COMS W4705 - Homework 5

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

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

from keras.applications.inception_v3 import InceptionV3

from keras.optimizers import Adam

from google.colab import drive
```

Access to the flickr8k data

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

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

I have uploaded all the data and model files you'll need to my GDrive and you can access the folder he
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 upl
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 t
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 whe
#copy paste that code in the text box that will appear below
# only used in colab
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call dr
```

Please look at the 'Files' tab on the left side and make sure you can see the 'hw5_data' folder that you

Part I: Image Encodings (14 pts)

The files Flickr_8k.trainImages.txt Flickr_8k.devImages.txt Flickr_8k.testImages.txt, contain a list of tr
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.trair
dev_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8k.devImag
test_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8k.testIr
```

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"
IMG_PATH = "/content/gdrive/My Drive/hw5_data/Flickr8k_Dataset"
```

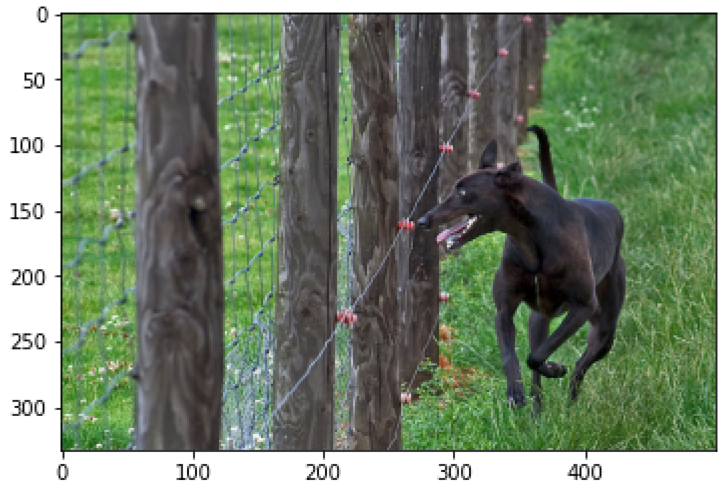
We can use PIL to open the image and matplotlib to display it.

```
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 0x7fd9245836d8>
```



We are going to use an off-the-shelf pre-trained image encoder, the Inception V3 network. The model i
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 archit
> Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).
> foundation.org/openaccess/content_cvpr_2016/html/Szegedy_Rethinking_the_Inception_CVPR_

The model requires that input images are presented as 299x299 pixels, with 3 color channels (RGB). T
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.

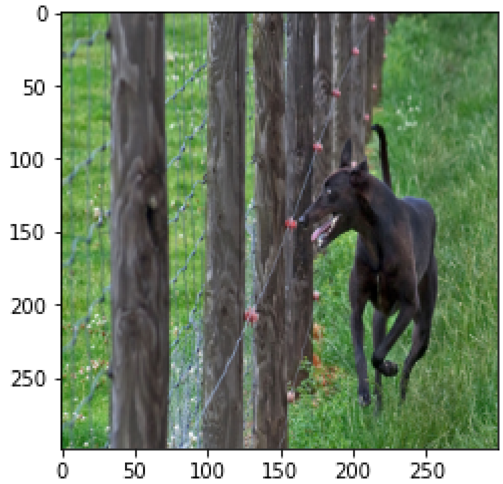
```
np.asarray(image)
```

```
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 0x7fd9245f6668>
```




```
new_image.shape
```

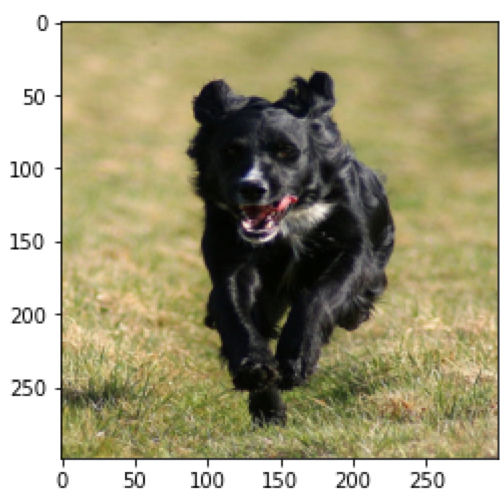
```
(299, 299, 3)
```

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 0x7fd91e8d32e8>
```



Next, we load the pre-trained Inception model.

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

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

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

This is a prediction model,so the output is typically a softmax-activated vector representing 1000 pos: interested in an encoded representation of the image we are just going to use the second-to-last layer 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 laye

```
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 moc
```

Let's try the encoder.

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

```
encoded_image
```

```
array([[0.638066  , 0.48872963, 0.05526248, ..., 0.64255667, 0.29595223,
        0.49004397]], dtype=float32)
```

