[-2].output
img_encoder = Model(new_input, new_output) # This is the final Keras image encoder model we will use.
```

Let's try the encoder.

In [21]:
```
encoded_image = img_encoder.predict(np.array([new_image]))
```

In [22]:
```
encoded_image
```

Out[22]:
```
array([[0.63806576, 0.4887299 , 0.05526242, ..., 0.64255714, 0.2959523 ,
        0.4900427 ]], dtype=float32)
```

TODO: We will need to create encodings for all images and store them in one big matrix (one for each dataset, train, dev, test). We can then save the matrices so that we never have to touch the bulky image data again.

To save memory (but slow the process down a little bit) we will read in the images lazily using a generator. We will encounter generators again later when we train the LSTM. If you are unfamiliar with generators, take a look at this page: https://wiki.python.org/moin/Generators (https://wiki.python.org/moin/Generators)

Write the following generator function, which should return one image at a time. `img_list` is a list of image file names (i.e. the train, dev, or test set). The return value should be a numpy array of shape (1,299,299,3).

In [23]:
```
def img_generator(img_list):
    #...
    for image in img_list:
      image = get_image(image)
      encoded_image = img_encoder.predict(np.array([image]))
      pil_image = PIL.Image.fromarray(encoded_image)
      pil_image = pil_image.resize((299, 299))
      pil_image = pil_image.convert("RGB")

      np_image = np.asarray(pil_image)
      np_image = np.expand_dims(np_image, axis = 0)
      yield np_image
    #...
```

Now we can encode all images (this takes a few minutes).

In [ ]:
```
enc_train = img_encoder.predict_generator(img_generator(train_list), steps=len(train_list), verbose=1)
```

```
WARNING:tensorflow:From <ipython-input-41-57cffb83e012>:1: Model.predict_generator (from tensorflow.python.keras.engine.training) is deprecated an
d will be removed in a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
6000/6000 [==============================] - 3510s 585ms/step
```

In [ ]:
```
enc_train[11]
```

Out[ ]:
```
array([1.04641184e-01, 9.12365876e-03, 1.17547631e-01, ...,
       4.96862829e-01, 3.39364313e-04, 1.29856955e-04], dtype=float32)
```

In [ ]:
```
enc_dev = img_encoder.predict_generator(img_generator(dev_list), steps=len(dev_list), verbose=1)
```

```
1000/1000 [==============================] - 649s 649ms/step
```

In [ ]:
```
enc_test = img_encoder.predict_generator(img_generator(test_list), steps=len(test_list), verbose=1)
```

```
1000/1000 [==============================] - 596s 596ms/step
```

It's a good idea to save the resulting matrices, so we do not have to run the encoder again.

In [ ]:
```
np.save("/content/drive/My Drive/"+my_data_dir+"/outputs/encoded_images_train.npy", enc_train)
np.save("/content/drive/My Drive/"+my_data_dir+"/outputs/encoded_images_dev.npy", enc_dev)
np.save("/content/drive/My Drive/"+my_data_dir+"/outputs/encoded_images_test.npy", enc_test)
```

Part II Text (Caption) Data Preparation (1.25 pts)

Next, we need to load the image captions and generate training data for the generator model.

Reading image descriptions

TODO: Write the following function that reads the image descriptions from the file `filename` and returns a dictionary in the following format. Take a look at the file `Flickr8k.token.txt` for the format of the input file. The keys of the dictionary should be image filenames. Each value should be a list of 5 captions. Each caption should be a list of tokens.

The captions in the file are already tokenized, so you can just split them at white spaces. You should convert each token to lower case. You should then pad each caption with a START token on the left and an END token on the right.

In [24]:
```
# 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

def read_image_descriptions(filename):
    image_descriptions = defaultdict(list)
    file = load_doc(filename)
    captions = file.split("\n")
    for caption in captions[:-1]:
        img, caption = caption.split('\t')
        desc = caption.split()
        #converts to lowercase
        desc = [word.lower() for word in desc]
        #convert back to string
        img_caption = ' '.join(desc)
        img_caption = '<START> '+''.join(img_caption) + ' <END>'
        l = list(img_caption.split())
        if img[:-2] not in image_descriptions:
            image_descriptions[img[:-2]] = [ l ]
        else:
            image_descriptions[img[:-2]].append(l)

    return image_descriptions
```

In [25]:
```
descriptions = read_image_descriptions("/content/drive/My Drive/"+my_data_dir+"/Flickr8k.token.txt")
```

In [26]:
```
print(descriptions[dev_list[0]])
```

```
[['<START>', 'the', 'boy', 'laying', 'face', 'down', 'on', 'a', 'skateboard', 'is', 'being', 'pushed', 'along', 'the', 'ground', 'by', 'another',
'boy', '.', '<END>'], ['<START>', 'two', 'girls', 'play', 'on', 'a', 'skateboard', 'in', 'a', 'courtyard', '.', '<END>'], ['<START>', 'two', 'peop
le', 'play', 'on', 'a', 'long', 'skateboard', '.', '<END>'], ['<START>', 'two', 'small', 'children', 'in', 'red', 'shirts', 'playing', 'on', 'a',
'skateboard', '.', '<END>'], ['<START>', 'two', 'young', 'children', 'on', 'a', 'skateboard', 'going', 'across', 'a', 'sidewalk', '<END>']]
```

Running the previous cell should print:

