COMS W4705 - Homework 4

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.

Getting Started

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

```
import os
from collections import defaultdict
import numpy as np
import PIL
from matplotlib import pyplot as plt
%matplotlib inline
from tensorflow import keras
from keras import Sequential, Model
from keras.layers import Embedding, LSTM, Dense, Input, Bidirectional, RepeatVector,
from keras.activations import softmax
from tensorflow.keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences

from keras.applications.inception_v3 import InceptionV3

from tensorflow.keras.optimizers import Adam
from google.colab import drive
import random
```

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

```
#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"
```

Mounting your GDrive so you can access the files from Colab

```
#running this command will generate a message that will ask you to click on a link wh
#copy paste that code in the text box that will appear below
drive.mount('/content/gdrive/',force_remount=True)

     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.

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.

```
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.trai
dev_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8k.devIma
test_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8k.testI
```

Let's see how many images there are

```
len(train_list), len(dev_list), len(test_list)
```

```
    (6000, 1000, 1000)
```

Each entry is an image filename.

```
dev_list[20]
```

```
    '3693961165_9d6c333d5b.jpg'
```

The images are located in a subdirectory.

```
IMG_PATH = '/content/gdrive/My Drive/'+my_data_dir+'/Flickr8k_Dataset'
```

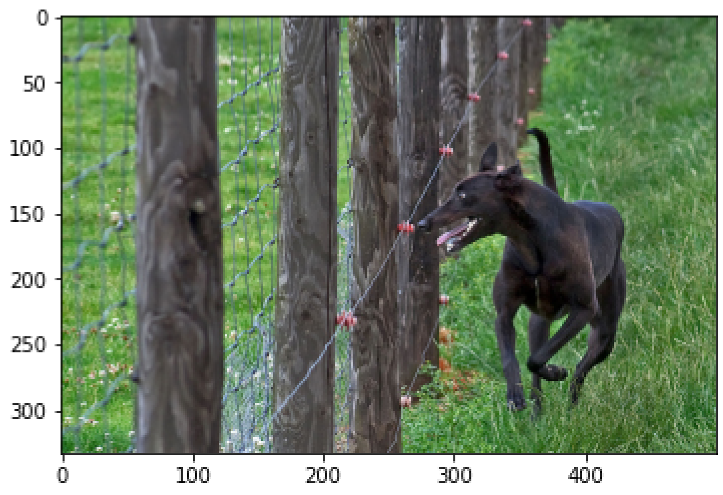
We can use PIL to open the image and matplotlib to display it.

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

if you can't see the image, try

```
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f5d8e98ea90>
```



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

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

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.

```
np.asarray(image).shape
```

```
(333, 500, 3)
```

The values range from 0 to 255.

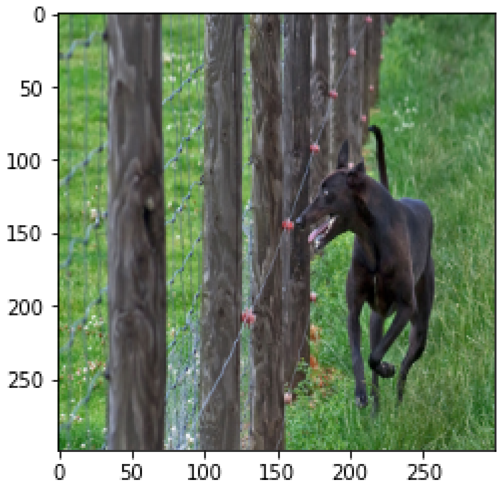
```
np.asarray(image)
```

```
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.

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

```
<matplotlib.image.AxesImage at 0x7f5d8e47d710>
```



```
new_image.shape
```

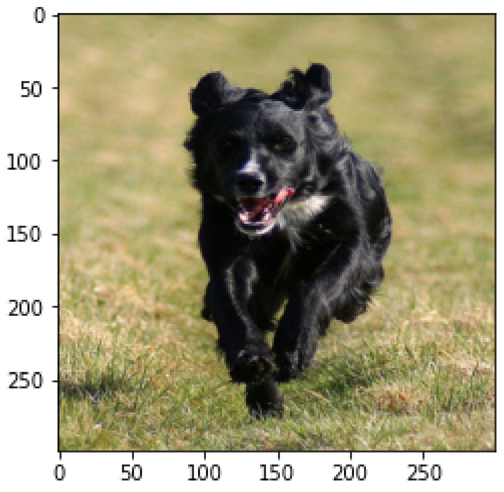
```
(299, 299, 3)
```

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

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

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

```
<matplotlib.image.AxesImage at 0x7f5d8e3f6490>
```