```
encoded_image.shape
```

```
(1, 2048)
```

TODO: We will need to create encodings for all images and store them in one big matrix (one for each 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 gener later when we train the LSTM. If you are unfamiliar with generators, take a look at this page: https://wi

Write the following generator function, which should return one image at a time. `img_list` is a list of 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:
        yield get_image(image_name).reshape(1,299,299,3)
```

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

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

```
6000/6000 [==============================] - 2825s 471ms/step
```

```
enc_train[11]
```

```
array([0.2681858 , 1.0321676 , 0.5851618 , ..., 1.2316744 , 0.17969292,
       0.2240531 ], dtype=float32)
```

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

```
1000/1000 [==============================] - 483s 483ms/step
```

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

```
1000/1000 [==============================] - 489s 489ms/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)
```

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 retu
Take a look at the file `Flickr8k.token.txt` for the format of the input file. The keys of the dictionary
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 c
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,'r') as image_list_f:
        for line in image_list_f:
            temp_list = line.strip().split("\t")
            caption_temp_list = []
            caption_temp_list.append('<START>')
            for token in temp_list[1].split(" "):
                caption_temp_list.append(token.lower())
            caption_temp_list.append('<END>')
            image_descriptions[temp_list[0].split("#")[0]].append(caption_temp_list)

    return image_descriptions
```

```
descriptions = read_image_descriptions("/content/gdrive/My Drive/"+my_data_dir+"/Flick

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

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

Running the previous cell should print:

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

Creating Word Indices

Next, we need to create a lookup table from the **training** data mapping words to integer indices, so we using numeric representations. **TODO** create the dictionaries id_to_word and word_to_id, which shoul 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 wa will always get the same dictionaries.

```
# you can reload train_list again or not
train_list = load_image_list('/content/gdrive/My Drive/'+my_data_dir+'/Flickr_8k.trair
# to do: get dict: id_to_word, and word_to_id
# the set that contains all tokens in train data
token_set = set()
for image_name in train_list:
    caption_list = descriptions[image_name]
    for caption in caption_list:
        for token in caption:
            token_set.add(token)

word_list = list(token_set)
word_list.sort()

id_to_word = defaultdict(None)

word_to_id = defaultdict(int)
for i in range(len(word_list)):
    id_to_word[i] = word_list[i]
```

```
    word_to_id[word_list[i]] = i
```

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

```
1985
```

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

```
'dog'
```

Note that we do not need an UNK word token because we are generating. The generated text will only

Part III Basic Decoder Model (24 pts)

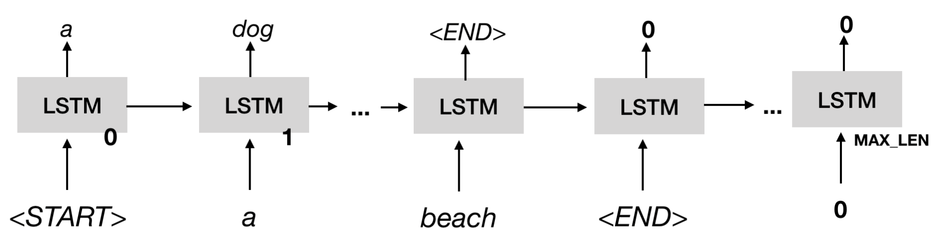
For now, we will just train a model for text generation without conditioning the generator on the image

There are different ways to do this and our approach will be slightly different from the generator discu

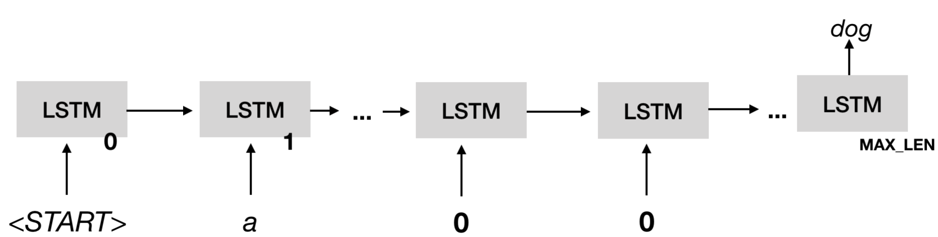
The core idea here is that the Keras recurrent layers (including LSTM) create an "unrolled" RNN. Each t
unit, but the weights for these units are shared. We are going to use the constant MAX_LEN to refer to
which turns out to be 40 words in this data set (including START and END).

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

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



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



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

`['<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 seq predicts the output. Also note the `return_sequence=False` parameter, which causes the LSTM to re per state.

Note also that we use an embedding layer for the input words. The weights are shared between all uni 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=['accuracy

model.summary()
```

```
Model: "model_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         (None, 40)                0
_________________________________________________________________
embedding_5 (Embedding)      (None, 40, 300)           2312100
_________________________________________________________________
bidirectional_5 (Bidirection (None, 1024)              3330048
_________________________________________________________________
dense_7 (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 vectoı an integer representing a word (according to the `word_to_id` dictionary). If the input sequence is shc should be padded with 0.

