homework4

December 16, 2020

1 COMS W4705 - Homework 4

1.1 Image Captioning with Conditioned LSTM Generators

Yassine Benajiba yb2235@cs.columbia.edu

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.
- Implement beam search for the image caption generator.

Please submit a copy of this notebook only, including all outputs. Do not submit any of the data files.

1.1.1 Must-run setups (including setups from P1 and P2 for testing)

[1]:
```
import os
from collections import defaultdict
import numpy as np
import PIL
from matplotlib import pyplot as plt
%matplotlib inline
from pandas.core.common import flatten
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
import tensorflow as tf
tf.config.run_functions_eagerly(True)
from google.colab import drive
my_data_dir="hw5_data"
drive.mount('/content/gdrive')
```

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

[2]:
```
def load_image_list(filename):
    with open(filename,'r') as image_list_f:
        return [line.strip() for line in image_list_f]
train_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/
 ↪Flickr_8k.trainImages.txt')
dev_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8k.
 ↪devImages.txt')
test_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8k.
 ↪testImages.txt')
```

1.1.2 Getting Started

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

[38]:
```
import os
from collections import defaultdict
import numpy as np
import PIL
from matplotlib import pyplot as plt
%matplotlib inline
from pandas.core.common import flatten
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
import tensorflow as tf
tf.config.run_functions_eagerly(True)
from keras.applications.inception_v3 import InceptionV3

from keras.optimizers import Adam

from google.colab import drive
```

1.1.3 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 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/1i9Iun4h3EN1vSd1A1woez0mXJ9vRjFlT?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

[2]:
```python
#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="hw5_data"
```

1.1.4 Mounting your GDrive so you can access the files from Colab

[3]:
```python
#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
```

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.

1.2 Part I: Image Encodings (14 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.

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

[5]:
```python
train_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/
↪Flickr_8k.trainImages.txt')
dev_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8k.
↪devImages.txt')
test_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8k.
↪testImages.txt')
```

Let's see how many images there are

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

[6]: (6000, 1000, 1000)

Each entry is an image filename.

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

[7]: '3693961165_9d6c333d5b.jpg'

The images are located in a subdirectory.

[3]:
```
root_dir = '/content/gdrive/My Drive/hw5_data/'
IMG_PATH = "Flickr8k_Dataset"
```

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

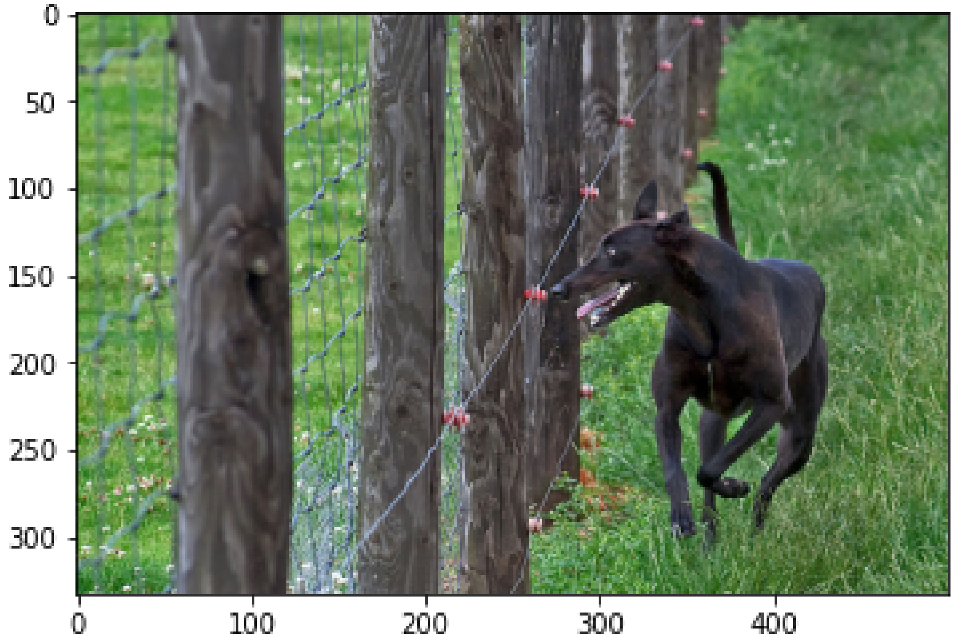
[9]:
```
image = PIL.Image.open(os.path.join(root_dir + IMG_PATH, dev_list[20]))
image
```

[9]:

if you can't see the image, try

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

[ ]: <matplotlib.image.AxesImage at 0x7f26d58e9048>



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_pa

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.

[10]:
```
np.asarray(image).shape
```

[10]:
```
(333, 500, 3)
```

The values range from 0 to 255.

[11]:
```
np.asarray(image)
```

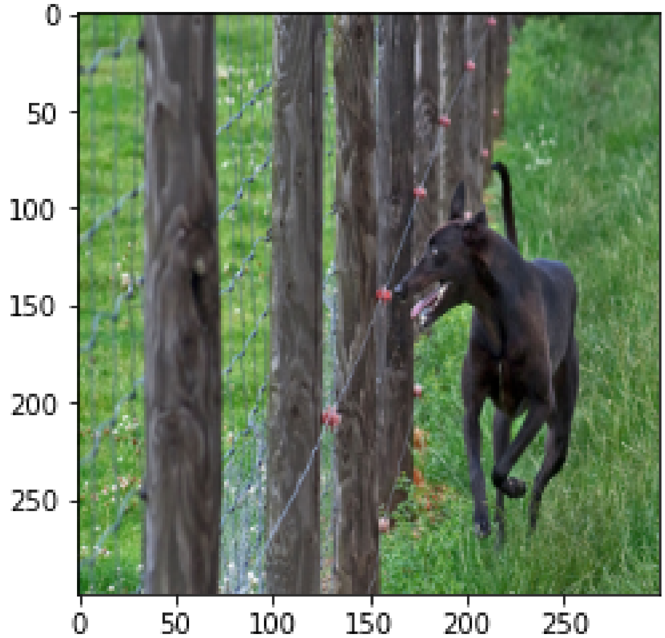
[11]:
```
array([[[118, 161,  89],
        [120, 164,  89],
        [111, 157,  82],
        ...,
        [ 68, 106,  65],
        [ 64, 102,  61],
        [ 65, 104,  60]],

       [[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.

[12]:
```
new_image = np.asarray(image.resize((299,299))) / 255.0
plt.imshow(new_image)
```

[12]:
```
<matplotlib.image.AxesImage at 0x7f0f3f939f28>
```



[13]:
```
new_image.shape
```

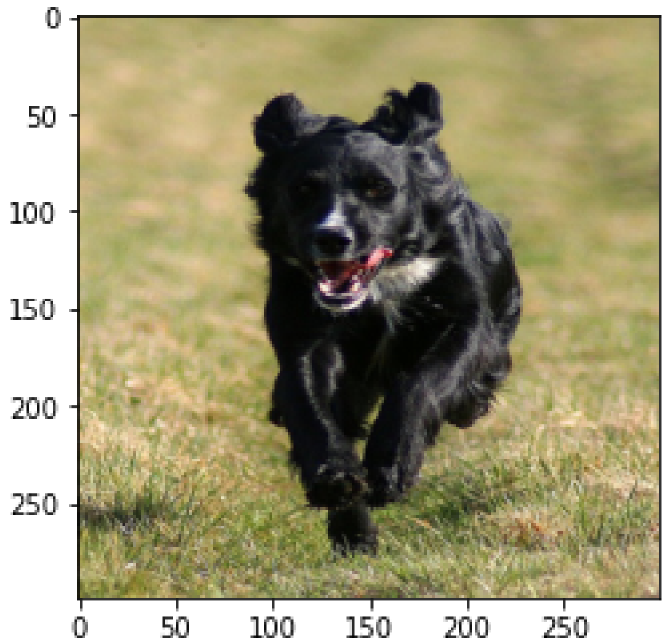
[13]:
```
(299, 299, 3)
```

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

[14]:
```
def get_image(image_name):
    image = PIL.Image.open(os.path.join(root_dir + IMG_PATH, image_name))
    return np.asarray(image.resize((299,299))) / 255.0
```

[15]:
```
plt.imshow(get_image(dev_list[25]))
```

[15]:
```
<matplotlib.image.AxesImage at 0x7f0f3f413710>
```



Next, we load the pre-trained Inception model.

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

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

```
Model: "inception_v3"

--------------------------------------------------------------------------------
------------------
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_2 (InputLayer)            [(None, 299, 299, 3) 0

--------------------------------------------------------------------------------
------------------
conv2d_94 (Conv2D)              (None, 149, 149, 32) 864         input_2[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_94 (BatchNo (None, 149, 149, 32) 96          conv2d_94[0][0]

--------------------------------------------------------------------------------
------------------
activation_94 (Activation)      (None, 149, 149, 32) 0
batch_normalization_94[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_95 (Conv2D)              (None, 147, 147, 32) 9216
```

```
activation_94[0][0]
__________________________________________________________________________________________________
batch_normalization_95 (BatchNo (None, 147, 147, 32) 96          conv2d_95[0][0]
__________________________________________________________________________________________________
activation_95 (Activation)      (None, 147, 147, 32) 0
batch_normalization_95[0][0]
__________________________________________________________________________________________________
conv2d_96 (Conv2D)              (None, 147, 147, 64) 18432
activation_95[0][0]
__________________________________________________________________________________________________
batch_normalization_96 (BatchNo (None, 147, 147, 64) 192         conv2d_96[0][0]
__________________________________________________________________________________________________
activation_96 (Activation)      (None, 147, 147, 64) 0
batch_normalization_96[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 73, 73, 64)   0
activation_96[0][0]
__________________________________________________________________________________________________
conv2d_97 (Conv2D)              (None, 73, 73, 80)   5120
max_pooling2d_4[0][0]
__________________________________________________________________________________________________
batch_normalization_97 (BatchNo (None, 73, 73, 80)   240         conv2d_97[0][0]
__________________________________________________________________________________________________
activation_97 (Activation)      (None, 73, 73, 80)   0
batch_normalization_97[0][0]
__________________________________________________________________________________________________
conv2d_98 (Conv2D)              (None, 71, 71, 192)  138240
activation_97[0][0]
__________________________________________________________________________________________________
batch_normalization_98 (BatchNo (None, 71, 71, 192)  576         conv2d_98[0][0]
__________________________________________________________________________________________________
activation_98 (Activation)      (None, 71, 71, 192)  0
batch_normalization_98[0][0]
__________________________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)  (None, 35, 35, 192)  0
```

