Image Captioning with Conditioned LSTM Generators

Note: Some of the traning in this notebook may take a while to run. To save time, you can download the encoded outputs and models from the `output` folder and upload to your google drive. Then by mounting GDrive you can access the models without traning it again.

Environment Setup

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

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

from keras.applications.inception_v3 import InceptionV3

from tensorflow.keras.optimizers import Adam

from google.colab import drive
```

Access to the flickr8k data

In this project we use the flickr8k data set, which is accessed here: https://forms.illinois.edu/sec/1713398

Reference:

> 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 when discussing our results

In [2]:
```
my_data_dir="data"
```

Mounting GDrive so we can access the files from Colab

In [28]:
```
drive.mount('/content/gdrive')
```

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

Image Encodings

The files Flickr_8k.trainImages.txt Flickr_8k.devImages.txt Flickr_8k.testImages.txt, contain a list of training, development, and test images, respectively.

In [4]:
```
def load_image_list(filename):
    with open(filename,'r') as image_list_f:
        return [line.strip() for line in image_list_f]
```

In [5]:
```
train_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr
dev_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8
test_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_
```

Let's see how many images there are

In [6]:
```
len(train_list), len(dev_list), len(test_list)
```

Out[6]:
```
(6000, 1000, 1000)
```

Each entry is an image filename.

In [7]:
```
dev_list[20]
```

Out[7]:
```
'3693961165_9d6c333d5b.jpg'
```

The images are located in a subdirectory.

In [10]:
```
IMG_PATH = "/content/gdrive/My Drive/data/Flickr8k_Dataset"
```

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

In [11]:
```
image = PIL.Image.open(os.path.join(IMG_PATH, dev_list[20]))
image
```

Out[11]:

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.

Reference:

> 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_

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)
```

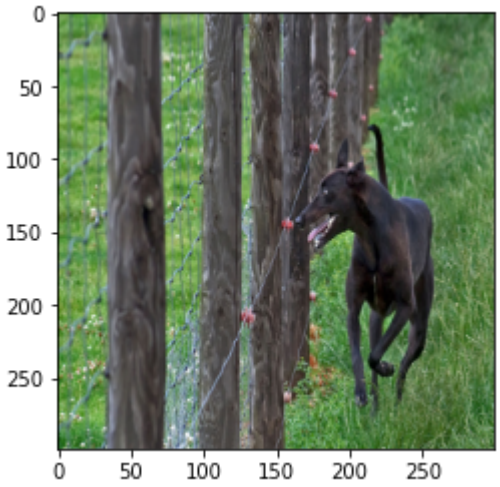
Out[13]:
```
array([[[118, 161,  89],
        [120, 164,  89],
        [111, 157,  82],
        ...,
        [ 68, 106,  65],
        [ 64, 102,  61],
        [ 65, 104,  60]],

       [[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 0x7f87de0b0290>
```



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

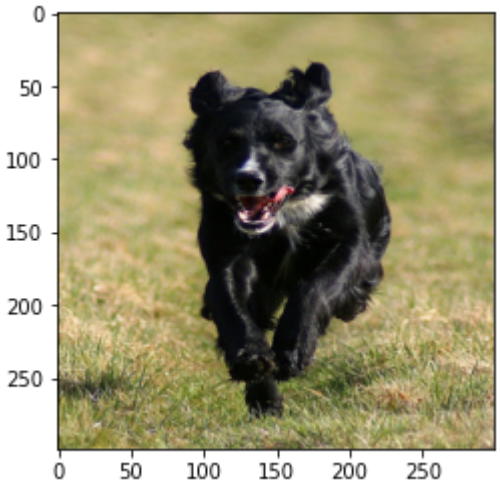
Out[15]:
```
(299, 299, 3)
```

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 0x7f87ddb94850>
```



Load the pre-trained Inception model.

Inception v3 is an image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model is the culmination of many ideas developed by multiple researchers over the years. It is based on the original paper: "Rethinking the Inception Architecture for Computer Vision" by Szegedy, et. al.

In [19]:
```
# Might take a while to load
img_model = InceptionV3(weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
ions/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96116736/96112376 [==============================] - 1s 0us/step
96124928/96112376 [==============================] - 1s 0us/step
```

In [20]:
```
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 (BatchNorm  (None, 149, 149, 32  96         ['conv2d[0]
[0]']
 alization)                     )

 activation (Activation)        (None, 149, 149, 32  0           ['batch_norma
lization[0][0]']
                                )

 conv2d_1 (Conv2D)              (None, 147, 147, 32  9216        ['activation
[0][0]']
                                )

 batch_normalization_1 (BatchNo  (None, 147, 147, 32  96         ['conv2d_1[0]
[0]']
 rmalization)                   )
```

```
 activation_1 (Activation)      (None, 147, 147, 32  0           ['batch_norma
lization_1[0][0]']
                                )

 conv2d_2 (Conv2D)              (None, 147, 147, 64  18432       ['activation_
1[0][0]']
                                )

 batch_normalization_2 (BatchNo  (None, 147, 147, 64  192        ['conv2d_2[0]
[0]']
 rmalization)                   )

 activation_2 (Activation)      (None, 147, 147, 64  0           ['batch_norma
lization_2[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 73, 73, 64)   0           ['activation_
2[0][0]']

 conv2d_3 (Conv2D)              (None, 73, 73, 80)   5120        ['max_pooling
2d[0][0]']

 batch_normalization_3 (BatchNo  (None, 73, 73, 80)  240         ['conv2d_3[0]
[0]']
 rmalization)

 activation_3 (Activation)      (None, 73, 73, 80)   0           ['batch_norma
lization_3[0][0]']

 conv2d_4 (Conv2D)              (None, 71, 71, 192)  138240      ['activation_
3[0][0]']

 batch_normalization_4 (BatchNo  (None, 71, 71, 192)  576        ['conv2d_4[0]
[0]']
 rmalization)

 activation_4 (Activation)      (None, 71, 71, 192)  0           ['batch_norma
lization_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_pooling
2d_1[0][0]']

 batch_normalization_8 (BatchNo  (None, 35, 35, 64)  192         ['conv2d_8[0]
[0]']
 rmalization)

 activation_8 (Activation)      (None, 35, 35, 64)   0           ['batch_norma
lization_8[0][0]']

 conv2d_6 (Conv2D)              (None, 35, 35, 48)   9216        ['max_pooling
2d_1[0][0]']

 conv2d_9 (Conv2D)              (None, 35, 35, 96)   55296       ['activation_
8[0][0]']

 batch_normalization_6 (BatchNo  (None, 35, 35, 48)  144         ['conv2d_6[0]
[0]']
 rmalization)

 batch_normalization_9 (BatchNo  (None, 35, 35, 96)  288         ['conv2d_9[0]
```