For each input example, the model returns a softmax activated vector (a probability distribution) over

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 tra will produce up to MAX_LEN input/output pairs, one for each word), it is better to produce the training generator (recall the image generator in part I).

Write the function `text_training_generator` below, that takes as a paramater the batch_size and r is a `(batch_size, MAX_LEN)` ndarray of partial input sequences, `output` contains the next words pı 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 *tr* a list of training images. A batch may contain input/output examples extracted from different descrip

You can just refer back to the variables you have defined above, including `descriptions`, `train_lis`

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

```
def text_training_generator(batch_size=128):
    # ...
    batch_number = 0
    batch_inputs = np.zeros([batch_size, MAX_LEN])
    batch_outputs = np.zeros([batch_size, vocab_size])
    while True:
        for image in train_list:
            for caption in descriptions[image]:
                # for create input and output pairs
                # include punctuations
                temp_array = np.zeros([1,MAX_LEN])
                for i in range(len(caption)-1):
                    temp_array[0,i] = word_to_id[caption[i]]

                    batch_inputs[batch_number][0:i+1] = temp_array[0,0:i+1]
                    batch_outputs[batch_number][word_to_id[caption[i+1]]] = 1
                    batch_number += 1

                    if(batch_number == 128):
                        yield (batch_inputs, batch_outputs)
```

```
                batch_number = 0
                batch_inputs = np.zeros([batch_size, MAX_LEN])
                batch_outputs = np.zeros([batch_size, vocab_size])
```

Training the Model

We will use the `fit_generator` method of the model to train the model. fit_generator needs to know epoch.

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

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

```
model.fit_generator(generator, steps_per_epoch=steps, verbose=True, epochs=10)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Epoch 1/10
1875/1875 [==============================] - 270s 144ms/step - loss: 4.2656 - acc
Epoch 2/10
1875/1875 [==============================] - 265s 141ms/step - loss: 3.6959 - acc
Epoch 3/10
1875/1875 [==============================] - 265s 142ms/step - loss: 3.5193 - acc
Epoch 4/10
1875/1875 [==============================] - 267s 143ms/step - loss: 3.4149 - acc
Epoch 5/10
1875/1875 [==============================] - 268s 143ms/step - loss: 3.3752 - acc
Epoch 6/10
1875/1875 [==============================] - 267s 142ms/step - loss: 3.3006 - acc
Epoch 7/10
1875/1875 [==============================] - 269s 143ms/step - loss: 3.2670 - acc
Epoch 8/10
1875/1875 [==============================] - 267s 142ms/step - loss: 3.2589 - acc
Epoch 9/10
1875/1875 [==============================] - 263s 140ms/step - loss: 3.2697 - acc
Epoch 10/10
1875/1875 [==============================] - 268s 143ms/step - loss: 3.2507 - acc
<keras.callbacks.callbacks.History at 0x7fd925e66320>
```

```
# save model for the above model
# another model saved in the name of"model2.h5"
# basic decoder model
model.save_weights("gdrive/My Drive/"+my_data_dir+"/outputs/model.h5")
```

```
model.load_weights("gdrive/My Drive/"+my_data_dir+"/outputs/model.h5")
```

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 t
append the word to the sequence and then continue until `"<END>"` is predicted or the sequence reach

```
def decoder():
    # ...
    result = []
    result.append('<START>')
    count = 0
    inputs = np.zeros([1, MAX_LEN])
    while count < 39:
        # predict next word
        inputs[0, count] = word_to_id[result[count]]
        temp_output = model.predict(inputs)
        predicted_word = id_to_word[np.argmax(temp_output)]
        result.append(predicted_word)
        count += 1
        if predicted_word == "<END>":
            break
    return result
```

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

```
['<START>', 'a', 'man', 'and', 'woman', 'are', 'standing', 'on', 'a', 'bench', '.
```

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

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

```
def sample_decoder():
    result = []
    result.append('<START>')
    count = 0
    inputs = np.zeros([1, MAX_LEN])
    while count < 39:
        # predict next word
        inputs[0, count] = word_to_id[result[count]]
        temp_output = model.predict(inputs)
```

```
        #temp_norm = temp_output.reshape(vocab_size) / np.sum(temp_output.reshape(voca
        # create the following three columns in order to avoid value error: "sum(pvals
        temp_dis = np.asarray(temp_output).astype('float64')
        temp_dis = temp_dis.reshape(vocab_size) / np.sum(temp_dis.reshape(vocab_size))
        sample_distribution = np.random.multinomial(vocab_size, temp_dis)
        predicted_word = id_to_word[np.argmax(sample_distribution)]
        result.append(predicted_word)
        count += 1
        if predicted_word == "<END>":
            break

    return result
```