Next, we load the pre-trained Inception model.

```
img_model = InceptionV3(weights='imagenet') # This will download the weight files for
```

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

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

```
 conv2d_84 (Conv2D)             (None, 8, 8, 192)    245760      ['average_poo

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

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

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

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

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

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

 activation_76 (Activation)     (None, 8, 8, 320)    0           ['batch_norma

 mixed9_0 (Concatenate)         (None, 8, 8, 768)    0           ['activation_
                                                                  'activation_

 concatenate (Concatenate)      (None, 8, 8, 768)    0           ['activation_
                                                                  'activation_

 activation_84 (Activation)     (None, 8, 8, 192)    0           ['batch_norma

 mixed9 (Concatenate)           (None, 8, 8, 2048)   0           ['activation_
                                                                  'mixed9_0[0]
                                                                  'concatenate
                                                                  'activation_

 conv2d_89 (Conv2D)             (None, 8, 8, 448)    917504      ['mixed9[0][0

 batch_normalization_89 (BatchN  (None, 8, 8, 448)   1344        ['conv2d_89[0
 ormalization)

 activation_89 (Activation)     (None, 8, 8, 448)    0           ['batch_norma

 conv2d_86 (Conv2D)             (None, 8, 8, 384)    786432      ['mixed9[0][0

 conv2d_90 (Conv2D)             (None, 8, 8, 384)    1548288     ['activation_

 batch_normalization_86 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_86[0
 ormalization)
```

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

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

activation_90 (Activation)     (None, 8, 8, 384)    0           ['batch_norma

conv2d_87 (Conv2D)             (None, 8, 8, 384)    442368      ['activation_

conv2d_88 (Conv2D)             (None, 8, 8, 384)    442368      ['activation_

conv2d_91 (Conv2D)             (None, 8, 8, 384)    442368      ['activation_
```

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 mo
```

Let's try the encoder.

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

```
encoded_image
```

```
    array([[0.6380658 , 0.48873055, 0.05526222, ..., 0.6425573 , 0.29595232,
            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 img_name in img_list:
      image = get_image(img_name)
      resized_image = np.array([image])
      yield resized_image
```

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

```
#NOTE: I stopped this early because I was loading from already created list of weight

enc_train = img_encoder.predict_generator(img_generator(train_list), steps=len(train_
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `Mc
  """Entry point for launching an IPython kernel.
1642/6000 [=======>......................] - ETA: 38:41
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-26-57cffb83e012> in <module>()
----> 1 enc_train = img_encoder.predict_generator(img_generator(train_list),
steps=len(train_list), verbose=1)

                                    9 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     57     ctx.ensure_initialized()
     58     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,
op_name,
---> 59                                         inputs, attrs, num_outputs)
     60   except core._NotOkStatusException as e:
     61     if name is not None:

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
enc_train[11]
```

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Tue Dec  7 22:52:18 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 495.44       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   46C    P0    35W / 250W |   3885MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
enc_dev = img_encoder.predict_generator(img_generator(dev_list), steps=len(dev_list),
```

```
enc_test = img_encoder.predict_generator(img_generator(test_list), steps=len(test_lis
```

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)
```

```
# dont re-run above lines after saving
enc_train = np.load("/content/gdrive/MyDrive/"+my_data_dir+"/outputs/encoded_images_t
enc_dev = np.load("/content/gdrive/MyDrive/"+my_data_dir+"/outputs/encoded_images_dev
enc_test = np.load("/content/gdrive/MyDrive/"+my_data_dir+"/outputs/encoded_images_te
```