```
[['<START>', 'the', 'boy', 'laying', 'face', 'down', 'on', 'a', 'skateboard', 'is', 'being', 'pushed', 'along', 'the', 'ground', 'by', 'another', 'boy', '.',
'<END>'], ['<START>', 'two', 'girls', 'play', 'on', 'a', 'skateboard', 'in', 'a', 'courtyard', '.', '<END>'], ['<START>', 'two', 'people', 'play', 'on', 'a',
'long', 'skateboard', '.', '<END>'], ['<START>', 'two', 'small', 'children', 'in', 'red', 'shirts', 'playing', 'on', 'a', 'skateboard', '.', '<END>'],
['<START>', 'two', 'young', 'children', 'on', 'a', 'skateboard', 'going', 'across', 'a', 'sidewalk', '<END>']]
```

Creating Word Indices

Next, we need to create a lookup table from the **training** data mapping words to integer indices, so we can encode input and output sequences using numeric representations. **TODO** create the dictionaries id_to_word and word_to_id, which should map tokens to numeric ids and numeric ids to tokens.
Hint: Create a set of tokens in the training data first, then convert the set into a list and sort it. This way if you run the code multiple times, you will always get the same dictionaries.

In [27]:
```
# Create the vocabulary.
vocabulary = set()
for key in descriptions.keys():
  for d in descriptions[key]:
    for c in d:
      vocabulary.add(c)
vocabulary = list(vocabulary)
vocabulary.sort()

# integer to token
id_to_word = {}

# token to integer
word_to_id = {}
id = 1
for w in vocabulary:
  word_to_id[w] = id
  id_to_word[id] = w
  id += 1
```

In [22]:
```
word_to_id['dog'] # should print an integer
```

Out[22]: 2310

In [23]:
```
id_to_word[2310] # should print a token
```

Out[23]: 'dog'

Note that we do not need an UNK word token because we are generating. The generated text will only contain tokens seen at training time.

Part III Basic Decoder Model (2.5 pts)

For now, we will just train a model for text generation without conditioning the generator on the image input.

There are different ways to do this and our approach will be slightly different from the generator discussed in class.

The core idea here is that the Keras recurrent layers (including LSTM) create an "unrolled" RNN. Each time-step is represented as a different unit, but the weights for these units are shared. We are going to use the constant MAX_LEN to refer to the maximum length of a sequence, which turns out to be 40 words in this data set (including START and END).

In [28]:
```
max(len(description) for image_id in train_list for description in descriptions[image_id])
```

Out[28]: 40

In class, we discussed LSTM generators as transducers that map each word in the input sequence to the next word.

Instead, we will use the model to predict one word at a time, given a partial sequence. For example, given the sequence ["START","a"], the model might predict "dog" as the most likely word. We are basically using the LSTM to encode the input sequence up to this point.

To train the model, we will convert each description into a set of input output pairs as follows. For example, consider the sequence

```
['<START>', 'a', 'black', 'dog', '.', '<END>']
```

We would train the model using the following input/output pairs

i	input	output
0	[START]	a
1	[START , a]	black
2	[START , a , black]	dog
3	[START , a , black , dog]	END

Here is the model in Keras Keras. Note that we are using a Bidirectional LSTM, which encodes the sequence from both directions and then predicts the output. Also note the `return_sequence=False` parameter, which causes the LSTM to return a single output instead of one output per state.

Note also that we use an embedding layer for the input words. The weights are shared between all units of the unrolled LSTM. We will train these embeddings with the model.

In [29]:
```
MAX_LEN = 40
EMBEDDING_DIM=300
vocab_size = len(word_to_id)

# Text input
text_input = Input(shape=(MAX_LEN,))
embedding = Embedding(vocab_size, EMBEDDING_DIM, input_length=MAX_LEN)(text_input)
x = Bidirectional(LSTM(512, return_sequences=False))(embedding)
pred = Dense(vocab_size, activation='softmax')(x)
model = Model(inputs=[text_input],outputs=pred)
model.compile(loss='categorical_crossentropy', optimizer='RMSprop', metrics=['accuracy'])

model.summary()
```

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 40)]              0
_________________________________________________________________
embedding (Embedding)        (None, 40, 300)           2676000
_________________________________________________________________
bidirectional (Bidirectional (None, 1024)              3330048
_________________________________________________________________
dense (Dense)                (None, 8920)              9143000
=================================================================
Total params: 15,149,048
Trainable params: 15,149,048
Non-trainable params: 0
_________________________________________________________________
```

The model input is a numpy ndarray (a tensor) of size `(batch_size, MAX_LEN)`. Each row is a vector of size MAX_LEN in which each entry is an integer representing a word (according to the `word_to_id` dictionary). If the input sequence is shorter than MAX_LEN, the remaining entries should be padded with 0.