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

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

```
['<START>', 'a', 'man', 'and', 'woman', 'are', 'standing', 'on', 'a', 'bench', '.
['<START>', 'a', 'man', 'and', 'woman', 'are', 'standing', 'on', 'a', 'bench', '.
['<START>', 'a', 'man', 'and', 'woman', 'are', 'sitting', 'on', 'a', 'bench', '.'
['<START>', 'a', 'man', 'and', 'woman', 'are', 'standing', 'on', 'a', 'bench', '.
['<START>', 'a', 'man', 'and', 'woman', 'are', 'sitting', 'on', 'a', 'bench', '.'
['<START>', 'a', 'man', 'and', 'woman', 'are', 'standing', 'on', 'a', 'bench', '.
['<START>', 'a', 'man', 'and', 'woman', 'are', 'standing', 'on', 'a', 'bench', '.
['<START>', 'a', 'man', 'and', 'woman', 'are', 'sitting', 'on', 'a', 'bench', '.'
['<START>', 'a', 'man', 'and', 'woman', 'are', 'standing', 'on', 'a', 'bench', '.
['<START>', 'a', 'man', 'and', 'woman', 'are', 'sitting', 'on', 'a', 'bench', '.'
```

Part IV - 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

We will project the 2048-dimensional image encoding to a 300-dimensional hidden layer. We then con
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=['accuracy

model.summary()
```

```
Model: "model_8"
________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
========================================================================================
input_10 (InputLayer)           (None, 2048)         0
________________________________________________________________________________________
dense_8 (Dense)                 (None, 300)          614700      input_10[0][0]
________________________________________________________________________________________
input_11 (InputLayer)           (None, 40)           0
________________________________________________________________________________________
repeat_vector_3 (RepeatVector)  (None, 40, 300)      0           dense_8[0][0]
________________________________________________________________________________________
embedding_6 (Embedding)         (None, 40, 300)      2312100     input_11[0][0]
________________________________________________________________________________________
concatenate_7 (Concatenate)     (None, 40, 600)      0           repeat_vector_3[
                                                                 embedding_6[0][0
________________________________________________________________________________________
bidirectional_6 (Bidirectional) (None, 512)          1755136     concatenate_7[0]
________________________________________________________________________________________
dense_9 (Dense)                 (None, 7707)         3953691     bidirectional_6[
========================================================================================
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 gener
following format: `([image_inputs, text_inputs], next_words)`. Where each element is an ndarr

You need to find the image encoding that belongs to each image. You can use the fact that the index
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")
```

```
enc_dev = np.load('gdrive/My Drive/'+my_data_dir+'/outputs/encoded_images_dev.npy')

def training_generator(batch_size=128):
    #...
    # ...
    batch_number = 0
    batch_text_inputs = np.zeros([batch_size, MAX_LEN])
    batch_image_inputs = np.zeros([batch_size, 2048])
    batch_outputs = np.zeros([batch_size, vocab_size])

    while True:
        for index in range(len(train_list)):
            batch_image_inputs[batch_number, 0:2048] = enc_train[index,:]
            for caption in descriptions[train_list[index]]:
                # for create input and output pairs
                # include punctuations
                temp_array = np.zeros([1,MAX_LEN])
                for j in range(len(caption)-1):
                    temp_array[0,j] = word_to_id[caption[j]]

                    batch_text_inputs[batch_number,0:j+1] = temp_array[0,0:j+1]
                    batch_image_inputs[batch_number,:] = enc_train[index, :]
                    batch_outputs[batch_number, word_to_id[caption[j+1]]] = 1
                    batch_number += 1