```
[0]']
 rmalization)

 activation_6 (Activation)      (None, 35, 35, 48)   0           ['batch_norma
lization_6[0][0]']

 activation_9 (Activation)      (None, 35, 35, 96)   0           ['batch_norma
lization_9[0][0]']

 average_pooling2d (AveragePool  (None, 35, 35, 192)  0          ['max_pooling
2d_1[0][0]']
 ing2D)

 conv2d_5 (Conv2D)              (None, 35, 35, 64)   12288       ['max_pooling
2d_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_poo
ling2d[0][0]']

 batch_normalization_5 (BatchNo  (None, 35, 35, 64)  192         ['conv2d_5[0]
[0]']
 rmalization)

 batch_normalization_7 (BatchNo  (None, 35, 35, 64)  192         ['conv2d_7[0]
[0]']
 rmalization)

 batch_normalization_10 (BatchN  (None, 35, 35, 96)  288         ['conv2d_10
[0][0]']
 ormalization)

 batch_normalization_11 (BatchN  (None, 35, 35, 32)  96          ['conv2d_11
[0][0]']
 ormalization)

 activation_5 (Activation)      (None, 35, 35, 64)   0           ['batch_norma
lization_5[0][0]']

 activation_7 (Activation)      (None, 35, 35, 64)   0           ['batch_norma
lization_7[0][0]']

 activation_10 (Activation)     (None, 35, 35, 96)   0           ['batch_norma
lization_10[0][0]']

 activation_11 (Activation)     (None, 35, 35, 32)   0           ['batch_norma
lization_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 (BatchN  (None, 35, 35, 64)  192         ['conv2d_15
[0][0]']
 ormalization)

 activation_15 (Activation)     (None, 35, 35, 64)   0           ['batch_norma
lization_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 (BatchN  (None, 35, 35, 48)  144         ['conv2d_13
[0][0]']
 ormalization)

 batch_normalization_16 (BatchN  (None, 35, 35, 96)  288         ['conv2d_16
[0][0]']
 ormalization)

 activation_13 (Activation)     (None, 35, 35, 48)   0           ['batch_norma
lization_13[0][0]']

 activation_16 (Activation)     (None, 35, 35, 96)   0           ['batch_norma
lization_16[0][0]']

 average_pooling2d_1 (AveragePo  (None, 35, 35, 256)  0          ['mixed0[0]
[0]']
 oling2D)

 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_poo
ling2d_1[0][0]']

 batch_normalization_12 (BatchN  (None, 35, 35, 64)  192         ['conv2d_12
[0][0]']
 ormalization)

 batch_normalization_14 (BatchN  (None, 35, 35, 64)  192         ['conv2d_14
[0][0]']
 ormalization)

 batch_normalization_17 (BatchN  (None, 35, 35, 96)  288         ['conv2d_17
[0][0]']
 ormalization)

 batch_normalization_18 (BatchN  (None, 35, 35, 64)  192         ['conv2d_18
[0][0]']
 ormalization)

 activation_12 (Activation)     (None, 35, 35, 64)   0           ['batch_norma
lization_12[0][0]']

 activation_14 (Activation)     (None, 35, 35, 64)   0           ['batch_norma
```

```
lization_14[0][0]']

 activation_17 (Activation)     (None, 35, 35, 96)   0           ['batch_norma
lization_17[0][0]']

 activation_18 (Activation)     (None, 35, 35, 64)   0           ['batch_norma
lization_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 (BatchN  (None, 35, 35, 64)  192         ['conv2d_22
[0][0]']
 ormalization)

 activation_22 (Activation)     (None, 35, 35, 64)   0           ['batch_norma
lization_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 (BatchN  (None, 35, 35, 48)  144         ['conv2d_20
[0][0]']
 ormalization)

 batch_normalization_23 (BatchN  (None, 35, 35, 96)  288         ['conv2d_23
[0][0]']
 ormalization)

 activation_20 (Activation)     (None, 35, 35, 48)   0           ['batch_norma
lization_20[0][0]']

 activation_23 (Activation)     (None, 35, 35, 96)   0           ['batch_norma
lization_23[0][0]']

 average_pooling2d_2 (AveragePo  (None, 35, 35, 288)  0         ['mixed1[0]
[0]']
 oling2D)

 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_poo
ling2d_2[0][0]']

 batch_normalization_19 (BatchN  (None, 35, 35, 64)  192         ['conv2d_19
```

```
[0][0]']
 ormalization)

 batch_normalization_21 (BatchN  (None, 35, 35, 64)  192         ['conv2d_21
[0][0]']
 ormalization)

 batch_normalization_24 (BatchN  (None, 35, 35, 96)  288         ['conv2d_24
[0][0]']
 ormalization)

 batch_normalization_25 (BatchN  (None, 35, 35, 64)  192         ['conv2d_25
[0][0]']
 ormalization)

 activation_19 (Activation)     (None, 35, 35, 64)   0           ['batch_norma
lization_19[0][0]']

 activation_21 (Activation)     (None, 35, 35, 64)   0           ['batch_norma
lization_21[0][0]']

 activation_24 (Activation)     (None, 35, 35, 96)   0           ['batch_norma
lization_24[0][0]']

 activation_25 (Activation)     (None, 35, 35, 64)   0           ['batch_norma
lization_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 (BatchN  (None, 35, 35, 64)  192         ['conv2d_27
[0][0]']
 ormalization)

 activation_27 (Activation)     (None, 35, 35, 64)   0           ['batch_norma
lization_27[0][0]']

 conv2d_28 (Conv2D)             (None, 35, 35, 96)   55296       ['activation_
27[0][0]']

 batch_normalization_28 (BatchN  (None, 35, 35, 96)  288         ['conv2d_28
[0][0]']
 ormalization)

 activation_28 (Activation)     (None, 35, 35, 96)   0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 384)  1152       ['conv2d_26
[0][0]']
```

```
 ormalization)

 batch_normalization_29 (BatchN  (None, 17, 17, 96)  288         ['conv2d_29
[0][0]']
 ormalization)

 activation_26 (Activation)     (None, 17, 17, 384)  0           ['batch_norma
lization_26[0][0]']

 activation_29 (Activation)     (None, 17, 17, 96)   0           ['batch_norma
lization_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_pooling
2d_2[0][0]']

 conv2d_34 (Conv2D)             (None, 17, 17, 128)  98304       ['mixed3[0]
[0]']

 batch_normalization_34 (BatchN  (None, 17, 17, 128)  384        ['conv2d_34
[0][0]']
 ormalization)

 activation_34 (Activation)     (None, 17, 17, 128)  0           ['batch_norma
lization_34[0][0]']

 conv2d_35 (Conv2D)             (None, 17, 17, 128)  114688      ['activation_
34[0][0]']

 batch_normalization_35 (BatchN  (None, 17, 17, 128)  384        ['conv2d_35
[0][0]']
 ormalization)

 activation_35 (Activation)     (None, 17, 17, 128)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 128)  384        ['conv2d_31
[0][0]']
 ormalization)

 batch_normalization_36 (BatchN  (None, 17, 17, 128)  384        ['conv2d_36
[0][0]']
 ormalization)

 activation_31 (Activation)     (None, 17, 17, 128)  0           ['batch_norma
lization_31[0][0]']

 activation_36 (Activation)     (None, 17, 17, 128)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 128)  384        ['conv2d_32
[0][0]']
 ormalization)

 batch_normalization_37 (BatchN  (None, 17, 17, 128)  384        ['conv2d_37
[0][0]']
 ormalization)

 activation_32 (Activation)     (None, 17, 17, 128)  0          ['batch_norma
lization_32[0][0]']

 activation_37 (Activation)     (None, 17, 17, 128)  0          ['batch_norma
lization_37[0][0]']

 average_pooling2d_3 (AveragePo  (None, 17, 17, 768)  0         ['mixed3[0]
[0]']
 oling2D)