```
activation_98[0][0]
__________________________________________________________________________________________________
conv2d_102 (Conv2D)             (None, 35, 35, 64)   12288
max_pooling2d_5[0][0]
__________________________________________________________________________________________________
batch_normalization_102 (BatchN (None, 35, 35, 64)   192
conv2d_102[0][0]
__________________________________________________________________________________________________
activation_102 (Activation)     (None, 35, 35, 64)   0
batch_normalization_102[0][0]
__________________________________________________________________________________________________
conv2d_100 (Conv2D)             (None, 35, 35, 48)   9216
max_pooling2d_5[0][0]
__________________________________________________________________________________________________
conv2d_103 (Conv2D)             (None, 35, 35, 96)   55296
activation_102[0][0]
__________________________________________________________________________________________________
batch_normalization_100 (BatchN (None, 35, 35, 48)   144
conv2d_100[0][0]
__________________________________________________________________________________________________
batch_normalization_103 (BatchN (None, 35, 35, 96)   288
conv2d_103[0][0]
__________________________________________________________________________________________________
activation_100 (Activation)     (None, 35, 35, 48)   0
batch_normalization_100[0][0]
__________________________________________________________________________________________________
activation_103 (Activation)     (None, 35, 35, 96)   0
batch_normalization_103[0][0]
__________________________________________________________________________________________________
average_pooling2d_9 (AveragePoo (None, 35, 35, 192)  0
max_pooling2d_5[0][0]
__________________________________________________________________________________________________
conv2d_99 (Conv2D)              (None, 35, 35, 64)   12288
max_pooling2d_5[0][0]
__________________________________________________________________________________________________
conv2d_101 (Conv2D)             (None, 35, 35, 64)   76800
```

```
activation_100[0][0]
________________________________________________________________________________
__________________
conv2d_104 (Conv2D)             (None, 35, 35, 96)   82944
activation_103[0][0]
________________________________________________________________________________
__________________
conv2d_105 (Conv2D)             (None, 35, 35, 32)   6144
average_pooling2d_9[0][0]
________________________________________________________________________________
__________________
batch_normalization_99 (BatchNo (None, 35, 35, 64)   192         conv2d_99[0][0]
________________________________________________________________________________
__________________
batch_normalization_101 (BatchN (None, 35, 35, 64)   192
conv2d_101[0][0]
________________________________________________________________________________
__________________
batch_normalization_104 (BatchN (None, 35, 35, 96)   288
conv2d_104[0][0]
________________________________________________________________________________
__________________
batch_normalization_105 (BatchN (None, 35, 35, 32)   96
conv2d_105[0][0]
________________________________________________________________________________
__________________
activation_99 (Activation)      (None, 35, 35, 64)   0
batch_normalization_99[0][0]
________________________________________________________________________________
__________________
activation_101 (Activation)     (None, 35, 35, 64)   0
batch_normalization_101[0][0]
________________________________________________________________________________
__________________
activation_104 (Activation)     (None, 35, 35, 96)   0
batch_normalization_104[0][0]
________________________________________________________________________________
__________________
activation_105 (Activation)     (None, 35, 35, 32)   0
batch_normalization_105[0][0]
________________________________________________________________________________
__________________
mixed0 (Concatenate)            (None, 35, 35, 256)  0
activation_99[0][0]
activation_101[0][0]
activation_104[0][0]
activation_105[0][0]
________________________________________________________________________________
```

```
__________________
conv2d_109 (Conv2D)             (None, 35, 35, 64)   16384       mixed0[0][0]

________________________________________________________________________________
__________________
batch_normalization_109 (BatchN (None, 35, 35, 64)   192
conv2d_109[0][0]

________________________________________________________________________________
__________________
activation_109 (Activation)     (None, 35, 35, 64)   0
batch_normalization_109[0][0]

________________________________________________________________________________
__________________
conv2d_107 (Conv2D)             (None, 35, 35, 48)   12288       mixed0[0][0]

________________________________________________________________________________
__________________
conv2d_110 (Conv2D)             (None, 35, 35, 96)   55296
activation_109[0][0]

________________________________________________________________________________
__________________
batch_normalization_107 (BatchN (None, 35, 35, 48)   144
conv2d_107[0][0]

________________________________________________________________________________
__________________
batch_normalization_110 (BatchN (None, 35, 35, 96)   288
conv2d_110[0][0]

________________________________________________________________________________
__________________
activation_107 (Activation)     (None, 35, 35, 48)   0
batch_normalization_107[0][0]

________________________________________________________________________________
__________________
activation_110 (Activation)     (None, 35, 35, 96)   0
batch_normalization_110[0][0]

________________________________________________________________________________
__________________
average_pooling2d_10 (AveragePo (None, 35, 35, 256)  0           mixed0[0][0]

________________________________________________________________________________
__________________
conv2d_106 (Conv2D)             (None, 35, 35, 64)   16384       mixed0[0][0]

________________________________________________________________________________
__________________
conv2d_108 (Conv2D)             (None, 35, 35, 64)   76800
activation_107[0][0]

________________________________________________________________________________
__________________
conv2d_111 (Conv2D)             (None, 35, 35, 96)   82944
activation_110[0][0]

________________________________________________________________________________
```

```
__________________
conv2d_112 (Conv2D)             (None, 35, 35, 64)   16384
average_pooling2d_10[0][0]
________________________________________________________________________________
__________________
batch_normalization_106 (BatchN (None, 35, 35, 64)   192
conv2d_106[0][0]
________________________________________________________________________________
__________________
batch_normalization_108 (BatchN (None, 35, 35, 64)   192
conv2d_108[0][0]
________________________________________________________________________________
__________________
batch_normalization_111 (BatchN (None, 35, 35, 96)   288
conv2d_111[0][0]
________________________________________________________________________________
__________________
batch_normalization_112 (BatchN (None, 35, 35, 64)   192
conv2d_112[0][0]
________________________________________________________________________________
__________________
activation_106 (Activation)     (None, 35, 35, 64)   0
batch_normalization_106[0][0]
________________________________________________________________________________
__________________
activation_108 (Activation)     (None, 35, 35, 64)   0
batch_normalization_108[0][0]
________________________________________________________________________________
__________________
activation_111 (Activation)     (None, 35, 35, 96)   0
batch_normalization_111[0][0]
________________________________________________________________________________
__________________
activation_112 (Activation)     (None, 35, 35, 64)   0
batch_normalization_112[0][0]
________________________________________________________________________________
__________________
mixed1 (Concatenate)            (None, 35, 35, 288)  0
activation_106[0][0]
activation_108[0][0]
activation_111[0][0]
activation_112[0][0]
________________________________________________________________________________
__________________
conv2d_116 (Conv2D)             (None, 35, 35, 64)   18432       mixed1[0][0]
________________________________________________________________________________
__________________
batch_normalization_116 (BatchN (None, 35, 35, 64)   192
```

```
conv2d_116[0][0]
__________________________________________________________________________________________________
activation_116 (Activation)     (None, 35, 35, 64)   0
batch_normalization_116[0][0]
__________________________________________________________________________________________________
conv2d_114 (Conv2D)             (None, 35, 35, 48)   13824       mixed1[0][0]
__________________________________________________________________________________________________
conv2d_117 (Conv2D)             (None, 35, 35, 96)   55296
activation_116[0][0]
__________________________________________________________________________________________________
batch_normalization_114 (BatchN (None, 35, 35, 48)   144
conv2d_114[0][0]
__________________________________________________________________________________________________
batch_normalization_117 (BatchN (None, 35, 35, 96)   288
conv2d_117[0][0]
__________________________________________________________________________________________________
activation_114 (Activation)     (None, 35, 35, 48)   0
batch_normalization_114[0][0]
__________________________________________________________________________________________________
activation_117 (Activation)     (None, 35, 35, 96)   0
batch_normalization_117[0][0]
__________________________________________________________________________________________________
average_pooling2d_11 (AveragePo (None, 35, 35, 288)  0           mixed1[0][0]
__________________________________________________________________________________________________
conv2d_113 (Conv2D)             (None, 35, 35, 64)   18432       mixed1[0][0]
__________________________________________________________________________________________________
conv2d_115 (Conv2D)             (None, 35, 35, 64)   76800
activation_114[0][0]
__________________________________________________________________________________________________
conv2d_118 (Conv2D)             (None, 35, 35, 96)   82944
activation_117[0][0]
__________________________________________________________________________________________________
conv2d_119 (Conv2D)             (None, 35, 35, 64)   18432
average_pooling2d_11[0][0]
__________________________________________________________________________________________________
```

```
batch_normalization_113 (BatchN (None, 35, 35, 64)   192
conv2d_113[0][0]
__________________________________________________________________________________________________
batch_normalization_115 (BatchN (None, 35, 35, 64)   192
conv2d_115[0][0]
__________________________________________________________________________________________________
batch_normalization_118 (BatchN (None, 35, 35, 96)   288
conv2d_118[0][0]
__________________________________________________________________________________________________
batch_normalization_119 (BatchN (None, 35, 35, 64)   192
conv2d_119[0][0]
__________________________________________________________________________________________________
activation_113 (Activation)     (None, 35, 35, 64)   0
batch_normalization_113[0][0]
__________________________________________________________________________________________________
activation_115 (Activation)     (None, 35, 35, 64)   0
batch_normalization_115[0][0]
__________________________________________________________________________________________________
activation_118 (Activation)     (None, 35, 35, 96)   0
batch_normalization_118[0][0]
__________________________________________________________________________________________________
activation_119 (Activation)     (None, 35, 35, 64)   0
batch_normalization_119[0][0]
__________________________________________________________________________________________________
mixed2 (Concatenate)            (None, 35, 35, 288)  0
activation_113[0][0]
activation_115[0][0]
activation_118[0][0]
activation_119[0][0]
__________________________________________________________________________________________________
conv2d_121 (Conv2D)             (None, 35, 35, 64)   18432       mixed2[0][0]
__________________________________________________________________________________________________
batch_normalization_121 (BatchN (None, 35, 35, 64)   192
conv2d_121[0][0]
__________________________________________________________________________________________________
activation_121 (Activation)     (None, 35, 35, 64)   0
batch_normalization_121[0][0]
```

```
__________________________________________________________________________________
__________________
conv2d_122 (Conv2D)             (None, 35, 35, 96)   55296
activation_121[0][0]
__________________________________________________________________________________
__________________
batch_normalization_122 (BatchN (None, 35, 35, 96)   288
conv2d_122[0][0]
__________________________________________________________________________________
__________________
activation_122 (Activation)     (None, 35, 35, 96)   0
batch_normalization_122[0][0]
__________________________________________________________________________________
__________________
conv2d_120 (Conv2D)             (None, 17, 17, 384)  995328      mixed2[0][0]
__________________________________________________________________________________
__________________
conv2d_123 (Conv2D)             (None, 17, 17, 96)   82944
activation_122[0][0]
__________________________________________________________________________________
__________________
batch_normalization_120 (BatchN (None, 17, 17, 384)  1152
conv2d_120[0][0]
__________________________________________________________________________________
__________________
batch_normalization_123 (BatchN (None, 17, 17, 96)   288
conv2d_123[0][0]
__________________________________________________________________________________
__________________
activation_120 (Activation)     (None, 17, 17, 384)  0
batch_normalization_120[0][0]
__________________________________________________________________________________
__________________
activation_123 (Activation)     (None, 17, 17, 96)   0
batch_normalization_123[0][0]
__________________________________________________________________________________
__________________
max_pooling2d_6 (MaxPooling2D)  (None, 17, 17, 288)  0           mixed2[0][0]
__________________________________________________________________________________
__________________
mixed3 (Concatenate)            (None, 17, 17, 768)  0
activation_120[0][0]
activation_123[0][0]
max_pooling2d_6[0][0]
__________________________________________________________________________________
__________________
conv2d_128 (Conv2D)             (None, 17, 17, 128)  98304       mixed3[0][0]
__________________________________________________________________________________
```