For each input example, the model returns a softmax activated vector (a probability distribution) over possible output words. The model output is a numpy ndarray of size `(batch_size, vocab_size)`. vocab_size is the number of vocabulary words.

Creating a Generator for the Training Data

TODO:

We could simply create one large numpy ndarray for all the training data. Because we have a lot of training instances (each training sentence will produce up to MAX_LEN input/output pairs, one for each word), it is better to produce the training examples *lazily*, i.e. in batches using a generator (recall the image generator in part I).

Write the function `text_training_generator` below, that takes as a paramater the batch_size and returns an `(input, output)` pair. `input` is a `(batch_size, MAX_LEN)` ndarray of partial input sequences, `output` contains the next words predicted for each partial input sequence, encoded as a `(batch_size, vocab_size)` ndarray.

Each time the next() function is called on the generator instance, it should return a new batch of the *training* data. You can use `train_list` as a list of training images. A batch may contain input/output examples extracted from different descriptions or even from different images.

You can just refer back to the variables you have defined above, including `descriptions`, `train_list`, `vocab_size`, etc.

Hint: To prevent issues with having to reset the generator for each epoch and to make sure the generator can always return exactly `batch_size` input/output pairs in each step, wrap your code into a `while True:` loop. This way, when you reach the end of the training data, you will just continue adding training data from the beginning into the batch.

In [30]:
```
def text_training_generator(batch_size=128):
    while True:
      count = 0
      input = []
      output = []

      for image in train_list:
        for cap in descriptions[image]:
          for i in range(len(cap)-1):
            partial = [word_to_id[txt] for txt in cap[:i+1]]
            input.append(partial)
            op = np.zeros(vocab_size)
            op[word_to_id[cap[i+1]]] = 1
            output.append(op)
            count += 1

            if count >= batch_size:
              output = np.asarray(output)
              input = pad_sequences(input, maxlen=MAX_LEN, padding='post')
              yield input, output
              count = 0
              input = []
              output = []
```

Training the Model

We will use the `fit_generator` method of the model to train the model. fit_generator needs to know how many iterator steps there are per epoch.

Because there are len(train_list) training samples with up to `MAX_LEN` words, an upper bound for the number of total training instances is `len(train_list)*MAX_LEN` . Because the generator returns these in batches, the number of steps is len(train_list) * MAX_LEN // batch_size

In [31]:
```
batch_size = 128
generator = text_training_generator(batch_size)
steps = len(train_list) * MAX_LEN // batch_size
```

In [ ]:
```
model.fit_generator(generator, steps_per_epoch=steps, verbose=True, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 149s 79ms/step - loss: 3.7503 - accuracy: 0.4160
Epoch 2/10
1875/1875 [==============================] - 147s 78ms/step - loss: 3.3738 - accuracy: 0.4211
Epoch 3/10
1875/1875 [==============================] - 146s 78ms/step - loss: 3.3801 - accuracy: 0.4239
Epoch 4/10
1875/1875 [==============================] - 146s 78ms/step - loss: 3.4938 - accuracy: 0.4237
Epoch 5/10
1875/1875 [==============================] - 147s 78ms/step - loss: 3.3163 - accuracy: 0.4295
Epoch 6/10
1875/1875 [==============================] - 146s 78ms/step - loss: 3.2990 - accuracy: 0.4330
Epoch 7/10
1875/1875 [==============================] - 146s 78ms/step - loss: 3.2795 - accuracy: 0.4331
Epoch 8/10
1875/1875 [==============================] - 146s 78ms/step - loss: 3.1888 - accuracy: 0.4382
Epoch 9/10
1875/1875 [==============================] - 146s 78ms/step - loss: 3.1478 - accuracy: 0.4430
Epoch 10/10
1875/1875 [==============================] - 146s 78ms/step - loss: 3.1070 - accuracy: 0.4452
```

Out[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8efe11c8d0>
```

Continue to train the model until you reach an accuracy of at least 40%.

Greedy Decoder

TODO Next, you will write a decoder. The decoder should start with the sequence `["<START>"]`, use the model to predict the most likely word, append the word to the sequence and then continue until `"<END>"` is predicted or the sequence reaches `MAX_LEN` words.

In [34]:
```
def decoder():
    start = ['<START>']
    while True:
      sequence = [word_to_id[w] for w in start]
      sequence = pad_sequences([sequence], maxlen=MAX_LEN, padding='post')
      yhat = model.predict(sequence)
      yhat = np.argmax(yhat)
      word = id_to_word[yhat]
      start.append(word)
      if word == '<END>' or len(start) > MAX_LEN:
        break
    return start
```

In [35]:
```
print(decoder())
```