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

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

Reading image descriptions

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

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

```
def read_image_descriptions(filename):
    image_descriptions = defaultdict(list)
    with open(filename) as file:
      for line in file:
        split_line = line.split("\t")
        jpg_name = split_line[0].split("#")[0]
        description = split_line[1].split("\n")[0].split(" ")
        lower =  (map(lambda x: x.lower(), description))
        lower_list = list(lower)
        lower_list.insert(0, '<START>')
        lower_list.append('<END>')
        if jpg_name not in image_descriptions:
          image_descriptions[jpg_name] = []

        curr_list = image_descriptions[jpg_name]
        curr_list.append(lower_list)
        image_descriptions[jpg_name] = curr_list

    return image_descriptions


descriptions = read_image_descriptions("/content/gdrive/MyDrive/"+my_data_dir+"/Flick


print(descriptions[dev_list[0]])
```

```
[['<START>', 'the', 'boy', 'laying', 'face', 'down', 'on', 'a', 'skateboard', 'i
```

Running the previous cell should print:

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

Creating Word Indices

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

```
dict_values = [*descriptions.values()]
words = set()
for jpg in dict_values:
  for sentence in jpg:
    for word in sentence:
      words.add(word)

words_list = list(words)
words_list.sort()


id_to_word = defaultdict()
word_to_id = defaultdict()
count=1
for word in words_list:
  word_to_id[word] = count
  id_to_word[count] = word
  count = count + 1
```

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

```
     2310
```

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

```
     'crows'
```

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

Part III Basic Decoder Model (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

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).

```
max(len(description) for image_id in train_list for description in descriptions[image
```

```
    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.

```
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=['accurac

model.summary()
```

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

 embedding (Embedding)       (None, 40, 300)           2676000

 bidirectional (Bidirectiona  (None, 1024)             3330048
 l)

 dense (Dense)               (None, 8920)              9143000

=================================================================
Total params: 15,149,048
Trainable params: 15,149,048
Non-trainable params: 0
_________________________________________________________________
```

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

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

Creating a Generator for the Training Data

TODO:

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

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

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

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

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

```
training_size = len(train_list)
def text_training_generator(batch_size=128):
    while True:
      input = np.zeros(shape=(batch_size,MAX_LEN))
      output = np.zeros(shape=(batch_size,vocab_size))

      for i in range(batch_size):
        train_index = random.randint(0, training_size - 1)
        description_index = random.randint(0, 4)

        jpg_name = train_list[train_index]
        description = descriptions[jpg_name][description_index]

        word_index = random.randint(0, len(description) - 2) #2 since last is always
        #no point predicting after that

        for j in range(word_index+1):
          input[i][j] = word_to_id[description[j]]
        next_word = description[word_index+1]
        output[i][word_to_id[next_word]] = 1

      yield (input, output)



def test_generator(batch_size=1):
  input = np.zeros(shape=(batch_size,MAX_LEN))
  output = np.zeros(shape=(batch_size,vocab_size))
```

```
    for i in range(batch_size):
      train_index = random.randint(0, training_size - 1)
      description_index = random.randint(0, 4)
      print(description_index)
      jpg_name = train_list[train_index]
      description = descriptions[jpg_name][description_index]
      word_index = random.randint(0, len(description) - 2) #2 since last is always <END
      #no point predicting after that
      for j in range(word_index+1):
        input[i][j] = word_to_id[description[j]]
      next_word = description[word_index+1]
      print(next_word)
      output[i][word_to_id[next_word]] = 1
      print(input[i])

test_generator()
```

```
2
dog
[  70.   72. 8684. 2310. 4013. 5856. 5170.   72. 1069.    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.]
```

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

```
batch_size = 128
generator = text_training_generator(batch_size)
steps = len(train_list) * MAX_LEN * 5 // batch_size
```

```
model.fit_generator(generator, steps_per_epoch=steps, verbose=True, epochs=2)
```

```
Epoch 1/2
   3/9375 [..............................] - ETA: 6:01 - loss: 2.5461 - accuracy
  """Entry point for launching an IPython kernel.
9375/9375 [==============================] - 328s 35ms/step - loss: 2.7908 - acc
Epoch 2/2
9375/9375 [==============================] - 328s 35ms/step - loss: 2.6408 - acc
<keras.callbacks.History at 0x7f5ba23490d0>
```

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

Greedy Decoder

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

```
def decoder():
    input = np.zeros(shape=(1,MAX_LEN))
    start_id = word_to_id['<START>']
    input[0][0] = start_id
    foundEnd=False
    index = 1
    while not foundEnd:
      output = model.predict(input)
      index_of_most_likely = np.argmax(output)
      most_likely_word = id_to_word[index_of_most_likely]
      input[0][index] = index_of_most_likely
      index = index + 1
      if word_to_id['<END>'] == index_of_most_likely or index == MAX_LEN:
        foundEnd=True
    return [input]
```

```
print(decoder())
```

```
    [array([[  70.,   72., 4578., 3920.,   72., 8684., 6802.,  263., 4051.,
            4013., 6942., 5170.,   72.,  728., 3754.,   72., 6199.,  541.,
              15.,   69.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
               0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
               0.,    0.,    0.,    0.]])]
```

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.

```
def sample_decoder():
    input = np.zeros(shape=(1,MAX_LEN))
    start_id = word_to_id['<START>']
    input[0][0] = start_id
    foundEnd=False
    index = 1
    while not foundEnd:
      output = model.predict(input)
      #print(output[0])
      if sum(output[0]) > 1:
        output = [element * .999999 for element in output]
      sampled_index_array = np.random.multinomial(1, output[0])
      sampled_index = np.argmax(sampled_index_array)
      most_likely_word = id_to_word[sampled_index]
      print(most_likely_word)
      input[0][index] = sampled_index
      index = index + 1
      if word_to_id['<END>'] == sampled_index or index == MAX_LEN:
        foundEnd=True

    return [input]
```

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.

```
for i in range(10):
    print(sample_decoder())
```

```
.
<END>
[array([[  70.,   72., 7685., 3920., 7920., 8595.,   15.,   69.,    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.]])]
two
dogs
are
playing
in
the
grass
.
<END>
[array([[  70., 8283., 2315.,  344., 5722., 3920., 7920., 3402.,   15.,
          69.,    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.]])]
a
small
toddler
sits
on
a
beach
.
<END>
```

```
[array([[   70.,    72., 7074., 8029., 6941., 5170.,    72.,   654.,    15.,
           69.,     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.]])]
a
young
girl
wearing
a
black
dress
and
green
blue
hair
has
a
a
ball
in
a
wood-stacked
.
<END>
[array([[   70.,    72., 8891., 3277., 8605.,    72.,   803., 2395.,   263.,
         3422.,   864., 3518., 3607.,    72.,    72.,   541., 3920.,    72.,
         8795.,    15.,    69.,     0.,     0.,     0.,     0.,     0.,     0.,
            0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
```

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:

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

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

# Text input
text_input = Input(shape=(MAX_LEN,))
```

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

model.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 input_3 (InputLayer)           [(None, 2048)]       0           []

 dense_1 (Dense)                (None, 300)          614700      ['input_3[0][0]

 input_4 (InputLayer)           [(None, 40)]         0           []

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

 embedding_1 (Embedding)        (None, 40, 300)      2676000     ['input_4[0][0]

 concatenate_2 (Concatenate)    (None, 40, 600)      0           ['repeat_vector
                                                                  'embedding_1[0

 bidirectional_1 (Bidirectional  (None, 512)         1755136     ['concatenate_2
 )

 dense_2 (Dense)                (None, 8920)         4575960     ['bidirectional

==========================================================================================
Total params: 9,621,796
Trainable params: 9,621,796
Non-trainable params: 0
__________________________________________________________________________________________
```

The model now takes two inputs:

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

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

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

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

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

```
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")
print(enc_train[4])
```

```
    [0.22433461 0.817485   0.03405499 ... 0.02048848 0.7156485  0.9570548 ]
```

```
training_size = len(train_list)
def training_generator(batch_size=128):
    while True:
      input_text = np.zeros(shape=(batch_size,MAX_LEN))
      input_encoded_image = np.zeros(shape=(batch_size,2048))
      output = np.zeros(shape=(batch_size,vocab_size))

      for i in range(batch_size):
        train_index = random.randint(0, training_size - 1)
        encoded_image = enc_train[train_index]
        input_encoded_image[i] = encoded_image
        description_index = random.randint(0, 4)

        jpg_name = train_list[train_index]
        description = descriptions[jpg_name][description_index]

        word_index = random.randint(0, len(description) - 2) #2 since last is always
        #no point predicting after that

        for j in range(word_index+1):
          input_text[i][j] = word_to_id[description[j]]
        next_word = description[word_index+1]
        output[i][word_to_id[next_word]] = 1

      yield ([input_encoded_image,input_text], output)
```

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 * 5 // batch_size
```

```
model.fit_generator(generator, steps_per_epoch=steps, verbose=True, epochs=2)
```

```
    Epoch 1/2
       3/9375 [..............................] - ETA: 4:19 - loss: 3.0058 - accuracy
      """Entry point for launching an IPython kernel.
    9375/9375 [==============================] - 245s 26ms/step - loss: 2.9234 - acc
    Epoch 2/2
```

```
9375/9375 [==============================] - 244s 26ms/step - loss: 2.8274 - acc
<keras.callbacks.History at 0x7f5ce9bd7a10>
```

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")
```

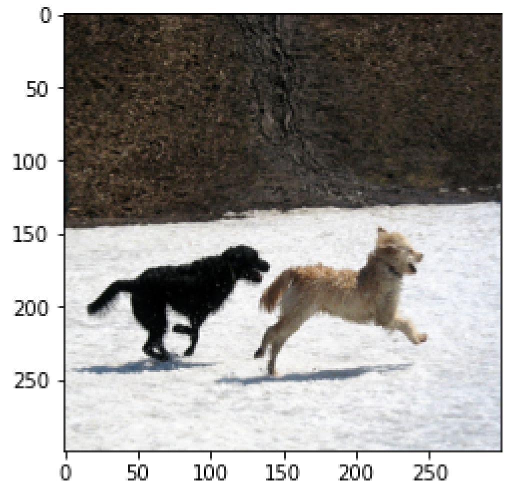
TODO: Now we are ready to actually generate image captions using the trained model. Modify the simple greedy decoder you wrote for the text-only generator, so that it takes an encoded image (a vector of length 2048) as input, and returns a sequence.

```
def image_decoder(enc_image):
    input = np.zeros(shape=(1,MAX_LEN))
    input_encoded_image = np.zeros(shape=(1,2048))
    input_encoded_image[0] = enc_image
    start_id = word_to_id['<START>']
    input[0][0] = start_id
    foundEnd=False
    index = 1
    while not foundEnd:
      output = model.predict([input_encoded_image,input])
      index_of_most_likely = np.argmax(output)
      most_likely_word = id_to_word[index_of_most_likely]
      #print(most_likely_word)
      input[0][index] = index_of_most_likely
      index = index + 1
      if word_to_id['<END>'] == index_of_most_likely or index == MAX_LEN:
        foundEnd=True

    input_to_words(input)
    return [input]
```

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

```
plt.imshow(get_image(train_list[0]))
image_decoder(enc_train[0])
#print(enc_train[0].shape)
```

```
a
man
and
a
dog
play
in
the
snow
.
<END>
[array([[  70.,   72., 4578.,  263.,   72., 2310., 5710., 3920., 7920.,
         7123.,   15.,   69.,    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 also be able to apply the model to dev images and get reasonable captions:

```
plt.imshow(get_image(dev_list[1]))
image_decoder(enc_dev[1])
```

```
a
skateboarder
jumps
into
the
air
.
<END>
[array([[  70.,   72., 6951., 4117., 3995., 7920.,  193.,   15.,   69.,
            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.]])]
```

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.

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.

```
import copy
def img_beam_decoder(n, image_enc):
    prob_and_seq = list()
    input = np.zeros(shape=(1,MAX_LEN))
    start_id = word_to_id['<START>']
    input[0][0] = start_id
    prob_and_seq.append((1,input))
    input_encoded_image = np.zeros(shape=(1,2048))
    input_encoded_image[0] = image_enc

    for index in range(1,40):
      next_step_prob_and_seq = list()
      for i in range(min(n, len(prob_and_seq))) :
        curr_tuple = prob_and_seq[i]
        curr_prob = curr_tuple[0]
        curr_seq = curr_tuple[1]

        #check if this sequence has reached an <END> - if so, do not predict
        #further on this branch
        if word_to_id['<END>'] not in curr_seq[0]:
          output = model.predict([input_encoded_image,curr_seq])
          indices_of_n_most_likely = np.argpartition(output[0], -n)[-n:]
          for likely_index in indices_of_n_most_likely:
            likely_index_prob = output[0][likely_index]
            updated_prob = curr_prob * likely_index_prob
            updated_seq_copy = copy.deepcopy(curr_seq)
            updated_seq_copy[0][index] = likely_index
            next_step_prob_and_seq.append((updated_prob, updated_seq_copy))

        else:
          next_step_prob_and_seq.append((curr_prob, curr_seq))

      next_step_prob_and_seq.sort()
      prob_and_seq = next_step_prob_and_seq[-n:]

    for sent in prob_and_seq:
      seq = sent[1]
      input_to_words(seq)
    return [prob_and_seq]



import copy
def test_img_beam_decoder(n, image_enc):
    prob_and_seq = list()
    input = np.zeros(shape=(1,MAX_LEN))
    start_id = word_to_id['<START>']
    input[0][0] = start_id
    prob_and_seq.append((1,input))

    input_encoded_image = np.zeros(shape=(1,2048))
    input_encoded_image[0] = image_enc

    for index in range(1,40):
      next_step_prob_and_seq = list()
      for i in range(min(n, len(prob_and_seq))) :
        curr_tuple = prob_and_seq[i]
        curr_prob = curr_tuple[0]
        curr_seq = curr_tuple[1]

        #check if this sequence has reached an <END> - if so, do not predict
        #further on this branch
        if word_to_id['<END>'] not in curr_seq[0]:
          output = model.predict([input_encoded_image,curr_seq])
          indices_of_n_most_likely = np.argpartition(output[0], -n)[-n:]
          #print(indices_of_n_most_likely)
          #input_to_words_1D(indices_of_n_most_likely)
          for likely_index in indices_of_n_most_likely:
            likely_index_prob = output[0][likely_index]
            updated_prob = curr_prob * likely_index_prob
            updated_seq_copy = copy.deepcopy(curr_seq)
            updated_seq_copy[0][index] = likely_index
            next_step_prob_and_seq.append((updated_prob, updated_seq_copy))

            #print('adding sentence to potential list\n')
            #input_to_words(updated_seq_copy)
        else:
          next_step_prob_and_seq.append((curr_prob, curr_seq))
          #print('adding sentence which already ended\n')

      #prune to top n
      debug_list_tuples(next_step_prob_and_seq)
      next_step_prob_and_seq.sort()
      topn = next_step_prob_and_seq[-n:]
      print('now consider top n moving on')

      prob_and_seq = topn
      debug_list_tuples(prob_and_seq)

    for sent in prob_and_seq:
      seq = sent[1]
      input_to_words(seq)
    return [prob_and_seq]

test_img_beam_decoder(3, enc_dev[2])
```

```
  end round


  now consider top n moving on

   Tuple prob=0.0018276724395120752 words=  <START> a young child in


   Tuple prob=0.0027218944031903547 words=  <START> a young child is


   Tuple prob=0.015103752004788376 words=  <START> a man in a

  end round
```

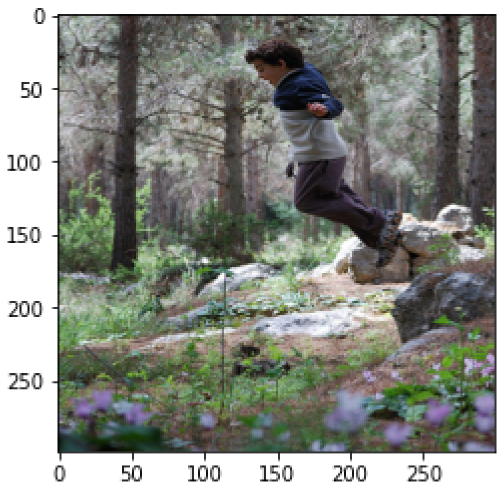
```
 Tuple prob=2.5782284181446084e-05 words=  <START> a young child in black

 Tuple prob=6.357657076270728e-05 words=  <START> a young child in blue

 Tuple prob=0.0016067578695936084 words=  <START> a young child in a

 Tuple prob=0.00014870399687458402 words=  <START> a young child is being

 Tuple prob=0.00019747329361906383 words=  <START> a young child is boys

 Tuple prob=0.0002679116052418473 words=  <START> a young child is eating

 Tuple prob=0.00098059174453507 words=  <START> a man in a green

 Tuple prob=0.0013145064469303937 words=  <START> a man in a black

 Tuple prob=0.005017688188826363 words=  <START> a man in a blue
end round


now consider top n moving on
 Tuple prob=0.0013145064469303937 words=  <START> a man in a black

 Tuple prob=0.0016067578695936084 words=  <START> a young child in a

 Tuple prob=0.005017688188826363 words=  <START> a man in a blue
```

```
def input_to_words(input):
  output = ''
  for index in input[0]:
    if int(index) != 0:
      output = output + ' ' + id_to_word[int(index)]
  print(output)

def input_to_words_string(input):
  output = ''
  for index in input[0]:
    if int(index) != 0:
      output = output + ' ' + id_to_word[int(index)]
  return output

def input_to_words_1D(input):
  for index in input:
    if int(index) != 0:
      print(id_to_word[int(index)])

print(id_to_word[72])

def debug_list_tuples(prob_and_sequence):
  for option in prob_and_sequence:
    prob = option[0]
    sequence = option[1]
    print('\n Tuple prob=' + str(prob) + ' words= ' + input_to_words_string(sequence)
  print('end round \n \n ')
```

```
     a
```

```
img_beam_decoder(3, enc_dev[1])
plt.imshow(get_image(dev_list[1]))
```

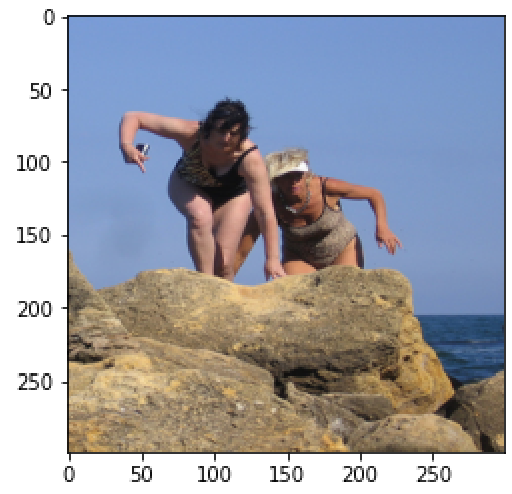
```
      <START> a boy in a blue shirt is climbing a rock wall with a blue bucket . <END
      <START> a boy in a blue shirt is climbing a rock wall . <END>
      <START> a skateboarder jumps into the air . <END>
     <matplotlib.image.AxesImage at 0x7f6ff2609b10>
```



```
#Below are 5 dev photos with captions


print('\n beam search for n=3\n')
img_beam_decoder(3, enc_dev[4])
print('\n beam search for n=5\n')
img_beam_decoder(5, enc_dev[4])
print('\n greedy output\n')
image_decoder(enc_dev[4])
print('\n')
```

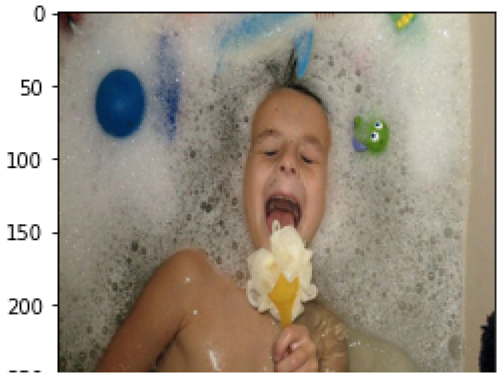
```
plt.imshow(get_image(dev_list[4]))
```

```
 beam search for n=3

<START> a woman with a backpack climbs a rock wall <END>
<START> a woman with a backpack climbs a rock wall . <END>
<START> a woman with a backpack climbs a rock <END>

 beam search for n=5

<START> a woman with a bikini climbs on the rocks with a bikini on it . <END>
<START> a woman with a backpack climbs a rock overlooking a city . <END>
<START> a woman with a backpack climbs a rock wall <END>
<START> a woman with a backpack climbs a rock wall . <END>
<START> a woman with a backpack climbs a rock <END>

 greedy output

<START> a woman in a bikini is climbing on a rock overlooking a city . <END>


<matplotlib.image.AxesImage at 0x7f6ff2ee5450>
```



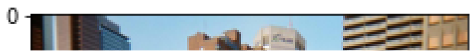
```
print('\n beam search for n=3\n')
img_beam_decoder(3, enc_dev[5])
print('\n beam search for n=5\n')
img_beam_decoder(5, enc_dev[5])
print('\n greedy output\n')
image_decoder(enc_dev[5])
print('\n')
plt.imshow(get_image(dev_list[5]))
```

```
 beam search for n=3

<START> a baby with an orange shirt and blue shirt is smiling . <END>
<START> a baby with an orange dress and blue shirt is smiling . <END>
<START> a baby in a baby smiling . <END>

 beam search for n=5

<START> a baby with an orange dress and blue shirt and a blue shirt is smiling
<START> a baby with an orange shirt and blue shirt is smiling over the camera .
<START> a little boy with a blue shirt and blue shirt is smiling . <END>
<START> a baby with an orange shirt and blue shirt is smiling . <END>
<START> a baby with an orange dress and blue shirt is smiling . <END>

 greedy output

<START> a baby in a baby smiling . <END>


<matplotlib.image.AxesImage at 0x7f6ff279dbd0>
```



```
print('\n beam search for n=3\n')
img_beam_decoder(3, enc_dev[6])
print('\n beam search for n=5\n')
img_beam_decoder(5, enc_dev[6])
print('\n greedy output\n')
image_decoder(enc_dev[6])
print('\n')
plt.imshow(get_image(dev_list[6]))
```

```
beam search for n=3

<START> a crowd of people are gathered in a lake <END>
<START> a crowd of people are gathered in a river <END>
<START> a crowd of people are gathered in a lake . <END>

beam search for n=5

<START> a crowd of people are gathered in a lake with many dogs in the backgrou
<START> a crowd of people are gathered in a lake with many dogs . <END>
<START> a crowd of people are gathered in a lake <END>
<START> a crowd of people are gathered in a river <END>
<START> a crowd of people are gathered in a lake . <END>

greedy output

<START> a crowd of people are gathered in a lake . <END>


<matplotlib.image.AxesImage at 0x7f6ff225d7d0>
```



```
print('\n beam search for n=3\n')
img_beam_decoder(3, enc_dev[80])
print('\n beam search for n=5\n')
img_beam_decoder(5, enc_dev[80])
print('\n greedy output\n')
image_decoder(enc_dev[80])
print('\n')
plt.imshow(get_image(dev_list[80]))
```

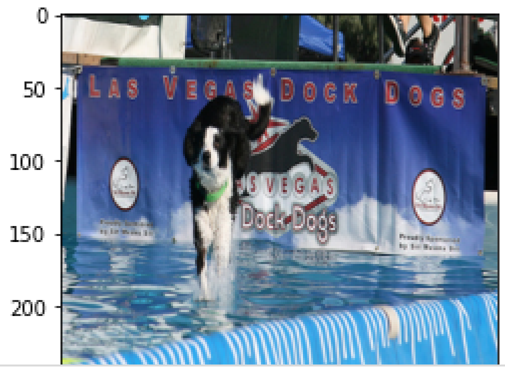
```
beam search for n=3

<START> a man in a blue shirt and blue jeans is standing in the street with a b
<START> a man in a blue shirt and blue jeans is holding a baby in a blue shirt
<START> a man in a blue shirt and blue jeans is standing in the street . <END>

beam search for n=5

<START> a man in a blue shirt is standing in front of a crowd of people on a be
<START> a man in a blue shirt is standing in front of a crowd of people on a si
<START> a man in a blue shirt is standing in front of a crowd of people on the
<START> a man in a blue shirt and blue jeans is standing in the street . <END>
```

```
print('\n beam search for n=3\n')
img_beam_decoder(3, enc_dev[81])
print('\n beam search for n=5\n')
img_beam_decoder(5, enc_dev[81])
print('\n greedy output\n')
image_decoder(enc_dev[81])
print('\n')
plt.imshow(get_image(dev_list[81]))
```

```
beam search for n=3

<START> a black dog leaps into the pool . <END>
```

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.

```
<START> a dog leaps into the pool . <END>
<START> a black dog leaps into the pool . <END>
<START> a dog jumps into a pool . <END>
<START> a black dog jumping into a pool . <END>
<START> a black dog jumps into a pool . <END>

greedy output

<START> a black dog jumps into a pool . <END>


<matplotlib.image.AxesImage at 0x7f6ff25451d0>
```



✓ 4s completed at 11:10 AM