```
__________________
batch_normalization_128 (BatchN (None, 17, 17, 128)  384
conv2d_128[0][0]
________________________________________________________________________________
__________________
activation_128 (Activation)     (None, 17, 17, 128)  0
batch_normalization_128[0][0]
________________________________________________________________________________
__________________
conv2d_129 (Conv2D)             (None, 17, 17, 128)  114688
activation_128[0][0]
________________________________________________________________________________
__________________
batch_normalization_129 (BatchN (None, 17, 17, 128)  384
conv2d_129[0][0]
________________________________________________________________________________
__________________
activation_129 (Activation)     (None, 17, 17, 128)  0
batch_normalization_129[0][0]
________________________________________________________________________________
__________________
conv2d_125 (Conv2D)             (None, 17, 17, 128)  98304       mixed3[0][0]
________________________________________________________________________________
__________________
conv2d_130 (Conv2D)             (None, 17, 17, 128)  114688
activation_129[0][0]
________________________________________________________________________________
__________________
batch_normalization_125 (BatchN (None, 17, 17, 128)  384
conv2d_125[0][0]
________________________________________________________________________________
__________________
batch_normalization_130 (BatchN (None, 17, 17, 128)  384
conv2d_130[0][0]
________________________________________________________________________________
__________________
activation_125 (Activation)     (None, 17, 17, 128)  0
batch_normalization_125[0][0]
________________________________________________________________________________
__________________
activation_130 (Activation)     (None, 17, 17, 128)  0
batch_normalization_130[0][0]
________________________________________________________________________________
__________________
conv2d_126 (Conv2D)             (None, 17, 17, 128)  114688
activation_125[0][0]
________________________________________________________________________________
__________________
```

```
conv2d_131 (Conv2D)             (None, 17, 17, 128)  114688
activation_130[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_126 (BatchN (None, 17, 17, 128)  384
conv2d_126[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_131 (BatchN (None, 17, 17, 128)  384
conv2d_131[0][0]
--------------------------------------------------------------------------------
------------------
activation_126 (Activation)     (None, 17, 17, 128)  0
batch_normalization_126[0][0]
--------------------------------------------------------------------------------
------------------
activation_131 (Activation)     (None, 17, 17, 128)  0
batch_normalization_131[0][0]
--------------------------------------------------------------------------------
------------------
average_pooling2d_12 (AveragePo (None, 17, 17, 768)  0           mixed3[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_124 (Conv2D)             (None, 17, 17, 192)  147456      mixed3[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_127 (Conv2D)             (None, 17, 17, 192)  172032
activation_126[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_132 (Conv2D)             (None, 17, 17, 192)  172032
activation_131[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_133 (Conv2D)             (None, 17, 17, 192)  147456
average_pooling2d_12[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_124 (BatchN (None, 17, 17, 192)  576
conv2d_124[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_127 (BatchN (None, 17, 17, 192)  576
conv2d_127[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_132 (BatchN (None, 17, 17, 192)  576
conv2d_132[0][0]
```

```
__________________________________________________________________________________
__________________
batch_normalization_133 (BatchN (None, 17, 17, 192)  576
conv2d_133[0][0]
__________________________________________________________________________________
__________________
activation_124 (Activation)     (None, 17, 17, 192)  0
batch_normalization_124[0][0]
__________________________________________________________________________________
__________________
activation_127 (Activation)     (None, 17, 17, 192)  0
batch_normalization_127[0][0]
__________________________________________________________________________________
__________________
activation_132 (Activation)     (None, 17, 17, 192)  0
batch_normalization_132[0][0]
__________________________________________________________________________________
__________________
activation_133 (Activation)     (None, 17, 17, 192)  0
batch_normalization_133[0][0]
__________________________________________________________________________________
__________________
mixed4 (Concatenate)            (None, 17, 17, 768)  0
activation_124[0][0]
activation_127[0][0]
activation_132[0][0]
activation_133[0][0]
__________________________________________________________________________________
__________________
conv2d_138 (Conv2D)             (None, 17, 17, 160)  122880      mixed4[0][0]
__________________________________________________________________________________
__________________
batch_normalization_138 (BatchN (None, 17, 17, 160)  480
conv2d_138[0][0]
__________________________________________________________________________________
__________________
activation_138 (Activation)     (None, 17, 17, 160)  0
batch_normalization_138[0][0]
__________________________________________________________________________________
__________________
conv2d_139 (Conv2D)             (None, 17, 17, 160)  179200
activation_138[0][0]
__________________________________________________________________________________
__________________
batch_normalization_139 (BatchN (None, 17, 17, 160)  480
conv2d_139[0][0]
__________________________________________________________________________________
__________________
```

```
activation_139 (Activation)     (None, 17, 17, 160)  0
batch_normalization_139[0][0]

__________________________________________________________________________________________________
conv2d_135 (Conv2D)             (None, 17, 17, 160)  122880      mixed4[0][0]

__________________________________________________________________________________________________
conv2d_140 (Conv2D)             (None, 17, 17, 160)  179200
activation_139[0][0]

__________________________________________________________________________________________________
batch_normalization_135 (BatchN (None, 17, 17, 160)  480
conv2d_135[0][0]

__________________________________________________________________________________________________
batch_normalization_140 (BatchN (None, 17, 17, 160)  480
conv2d_140[0][0]

__________________________________________________________________________________________________
activation_135 (Activation)     (None, 17, 17, 160)  0
batch_normalization_135[0][0]

__________________________________________________________________________________________________
activation_140 (Activation)     (None, 17, 17, 160)  0
batch_normalization_140[0][0]

__________________________________________________________________________________________________
conv2d_136 (Conv2D)             (None, 17, 17, 160)  179200
activation_135[0][0]

__________________________________________________________________________________________________
conv2d_141 (Conv2D)             (None, 17, 17, 160)  179200
activation_140[0][0]

__________________________________________________________________________________________________
batch_normalization_136 (BatchN (None, 17, 17, 160)  480
conv2d_136[0][0]

__________________________________________________________________________________________________
batch_normalization_141 (BatchN (None, 17, 17, 160)  480
conv2d_141[0][0]

__________________________________________________________________________________________________
activation_136 (Activation)     (None, 17, 17, 160)  0
batch_normalization_136[0][0]

__________________________________________________________________________________________________
activation_141 (Activation)     (None, 17, 17, 160)  0
```

```
batch_normalization_141[0][0]
________________________________________________________________________________
__________________
average_pooling2d_13 (AveragePo (None, 17, 17, 768)  0           mixed4[0][0]
________________________________________________________________________________
__________________
conv2d_134 (Conv2D)             (None, 17, 17, 192)  147456      mixed4[0][0]
________________________________________________________________________________
__________________
conv2d_137 (Conv2D)             (None, 17, 17, 192)  215040
activation_136[0][0]
________________________________________________________________________________
__________________
conv2d_142 (Conv2D)             (None, 17, 17, 192)  215040
activation_141[0][0]
________________________________________________________________________________
__________________
conv2d_143 (Conv2D)             (None, 17, 17, 192)  147456
average_pooling2d_13[0][0]
________________________________________________________________________________
__________________
batch_normalization_134 (BatchN (None, 17, 17, 192)  576
conv2d_134[0][0]
________________________________________________________________________________
__________________
batch_normalization_137 (BatchN (None, 17, 17, 192)  576
conv2d_137[0][0]
________________________________________________________________________________
__________________
batch_normalization_142 (BatchN (None, 17, 17, 192)  576
conv2d_142[0][0]
________________________________________________________________________________
__________________
batch_normalization_143 (BatchN (None, 17, 17, 192)  576
conv2d_143[0][0]
________________________________________________________________________________
__________________
activation_134 (Activation)     (None, 17, 17, 192)  0
batch_normalization_134[0][0]
________________________________________________________________________________
__________________
activation_137 (Activation)     (None, 17, 17, 192)  0
batch_normalization_137[0][0]
________________________________________________________________________________
__________________
activation_142 (Activation)     (None, 17, 17, 192)  0
batch_normalization_142[0][0]
________________________________________________________________________________
```

```
__________________
activation_143 (Activation)     (None, 17, 17, 192)  0
batch_normalization_143[0][0]
________________________________________________________________________________
__________________
mixed5 (Concatenate)            (None, 17, 17, 768)  0
activation_134[0][0]
activation_137[0][0]
activation_142[0][0]
activation_143[0][0]
________________________________________________________________________________
__________________
conv2d_148 (Conv2D)             (None, 17, 17, 160)  122880      mixed5[0][0]
________________________________________________________________________________
__________________
batch_normalization_148 (BatchN (None, 17, 17, 160)  480
conv2d_148[0][0]
________________________________________________________________________________
__________________
activation_148 (Activation)     (None, 17, 17, 160)  0
batch_normalization_148[0][0]
________________________________________________________________________________
__________________
conv2d_149 (Conv2D)             (None, 17, 17, 160)  179200
activation_148[0][0]
________________________________________________________________________________
__________________
batch_normalization_149 (BatchN (None, 17, 17, 160)  480
conv2d_149[0][0]
________________________________________________________________________________
__________________
activation_149 (Activation)     (None, 17, 17, 160)  0
batch_normalization_149[0][0]
________________________________________________________________________________
__________________
conv2d_145 (Conv2D)             (None, 17, 17, 160)  122880      mixed5[0][0]
________________________________________________________________________________
__________________
conv2d_150 (Conv2D)             (None, 17, 17, 160)  179200
activation_149[0][0]
________________________________________________________________________________
__________________
batch_normalization_145 (BatchN (None, 17, 17, 160)  480
conv2d_145[0][0]
________________________________________________________________________________
__________________
batch_normalization_150 (BatchN (None, 17, 17, 160)  480
conv2d_150[0][0]
```

```
--------------------------------------------------------------------------------
------------------
activation_145 (Activation)     (None, 17, 17, 160)  0
batch_normalization_145[0][0]
--------------------------------------------------------------------------------
------------------
activation_150 (Activation)     (None, 17, 17, 160)  0
batch_normalization_150[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_146 (Conv2D)             (None, 17, 17, 160)  179200
activation_145[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_151 (Conv2D)             (None, 17, 17, 160)  179200
activation_150[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_146 (BatchN (None, 17, 17, 160)  480
conv2d_146[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_151 (BatchN (None, 17, 17, 160)  480
conv2d_151[0][0]
--------------------------------------------------------------------------------
------------------
activation_146 (Activation)     (None, 17, 17, 160)  0
batch_normalization_146[0][0]
--------------------------------------------------------------------------------
------------------
activation_151 (Activation)     (None, 17, 17, 160)  0
batch_normalization_151[0][0]
--------------------------------------------------------------------------------
------------------
average_pooling2d_14 (AveragePo (None, 17, 17, 768)  0           mixed5[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_144 (Conv2D)             (None, 17, 17, 192)  147456      mixed5[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_147 (Conv2D)             (None, 17, 17, 192)  215040
activation_146[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_152 (Conv2D)             (None, 17, 17, 192)  215040
activation_151[0][0]
--------------------------------------------------------------------------------
------------------
```