```
['<START>', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos',
'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'logos', 'verizon', 'verizon', 'verizon', 'verizon', 'verizon', 'verizon',
'verizon', 'vessel', 'blowup', 'blowup', 'flickr', 'blowup', 'flickr', 'flickr', 'san', 'pilar']
```

This simple decoder will of course always predict the same sequence (and it's not necessarily a good one).

Modify the decoder as follows. Instead of choosing the most likely word in each step, sample the next word from the distribution (i.e. the softmax activated output) returned by the model. Take a look at the np.random.multinomial (https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.multinomial.html) function to do this.

In [36]:
```
def sample_decoder():
    start = ['<START>']
    while True:
      sequence = [word_to_id[w] for w in start]
      sequence = pad_sequences([sequence], maxlen=MAX_LEN, padding='post')
      pred = list(np.squeeze(model.predict(sequence.reshape(1, -1))).astype(np.float64))
      predict = pred / np.sum(pred)
      probs = np.random.multinomial(1, predict, 1)
      yhat = np.argmax(probs)
      word = id_to_word[yhat]
      start.append(word)
      if word == '<END>' or len(start) > MAX_LEN:
        break
    return start
```

You should now be able to see some interesting output that looks a lot like flickr8k image captions -- only that the captions are generated randomly without any image input.

In [37]:
```
for i in range(10):
    print(sample_decoder())
```

```
['<START>', 'pipeline', 'pinata', 'observe', 'slouched', 'tires', 'logo', 'five', 'come', 'skeleton-printed', 'tge', 'waeribng', 'castle', 'prin
t', 'kneeled', 'teens', 'impeach', 'inflated', 'inspects', 'objects', 'sniff', 'smiff', 'actors', 'robes', 'funky', 'vampires', 'walking', 'rainst
orm', 'strings', 'collides', 'modifications', 'bare-back', 'slice', 'these', 'sifting', 'bare-back', 'mr.', 'shack', 'wasteland', 'broadway', 'mag
azine']
['<START>', 'oxen', 'heavily', 'manually', 'burbur', 'itself', 'tourquoise', 'coverings', 'blonde-haired', 'sparring', 'best', 'two-wheeled', 'ami
dst', 'hoddie', 'licked', 'mechanisms', 'quarterpipe', 'ou', 'mouthed', 'stomachs', 'conoes', 'sifting', 'fists', 'pretend', 'more', 'magazine',
'cartwheels', 'army', 'mountainous', 'storm', 'valleys', 'rooftop', 'ball', 'goatee', '80', 'entrance', 'gesture', 'wide-legged', 'viewer', 'rolls
kating', 'kitty']
['<START>', 'furnace', 'throughwindow', 'booths', 'menacingly', 'pickup', 'belly-smacker', 'paraglider', 'works', 'dappled', 'prison', 'evade', 's
everal', 'punch', 'youngle', 'cycle', 'wedgie', 'weilding', 'knelt', 'pudding', 'mountaineers', 'apron', 'yankees', 'brought', 'source', 'raling',
'sound', 'soles', 'fog', 'otuside', 'snowpants', 'ipod', 'brown-and-white', 'any', 'bloody', 'kerry', 'bus', 'crime', 'instructor', 'crossing', 'a
lligator']
['<START>', 'drummer', 'closeup', 'practice', 'looked', 'sunflower', 'arrows', 'patiently', 'sailboats', 'several', 'turquiose', 'celtics', 'snows
hovel', 'cardigan', 'docking', 'chins', 'horned', 'pillared', 'c', 'vessel', 'pinkish', 'furry', 'types', 'noisemaker', 'perfors', 'sidewalks', 'l
augh', 'moving', 'rungs', 'redwood', 'holey', 'motorboat', 'mission', 'vacant', 'times', 'tambourines', '5', 'barrior', 'concentrating', 'indescri
pt', 'wants']
['<START>', 'drive', 'em', 'blazing', 'arrow', 'thows', 'evil', 'saxaphones', 'twho', 'fantasy', 'hooker', 'gathers', 'ended', 'chiseling', 'jumps
uites', 'long-haired', 'furiously', 'puddles', 'cub', 'rummage', 'garbage', 'stripped', 'grab', 'railling', 'flamboyant', 'berets', 'hilltops', 'e
xplosions', 'ovals', 'hardscape', 'tankini', 'plane', 'pastures', 'had', 'holds', 'paralell', 'dramatically', 'telephot', 'treefilled', 'support',
'joins']
['<START>', 'informal', 'gymnasium', 'baton', 'pride', 'teases', 'angerly', 'backview', 'kitty', 'yellow-gold', 'fleeing', 'fireside', 'nubby', 's
howgirls', 'cloth', 'tandom', 'bowl', 'griding', 'david', 'cartwheeling', 'flashlight', 'sabre', 'last', 'gray-green', 'quinta', 'zaftig', 'lill
y', 'swampy', 'off-roading', 'coarse', 'hallways', 'aerodynamic', 'shovels', 'himself', 'distance', 'barge', 'clibing', 'sheilding', 'is', 'neptun
o', 'black-robed']
['<START>', 'tin', 'closed', 'crag', 'yellow-suited', 'salon', 'swimcap', 'presentations', 'meter', 'surrounds', 'gatorade', 'williams', 'goose',
'canal', 'nash', 'snowshovel', 'arts', 'ally', 'flying', 'lited', 'cards', 'presents', 'slender', '10', 'waders', 'venue', 'bathrooms', 'mid-swin
g', 'school', 'whil', 'pac-man', 'bar-type', 'about', 'easels', 'department', 'stair', 'snowpants', 'sea', 'ceremonial', 'handstands', 'pecks']
['<START>', 'contraption', 'showgirl', 'released', 'powerlines', 'rugby', 'construction', 'snowy', 'cream', 'spewing', 'beak', 'ethnic', 'wheelbar
row', 'thinking', 'instructs', 'crows', '30', 'neon', 'mudfight', 'train', 'tote', 'flaps', 'smartly', 'quiet', 'followed', 'lav', 'daschunds', 'k
nit', 'whack', 'lonely', 'floral', 'taught', 'cardigan', 'walkways', 'wanting', 'offf', 'beds', 'bruised', 'san', 'wake', 'fix']
['<START>', 'dim', 'enviorment', 'shephard', 'fencers', 'flume', 'buggy', 'bad', 'sun-dappled', 'africans', 'dome', 'midstride', 'parachutes', 'ki
ckflip', 'kissing', 'contorts', 'high-fiving', 'meat', 'sunshine', 'fairway', 'facefirst', 'bicyler', 'mill', 'railgrind', 'wetland', 'rope', 'suc
kles', 'peddal', 'eyeshadow', 'astride', 'grinding', 'york', 'disc', 'coppery', 'digger', 'juggler', 'prairie', 'dozes', 'bigs', 'hero', 'ant']
['<START>', 'cascades', 'sprinkling', 'uniform', 'tw', 'tobaggons', 'roses', 'daisies', 'trimmed', 'to', 'masked', 'prances', 'piste', 'parasurf
s', 'bullfight', 'guitarists', 'overcoat', 'anticipation', 'icicles', 'zoo', 'whoa', 'snowboard', 'impersonators', 'drooping', 'shoot', 'mottled',
'bottles', 'flurry', 'seventh', 'attrative', 'iove', 'bald', 'shoulder', 'gettnig', 'orbs', 'rodents', 'supported', 'mid-flight', 'infant', 'trenc
h', 'bangles']
```

Part III - Conditioning on the Image (3.75 pts)

We will now extend the model to condition the next word not only on the partial sequence, but also on the encoded image.

We will project the 2048-dimensional image encoding to a 300-dimensional hidden layer. We then concatenate this vector with each embedded input word, before applying the LSTM.

Here is what the Keras model looks like:

In [38]:

```
MAX_LEN = 40
EMBEDDING_DIM=300
IMAGE_ENC_DIM=300

# Image input
img_input = Input(shape=(2048,))
img_enc = Dense(300, activation="relu") (img_input)
images = RepeatVector(MAX_LEN)(img_enc)

# Text input
text_input = Input(shape=(MAX_LEN,))
embedding = Embedding(vocab_size, EMBEDDING_DIM, input_length=MAX_LEN)(text_input)
x = Concatenate()([images,embedding])
y = Bidirectional(LSTM(256, return_sequences=False))(x)
pred = Dense(vocab_size, activation='softmax')(y)
model = Model(inputs=[img_input,text_input],outputs=pred)
model.compile(loss='categorical_crossentropy', optimizer="RMSProp", metrics=['accuracy'])

model.summary()
```

```
Model: "functional_5"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 2048)]       0
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 300)          614700      input_3[0][0]
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 40)]         0
__________________________________________________________________________________________________
repeat_vector (RepeatVector)    (None, 40, 300)      0           dense_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 40, 300)      2676000     input_4[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 40, 600)      0           repeat_vector[0][0]
                                                                 embedding_1[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 512)          1755136     concatenate_2[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 8920)         4575960     bidirectional_1[0][0]
==================================================================================================
Total params: 9,621,796
Trainable params: 9,621,796
Non-trainable params: 0
__________________________________________________________________________________________________
```

The model now takes two inputs:

1. a `(batch_size, 2048)` ndarray of image encodings.
2. a `(batch_size, MAX_LEN)` ndarray of partial input sequences.

And one output as before: a `(batch_size, vocab_size)` ndarray of predicted word distributions.

TODO: Modify the training data generator to include the image with each input/output pair. Your generator needs to return an object of the following format: `([image_inputs, text_inputs], next_words)` . Where each element is an ndarray of the type described above.

You need to find the image encoding that belongs to each image. You can use the fact that the index of the image in `train_list` is the same as the index in enc_train and enc_dev.

If you have previously saved the image encodings, you can load them from disk:

In [39]:
```
enc_train = np.load("drive/My Drive/"+my_data_dir+"/outputs/encoded_images_train.npy")
enc_dev = np.load("drive/My Drive/"+my_data_dir+"/outputs/encoded_images_dev.npy")
```

In [40]:
```
def training_generator(batch_size=128):
  while True:
      count = 0
      text_inputs = []
      next_words = []
      image_inputs = []