                    if(batch_number == 128):
                        yield ([batch_image_inputs,batch_text_inputs], batch_outputs)
                        batch_number = 0
                        batch_text_inputs = np.zeros([batch_size, MAX_LEN])
                        batch_image_inputs = np.zeros([batch_size, 2048])
                        batch_image_inputs[batch_number, 0:2048] = enc_train[index,:]
                        batch_outputs = np.zeros([batch_size, vocab_size])
```

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)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Epoch 1/20
1875/1875 [==============================] - 277s 148ms/step - loss: 4.4097 - acc
Epoch 2/20
1875/1875 [==============================] - 276s 147ms/step - loss: 3.6382 - acc
Epoch 3/20
1875/1875 [==============================] - 269s 143ms/step - loss: 3.4348 - acc
Epoch 4/20
1875/1875 [==============================] - 267s 143ms/step - loss: 3.3248 - acc
Epoch 5/20
1875/1875 [==============================] - 266s 142ms/step - loss: 3.2622 - acc
Epoch 6/20
1875/1875 [==============================] - 267s 143ms/step - loss: 3.1897 - acc
Epoch 7/20
1875/1875 [==============================] - 268s 143ms/step - loss: 3.1993 - acc
Epoch 8/20
1875/1875 [==============================] - 267s 142ms/step - loss: 3.1833 - acc
Epoch 9/20
1875/1875 [==============================] - 266s 142ms/step - loss: 3.1938 - acc
Epoch 10/20
1875/1875 [==============================] - 267s 143ms/step - loss: 3.1876 - acc
Epoch 11/20
1875/1875 [==============================] - 271s 144ms/step - loss: 3.2022 - acc
Epoch 12/20
1875/1875 [==============================] - 267s 142ms/step - loss: 3.2019 - acc
Epoch 13/20
1875/1875 [==============================] - 267s 142ms/step - loss: 3.2185 - acc
Epoch 14/20
1875/1875 [==============================] - 267s 142ms/step - loss: 3.1897 - acc
Epoch 15/20
1875/1875 [==============================] - 267s 142ms/step - loss: 3.1933 - acc
Epoch 16/20
1875/1875 [==============================] - 265s 141ms/step - loss: 3.1925 - acc
Epoch 17/20
1875/1875 [==============================] - 266s 142ms/step - loss: 3.1913 - acc
Epoch 18/20
1875/1875 [==============================] - 265s 141ms/step - loss: 3.1949 - acc
Epoch 19/20
1875/1875 [==============================] - 264s 141ms/step - loss: 3.2332 - acc
Epoch 20/20
1875/1875 [==============================] - 267s 142ms/step - loss: 3.2485 - acc
<keras.callbacks.callbacks.History at 0x7fd9251b7a58>
```

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

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

```
# model2.h5: the captioning model
model.save_weights("gdrive/My Drive/"+my_data_dir+"/outputs/model2.h5")
```

to load the model:

```
# model2.h5: the captioning model
model.load_weights("gdrive/My Drive/"+my_data_dir+"/outputs/model2.h5")
```

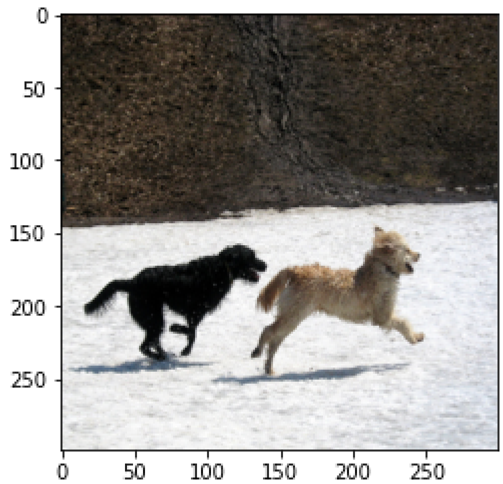
TODO: Now we are ready to actually generate image captions using the trained model. Modify the sim
only generator, so that it takes an encoded image (a vector of length 2048) as input, and returns a seq

```
def image_decoder(enc_image):
    # ...
    result = []
    result.append('<START>')
    count = 0
    text_inputs = np.zeros([1, MAX_LEN])
    while count < 39:
        # predict next word
        text_inputs[0, count] = word_to_id[result[count]]
        temp_output = model.predict([enc_image.reshape(1,2048),text_inputs])
        predicted_word = id_to_word[np.argmax(temp_output)]
        result.append(predicted_word)
        count += 1

        if predicted_word == "<END>":
           break
    return result
```

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

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

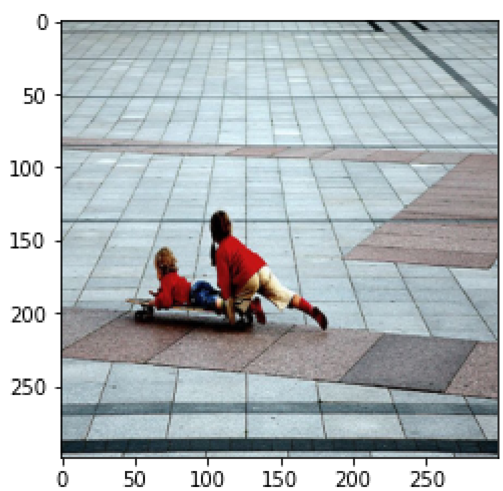
```
['<START>',
 'a',
 'dog',
 'with',
 'a',
 'stick',
 'in',
 'its',
 'mouth',
 '.',
 '<END>']
```



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

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

```
['<START>', 'a', 'skateboarder', 'in', 'the', 'air', '.', '<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 poir produce good descriptions for the dev images.

Part V - Beam Search Decoder (24 pts)

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

Then, for each sequence, compute the n most likely successor words. Append the word to produce n 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.

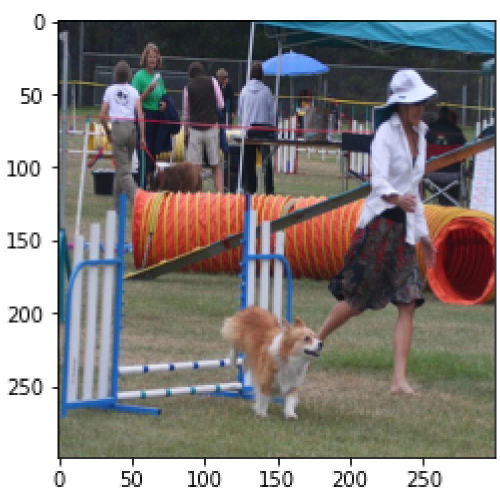
Note that you cannot use the occurence of the `"<END>"` tag to terminate generation, because the tag different entries in the beam.

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

```
def img_beam_decoder(n, image_enc):
    # ...
    possible_sequences_n = []
    possible_sequences_n_double = []
    # initialization, add the first n possible sequence based on the first words is "<
    # and the possibility of the first
    ini_sequence = []
    ini_sequence.append('<START>')
    possible_sequences_n = max_n_sequence(n, 1.0, ini_sequence, image_enc )

    count = 1
    while count < 39:
        possible_sequences_n_double = []
        for pair in possible_sequences_n:
            for max_n_pair in max_n_sequence(n, pair[0], pair[1], image_enc):
                possible_sequences_n_double.append(max_n_pair)
        # find n sequences with the largest possibility
        possible_sequences_n_double.sort(key=lambda x:x[1])
        possible_sequences_n = possible_sequences_n_double[-n:]
        count += 1

    # finally find the sequence with the sequence with the largest probability
    possible_sequences_n.sort(key=lambda x:x[1])
    return possible_sequences_n[-1][1]
```

```
def max_n_sequence(n, possibility, sequence, enc_image):
    # contains a list of tuples(possibility, sequence)
    max_n_sequence = []
    text_input = sequence_to_text_inputs(sequence)
    output = model.predict([enc_image.reshape(1,2048),text_input])
    indexes = np.argpartition(output.reshape(vocab_size), -n)[-n:]

    for i in indexes:
        temp_sequence = []
        temp_sequence = sequence[:]
        temp_sequence.append(id_to_word[i])
        max_n_sequence.append((output.reshape(vocab_size)[i]*possibility, temp_sequenc

    return max_n_sequence


def sequence_to_text_inputs(sequence):
    # sequence is a list
    text_input = np.zeros([1, MAX_LEN])
    for i in range(len(sequence)):
        text_input[0,i] = word_to_id[sequence[i]]
    return text_input


#img_beam_decoder(3, dev_list[1])
plt.imshow(get_image(dev_list[2]))
img_beam_decoder(3, enc_dev[2])
```

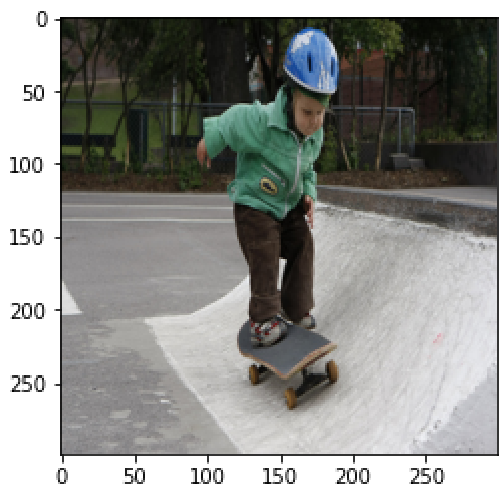
```
['<START>',
 'two',
 'dogs',
 'running',
 'through',
 'the',
 'snow',
 'with',
 'two',
 'dogs',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'grass',
 'in',
 'the',
 'the',
 'ground',
 'in',
 'the']
```



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

```
# test 5 development images
# test 1
# greedy output
plt.imshow(get_image(dev_list[8]))
image_decoder(enc_dev[8])
```

```
['<START>',
 'a',
 'skateboarder',
 'jumps',
 'over',
 'a',
 'wooden',
 'fence',
 '.',
 '<END>']
```



```
# beam search with n = 3
img_beam_decoder(3, enc_dev[8])
```

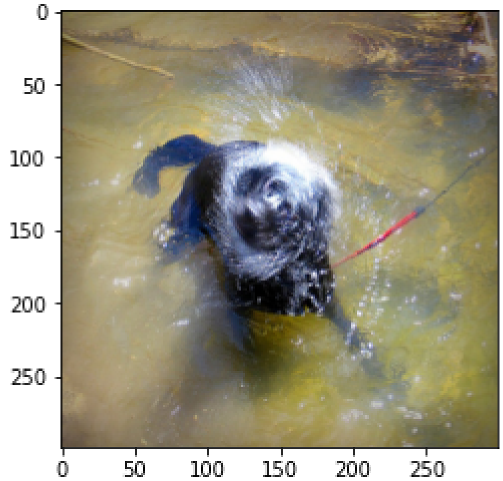
```
['<START>',
 'two',
 'snowboarder',
 'play',
 'with',
 'the',
 'water',
 'on',
 'the',
 'side',
 'of',
 'the',
 'street',
 'in',
 'the',
 'woods',
 'in',
 'the',
 'woods',
 'in',
 'the',
 'blue',
 'woods',
 'on',
 'the',
 'and',
 'the',
 'background',
 'on',
 'the',
 'blue',
 'city',
 'in',
 'the',
 'on',
 'the',
 'white',
 'surfer',
 'on',
 'the']
```

```
# beam search with n = 5
img_beam_decoder(5, enc_dev[8])
```

```
['<START>',
 'two',
 'young',
 'men',
 'skateboarding',
 'on',
 'top',
 'on',
 'top',
 'with',
 'two',
 'other',
 'people',
 'on',
 'the',
 'side',
 'on',
 'the',
 'sidewalk',
 'with',
 'trees',
 'on',
 'their',
 'side',
 'on',
 'the',
 'wall',
 'with',
 'trees',
 'on',
 'the',
 'side',
 'on',
 'the',
 'side',
 'on',
 'the',
 'sidewalk',
 'on',
 'the']
```

```
# test 2
# greedy output
plt.imshow(get_image(dev_list[12]))
image_decoder(enc_dev[12])
```

```
['<START>', 'a', 'dog', 'in', 'the', 'water', '.', '<END>']
```



```
# beam search with n = 3
img_beam_decoder(3, enc_dev[12])
```

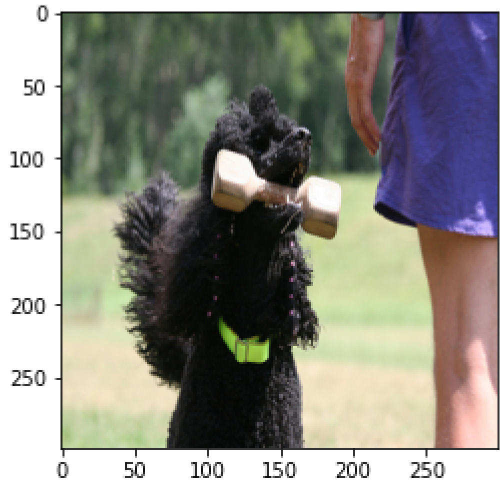
```
['<START>',
 'two',
 'dogs',
 'running',
 'through',
 'water',
 'with',
 'the',
 'waves',
 'in',
 'the',
 'ocean',
 'and',
 'the',
 'water',
 'in',
 'water',
 'and',
 'the',
 'water',
 'in',
 'water',
 'in',
 'water',
 'in',
 'water',
 'in',
 'water',
 'in',
 'water',
 'in',
 'water',
 'in',
 'water',
 'in',
 'the',
 'water',
 'in',
 'the',
 'distance']
```

```
# beam search with n = 5
img_beam_decoder(5, enc_dev[12])
```

```
['<START>',
 'two',
 'white',
 'dogs',
 'with',
 'water',
 'swimming',
 'through',
 'water',
 'with',
 'the',
 'water',
 'in',
 'the',
 'water',
 'with',
 'the',
 'water',
 'in',
 'the',
 'water',
 'with',
 'the',
 'water',
 'on',
 'the',
 'water',
 'with',
 'the',
 'water',
 'on',
 'the',
 'water',
 'with',
 'two',
 'other',
 'people',
 'on',
 'the',
 'surfer']
```

```
# test 3
plt.imshow(get_image(dev_list[28]))
image_decoder(enc_dev[28])
```

```
['<START>',
 'a',
 'dog',
 'with',
 'a',
 'stick',
 'in',
 'its',
 'mouth',
 '.',
 '<END>']
```




```
# beam search with n = 3
img_beam_decoder(3, enc_dev[28])
```

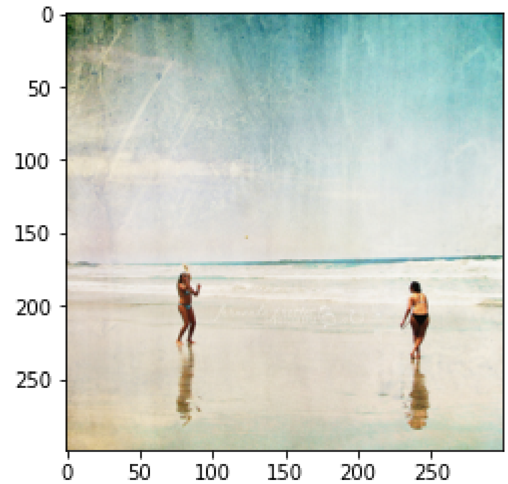
```
['<START>',
 'two',
 'dogs',
 'running',
 'through',
 'the',
 'water',
 'with',
 'two',
 'dogs',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'snow',
 'in',
 'the',
 'the',
 'grass',
 'in',
 'the',
 'the',
 'the',
 'the',
 'the',
 'the',
 'side']
```

```
# beam search with n = 5
img_beam_decoder(5, enc_dev[28])
```

```
['<START>',
 'two',
 'tan',
 'dogs',
 'running',
 'with',
 'two',
 'grey',
 'each',
 'other',
 'with',
 'two',
 'other',
 'people',
 'on',
 'the',
 'side',
 'on',
 'the',
 'snow',
 'with',
 'two',
 'trees',
 'in',
 'the',
 'the',
 'mouth',
 'of',
 'them',
 'in',
 'the',
 'snow',
 'with',
 'two',
 'white',
 'snow',
 'in',
 'the',
 'the',
 'ground']
```

```
# test 4
# greedy output
plt.imshow(get_image(dev_list[32]))
image_decoder(enc_dev[32])
```

```
['<START>',
 'two',
 'children',
 'are',
 'playing',
 'on',
 'a',
 'beach',
 '.',
 '<END>']
```



```
# beam search with n = 3
img_beam_decoder(3, enc_dev[32])
```

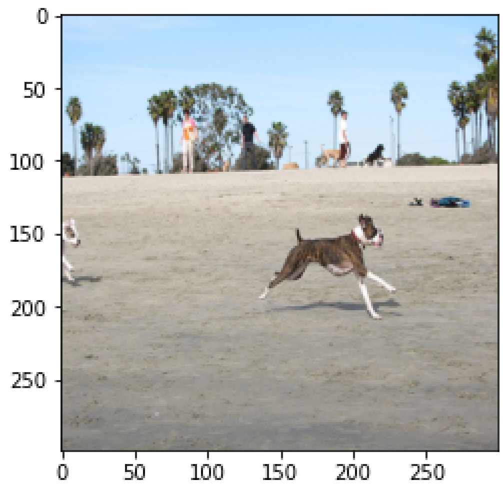
```
['<START>',
 'two',
 'kids',
 'run',
 'through',
 'water',
 'with',
 'two',
 'dogs',
 'in',
 'water',
 'in',
 'the',
 'grass',
 'and',
 'one',
 'is',
 'watching',
 'the',
 'side',
 'in',
 'water',
 'in',
 'the',
 'park',
 'in',
 'the',
 'other',
 'and',
 'the',
 'side',
 'the',
 'side',
 'in',
 'the',
 'other',
 'the',
 'side',
 'in',
 'surfer']
```

```
# beam search with n = 5
img_beam_decoder(5, enc_dev[32])
```

```
['<START>',
 'two',
 'people',
 'stand',
 'outside',
 'on',
 'water',
 'with',
 'two',
 'trees',
 'on',
 'the',
 'side',
 'of',
 'them',
 'with',
 'them',
 'on',
 'the',
 'side',
 'of',
 'them',
 'in',
 'water',
 'with',
 'water',
 'on',
 'the',
 'side',
 'of',
 'them',
 'with',
 'water',
 'on',
 'the',
 'side',
 'the',
 'side',
 'with',
 'they']
```

```
# test 5
# greedy output
plt.imshow(get_image(dev_list[66]))
image_decoder(enc_dev[66])
```

```
['<START>',
 'a',
 'dog',
 'and',
 'a',
 'dog',
 'run',
 'in',
 'the',
 'grass',
 '.',
 '<END>']
```



```
# beam search with n = 3
img_beam_decoder(3, enc_dev[66])
```

```
['<START>',
 'two',
 'dogs',
 'running',
 'through',
 'the',
 'green',
 'snow',
 'with',
 'the',
 'trees',
 'in',
 'the',
 'mouth',
 'and',
 'the',
 'other',
 'is',
 'running',
 'in',
 'the',
 'park',
 'in',
 'the',
 'park',
 'in',
 'the',
 'park',
 'in',
 'the',
 'park',
 'in',
 'the',
 'park',
 'in',
 'the',
 'park',
 'in',
 'the',
 'day']
```

```
# beam search with n = 5
img_beam_decoder(5, enc_dev[66])
```

```
['<START>',
 'two',
 'tan',
 'dogs',
 'running',
 'with',
 'two',
 'women',
 'with',
 'two',
 'running',
 'on',
 'the',
 'ground',
 'with',
 'two',
 'trees',
 'in',
 'the',
 'other',
 'in',
 'the',
 'other',
 'one',
 'one',
 'is',
 'running',
 'on',
 'the',
 'side',
 'on',
 'the',
 'other',
 'side',
 'the',
 'side',
 'the',
 'side',
 'with',
 'other']
```