```
conv2d_153 (Conv2D)            (None, 17, 17, 192)  147456
average_pooling2d_14[0][0]
__________________________________________________________________________________________________
batch_normalization_144 (BatchN (None, 17, 17, 192)  576
conv2d_144[0][0]
__________________________________________________________________________________________________
batch_normalization_147 (BatchN (None, 17, 17, 192)  576
conv2d_147[0][0]
__________________________________________________________________________________________________
batch_normalization_152 (BatchN (None, 17, 17, 192)  576
conv2d_152[0][0]
__________________________________________________________________________________________________
batch_normalization_153 (BatchN (None, 17, 17, 192)  576
conv2d_153[0][0]
__________________________________________________________________________________________________
activation_144 (Activation)    (None, 17, 17, 192)  0
batch_normalization_144[0][0]
__________________________________________________________________________________________________
activation_147 (Activation)    (None, 17, 17, 192)  0
batch_normalization_147[0][0]
__________________________________________________________________________________________________
activation_152 (Activation)    (None, 17, 17, 192)  0
batch_normalization_152[0][0]
__________________________________________________________________________________________________
activation_153 (Activation)    (None, 17, 17, 192)  0
batch_normalization_153[0][0]
__________________________________________________________________________________________________
mixed6 (Concatenate)           (None, 17, 17, 768)  0
activation_144[0][0]
activation_147[0][0]
activation_152[0][0]
activation_153[0][0]
__________________________________________________________________________________________________
conv2d_158 (Conv2D)            (None, 17, 17, 192)  147456      mixed6[0][0]
__________________________________________________________________________________________________
batch_normalization_158 (BatchN (None, 17, 17, 192)  576
conv2d_158[0][0]
```

```
________________________________________________________________________________
__________________
activation_158 (Activation)     (None, 17, 17, 192)  0
batch_normalization_158[0][0]
________________________________________________________________________________
__________________
conv2d_159 (Conv2D)             (None, 17, 17, 192)  258048
activation_158[0][0]
________________________________________________________________________________
__________________
batch_normalization_159 (BatchN (None, 17, 17, 192)  576
conv2d_159[0][0]
________________________________________________________________________________
__________________
activation_159 (Activation)     (None, 17, 17, 192)  0
batch_normalization_159[0][0]
________________________________________________________________________________
__________________
conv2d_155 (Conv2D)             (None, 17, 17, 192)  147456      mixed6[0][0]
________________________________________________________________________________
__________________
conv2d_160 (Conv2D)             (None, 17, 17, 192)  258048
activation_159[0][0]
________________________________________________________________________________
__________________
batch_normalization_155 (BatchN (None, 17, 17, 192)  576
conv2d_155[0][0]
________________________________________________________________________________
__________________
batch_normalization_160 (BatchN (None, 17, 17, 192)  576
conv2d_160[0][0]
________________________________________________________________________________
__________________
activation_155 (Activation)     (None, 17, 17, 192)  0
batch_normalization_155[0][0]
________________________________________________________________________________
__________________
activation_160 (Activation)     (None, 17, 17, 192)  0
batch_normalization_160[0][0]
________________________________________________________________________________
__________________
conv2d_156 (Conv2D)             (None, 17, 17, 192)  258048
activation_155[0][0]
________________________________________________________________________________
__________________
conv2d_161 (Conv2D)             (None, 17, 17, 192)  258048
activation_160[0][0]
________________________________________________________________________________
```

```
__________________
batch_normalization_156 (BatchN (None, 17, 17, 192)  576
conv2d_156[0][0]
________________________________________________________________________________
__________________
batch_normalization_161 (BatchN (None, 17, 17, 192)  576
conv2d_161[0][0]
________________________________________________________________________________
__________________
activation_156 (Activation)     (None, 17, 17, 192)  0
batch_normalization_156[0][0]
________________________________________________________________________________
__________________
activation_161 (Activation)     (None, 17, 17, 192)  0
batch_normalization_161[0][0]
________________________________________________________________________________
__________________
average_pooling2d_15 (AveragePo (None, 17, 17, 768)  0           mixed6[0][0]
________________________________________________________________________________
__________________
conv2d_154 (Conv2D)             (None, 17, 17, 192)  147456      mixed6[0][0]
________________________________________________________________________________
__________________
conv2d_157 (Conv2D)             (None, 17, 17, 192)  258048
activation_156[0][0]
________________________________________________________________________________
__________________
conv2d_162 (Conv2D)             (None, 17, 17, 192)  258048
activation_161[0][0]
________________________________________________________________________________
__________________
conv2d_163 (Conv2D)             (None, 17, 17, 192)  147456
average_pooling2d_15[0][0]
________________________________________________________________________________
__________________
batch_normalization_154 (BatchN (None, 17, 17, 192)  576
conv2d_154[0][0]
________________________________________________________________________________
__________________
batch_normalization_157 (BatchN (None, 17, 17, 192)  576
conv2d_157[0][0]
________________________________________________________________________________
__________________
batch_normalization_162 (BatchN (None, 17, 17, 192)  576
conv2d_162[0][0]
________________________________________________________________________________
__________________
batch_normalization_163 (BatchN (None, 17, 17, 192)  576
```

```
conv2d_163[0][0]
__________________________________________________________________________________
__________________
activation_154 (Activation)     (None, 17, 17, 192)  0
batch_normalization_154[0][0]
__________________________________________________________________________________
__________________
activation_157 (Activation)     (None, 17, 17, 192)  0
batch_normalization_157[0][0]
__________________________________________________________________________________
__________________
activation_162 (Activation)     (None, 17, 17, 192)  0
batch_normalization_162[0][0]
__________________________________________________________________________________
__________________
activation_163 (Activation)     (None, 17, 17, 192)  0
batch_normalization_163[0][0]
__________________________________________________________________________________
__________________
mixed7 (Concatenate)            (None, 17, 17, 768)  0
activation_154[0][0]
activation_157[0][0]
activation_162[0][0]
activation_163[0][0]
__________________________________________________________________________________
__________________
conv2d_166 (Conv2D)             (None, 17, 17, 192)  147456      mixed7[0][0]
__________________________________________________________________________________
__________________
batch_normalization_166 (BatchN (None, 17, 17, 192)  576
conv2d_166[0][0]
__________________________________________________________________________________
__________________
activation_166 (Activation)     (None, 17, 17, 192)  0
batch_normalization_166[0][0]
__________________________________________________________________________________
__________________
conv2d_167 (Conv2D)             (None, 17, 17, 192)  258048
activation_166[0][0]
__________________________________________________________________________________
__________________
batch_normalization_167 (BatchN (None, 17, 17, 192)  576
conv2d_167[0][0]
__________________________________________________________________________________
__________________
activation_167 (Activation)     (None, 17, 17, 192)  0
batch_normalization_167[0][0]
__________________________________________________________________________________
```

```
------------------
conv2d_164 (Conv2D)             (None, 17, 17, 192)  147456      mixed7[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_168 (Conv2D)             (None, 17, 17, 192)  258048
activation_167[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_164 (BatchN (None, 17, 17, 192)  576
conv2d_164[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_168 (BatchN (None, 17, 17, 192)  576
conv2d_168[0][0]
--------------------------------------------------------------------------------
------------------
activation_164 (Activation)     (None, 17, 17, 192)  0
batch_normalization_164[0][0]
--------------------------------------------------------------------------------
------------------
activation_168 (Activation)     (None, 17, 17, 192)  0
batch_normalization_168[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_165 (Conv2D)             (None, 8, 8, 320)    552960
activation_164[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_169 (Conv2D)             (None, 8, 8, 192)    331776
activation_168[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_165 (BatchN (None, 8, 8, 320)    960
conv2d_165[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_169 (BatchN (None, 8, 8, 192)    576
conv2d_169[0][0]
--------------------------------------------------------------------------------
------------------
activation_165 (Activation)     (None, 8, 8, 320)    0
batch_normalization_165[0][0]
--------------------------------------------------------------------------------
------------------
activation_169 (Activation)     (None, 8, 8, 192)    0
batch_normalization_169[0][0]
--------------------------------------------------------------------------------
------------------
```

```
max_pooling2d_7 (MaxPooling2D)  (None, 8, 8, 768)    0           mixed7[0][0]
________________________________________________________________________________
__________________
mixed8 (Concatenate)            (None, 8, 8, 1280)   0
activation_165[0][0]
activation_169[0][0]
max_pooling2d_7[0][0]
________________________________________________________________________________
__________________
conv2d_174 (Conv2D)             (None, 8, 8, 448)    573440      mixed8[0][0]
________________________________________________________________________________
__________________
batch_normalization_174 (BatchN (None, 8, 8, 448)    1344
conv2d_174[0][0]
________________________________________________________________________________
__________________
activation_174 (Activation)     (None, 8, 8, 448)    0
batch_normalization_174[0][0]
________________________________________________________________________________
__________________
conv2d_171 (Conv2D)             (None, 8, 8, 384)    491520      mixed8[0][0]
________________________________________________________________________________
__________________
conv2d_175 (Conv2D)             (None, 8, 8, 384)    1548288
activation_174[0][0]
________________________________________________________________________________
__________________
batch_normalization_171 (BatchN (None, 8, 8, 384)    1152
conv2d_171[0][0]
________________________________________________________________________________
__________________
batch_normalization_175 (BatchN (None, 8, 8, 384)    1152
conv2d_175[0][0]
________________________________________________________________________________
__________________
activation_171 (Activation)     (None, 8, 8, 384)    0
batch_normalization_171[0][0]
________________________________________________________________________________
__________________
activation_175 (Activation)     (None, 8, 8, 384)    0
batch_normalization_175[0][0]
________________________________________________________________________________
__________________
conv2d_172 (Conv2D)             (None, 8, 8, 384)    442368
activation_171[0][0]
________________________________________________________________________________
__________________
conv2d_173 (Conv2D)             (None, 8, 8, 384)    442368
```

```
activation_171[0][0]
__________________________________________________________________________________________________
conv2d_176 (Conv2D)             (None, 8, 8, 384)    442368
activation_175[0][0]
__________________________________________________________________________________________________
conv2d_177 (Conv2D)             (None, 8, 8, 384)    442368
activation_175[0][0]
__________________________________________________________________________________________________
average_pooling2d_16 (AveragePo (None, 8, 8, 1280)   0           mixed8[0][0]
__________________________________________________________________________________________________
conv2d_170 (Conv2D)             (None, 8, 8, 320)    409600      mixed8[0][0]
__________________________________________________________________________________________________
batch_normalization_172 (BatchN (None, 8, 8, 384)    1152
conv2d_172[0][0]
__________________________________________________________________________________________________
batch_normalization_173 (BatchN (None, 8, 8, 384)    1152
conv2d_173[0][0]
__________________________________________________________________________________________________
batch_normalization_176 (BatchN (None, 8, 8, 384)    1152
conv2d_176[0][0]
__________________________________________________________________________________________________
batch_normalization_177 (BatchN (None, 8, 8, 384)    1152
conv2d_177[0][0]
__________________________________________________________________________________________________
conv2d_178 (Conv2D)             (None, 8, 8, 192)    245760
average_pooling2d_16[0][0]
__________________________________________________________________________________________________
batch_normalization_170 (BatchN (None, 8, 8, 320)    960
conv2d_170[0][0]
__________________________________________________________________________________________________
activation_172 (Activation)     (None, 8, 8, 384)    0
batch_normalization_172[0][0]
__________________________________________________________________________________________________
activation_173 (Activation)     (None, 8, 8, 384)    0
batch_normalization_173[0][0]
__________________________________________________________________________________________________
```