      for idx, image in enumerate(train_list):
        for cap in descriptions[image]:
          for i in range(len(cap)-1):
            partial = [word_to_id[txt] for txt in cap[:i+1]]
            text_inputs.append(partial)
            op = np.zeros(vocab_size)
            op[word_to_id[cap[i+1]]] = 1
            next_words.append(op)
            count += 1
            image_inputs.append(enc_train[idx])

            if count >= batch_size:
              image_inputs = np.asarray(image_inputs)
              next_words = np.asarray(next_words)
              text_inputs = pad_sequences(text_inputs, maxlen=MAX_LEN, padding='post')
              yield ([image_inputs, text_inputs], next_words)
              count = 0
              text_inputs = []
              next_words = []
              image_inputs = []
```

You should now be able to train the model as before:

In [37]:
```
batch_size = 128
generator = training_generator(batch_size)
steps = len(train_list) * MAX_LEN // batch_size
```

In [ ]:
```
model.fit_generator(generator, steps_per_epoch=steps, verbose=True, epochs=20)
```

```
Epoch 1/20
1875/1875 [==============================] - 149s 79ms/step - loss: 4.3523 - accuracy: 0.2856
Epoch 2/20
1875/1875 [==============================] - 149s 79ms/step - loss: 3.7501 - accuracy: 0.3537
Epoch 3/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.5809 - accuracy: 0.3706
Epoch 4/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.4799 - accuracy: 0.3797
Epoch 5/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.4193 - accuracy: 0.3861
Epoch 6/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.3569 - accuracy: 0.3926
Epoch 7/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.3513 - accuracy: 0.3951
Epoch 8/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.3613 - accuracy: 0.3970
Epoch 9/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.3470 - accuracy: 0.3993
Epoch 10/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.3789 - accuracy: 0.3993
Epoch 11/20
1875/1875 [==============================] - 149s 79ms/step - loss: 3.3810 - accuracy: 0.4015
Epoch 12/20
1875/1875 [==============================] - 149s 79ms/step - loss: 3.4007 - accuracy: 0.4022
Epoch 13/20
1875/1875 [==============================] - 149s 80ms/step - loss: 3.4075 - accuracy: 0.4062
Epoch 14/20
1875/1875 [==============================] - 149s 79ms/step - loss: 3.3851 - accuracy: 0.4101
Epoch 15/20
1875/1875 [==============================] - 149s 79ms/step - loss: 3.3631 - accuracy: 0.4128
Epoch 16/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.3364 - accuracy: 0.4139
Epoch 17/20
1875/1875 [==============================] - 148s 79ms/step - loss: 3.3331 - accuracy: 0.4161
Epoch 18/20
1875/1875 [==============================] - 147s 78ms/step - loss: 3.4402 - accuracy: 0.4144
Epoch 19/20
1875/1875 [==============================] - 147s 79ms/step - loss: 3.4483 - accuracy: 0.4124
Epoch 20/20
1875/1875 [==============================] - 147s 79ms/step - loss: 3.7653 - accuracy: 0.4132
```

Out[ ]: <tensorflow.python.keras.callbacks.History at 0x7f8f06687198>

Again, continue to train the model until you hit an accuracy of about 40%. This may take a while. I strongly encourage you to experiment with cloud GPUs using the GCP voucher for the class.

You can save your model weights to disk and continue at a later time.

In [ ]:
```
model.save_weights("drive/My Drive/"+my_data_dir+"/outputs/model.h5")
```

to load the model:

In [41]:
```
model.load_weights("drive/My Drive/"+my_data_dir+"/outputs/model.h5")
```

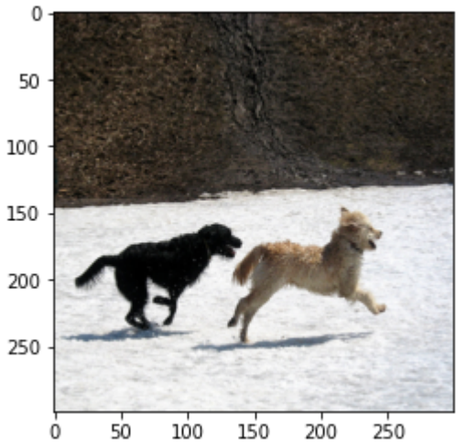
TODO: Now we are ready to actually generate image captions using the trained model. Modify the simple greedy decoder you wrote for the text-only generator, so that it takes an encoded image (a vector of length 2048) as input, and returns a sequence.

In [42]:
```
def image_decoder(enc_image):
    enc_image = enc_image.reshape(1, 2048)
    start = ['<START>']
    while True:
      sequence = [word_to_id[w] for w in start]
      sequence = pad_sequences([sequence], maxlen=MAX_LEN, padding='post')
      pred = list(np.squeeze(model.predict([enc_image, sequence])).astype(np.float64))
      predict = pred / np.sum(pred)
      probs = np.random.multinomial(1, predict, 1)
      yhat = np.argmax(probs)
      word = id_to_word[yhat]
      start.append(word)
      if word == '<END>' or len(start) > MAX_LEN:
        break
    return ' '.join(start[1:-1])
```

As a sanity check, you should now be able to reproduce (approximately) captions for the training images.

In [112]:
```
plt.imshow(get_image(train_list[0]))
image_decoder(enc_train[0])
```

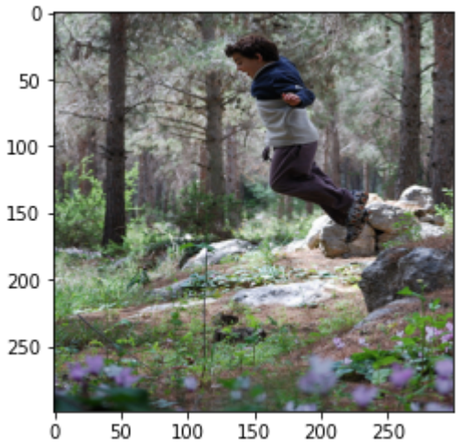
Out[112]: 'a brown and white dog jumps over a hurdle .'



You should also be able to apply the model to dev images and get reasonable captions:

In [138]:
```
plt.imshow(get_image(dev_list[1]))
image_decoder(enc_dev[0])
```

Out[138]: 'a young boy poses in the air .'



For this assignment we will not perform a formal evaluation.

Feel free to experiment with the parameters of the model or continue training the model. At some point, the model will overfit and will no longer produce good descriptions for the dev images.

TODO Modify the simple greedy decoder for the caption generator to use beam search. Instead of always selecting the most probable word, use a *beam*, which contains the n highest-scoring sequences so far and their total probability (i.e. the product of all word probabilities). I recommend that you use a list of `(probability, sequence)` tuples. After each time-step, prune the list to include only the n most probable sequences.

Then, for each sequence, compute the n most likely successor words. Append the word to produce n new sequences and compute their score. This way, you create a new list of n*n candidates.

Prune this list to the best n as before and continue until `MAX_LEN` words have been generated.

Note that you cannot use the occurence of the `"<END>"` tag to terminate generation, because the tag may occur in different positions for different entries in the beam.

Once `MAX_LEN` has been reached, return the most likely sequence out of the current n.

HOMEWORK2 BONUS

Part IV - Beam Search Decoder (2.5 pts)

TODO Modify the simple greedy decoder for the caption generator to use beam search. Instead of always selecting the most probable word, use a *beam*, which contains the n highest-scoring sequences so far and their total probability (i.e. the product of all word probabilities). I recommend that you use a list of `(probability, sequence)` tuples. After each time-step, prune the list to include only the n most probable sequences.

Then, for each sequence, compute the n most likely successor words. Append the word to produce n new sequences and compute their score. This way, you create a new list of n*n candidates.

Prune this list to the best n as before and continue until `MAX_LEN` words have been generated.

Note that you cannot use the occurence of the `"<END>"` tag to terminate generation, because the tag may occur in different positions for different entries in the beam.

Once `MAX_LEN` has been reached, return the most likely sequence out of the current n.

In [43]:
```
def img_beam_decoder(n, image_enc):
    enc_image = image_enc.reshape(1,2048)
    start = [word_to_id['<START>']]