 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_poo
ling2d_3[0][0]']

 batch_normalization_30 (BatchN  (None, 17, 17, 192)  576        ['conv2d_30
[0][0]']
 ormalization)

 batch_normalization_33 (BatchN  (None, 17, 17, 192)  576        ['conv2d_33
[0][0]']
 ormalization)

 batch_normalization_38 (BatchN  (None, 17, 17, 192)  576        ['conv2d_38
[0][0]']
 ormalization)

 batch_normalization_39 (BatchN  (None, 17, 17, 192)  576        ['conv2d_39
[0][0]']
 ormalization)

 activation_30 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_30[0][0]']

 activation_33 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_33[0][0]']

 activation_38 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_38[0][0]']

 activation_39 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_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 (BatchN  (None, 17, 17, 160)  480        ['conv2d_44
[0][0]']
 ormalization)

 activation_44 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_44[0][0]']

 conv2d_45 (Conv2D)             (None, 17, 17, 160)  179200      ['activation_
44[0][0]']

 batch_normalization_45 (BatchN  (None, 17, 17, 160)  480        ['conv2d_45
[0][0]']
 ormalization)

 activation_45 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 160)  480        ['conv2d_41
[0][0]']
 ormalization)

 batch_normalization_46 (BatchN  (None, 17, 17, 160)  480        ['conv2d_46
[0][0]']
 ormalization)

 activation_41 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_41[0][0]']

 activation_46 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 160)  480        ['conv2d_42
[0][0]']
 ormalization)

 batch_normalization_47 (BatchN  (None, 17, 17, 160)  480        ['conv2d_47
[0][0]']
 ormalization)

 activation_42 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_42[0][0]']
```

```
 activation_47 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_47[0][0]']

 average_pooling2d_4 (AveragePo  (None, 17, 17, 768)  0          ['mixed4[0]
[0]']
 oling2D)

 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_poo
ling2d_4[0][0]']

 batch_normalization_40 (BatchN  (None, 17, 17, 192)  576        ['conv2d_40
[0][0]']
 ormalization)

 batch_normalization_43 (BatchN  (None, 17, 17, 192)  576        ['conv2d_43
[0][0]']
 ormalization)

 batch_normalization_48 (BatchN  (None, 17, 17, 192)  576        ['conv2d_48
[0][0]']
 ormalization)

 batch_normalization_49 (BatchN  (None, 17, 17, 192)  576        ['conv2d_49
[0][0]']
 ormalization)

 activation_40 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_40[0][0]']

 activation_43 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_43[0][0]']

 activation_48 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_48[0][0]']

 activation_49 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 160)  480        ['conv2d_54
[0][0]']
 ormalization)

 activation_54 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
```

```
lization_54[0][0]']

 conv2d_55 (Conv2D)             (None, 17, 17, 160)  179200      ['activation_
54[0][0]']

 batch_normalization_55 (BatchN  (None, 17, 17, 160)  480        ['conv2d_55
[0][0]']
 ormalization)

 activation_55 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 160)  480        ['conv2d_51
[0][0]']
 ormalization)

 batch_normalization_56 (BatchN  (None, 17, 17, 160)  480        ['conv2d_56
[0][0]']
 ormalization)

 activation_51 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_51[0][0]']

 activation_56 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 160)  480        ['conv2d_52
[0][0]']
 ormalization)

 batch_normalization_57 (BatchN  (None, 17, 17, 160)  480        ['conv2d_57
[0][0]']
 ormalization)

 activation_52 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_52[0][0]']

 activation_57 (Activation)     (None, 17, 17, 160)  0           ['batch_norma
lization_57[0][0]']

 average_pooling2d_5 (AveragePo  (None, 17, 17, 768)  0          ['mixed5[0]
[0]']
 oling2D)

 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_poo
ling2d_5[0][0]']

 batch_normalization_50 (BatchN  (None, 17, 17, 192)  576        ['conv2d_50
[0][0]']
 ormalization)

 batch_normalization_53 (BatchN  (None, 17, 17, 192)  576        ['conv2d_53
[0][0]']
 ormalization)

 batch_normalization_58 (BatchN  (None, 17, 17, 192)  576        ['conv2d_58
[0][0]']
 ormalization)

 batch_normalization_59 (BatchN  (None, 17, 17, 192)  576        ['conv2d_59
[0][0]']
 ormalization)

 activation_50 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_50[0][0]']

 activation_53 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_53[0][0]']

 activation_58 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_58[0][0]']

 activation_59 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_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 (BatchN  (None, 17, 17, 192)  576        ['conv2d_64
[0][0]']
 ormalization)

 activation_64 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_64[0][0]']

 conv2d_65 (Conv2D)             (None, 17, 17, 192)  258048      ['activation_
64[0][0]']

 batch_normalization_65 (BatchN  (None, 17, 17, 192)  576        ['conv2d_65
[0][0]']
 ormalization)

 activation_65 (Activation)     (None, 17, 17, 192)  0          ['batch_norma
lization_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 (BatchN  (None, 17, 17, 192)  576        ['conv2d_61
[0][0]']
 ormalization)

 batch_normalization_66 (BatchN  (None, 17, 17, 192)  576        ['conv2d_66
[0][0]']
 ormalization)

 activation_61 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_61[0][0]']

 activation_66 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 192)  576        ['conv2d_62
[0][0]']
 ormalization)

 batch_normalization_67 (BatchN  (None, 17, 17, 192)  576        ['conv2d_67
[0][0]']
 ormalization)

 activation_62 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_62[0][0]']

 activation_67 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_67[0][0]']

 average_pooling2d_6 (AveragePo  (None, 17, 17, 768)  0          ['mixed6[0]
[0]']
 oling2D)

 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_poo
ling2d_6[0][0]']

 batch_normalization_60 (BatchN  (None, 17, 17, 192)  576        ['conv2d_60
[0][0]']
 ormalization)

 batch_normalization_63 (BatchN  (None, 17, 17, 192)  576        ['conv2d_63
[0][0]']
 ormalization)

 batch_normalization_68 (BatchN  (None, 17, 17, 192)  576        ['conv2d_68
[0][0]']
 ormalization)
```

```
 batch_normalization_69 (BatchN  (None, 17, 17, 192)  576         ['conv2d_69
[0][0]']
 ormalization)

 activation_60 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_60[0][0]']

 activation_63 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_63[0][0]']

 activation_68 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_68[0][0]']

 activation_69 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 192)  576         ['conv2d_72
[0][0]']
 ormalization)

 activation_72 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_72[0][0]']

 conv2d_73 (Conv2D)             (None, 17, 17, 192)  258048      ['activation_
72[0][0]']

 batch_normalization_73 (BatchN  (None, 17, 17, 192)  576         ['conv2d_73
[0][0]']
 ormalization)

 activation_73 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_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 (BatchN  (None, 17, 17, 192)  576         ['conv2d_70
[0][0]']
 ormalization)

 batch_normalization_74 (BatchN  (None, 17, 17, 192)  576         ['conv2d_74
[0][0]']
 ormalization)

 activation_70 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
lization_70[0][0]']

 activation_74 (Activation)     (None, 17, 17, 192)  0           ['batch_norma
```

```
lization_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 (BatchN  (None, 8, 8, 320)   960         ['conv2d_71
[0][0]']
 ormalization)

 batch_normalization_75 (BatchN  (None, 8, 8, 192)   576         ['conv2d_75
[0][0]']
 ormalization)

 activation_71 (Activation)     (None, 8, 8, 320)    0           ['batch_norma
lization_71[0][0]']

 activation_75 (Activation)     (None, 8, 8, 192)    0           ['batch_norma
lization_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_pooling
2d_3[0][0]']

 conv2d_80 (Conv2D)             (None, 8, 8, 448)    573440      ['mixed8[0]
[0]']

 batch_normalization_80 (BatchN  (None, 8, 8, 448)   1344        ['conv2d_80
[0][0]']
 ormalization)

 activation_80 (Activation)     (None, 8, 8, 448)    0           ['batch_norma
lization_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 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_77
[0][0]']
 ormalization)

 batch_normalization_81 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_81
[0][0]']
 ormalization)

 activation_77 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_77[0][0]']

 activation_81 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_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 (AveragePo  (None, 8, 8, 1280)  0           ['mixed8[0]
[0]']
 oling2D)

 conv2d_76 (Conv2D)             (None, 8, 8, 320)    409600      ['mixed8[0]
[0]']

 batch_normalization_78 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_78
[0][0]']
 ormalization)

 batch_normalization_79 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_79
[0][0]']
 ormalization)

 batch_normalization_82 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_82
[0][0]']
 ormalization)

 batch_normalization_83 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_83
[0][0]']
 ormalization)

 conv2d_84 (Conv2D)             (None, 8, 8, 192)    245760      ['average_poo
ling2d_7[0][0]']

 batch_normalization_76 (BatchN  (None, 8, 8, 320)   960         ['conv2d_76
[0][0]']
 ormalization)

 activation_78 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_78[0][0]']

 activation_79 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_79[0][0]']

 activation_82 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_82[0][0]']

 activation_83 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_83[0][0]']

 batch_normalization_84 (BatchN  (None, 8, 8, 192)   576         ['conv2d_84
[0][0]']
 ormalization)

 activation_76 (Activation)     (None, 8, 8, 320)    0           ['batch_norma
lization_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_norma
lization_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 (BatchN  (None, 8, 8, 448)   1344        ['conv2d_89
[0][0]']
 ormalization)

 activation_89 (Activation)     (None, 8, 8, 448)    0           ['batch_norma
lization_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 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_86
[0][0]']
 ormalization)

 batch_normalization_90 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_90
[0][0]']
 ormalization)

 activation_86 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_86[0][0]']

 activation_90 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_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 (AveragePo  (None, 8, 8, 2048)  0           ['mixed9[0]
[0]']
 oling2D)

 conv2d_85 (Conv2D)             (None, 8, 8, 320)    655360      ['mixed9[0]
```

```
[0]']

 batch_normalization_87 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_87
[0][0]']
 ormalization)

 batch_normalization_88 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_88
[0][0]']
 ormalization)

 batch_normalization_91 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_91
[0][0]']
 ormalization)

 batch_normalization_92 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_92
[0][0]']
 ormalization)

 conv2d_93 (Conv2D)             (None, 8, 8, 192)    393216      ['average_poo
ling2d_8[0][0]']

 batch_normalization_85 (BatchN  (None, 8, 8, 320)   960         ['conv2d_85
[0][0]']
 ormalization)

 activation_87 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_87[0][0]']

 activation_88 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_88[0][0]']

 activation_91 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_91[0][0]']

 activation_92 (Activation)     (None, 8, 8, 384)    0           ['batch_norma
lization_92[0][0]']

 batch_normalization_93 (BatchN  (None, 8, 8, 192)   576         ['conv2d_93
[0][0]']
 ormalization)

 activation_85 (Activation)     (None, 8, 8, 320)    0           ['batch_norma
lization_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_norma
lization_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 (GlobalAveragePooling  (None, 2048)        0           ['mixed10[0]
[0]']
 2D)

 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 [26]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [21]:
```
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 en
```

In [22]:
```
encoded_image = img_encoder.predict(np.array([new_image]))
```

In [23]:
```
encoded_image
```

Out[23]:
```
array([[0.63806605, 0.4887302 , 0.05526249, ..., 0.64255637, 0.2959523 ,
        0.49004343]], dtype=float32)
```

Create encodings for all images

In [24]:
```
# The following generator function will 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).
def img_generator(img_list):
  for file_name in img_list:
    image = get_image(file_name)
    yield np.array(image, ndmin=4)
```

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

In [29]:
```
enc_train = img_encoder.predict_generator(img_generator(train_list), steps=le
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `
Model.predict_generator` is deprecated and will be removed in a future versio
n. Please use `Model.predict`, which supports generators.
  """Entry point for launching an IPython kernel.
6000/6000 [==============================] - 1381s 230ms/step
```

In [30]:
```
enc_train[11]
```

Out[30]:
```
array([0.26818594, 1.0321662 , 0.5851619 , ..., 1.2316743 , 0.17969307,
       0.22405306], dtype=float32)
```

In [31]:
```
enc_dev = img_encoder.predict_generator(img_generator(dev_list), steps=len(de
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `
Model.predict_generator` is deprecated and will be removed in a future versio
n. Please use `Model.predict`, which supports generators.
  """Entry point for launching an IPython kernel.
1000/1000 [==============================] - 235s 235ms/step
```

In [32]:
```
enc_test = img_encoder.predict_generator(img_generator(test_list), steps=len(
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `
Model.predict_generator` is deprecated and will be removed in a future versio
n. Please use `Model.predict`, which supports generators.
  """Entry point for launching an IPython kernel.
1000/1000 [==============================] - 234s 234ms/step
```

Save the resulting matrices, so we do not have to run the encoder again.

In [33]:
```
np.save("gdrive/My Drive/"+my_data_dir+"/outputs/encoded_images_train.npy", e
np.save("gdrive/My Drive/"+my_data_dir+"/outputs/encoded_images_dev.npy", enc
np.save("gdrive/My Drive/"+my_data_dir+"/outputs/encoded_images_test.npy", en
```

Text (Caption) Data Preparation

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

Reading image descriptions

In [34]:
```
def read_image_descriptions(filename):
    image_descriptions = defaultdict(list)
    # ...
    files = open(filename, 'r')

    for file in files:
      file_name = file.split('#')[0]
      caption = file.split('#')[1].split('\t')[1].split()
      # Convert each token to lower case
      caption = [word.lower() for word in caption]
      # Pad each caption with a START token on the left
      caption.insert(0, '<START>')
      # and an END token on the right
      caption.append('<END>')

      image_descriptions[file_name].append(caption)
```

```
    return image_descriptions
```

In [35]:
```
descriptions = read_image_descriptions("gdrive/My Drive/"+my_data_dir+"/Flick
```

In [36]:
```
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', 'i
n', 'a', 'courtyard', '.', '<END>'], ['<START>', 'two', 'people', 'play', 'o
n', 'a', 'long', 'skateboard', '.', '<END>'], ['<START>', 'two', 'small', 'chi
ldren', 'in', 'red', 'shirts', 'playing', 'on', 'a', 'skateboard', '.', '<END
>'], ['<START>', 'two', 'young', 'children', 'on', 'a', 'skateboard', 'going',
'across', 'a', 'sidewalk', '<END>']]
```

Creating Word Indices

In [37]:
```
# Create a set of tokens in the training data
tokens = []
for captions in descriptions.values():
  for caption in captions:
    for word in caption:
      tokens.append(word)
tokens = set(tokens)

# Convert the set into a list and sort it
tokens = list(tokens)
tokens.sort()
```

In [38]:
```
id_to_word = defaultdict()
n = len(tokens)
for i in range(n):
  id_to_word[i] = tokens[i]
```

In [39]:
```
word_to_id = {value: k for k, value in id_to_word.items()}
```

In [40]:
```
word_to_id['dog'] # should print an integer
```

Out[40]:
```
2307
```

In [41]:
```
id_to_word[1985] # should print a token
```

Out[41]:
```
'crudely'
```

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.