```
------------------
activation_176 (Activation)     (None, 8, 8, 384)    0
batch_normalization_176[0][0]

--------------------------------------------------------------------------------
------------------
activation_177 (Activation)     (None, 8, 8, 384)    0
batch_normalization_177[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_178 (BatchN (None, 8, 8, 192)    576
conv2d_178[0][0]

--------------------------------------------------------------------------------
------------------
activation_170 (Activation)     (None, 8, 8, 320)    0
batch_normalization_170[0][0]

--------------------------------------------------------------------------------
------------------
mixed9_0 (Concatenate)          (None, 8, 8, 768)    0
activation_172[0][0]
activation_173[0][0]

--------------------------------------------------------------------------------
------------------
concatenate_2 (Concatenate)     (None, 8, 8, 768)    0
activation_176[0][0]
activation_177[0][0]

--------------------------------------------------------------------------------
------------------
activation_178 (Activation)     (None, 8, 8, 192)    0
batch_normalization_178[0][0]

--------------------------------------------------------------------------------
------------------
mixed9 (Concatenate)            (None, 8, 8, 2048)   0
activation_170[0][0]
                                                                mixed9_0[0][0]

concatenate_2[0][0]
activation_178[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_183 (Conv2D)             (None, 8, 8, 448)    917504      mixed9[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_183 (BatchN (None, 8, 8, 448)    1344
conv2d_183[0][0]

--------------------------------------------------------------------------------
------------------
activation_183 (Activation)     (None, 8, 8, 448)    0
batch_normalization_183[0][0]

--------------------------------------------------------------------------------
```

```
__________________
conv2d_180 (Conv2D)             (None, 8, 8, 384)    786432      mixed9[0][0]
________________________________________________________________________________
__________________
conv2d_184 (Conv2D)             (None, 8, 8, 384)    1548288
activation_183[0][0]
________________________________________________________________________________
__________________
batch_normalization_180 (BatchN (None, 8, 8, 384)    1152
conv2d_180[0][0]
________________________________________________________________________________
__________________
batch_normalization_184 (BatchN (None, 8, 8, 384)    1152
conv2d_184[0][0]
________________________________________________________________________________
__________________
activation_180 (Activation)     (None, 8, 8, 384)    0
batch_normalization_180[0][0]
________________________________________________________________________________
__________________
activation_184 (Activation)     (None, 8, 8, 384)    0
batch_normalization_184[0][0]
________________________________________________________________________________
__________________
conv2d_181 (Conv2D)             (None, 8, 8, 384)    442368
activation_180[0][0]
________________________________________________________________________________
__________________
conv2d_182 (Conv2D)             (None, 8, 8, 384)    442368
activation_180[0][0]
________________________________________________________________________________
__________________
conv2d_185 (Conv2D)             (None, 8, 8, 384)    442368
activation_184[0][0]
________________________________________________________________________________
__________________
conv2d_186 (Conv2D)             (None, 8, 8, 384)    442368
activation_184[0][0]
________________________________________________________________________________
__________________
average_pooling2d_17 (AveragePo (None, 8, 8, 2048)   0           mixed9[0][0]
________________________________________________________________________________
__________________
conv2d_179 (Conv2D)             (None, 8, 8, 320)    655360      mixed9[0][0]
________________________________________________________________________________
__________________
batch_normalization_181 (BatchN (None, 8, 8, 384)    1152
conv2d_181[0][0]
```

```
__________________________________________________________________________________________________
batch_normalization_182 (BatchN (None, 8, 8, 384)    1152
conv2d_182[0][0]
__________________________________________________________________________________________________
batch_normalization_185 (BatchN (None, 8, 8, 384)    1152
conv2d_185[0][0]
__________________________________________________________________________________________________
batch_normalization_186 (BatchN (None, 8, 8, 384)    1152
conv2d_186[0][0]
__________________________________________________________________________________________________
conv2d_187 (Conv2D)             (None, 8, 8, 192)    393216
average_pooling2d_17[0][0]
__________________________________________________________________________________________________
batch_normalization_179 (BatchN (None, 8, 8, 320)    960
conv2d_179[0][0]
__________________________________________________________________________________________________
activation_181 (Activation)     (None, 8, 8, 384)    0
batch_normalization_181[0][0]
__________________________________________________________________________________________________
activation_182 (Activation)     (None, 8, 8, 384)    0
batch_normalization_182[0][0]
__________________________________________________________________________________________________
activation_185 (Activation)     (None, 8, 8, 384)    0
batch_normalization_185[0][0]
__________________________________________________________________________________________________
activation_186 (Activation)     (None, 8, 8, 384)    0
batch_normalization_186[0][0]
__________________________________________________________________________________________________
batch_normalization_187 (BatchN (None, 8, 8, 192)    576
conv2d_187[0][0]
__________________________________________________________________________________________________
activation_179 (Activation)     (None, 8, 8, 320)    0
batch_normalization_179[0][0]
__________________________________________________________________________________________________
mixed9_1 (Concatenate)          (None, 8, 8, 768)    0
activation_181[0][0]
```

```
activation_182[0][0]
__________________________________________________________________________________
__________________
concatenate_3 (Concatenate)     (None, 8, 8, 768)    0
activation_185[0][0]
activation_186[0][0]
__________________________________________________________________________________
__________________
activation_187 (Activation)     (None, 8, 8, 192)    0
batch_normalization_187[0][0]
__________________________________________________________________________________
__________________
mixed10 (Concatenate)           (None, 8, 8, 2048)   0
activation_179[0][0]
                                                                 mixed9_1[0][0]
concatenate_3[0][0]
activation_187[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.

[ ]:
```
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.

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

[ ]:
```
encoded_image
```

[ ]:
```
array([[0.63806564, 0.4887301 , 0.0552623 , ..., 0.64255726, 0.29595244,
        0.49004275]], 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

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).

[ ]:
```
def img_generator(img_list):
    #...
    for image_name in img_list:
      img = get_image(image_name)
      yield np.array([img])
```

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

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

```
6000/6000 [==============================] - 88s 15ms/step
```

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

[ ]:
```
array([0.26818538, 1.0321677 , 0.58516157, ..., 1.2316749 , 0.17969316,
       0.22405301], dtype=float32)
```

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

```
1000/1000 [==============================] - 250s 250ms/step
```

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

```
1000/1000 [==============================] - 242s 242ms/step
```

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

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

1.3 Part II Text (Caption) Data Preparation (14 pts)

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

1.3.1 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.

[4]:
```
def read_image_descriptions(filename):
    file = open(filename, "r")
    image_descriptions = defaultdict(list)
    # dict[filename] = [list of 5 captions]
    start = ['<START>']
    end = ['<END>']
    for sentence in file:
      # print(s)
      name = sentence.split()[0][:-2] #get file name
      tokenized = sentence.lower().split()[1:] #get lowercase tokens
      lst = start + tokenized + end
      if name not in image_descriptions:
        image_descriptions[name] = [lst]
      else:
        image_descriptions[name].append(lst)
      # print(image_descriptions)

    return image_descriptions
```

[5]:
```
descriptions = read_image_descriptions("gdrive/My Drive/"+my_data_dir+"/
↪Flickr8k.token.txt")
```

[5]:
```
print(descriptions)
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

[6]:
```
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', '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>']]
```

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>']]
```

1.3.2 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.

[6]:
```
train_data = []
for train_f in train_list:
  train_data.extend(flatten(descriptions[train_f]))
  # print(descriptions[train_f])
token_set = list(set(train_data))
# token_set
```

[7]:
```
id_to_word = {ind: token for ind, token in enumerate(sorted(set(token_set)))}
# id_to_word
```

[8]:
```
word_to_id = {token: ind for ind, token in id_to_word.items()}
# word_to_id
```

[9]:
```
word_to_id['dog'] # should print an integer
```

[9]:
```
1985
```

[10]:
```
id_to_word[1985] # should print a token
```

[10]:
```
'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.

1.4 Part III Basic Decoder Model (24 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).

[11]:
```
max(len(description) for image_id in train_list for description in␣
↪descriptions[image_id])
```

[11]: 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.

[45]:
```
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_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        [(None, 40)]              0
_________________________________________________________________
embedding_9 (Embedding)      (None, 40, 300)           2312100
_________________________________________________________________
bidirectional_9 (Bidirection (None, 1024)              3330048
_________________________________________________________________
dense_9 (Dense)              (None, 7707)              7899675
=================================================================
Total params: 13,541,823
Trainable params: 13,541,823
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.

1.4.1 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.

[46]:
```
def text_training_generator(batch_size=128):
  curr = []
  input_arr = []
  output_arr = []
  while True:
    for train_id in range(len(train_list)):
      for sentence in descriptions[train_list[train_id]]:
        sentence_int = [word_to_id[word] for word in sentence]
        for ind in range(len(sentence_int)-1):
          if len(input_arr) < batch_size:
            curr = sentence_int[:ind+1]
            curr.extend([0]*(MAX_LEN-ind-1))
            input_arr.append(curr)

            output_temp = [0]*vocab_size
            output_temp[sentence_int[ind+1]] = 1
            output_arr.append(output_temp)
          else:
            yield (np.array(input_arr),np.array(output_arr))
            input_arr = []
            output_arr = []
```

[47]:
```
g = text_training_generator(128)
# print(next(g))
for i in range(3):
  inp, out = next(g)
#   if inp.shape != (128, 40) or out.shape != (128, 7707):
  print(inp.shape, out.shape)
#     print(111)
print(next(g))
```

```
(128, 40) (128, 7707)
(128, 40) (128, 7707)
(128, 40) (128, 7707)
(array([[  59,   61, 3963, ...,    0,    0,    0],
       [  59,   61, 3963, ...,    0,    0,    0],
       [  59,   61, 3963, ...,    0,    0,    0],
       ...,
       [  59, 6861, 3848, ...,    0,    0,    0],
       [  59, 6861, 3848, ...,    0,    0,    0],
       [  59, 6861, 3848, ...,    0,    0,    0]]), array([[0, 0, 0, ..., 0, 0,
0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
```

```
...,
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0]]))
```

1.4.2 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

[48]:
```
batch_size = 128
generator = text_training_generator(batch_size)
steps = len(train_list) * MAX_LEN // batch_size
```

[49]:
```
model.fit_generator(generator, steps_per_epoch=steps, verbose=True, epochs=10)
```

```
/usr/local/lib/python3.6/dist-
packages/tensorflow/python/data/ops/dataset_ops.py:3350: UserWarning: Even
though the tf.config.experimental_run_functions_eagerly option is set, this
option does not apply to tf.data functions. tf.data functions are still traced
and executed as graphs.
  "Even though the tf.config.experimental_run_functions_eagerly "

Epoch 1/10
1875/1875 [==============================] - 296s 158ms/step - loss: 4.2501 -
accuracy: 0.2954
Epoch 2/10
1875/1875 [==============================] - 298s 159ms/step - loss: 3.6942 -
accuracy: 0.3565
Epoch 3/10
1875/1875 [==============================] - 299s 160ms/step - loss: 3.5223 -
accuracy: 0.3734
Epoch 4/10
1875/1875 [==============================] - 280s 149ms/step - loss: 3.4144 -
accuracy: 0.3836
Epoch 5/10
1875/1875 [==============================] - 272s 145ms/step - loss: 3.3682 -
accuracy: 0.3899
Epoch 6/10
1875/1875 [==============================] - 276s 147ms/step - loss: 3.2939 -
accuracy: 0.3962
Epoch 7/10
1875/1875 [==============================] - 281s 150ms/step - loss: 3.2815 -
accuracy: 0.3987
Epoch 8/10
1875/1875 [==============================] - 283s 151ms/step - loss: 3.2824 -
```

```
accuracy: 0.3996
Epoch 9/10
1875/1875 [==============================] - 284s 151ms/step - loss: 3.2498 -
accuracy: 0.4031
Epoch 10/10
1875/1875 [==============================] - 301s 161ms/step - loss: 3.2710 -
accuracy: 0.4047
```

[49]:
```
<tensorflow.python.keras.callbacks.History at 0x7f693c38c9b0>
```

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

1.4.3 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.