    sequence = [[start, 0.0]]

    while len(sequence[0][0]) < MAX_LEN:
        candidate = []
        for s in sequence:
            partial_caps = pad_sequences([s[0]], maxlen=MAX_LEN, padding='post')
            pred = model.predict([enc_image, np.array(partial_caps)])
            word_pred = np.argsort(pred[0])[-n:]

            for w in word_pred:
                next_word, prob = s[0][:], s[1]
                next_word.append(w)
                prob += pred[0][w]
                candidate.append([next_word, prob])

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

    sequence = sequence[-1][0]
    inter_caption = [id_to_word[i] for i in sequence]

    caption = []

    for i in inter_caption:
        if i != '<END>':
            caption.append(i)
        else:
            break

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

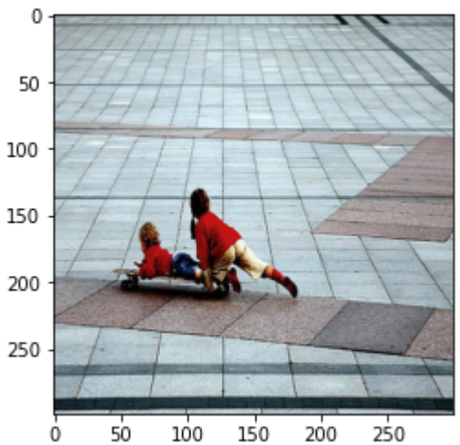
In [44]:
```
img_beam_decoder(3, enc_dev[1])
```

Out[44]:
```
'a group of men playing soccer'
```

TODO Finally, before you submit this assignment, please show 5 development images, each with 1) their greedy output, 2) beam search at n=3 3) beam search at n=5.