Basic Decoder Model

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

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 [42]:
```
max(len(description) for image_id in train_list for description in descriptio
```

Out[42]:
```
40
```

In [43]:
```
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_i
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=[

model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 40)]              0

 embedding (Embedding)       (None, 40, 300)           2675100

 bidirectional (Bidirectiona  (None, 1024)             3330048
 l)

 dense (Dense)               (None, 8917)              9139925

=================================================================
Total params: 15,145,073
Trainable params: 15,145,073
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

In [44]:
```
def text_training_generator(batch_size=128):
```

```
    input = []
    output = []

    while(True):
      for path in train_list:
        for caption in descriptions[path]:
          for i in range(len(caption)-1):
            count = 0
            generator_num = np.zeros(40) #If the input sequence is shorter th
            generator = caption[:i+1]
            word = caption[i+1]
            word_index = word_to_id[word]

            for word in generator:
              generator_num[count]=word_to_id[word]
              count += 1

            input.append(generator_num)
            one_hot_encoding = to_categorical(word_index, vocab_size)
            output.append(one_hot_encoding)

            if(len(input)==128):
              input = np.asarray(input)
              output = np.asarray(output)
              reshaped_input= np.reshape(input,(batch_size, 40))
              reshaped_output= np.reshape(output,(batch_size, vocab_size))
              yield (reshaped_input, reshaped_output)

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

The training might take a while.

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

```
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `
Model.fit_generator` is deprecated and will be removed in a future version. Pl
ease use `Model.fit`, which supports generators.
  """Entry point for launching an IPython kernel.
1875/1875 [==============================] - 204s 106ms/step - loss: 4.2589 -
accuracy: 0.2955
Epoch 2/10
1875/1875 [==============================] - 197s 105ms/step - loss: 3.7056 -
accuracy: 0.3574
```

```
Epoch 3/10
1875/1875 [==============================] - 197s 105ms/step - loss: 3.5340 -
accuracy: 0.3736
Epoch 4/10
1875/1875 [==============================] - 197s 105ms/step - loss: 3.4350 -
accuracy: 0.3832
Epoch 5/10
1875/1875 [==============================] - 197s 105ms/step - loss: 3.3821 -
accuracy: 0.3893
Epoch 6/10
1875/1875 [==============================] - 197s 105ms/step - loss: 3.3112 -
accuracy: 0.3956
Epoch 7/10
1875/1875 [==============================] - 197s 105ms/step - loss: 3.2918 -
accuracy: 0.3988
Epoch 8/10
1875/1875 [==============================] - 197s 105ms/step - loss: 3.2804 -
accuracy: 0.4011
Epoch 9/10
1875/1875 [==============================] - 197s 105ms/step - loss: 3.2524 -
accuracy: 0.4034
Epoch 10/10
1875/1875 [==============================] - 197s 105ms/step - loss: 3.3113 -
accuracy: 0.4023
```

Out[ ]:
```
<keras.callbacks.History at 0x7f46f504b290>
```

Greedy Decoder

In [45]:
```
def decoder():
  prediction = "<START>"
  output = ["<START>"]

  i = 1
  sequence = np.zeros(MAX_LEN)
  # Start with the sequence ["<START>"]
  sequence[0] = word_to_id["<START>"]

  while(i < MAX_LEN and prediction != "<END>" ):
    # Use the model to predict the most likely word
    word_pred = model.predict(np.array([sequence]))
    id = np.where(word_pred == np.amax(word_pred))
    sequence[i]= id[1][0]
    prediction = id_to_word[id[1][0]]
    output.append(prediction)

    i += 1

  return output
```

In [46]:
```
print(decoder())
```

```
['<START>', 'passers', 'passers', 'passers', 'passers', 'three-wheeled', 'thre
e-wheeled', 'three-wheeled', 'three-wheeled', 'three-wheeled', 'three-wheele
d', 'three-wheeled', 'three-wheeled', 'three-wheeled', 'three-wheeled', 'three
-wheeled', 'three-wheeled', 'three-wheeled', 'three-wheeled', 'three-wheeled',
'three-wheeled', 'three-wheeled', 'three-wheeled', 'three-wheeled', 'three-whe
eled', 'three-wheeled', 'three-wheeled', 'footprints', 'footprints', 'footprin
ts', 'footprints', 'footprints', 'footprints', 'footprints', 'footprints', 'fo
otprints', 'footprints', 'waters', 'waters', 'waters']
```

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.

Reference: np.random.multinomial

In [47]:
```
def sample_decoder():
  prediction = "<START>"
  output = ["<START>"]

  count = 1
  sequence = np.zeros(MAX_LEN)
  sequence[0] = word_to_id["<START>"]

  while(count < 40 and prediction != "<END>"):
    pred = model.predict(np.array([sequence]))
    pred = pred.tolist()[0]

    norm = []
    for i in pred:
      norm.append(i/sum(pred))

    multi = np.random.multinomial(10, norm, size=None)
    id = np.where(multi == np.amax(multi))

    sequence[count] = id[0][0]
    prediction = id_to_word[id[0][0]]
    output.append(prediction)
    count += 1

  return output
```

In [30]:
```
for i in range(10):
    print(sample_decoder())
```

```
['<START>', 'bagpipes', 'activities', 'bi-plane', 'bloodied', 'demonstrating',
'bared', 'adjust', 'badminton', 'attentively', 'bananas', 'brushes', 'admire
d', 'black-striped', 'ability', 'chidl', 'adjustments', 'aged', 'ads', 'backba
ck', 'bitten', 'asleep', 'huskey', 'beret', 'bear', 'approaches', 'applying',
'argues', 'boogieboard', 'crudely', 'bangles', 'carefully', 'adobe', 'blower',
'"', 'bicyclist', 'body', 'affectionately', 'better', 'beat']
['<START>', 'assist', 'brown-spotted', 'barbeque', 'carring', 'bring', 'brough
t', 'amazement', 'add', 'guns', 'complimentary', '52', 'calm', 'attempts', 'ba
ss', 'cross', 'beaks', 'buggies', 'bat', 'almost-dried', 'allow', 'comfortabl
e', 'bespectacled', 'appearing', 'african', 'barn-like', 'fend', 'hang', 'ac',
'any', 'cinderblock', 'aiming', 'determined', 'convenience', 'bill', 'blood',
'banister', 'bohemian', 'balanced', 'bumps']
['<START>', 'congregated', 'ascends', 'cruise', 'ambulance', 'breathing', 'bas
ketballs', 'campground', 'antenna', 'aided', 'adoring', 'area', 'created', 'bi
kinis', 'bow-like', 'ballerinas', 'acrobat', 'biek', 'birds', 'cluster', 'ante
nna', 'compact', '661', 'area', 'anticipates', 'age', '58', 'bark', 'brick',
'accelerates', 'hers', 'beak', 'african', 'apparantly', 'chalk', 'crocodile',
'album', '52', 'background', "'slide"]
['<START>', 'aside', 'celebration', 'bruised', 'cover', 'after', "'n'", 'archw
ay', 'bomber', 'affectionately', 'advance', 'bouncy', 'certificates', '1950s',
'body-board', 'diner', 'contents', 'carpenters', 'belts', 'bundled-up', 'bikin
g', 'blow-up', 'electric', 'endzone', 'beverage', 'attrative', 'bicyler', 'dus
ted', 'raging', 'buena', 'confronts', 'clover-filled', 'balanced', 'armbands',
'call', 'boats', 'blasts', '8', 'approachs', 'advertising']
['<START>', 'amidst', 'ax', '625', 'archeologist', 'cries', 'adjusts', 'carava
```

```
n', 'cattle', 'album', '<START>', 'biscuit', 'earphone', 'beers', 'been', 'at
e', 'clouds', 'acrobat', '57', 'ascends', 'bagpipers', 'announcer', 'cracked',
'cannonball', 'away', 'cards', 'amidst', 'around', 'bumpy', '21', 'brighty',
'alike', '57', 'bordering', 'cattle', 'aid', 'backgound', 'accident', '22', 'a
lcove']
['<START>', 'arabian', 'banana', 'acts', 'anti-tax', 'cloak', '19', 'ascend',
'drill', 'ascends', 'at', 'all-male', 'caring', 'bee', 'bald', 'athelete', 'bu
ll', 'blows', 'bigwheels', 'bohemians', 'arc', 'album', 'aloft', '4-wheeler',
'butchers', 'acrouss', 'bowed', 'backed', 'embankment', 'agile', 'beads', '3r
d', 'backseat', 'blue-gray', 'furious', 'above-ground', 'apparatus', 'bracin
g', 'beaks', 'card']
['<START>', 'barren', 'cardboard', 'bald', 'ace', 'backwards', 'alongside', 'b
usk', 'badges', 'chi', 'antelope', 'beachgoers', 'coached', 'contained', 'angl
es', 'convienience', 'aerial', 'angel', 'camps', 'brighly', 'annoyed', 'blur',
'badly', 'bobbed', 'community', 'begs', 'bridges', 'corn', 'aliens', 'barack',
'americans', 'attempts', 'bridesmaid', 'braiding', 'batsman', 'antlers', 'acto
r', 'brilliant', 'dandelion', 'beg']
['<START>', 'grownup', 'electronic', 'blank', 'automobile', 'attentive', '<STA
RT>', 'bark', 'biek', 'biking', 'bench', 'canoeing', 'again', 'bicyclist', '8
0', 'classes', 'comforts', 'airborne', 'blocked', 'also', 'biting', 'bullet',
'32', 'camel', 'backgound', 'attacks', 'aimed', '5', 'closer', 'atm', 'enteran
ce', 'curtain', 'alongside', 'anti-tax', 'berry', 'adolescents', 'bands', 'bla
ck-blue', 'black-green', 'cine']
['<START>', 'bathing', 'bumpy', 'atvs', 'air', 'attacks', 'containing', 'broth
er', 'bandaged', 'artificial', 'flamboyant', 'attempting', 'blond', 'bunny',
'accompanying', 'beige', 'blurred', 'baring', 'clergy', 'alleyway', 'ashtray',
'confronting', 'course', 'brightly-colored', 'crime', 'ash', 'bomber', 'atriu
m', 'fumble', 'compact', 'benches', 'back', 'africans', 'mannequins', 'condom
s', 'alleyway', 'adventures', 'breed', 'ashy', 'baggy']
['<START>', 'arena', 'adobe', 'bookcase', 'apron', 'butts', '.', 'autos', 'bow
led', 'affixed', 'articles', 'basett', 'assisting', 'bungee-type', 'atrium',
'eyed', 'along', 'balding', 'anticipates', 'archways', 'bluejean', 'bales', 'b
leachers', 'blossoming', 'bares', 'cellphone', 'brass', 'led', 'adjust', 'bann
ister', 'coppery', 'chip', 'attrative', 'bottle', 'birds', 'alter', 'across',
'acoustic', '93', 'bedroll']
```

Conditioning on the Image

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 [48]:

```
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_i
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=[

model.summary()
```

```
Model: "model_2"
__________________________________________________________________________________
_____________________
 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)      2675100     ['input_4[0]
[0]']

 concatenate_2 (Concatenate)    (None, 40, 600)      0           ['repeat_vect
or[0][0]',
                                                                  'embedding_1
[0][0]']

 bidirectional_1 (Bidirectional  (None, 512)         1755136     ['concatenate
_2[0][0]']
 )

 dense_2 (Dense)                (None, 8917)         4574421     ['bidirection
al_1[0][0]']

==================================================================================
====================
Total params: 9,619,357
Trainable params: 9,619,357
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.

In [49]:
```
enc_train = np.load("gdrive/My Drive/"+my_data_dir+"/outputs/encoded_images_t
enc_dev = np.load("gdrive/My Drive/"+my_data_dir+"/outputs/encoded_images_dev
```

In [50]:
```
def training_generator(batch_size=128):
  image_inputs = []
  text_inputs = []
  output = []

  while(True):
```

```
    for index in range(len(train_list)): # list of training image
      name = train_list[index]
      encoding = enc_train[index]
      for caption in descriptions[name]: # list of captions
        for i in range(len(caption)-1):
          count = 0
          generator_num = np.zeros(MAX_LEN)
          generator = caption[:i+1]
          word = caption[i+1]
          word_index = word_to_id[word]

          for word in generator:
              generator_num[count]=word_to_id[word]
              count += 1

          image_inputs.append(encoding)
          text_inputs.append(generator_num)
          one_hot_encoding = to_categorical(word_index, vocab_size)
          output.append(one_hot_encoding)

          if(len(text_inputs)==128):
            image_inputs = np.asarray(image_inputs)
            text_inputs = np.asarray(text_inputs)
            output = np.asarray(output)
            reshaped_image_inputs = np.reshape(image_inputs, (batch_size,2048
            reshaped_text_inputs = np.reshape(text_inputs,(batch_size, 40))
            reshaped_output = np.reshape(output,(batch_size, vocab_size))
            yield ([reshaped_image_inputs, reshaped_text_inputs], reshaped_ou

            image_inputs = []
            text_inputs = []
            output = []
```

In [34]:

```
batch_size = 128
generator = training_generator(batch_size)
steps = len(train_list) * MAX_LEN // batch_size
```

The training may take a while.

In [ ]:

```
model.fit_generator(generator, steps_per_epoch=steps, verbose=True, epochs=20
```

```
Epoch 1/20
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `
Model.fit_generator` is deprecated and will be removed in a future version. Pl
ease use `Model.fit`, which supports generators.
  """Entry point for launching an IPython kernel.
1875/1875 [==============================] - 134s 69ms/step - loss: 4.4991 - a
ccuracy: 0.2564
Epoch 2/20
1875/1875 [==============================] - 129s 69ms/step - loss: 3.6430 - a
ccuracy: 0.3646
Epoch 3/20
1875/1875 [==============================] - 129s 69ms/step - loss: 3.4418 - a
ccuracy: 0.3865
Epoch 4/20
1875/1875 [==============================] - 129s 69ms/step - loss: 3.3389 - a
ccuracy: 0.3949
Epoch 5/20
1875/1875 [==============================] - 129s 69ms/step - loss: 3.2923 - a
ccuracy: 0.4019
Epoch 6/20
```

```
1875/1875 [==============================] - 129s 69ms/step - loss: 3.2296 - a
ccuracy: 0.4073
Epoch 7/20
1875/1875 [==============================] - 129s 69ms/step - loss: 3.2357 - a
ccuracy: 0.4097
Epoch 8/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2388 - a
ccuracy: 0.4114
Epoch 9/20
1875/1875 [==============================] - 128s 69ms/step - loss: 3.2273 - a
ccuracy: 0.4152
Epoch 10/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2299 - a
ccuracy: 0.4165
Epoch 11/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2490 - a
ccuracy: 0.4152
Epoch 12/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2352 - a
ccuracy: 0.4175
Epoch 13/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2420 - a
ccuracy: 0.4200
Epoch 14/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2153 - a
ccuracy: 0.4233
Epoch 15/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2241 - a
ccuracy: 0.4240
Epoch 16/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2329 - a
ccuracy: 0.4234
Epoch 17/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2274 - a
ccuracy: 0.4269
Epoch 18/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2025 - a
ccuracy: 0.4274
Epoch 19/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2460 - a
ccuracy: 0.4278
Epoch 20/20
1875/1875 [==============================] - 128s 68ms/step - loss: 3.2191 - a
ccuracy: 0.4303
```

Out[ ]:
```
<keras.callbacks.History at 0x7f46f43b0f90>
```

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

The weights is save so we don't have to train the dataset again. To load the model:

In [51]:
```
model.load_weights("gdrive/My Drive/"+my_data_dir+"/outputs/model.h5")
```

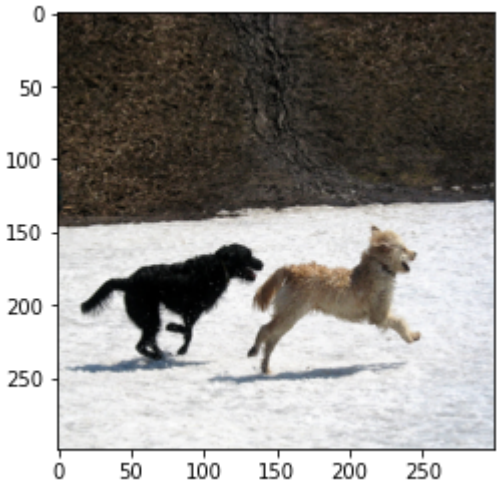
Modify the simple greedy decoder for the text-only generator so that it takes an encoded image (a vector of length 2048) as input, and returns a sequence.

In [52]:
```
def image_decoder(enc_image):
  prediction = "<START>"
  output = ["<START>"]

  i = 1
```

```
    sequence = np.zeros(MAX_LEN)
    sequence[0] = word_to_id["<START>"]

    while( i < 40 and prediction != "<END>" ):
      pred = model.predict([np.array([enc_image]),np.array([sequence])])
      id = np.where(pred == np.amax(pred))
      sequence[i]= id[1][0]
      prediction = id_to_word[id[1][0]]
      output.append(prediction)

      i += 1

    return output
```

In [37]:
```
plt.imshow(get_image(train_list[0]))
image_decoder(enc_train[0])
```

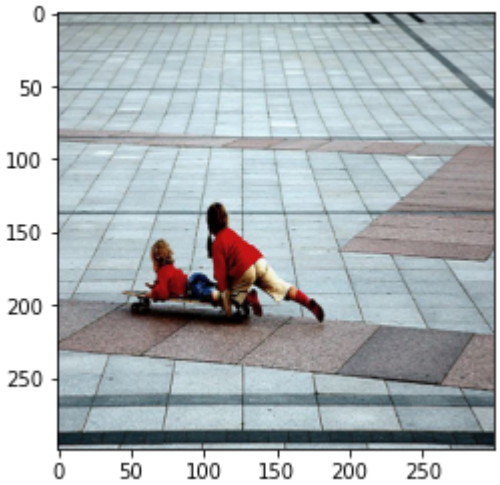
Out[37]:
```
['<START>', 'a', 'dog', 'runs', 'through', 'the', 'water', '.', '<END>']
```



Now we can apply the model to dev images and get reasonable captions:

In [38]:
```
plt.imshow(get_image(dev_list[0]))
image_decoder(enc_dev[0])
```

Out[38]:
```
['<START>',
 'a',
 'man',
 'sits',
 'on',
 'a',
 'bench',
 'near',
 'a',
 'wooden',
 'fence',
 '.',
 '<END>']
```



Beam Search Decoder

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 [53]:
```
def img_beam_decoder(n, image_enc):
  input = [word_to_id["<START>"]]
  array = [[input, 0.0]]

  while len(array[0][0]) < MAX_LEN:
    temp = []

    for seq in array:
      captions_num = pad_sequences([seq[0]], maxlen=40, padding ='post')
      captions_num_np = np.asarray([captions_num]).reshape((1,40))
      img = np.asarray([image_enc]).reshape((1,2048))
      probability = model.predict([img, captions_num_np])
      prob_seq = np.argsort(probability[0])[-n:]

      for i in prob_seq:
        p = seq[1]
        p += probability[0][i]
        next_caption = seq[0][:]
        next_caption.append(i)
        temp.append([next_caption, p])
```

```
    array = temp
    # Sort by highest total probability
    array = sorted(array, reverse=False, key= lambda x: x[1])
    array = array[-n:]

  array = array[-1][0]

  temporary_caption=[]
  for i in array:
    temporary_caption.append(id_to_word[i])

  # Most probable sequence
  caption = []
  for i in temporary_caption:
    if i != "<END>":
      caption.append(i)
    else:
        break

  caption.append("<END>")
  return caption
```

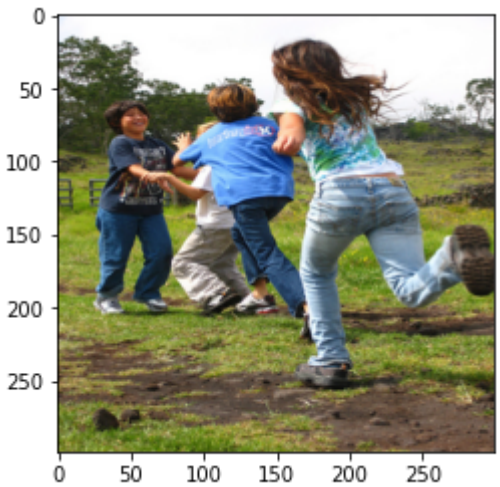
In [40]:
```
img_beam_decoder(3, enc_dev[1])
```

Out[40]:
```
['<START>', 'a', 'person', 'climbing', 'a', 'rock', 'face', '.', '<END>']
```

Now we can show images each with 1) their greedy output, 2) beam search at n=3 3) beam search at n=5.

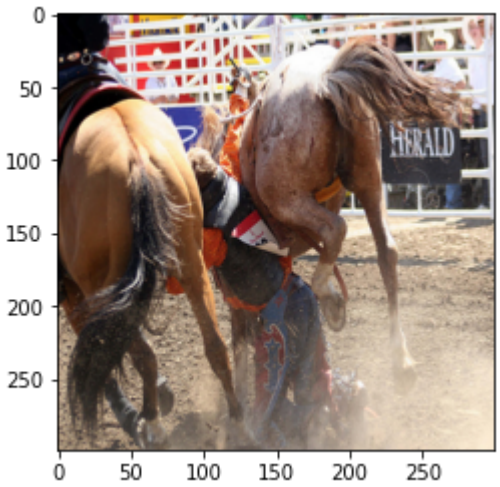
In [41]:
```
plt.imshow(get_image(dev_list[7]))
print("Greedy Output: ", image_decoder(enc_dev[7]))
print("Beam Search at n=3: ", img_beam_decoder(3, enc_dev[7]))
print("Beam Search at n=5:", img_beam_decoder(5, enc_dev[7]))
```