[50]:
```python
word_to_id['<END>']
id_to_word[0]
lst = np.nonzero(np.array([0,0,0,0,0,0,0,0,1,0,0]))
int(lst[0])
```

[50]:
```
8
```

[51]:
```python
def decoder():
    # ...
    sentence = ["<START>"]
    int_sentence = [word_to_id[sentence[0]]]
    int_sentence.extend([0]*(MAX_LEN-1))
    cnt = 0
    pred = "<START>"
    while cnt < MAX_LEN-1 and pred != "<END>":
      cnt += 1
      # new_w = model.predict([np.array([int_sentence])])
      pred = id_to_word[int(np.argmax(model.predict([np.
 ↪array([int_sentence])])))]
      # print(pred)
      sentence.append(pred)
      int_sentence[cnt] = word_to_id[pred]

    return sentence
```

[52]:
```python
print(decoder())
```

```
/usr/local/lib/python3.6/dist-
packages/tensorflow/python/data/ops/dataset_ops.py:3350: UserWarning: Even
though the tf.config.experimental_run_functions_eagerly option is set, this
option does not apply to tf.data functions. tf.data functions are still traced
and executed as graphs.
  "Even though the tf.config.experimental_run_functions_eagerly "
```

```
['<START>', 'a', 'man', 'in', 'a', 'red', 'and', 'white', 'shirt', 'is',
'playing', 'with', 'a', 'dog', '.', '<END>']
```

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 function to do this.

[53]:
```
rand = np.random.multinomial(20,[0.1,0.1,0.1,0.2,0.3,0.3],size=1)
rand
```

[53]:
```
array([[4, 2, 1, 1, 5, 7]])
```

[54]:
```
def sample_decoder():
    #...
    sentence = ["<START>"]
    int_sentence = [word_to_id[sentence[0]]]
    int_sentence.extend([0]*(MAX_LEN-1))
    cnt = 0
    pred = "<START>"
    while cnt < MAX_LEN-1 and pred != "<END>":
      cnt += 1
      new_w = model.predict([np.array([int_sentence])])
      w = np.array(new_w, dtype=np.float32)
      sum_1_w = w / (w.sum() + 0.01)
      pred = id_to_word[int(np.argmax(np.random.
 ↪multinomial(20,sum_1_w[0],size=1)))]
      # print(pred)
      sentence.append(pred)
      int_sentence[cnt] = word_to_id[pred]
    return sentence
```

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.

[55]:
```
for i in range(10):
    print(sample_decoder())
```

```
/usr/local/lib/python3.6/dist-
packages/tensorflow/python/data/ops/dataset_ops.py:3350: UserWarning: Even
though the tf.config.experimental_run_functions_eagerly option is set, this
option does not apply to tf.data functions. tf.data functions are still traced
and executed as graphs.
  "Even though the tf.config.experimental_run_functions_eagerly "

['<START>', 'a', 'man', 'sits', 'on', 'a', 'bench', 'and', 'carries', 'a',
'baby', 'in', 'a', 'and', 'blue', 'dress', '.', '<END>']
['<START>', 'a', 'man', 'in', 'a', 'green', 'hoodie', 'is', 'playing', 'with',
'a', 'dog', 'on', 'the', 'beach', '.', '<END>']
['<START>', 'a', 'man', 'in', 'a', 'red', 'shirt', 'is', 'riding', 'a', 'bike',
```

```
'on', 'a', 'beach', '.', '<END>']
['<START>', 'a', 'man', 'is', 'walking', 'on', 'a', 'sidewalk', '.', '<END>']
['<START>', 'a', 'man', 'on', 'a', 'bicycle', 'is', 'riding', 'a', 'bike', 'on',
'a', 'dirt', 'path', '.', '<END>']
['<START>', 'a', 'man', 'in', 'a', 'red', 'shirt', 'is', 'standing', 'on', 'a',
'sidewalk', '.', '<END>']
['<START>', 'a', 'young', 'boy', 'taking', 'a', 'back', 'of', 'his', 'back',
'while', 'the', 'boy', 'walks', 'in', 'the', 'background', '.', '<END>']
['<START>', 'a', 'man', 'is', 'doing', 'a', 'trick', 'on', 'a', 'bike', 'on',
'a', 'dirt', 'road', '.', '<END>']
['<START>', 'a', 'dog', 'is', 'running', 'through', 'the', 'snow', '.', '<END>']
['<START>', 'a', 'man', 'does', 'a', 'back', 'trick', 'on', 'a', 'skateboard',
'.', '<END>']
```

1.5 Part III - Conditioning on the Image (24 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:

[29]:
```
MAX_LEN = 40
EMBEDDING_DIM=300
IMAGE_ENC_DIM=300
vocab_size = len(word_to_id)
# 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_3"

--------------------------------------------------------------------------------
------------------
Layer (type)                    Output Shape          Param #     Connected to
```

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

------------------------------------------------------------------------------
------------------
dense_1 (Dense)                (None, 300)          614700      input_2[0][0]

------------------------------------------------------------------------------
------------------
input_3 (InputLayer)           [(None, 40)]         0

------------------------------------------------------------------------------
------------------
repeat_vector (RepeatVector)   (None, 40, 300)      0           dense_1[0][0]

------------------------------------------------------------------------------
------------------
embedding_1 (Embedding)        (None, 40, 300)      2312100     input_3[0][0]

------------------------------------------------------------------------------
------------------
concatenate (Concatenate)      (None, 40, 600)      0
repeat_vector[0][0]
embedding_1[0][0]

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

------------------------------------------------------------------------------
------------------
dense_2 (Dense)                (None, 7707)         3953691
bidirectional_1[0][0]
==============================================================================
==================
Total params: 8,635,627
Trainable params: 8,635,627
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:

[30]:
```
enc_train = np.load("gdrive/My Drive/"+my_data_dir+"/outputs/
↪encoded_images_train.npy")
enc_dev = np.load("gdrive/My Drive/"+my_data_dir+"/outputs/encoded_images_dev.
↪npy")
```

[ ]:
```
# enc_train
```

[31]:
```
def training_generator(batch_size=128):
  curr = []
  input_arr = []
  output_arr = []
  image_arr = np.asarray([])
  while True:
    for train_id in range(len(train_list)):
      for sentence in descriptions[train_list[train_id]]:
        sentence_int = [word_to_id[word] for word in sentence]
        for ind in range(len(sentence_int)-1):
          if len(input_arr) < batch_size:
            curr = sentence_int[:ind+1]
            curr.extend([0]*(MAX_LEN-ind-1))
            input_arr.append(curr)
            if len(image_arr) == 0:
              image_arr = enc_train[train_id]
            else:
              image_arr = np.vstack((image_arr,enc_train[train_id]))

            output_temp = [0]*vocab_size
            output_temp[sentence_int[ind+1]] = 1
            output_arr.append(output_temp)
          else:
            yield ([np.asarray(image_arr),np.asarray(input_arr)], \
                   np.asarray(output_arr))
            input_arr = []
            output_arr = []
            image_arr = np.asarray([])
```

[33]:
```
g = training_generator(128)
# print(next(g))
for i in range(3):
  inp, out = next(g)
#   if inp.shape != (128, 40) or out.shape != (128, 7707):
  print(inp[0].shape,inp[1].shape, out.shape)
#     print(111)
print(next(g))
```

```
(128, 2048) (128, 40) (128, 7707)
(128, 2048) (128, 40) (128, 7707)
(128, 2048) (128, 40) (128, 7707)
([array([[0.0741271 , 0.2739752 , 1.1527994 , ..., 0.0939803 , 0.6577325 ,
```

```
         0.8086839 ],
        [0.0741271 , 0.2739752 , 1.1527994 , ..., 0.0939803 , 0.6577325 ,
         0.8086839 ],
        [0.0741271 , 0.2739752 , 1.1527994 , ..., 0.0939803 , 0.6577325 ,
         0.8086839 ],
        ...,
        [0.27066442, 0.33927402, 0.19847967, ..., 0.3131588 , 0.16954991,
         0.        ],
        [0.27066442, 0.33927402, 0.19847967, ..., 0.3131588 , 0.16954991,
         0.        ],
        [0.27066442, 0.33927402, 0.19847967, ..., 0.3131588 , 0.16954991,
         0.        ]], dtype=float32), array([[  59,   61, 3963, ...,    0,    0,
0],
        [  59,   61, 3963, ...,    0,    0,    0],
        [  59,   61, 3963, ...,    0,    0,    0],
        ...,
        [  59, 6861, 3848, ...,    0,    0,    0],
        [  59, 6861, 3848, ...,    0,    0,    0],
        [  59, 6861, 3848, ...,    0,    0,    0]])], array([[0, 0, 0, ..., 0, 0,
0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]]))
```