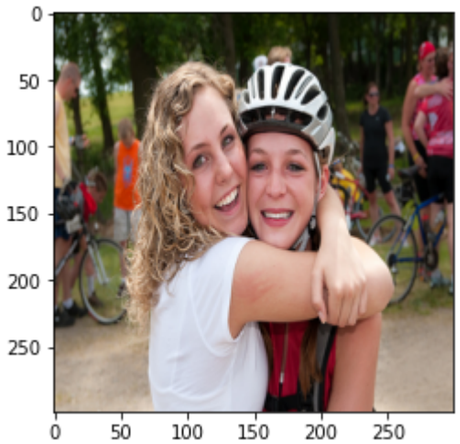
In [146]:
```
plt.imshow(get_image(dev_list[0]))
print(image_decoder(enc_dev[0]))
print(img_beam_decoder(3, enc_dev[0]))
print(img_beam_decoder(5, enc_dev[0]))
```

```
a man is surfing with his arms out .
a group of men playing soccer
a group of men playing soccer
```



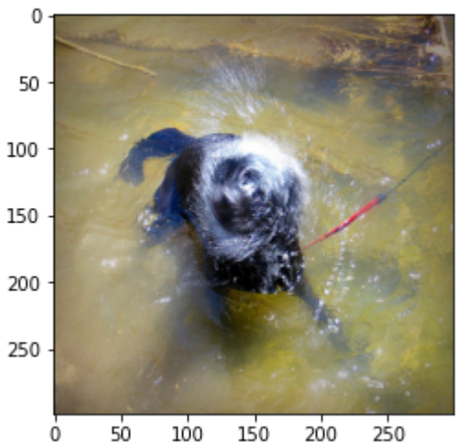
In [147]:
```
plt.imshow(get_image(dev_list[16]))
print(image_decoder(enc_dev[16]))
print(img_beam_decoder(3, enc_dev[16]))
print(img_beam_decoder(5, enc_dev[16]))
```

```
a brown dog is running after a yellow soccer ball .
a group of men playing soccer
a group of men playing soccer
```



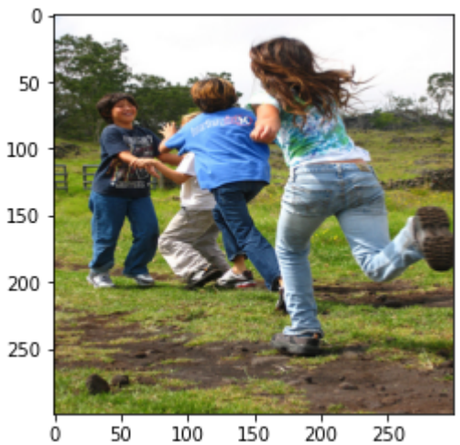
In [148]:
```
plt.imshow(get_image(dev_list[12]))
print(image_decoder(enc_dev[12]))
print(img_beam_decoder(3, enc_dev[12]))
print(img_beam_decoder(5, enc_dev[12]))
```

```
children play soccer .
a group of men playing soccer
a group of men playing soccer
```



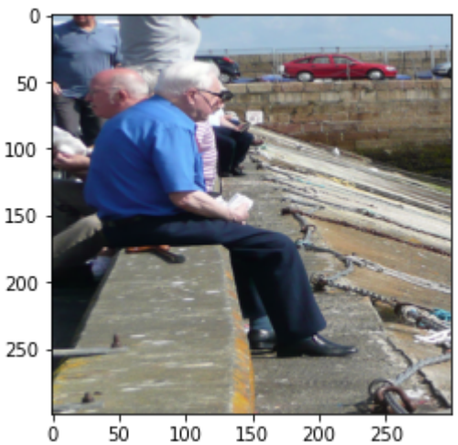
In [149]:
```
plt.imshow(get_image(dev_list[7]))
print(image_decoder(enc_dev[7]))
print(img_beam_decoder(3, enc_dev[7]))
print(img_beam_decoder(5, enc_dev[7]))
```

```
the opposing player is standing looking down
a group of men playing soccer
a group of men playing soccer
```



In [150]:
```
plt.imshow(get_image(dev_list[92]))
print(image_decoder(enc_dev[92]))
print(img_beam_decoder(3, enc_dev[92]))
print(img_beam_decoder(5, enc_dev[92]))
```

```
a man playing soccer
a group of men playing soccer
a group of men playing soccer
```



References

- https://github.com/AmritK10/Image_Captioning/blob/master/image_captioning.ipynb (https://github.com/AmritK10/Image_Captioning/blob/master/image_captioning.ipynb)
- https://github.com/yashk2810/Image-Captioning/blob/master/Image%20Captioning%20InceptionV3.ipynb (https://github.com/yashk2810/Image-Captioning/blob/master/Image%20Captioning%20InceptionV3.ipynb)
- https://www.machinecurve.com/index.php/2019/11/25/visualizing-keras-cnn-attention-saliency-maps/ (https://www.machinecurve.com/index.php/2019/11/25/visualizing-keras-cnn-attention-saliency-maps/)
- https://medium.com/datalogue/attention-in-keras-1892773a4f22 (https://medium.com/datalogue/attention-in-keras-1892773a4f22)