```
Greedy Output:  ['<START>', 'a', 'man', 'sits', 'on', 'a', 'bench', 'with',
'a', 'baby', 'in', 'his', 'hand', '.', '<END>']
Beam Search at n=3:  ['<START>', 'a', 'young', 'girl', 'sits', 'on', 'a', 'ben
ch', '.', '<END>']
Beam Search at n=5: ['<START>', 'a', 'little', 'girl', 'sitting', 'on', 'a',
'bench', '.', '<END>']
```



In [42]:
```
plt.imshow(get_image(dev_list[77]))
print("Greedy Output: ", image_decoder(enc_dev[77]))
print("Beam Search at n=3: ", img_beam_decoder(3, enc_dev[77]))
print("Beam Search at n=5:", img_beam_decoder(5, enc_dev[77]))
```

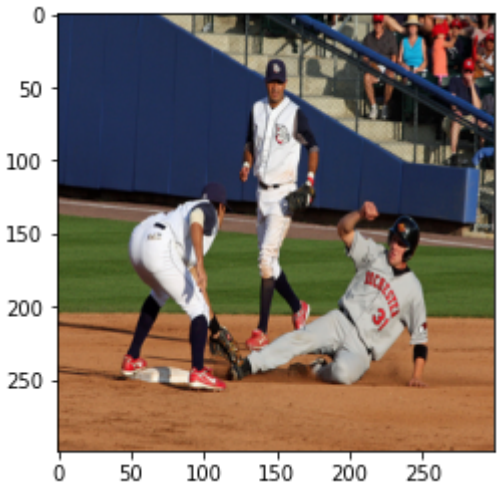
```
Greedy Output:  ['<START>', 'a', 'man', 'and', 'a', 'dog', 'are', 'walking',
'along', 'a', 'hill', 'looking', 'at', 'camera', '.', '<END>']
Beam Search at n=3:  ['<START>', 'there', 'are', 'two', 'dogs', 'looking', 'a
t', 'the', 'camera', '.', '<END>']
Beam Search at n=5: ['<START>', 'a', 'group', 'of', 'dogs', 'are', 'playing',
'dogs', '.', '<END>']
```



In [43]:

```
plt.imshow(get_image(dev_list[100]))
print("Greedy Output: ", image_decoder(enc_dev[100]))
print("Beam Search at n=3: ", img_beam_decoder(3, enc_dev[100]))
print("Beam Search at n=5:", img_beam_decoder(5, enc_dev[100]))
```

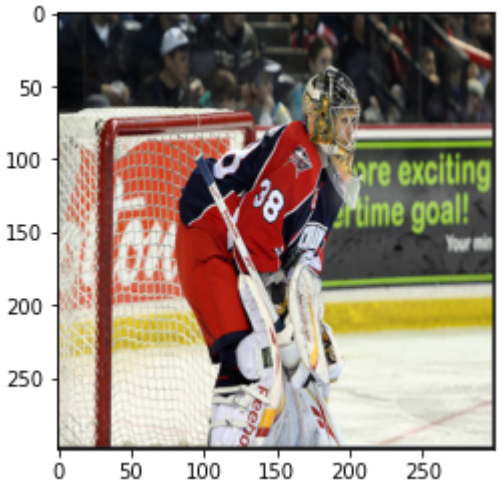
```
Greedy Output:  ['<START>', 'a', 'man', 'in', 'a', 'baseball', 'uniform', 'swi
ngs', 'a', 'tennis', 'bat', '.', '<END>']
Beam Search at n=3:  ['<START>', 'a', 'baseball', 'player', 'swings', 'the',
'ball', '.', '<END>']
Beam Search at n=5: ['<START>', 'a', 'baseball', 'player', 'playing', 'basebal
l', '.', '<END>']
```



In [44]:

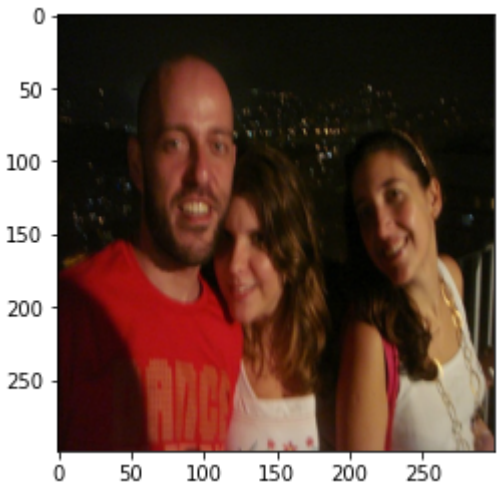
```
plt.imshow(get_image(dev_list[200]))
print("Greedy Output: ", image_decoder(enc_dev[200]))
print("Beam Search at n=3: ", img_beam_decoder(3, enc_dev[200]))
print("Beam Search at n=5:", img_beam_decoder(5, enc_dev[200]))
```

```
Greedy Output:  ['<START>', 'a', 'basketball', 'player', 'a', 'ball', '.', '<E
ND>']
Beam Search at n=3:  ['<START>', 'a', 'basketball', 'player', 'the', 'ball',
'.', '<END>']
Beam Search at n=5: ['<START>', 'a', 'basketball', 'player', 'the', 'ball',
'.', '<END>']
```



In [45]:
```
plt.imshow(get_image(dev_list[123]))
print("Greedy Output: ", image_decoder(enc_dev[123]))
print("Beam Search at n=3: ", img_beam_decoder(3, enc_dev[123]))
print("Beam Search at n=5:", img_beam_decoder(5, enc_dev[123]))
```

```
Greedy Output:  ['<START>', 'a', 'man', 'and', 'a', 'woman', 'posing', 'for',
'a', 'picture', '.', '<END>']
Beam Search at n=3:  ['<START>', 'a', 'group', 'of', 'young', 'women', 'are',
'posing', 'for', 'a', 'picture', '.', '<END>']
Beam Search at n=5: ['<START>', 'a', 'group', 'of', 'people', 'posing', 'for',
'a', 'picture', '.', '<END>']
```



Evaluation using BLEU

In [54]:
```
from nltk.translate.bleu_score import sentence_bleu
from nltk.translate.bleu_score import corpus_bleu
```

In [55]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [56]:
```
sentence_bleu(descriptions[dev_list[0]], image_decoder(enc_dev[0]))
```

Out[56]:
```
0.5266403878479265
```

We can calculate the mean BLEU score for the development dataset:

In [57]:
```
bleus = []

for i in range(len(dev_list)):
  actual = descriptions[dev_list[i]][1:-1]
  predicted = image_decoder(enc_dev[i])[1:-1]
  bleu = sentence_bleu(actual, predicted)
  bleus.append(bleu)

print("Mean BLEU {:4.3f}".format(np.mean(bleus)))
```

```
Mean BLEU 0.470
```

The following calculations may take a while to run.

In [58]:
```
bleus_beam_3 = []

for i in range(len(dev_list)):
  actual = descriptions[dev_list[i]][1:-1]
  predicted = img_beam_decoder(3, enc_dev[i])[1:-1]
  bleu = sentence_bleu(actual, predicted)
  bleus_beam_3.append(bleu)

print("Mean BLEU {:4.3f}".format(np.mean(bleus_beam_3)))
```

```
Mean BLEU 0.456
```

For evaluations on the Inception V2 model, refer to **inception_resnet_v2_LSTM.ipynb** in the doc folder.