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

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

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

```
Epoch 1/20
1875/1875 [==============================] - 219s 117ms/step - loss: 4.4000 -
accuracy: 0.2724
Epoch 2/20
1875/1875 [==============================] - 218s 116ms/step - loss: 3.6309 -
accuracy: 0.3654
Epoch 3/20
1875/1875 [==============================] - 218s 116ms/step - loss: 3.4506 -
accuracy: 0.3846
Epoch 4/20
1875/1875 [==============================] - 217s 116ms/step - loss: 3.3357 -
accuracy: 0.3948
Epoch 5/20
1875/1875 [==============================] - 217s 116ms/step - loss: 3.2785 -
```

```
accuracy: 0.4026
Epoch 6/20
1875/1875 [==============================] - 214s 114ms/step - loss: 3.2016 -
accuracy: 0.4079
Epoch 7/20
1875/1875 [==============================] - 213s 114ms/step - loss: 3.2074 -
accuracy: 0.4110
Epoch 8/20
1875/1875 [==============================] - 211s 113ms/step - loss: 3.1895 -
accuracy: 0.4129
Epoch 9/20
1875/1875 [==============================] - 211s 113ms/step - loss: 3.2054 -
accuracy: 0.4148
Epoch 10/20
1875/1875 [==============================] - 209s 112ms/step - loss: 3.2147 -
accuracy: 0.4155
Epoch 11/20
1875/1875 [==============================] - 210s 112ms/step - loss: 3.2307 -
accuracy: 0.4162
Epoch 12/20
1875/1875 [==============================] - 210s 112ms/step - loss: 3.2205 -
accuracy: 0.4184
Epoch 13/20
1875/1875 [==============================] - 213s 114ms/step - loss: 3.2389 -
accuracy: 0.4176
Epoch 14/20
1875/1875 [==============================] - 215s 114ms/step - loss: 3.2160 -
accuracy: 0.4197
Epoch 15/20
1875/1875 [==============================] - 214s 114ms/step - loss: 3.1961 -
accuracy: 0.4210
Epoch 16/20
1875/1875 [==============================] - 214s 114ms/step - loss: 3.2459 -
accuracy: 0.4184
Epoch 17/20
1875/1875 [==============================] - 212s 113ms/step - loss: 3.2124 -
accuracy: 0.4234
Epoch 18/20
1875/1875 [==============================] - 209s 111ms/step - loss: 3.2059 -
accuracy: 0.4229
Epoch 19/20
1875/1875 [==============================] - 207s 110ms/step - loss: 3.1859 -
accuracy: 0.4266
Epoch 20/20
1875/1875 [==============================] - 209s 112ms/step - loss: 3.2108 -
accuracy: 0.4248
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7f7f80609588>
```

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.

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

to load the model:

[ ]:
```
model.load_weights("gdrive/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.

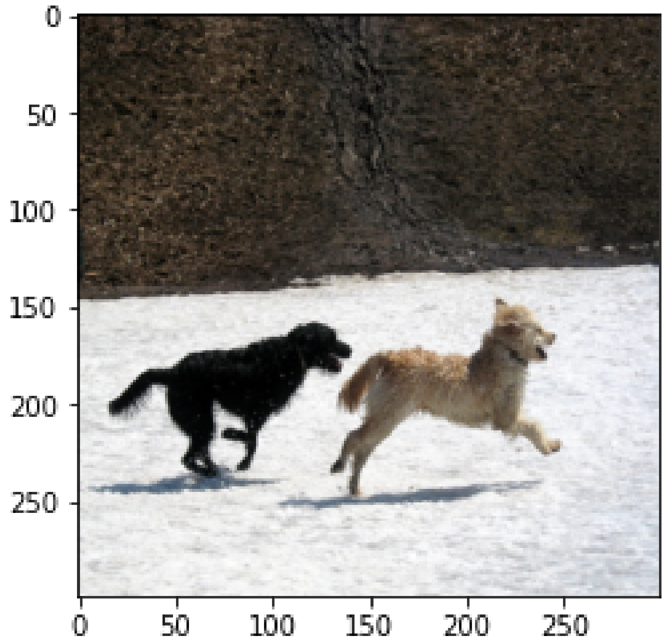
[ ]:
```
def img_decoder(enc_image):
    # ...
    sentence = ["<START>"]
    int_sentence = [word_to_id[sentence[0]]]
    int_sentence.extend([0]*(MAX_LEN-1))
    cnt = 0
    pred = "<START>"
    while cnt < MAX_LEN-1 and pred != "<END>":
      cnt += 1
      new_w = model.predict([np.array([enc_image]),np.array([int_sentence])])
      w = np.array(new_w, dtype=np.float32)
      sum_1_w = w / (w.sum() + 0.01)
      pred = id_to_word[int(np.argmax(np.random.
 ↪multinomial(20,sum_1_w[0],size=1)))]
      # print(pred)
      sentence.append(pred)
      int_sentence[cnt] = word_to_id[pred]

    return sentence
```

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

[ ]:
```
plt.imshow(get_image(train_list[0]))
img_decoder(enc_train[0])
```

[ ]:
```
['<START>',
 'a',
 'black',
 'dog',
 'is',
 'running',
 'through',
 'the',
 'water',
 '.',
```

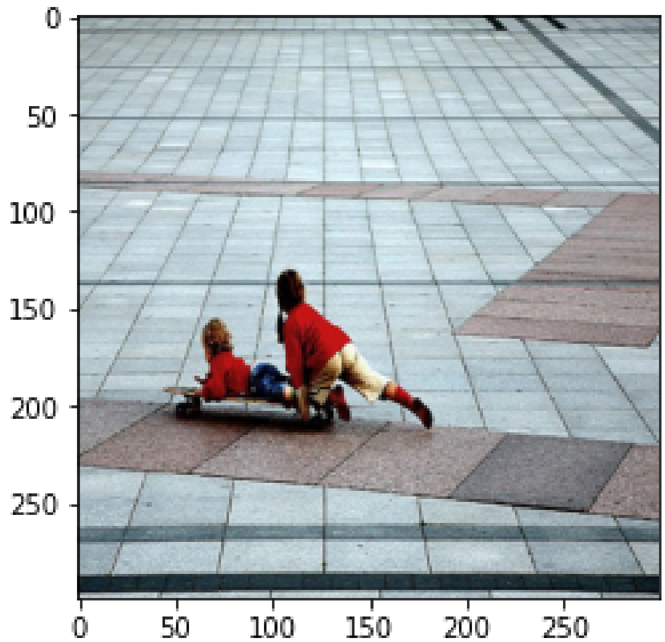
```
'<END>']
```



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

[ ]:
```
plt.imshow(get_image(dev_list[0]))
img_decoder(enc_dev[0])
```

[ ]:
```
['<START>',
 'a',
 'man',
 'in',
 'a',
 'hat',
 'and',
 'a',
 'white',
 'hat',
 'is',
 'standing',
 'on',
 'a',
 'city',
 'street',
 '.',
 '<END>']
```



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.

1.6 Part IV - Beam Search Decoder (24 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.

[ ]:
```
# dev_list[1]
enc_dev[0]
```

[ ]:
```
array([0.02197634, 0.57011366, 1.0859778 , ..., 1.1590774 , 0.01584431,
       0.2336205 ], dtype=float32)
```

[ ]:
```
def beam_decoder(n, image_enc):
    top_n = [(1,[word_to_id["<START>"]]) for i in range(n)]
    cnt = 1
```

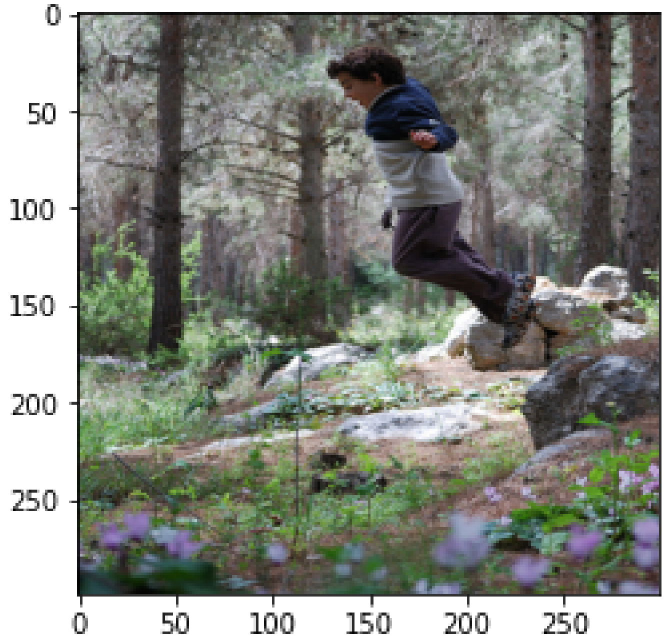
```
  while cnt < MAX_LEN-1:
    top_n_times_n = []

    # add all possible sq to the list
    for curr in top_n:
      # print(curr)
      curr_prob = curr[0]
      curr_s = curr[1]
      # print(2,curr_prob, curr_s)
      # curr = curr_s[1][-1] #last word
      curr_s.extend([0]*(MAX_LEN-cnt))
      # print(1,curr_s)
      # new_w = model.predict([np.array([enc_image]),np.array([int_sentence])])
      pred = model.predict([np.array([image_enc]),\
                            np.array([curr_s])])
      curr_s = curr_s[:cnt]
      # print(pred)
      w = np.array(pred, dtype=np.float32)
      # print(w)
      # w = np.array(new_w, dtype=np.float32)
      sum_1_w = w / (w.sum() + 0.01)
      # index of top n prob
      sorted_prob_ind = np.argsort(np.random.multinomial(n,\
                                                         sum_1_w[0],\
                                                         size=1))[::-1][:n]
      # print(sorted_prob_ind)
      sorted_sq = [(sum_1_w[0][int(i)]*curr_prob, curr_s+[int(i)]) \
                   for i in sorted_prob_ind[0]] # prob, seq pair
      top_n_times_n.extend(sorted_sq)

    # prune the list to top n seq
    top_n_times_n = sorted(top_n_times_n, key = lambda x: x[0])[::-1][:n]
    # update top n
    # top_n = [seq for (prob,seq) in top_n_times_n]
    top_n = top_n_times_n
    # print(top_n)
    cnt += 1


  top1 = sorted(top_n_times_n, key = lambda x: x[0])[-1]
  s = ''
  for ind,i in enumerate(top1[1]):
    if ind % 10 == 0:
      s = s +'\n' + str(id_to_word[int(i)])
    else:
      s = s +' ' + str(id_to_word[int(i)])
```

```
    return s
```

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.

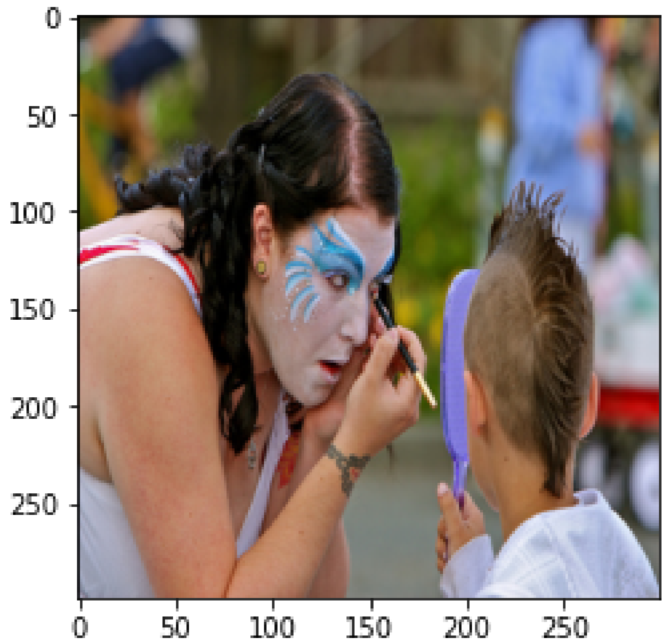
[ ]:
```
plt.imshow(get_image(dev_list[1]))
print("greedy output:")
print(img_decoder(enc_dev[1]))
n = 3
img = 1
print("beam search at n=3:")
print(beam_decoder(n, enc_dev[img]))
n = 5
print("beam search at n=5:")
print(beam_decoder(n, enc_dev[img]))
```

```
greedy output:
['<START>', 'a', 'man', 'in', 'the', 'mountains', '.', '<END>']
beam search at n=3:

<START> a man is standing on a rock overlooking a
mountain . <END> . <END> . <END> <END> . <END>
<END> . <END> <END> a . <END> <END> a a
. <END> <END> a a a a a a
beam search at n=5:

<START> a man is standing on a rock overlooking a
mountain . <END> . <END> . <END> <END> . <END>
<END> . <END> <END> a . <END> <END> a a
. <END> <END> a a a a a a
```



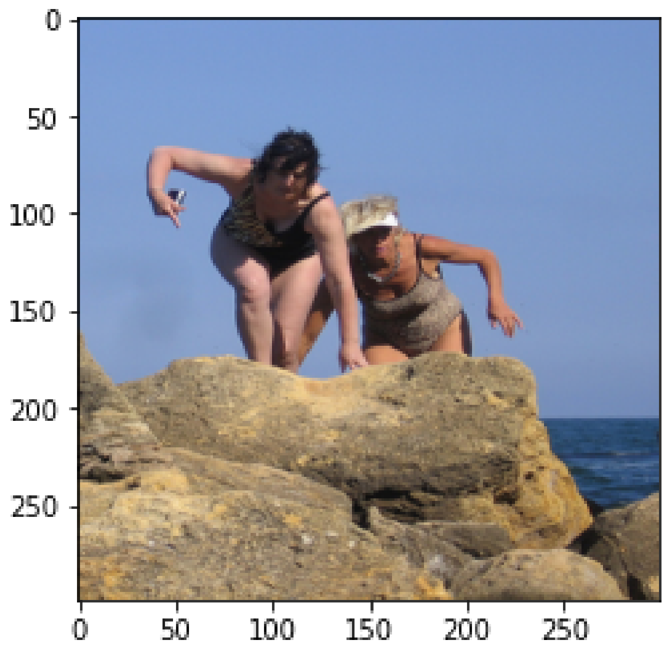
[ ]:
```
img = 19
plt.imshow(get_image(dev_list[img]))
print("greedy output:")
print(img_decoder(enc_dev[img]))
n = 3
print("beam search at n=3:")
print(beam_decoder(n, enc_dev[img]))
n = 5
print("beam search at n=5:")
print(beam_decoder(n, enc_dev[img]))
```

```
greedy output:
['<START>', 'a', 'young', 'boy', 'in', 'a', 'red', 'shirt', 'is', 'standing',
'in', 'a', 'field', '.', '<END>']
beam search at n=3:

<START> a young girl in a red shirt and a
man in a white shirt . <END> <END> . <END>
<END> a <END> a <END> a <END> a <END> a
<END> <END> <END> <END> <END> biker <END> surfer <END>
beam search at n=5:

<START> a young girl in a red shirt and a
man in a white shirt . <END> <END> . <END>
<END> a <END> a <END> a <END> a <END> a
<END> <END> <END> <END> <END> biker <END> surfer <END>
```



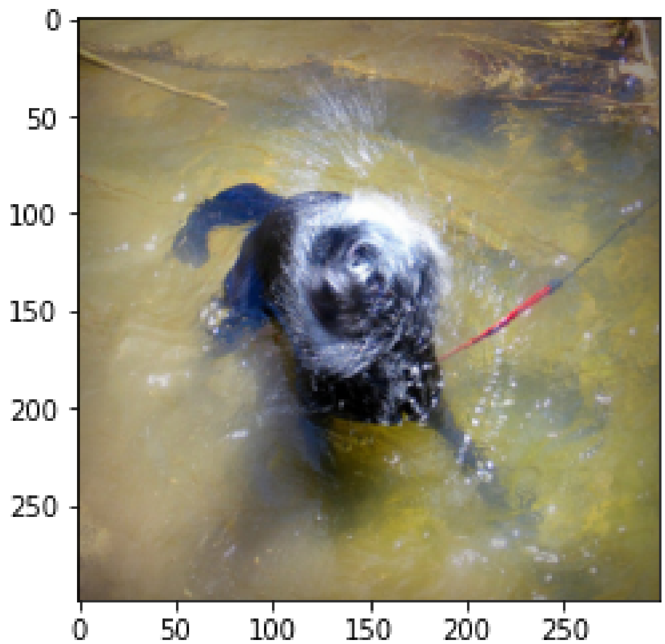
[ ]:
```python
img = 4
plt.imshow(get_image(dev_list[img]))
print("greedy output:")
print(img_decoder(enc_dev[img]))
n = 3
print("beam search at n=3:")
print(beam_decoder(n, enc_dev[img]))
n = 5
print("beam search at n=5:")
print(beam_decoder(n, enc_dev[img]))
```

```
greedy output:
['<START>', 'a', 'man', 'in', 'a', 'red', 'shirt', 'is', 'standing', 'on', 'a',
'rocky', 'beach', '.', '<END>']
beam search at n=3:

<START> a man in a white shirt and jeans is
standing on a rock . <END> . <END> <END> .
<END> <END> . <END> <END> . <END> <END> . <END>
<END> lake <END> . <END> <END> . <END> .
beam search at n=5:

<START> a man in a white shirt and jeans is
standing on a rock . <END> . <END> <END> .
<END> <END> . <END> <END> . <END> <END> . <END>
<END> lake <END> . <END> <END> . <END> .
```



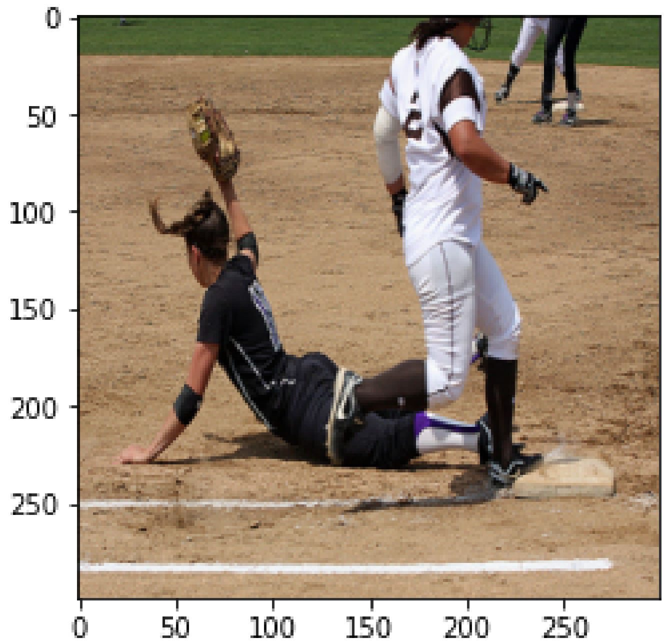
[ ]:
```
img = 12
plt.imshow(get_image(dev_list[img]))
print("greedy output:")
print(img_decoder(enc_dev[img]))
n = 3
print("beam search at n=3:")
print(beam_decoder(n, enc_dev[img]))
n = 5
print("beam search at n=5:")
print(beam_decoder(n, enc_dev[img]))
```

```
greedy output:
['<START>', 'a', 'man', 'in', 'a', 'black', 'shirt', 'is', 'walking', 'on', 'a',
'beach', '.', '<END>']
beam search at n=3:

<START> a man in a black shirt is standing in
the water . <END> <END> . <END> <END> <END> <END>
<END> <END> beach beach <END> beach beach <END> beach beach
<END> background <END> beach <END> <END> <END> beach <END>
beam search at n=5:

<START> a man in a black shirt is standing in
the water . <END> <END> . <END> <END> <END> <END>
<END> <END> beach beach <END> beach beach <END> beach beach
<END> background <END> beach <END> <END> <END> beach <END>
```



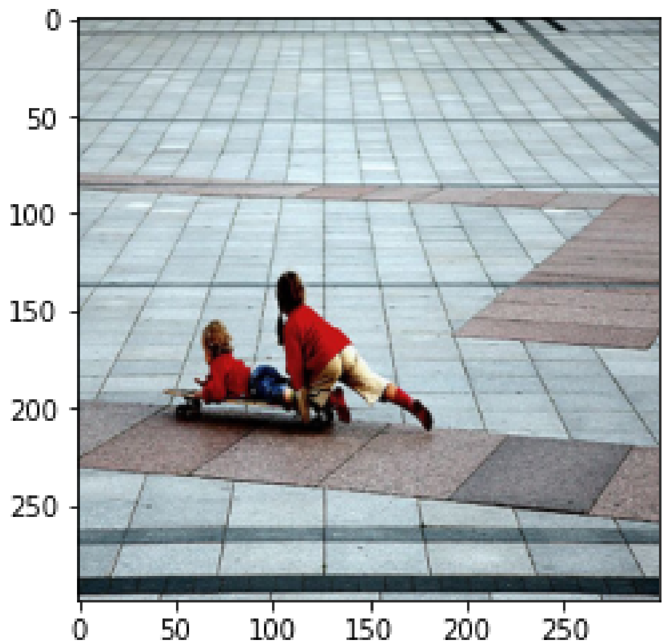

```
[ ]: img = 11
plt.imshow(get_image(dev_list[img]))
print("greedy output:")
print(img_decoder(enc_dev[img]))
n = 3
print("beam search at n=3:")
print(beam_decoder(n, enc_dev[img]))
n = 5
print("beam search at n=5:")
print(beam_decoder(n, enc_dev[img]))
```

```
greedy output:
['<START>', 'a', 'man', 'in', 'a', 'black', 'jacket', 'is', 'standing', 'in',
'the', 'water', '.', '<END>']
beam search at n=3:

<START> a man in a black shirt and a woman
in a white shirt . <END> <END> a trick .
<END> <END> a <END> . <END> <END> <END> <END> <END>
<END> <END> <END> <END> <END> <END> <END> <END> <END>
beam search at n=5:

<START> a man in a black shirt and a woman
in a white shirt . <END> <END> a trick .
<END> <END> a <END> . <END> <END> <END> <END> <END>
<END> <END> <END> <END> <END> <END> <END> <END> <END>
```



[ ]:
```
img = 0
plt.imshow(get_image(dev_list[img]))
print("greedy output:")
print(img_decoder(enc_dev[img]))
n = 3
print("beam search at n=3:")
print(beam_decoder(n, enc_dev[img]))
n = 5
print("beam search at n=5:")
print(beam_decoder(n, enc_dev[img]))
```

```
greedy output:
['<START>', 'a', 'man', 'in', 'a', 'hat', 'and', 'a', 'woman', 'in', 'a', 'red',
'jacket', '.', '<END>']
beam search at n=3:

<START> a man in a red shirt and a woman
in a red hat . <END> a <END> . <END>
<END> a a a <END> a a <END> a a
a a <END> a a <END> a a a
beam search at n=5:

<START> a man in a red shirt and a woman
in a red hat . <END> a <END> . <END>
<END> a a a <END> a a <END> a a
a a <END> a a <END> a a a
```


0
50
100
150
200
250
0
50
100
150
200
250

[ ]: