Image Captioning with LSTM

This is a partial implementation of "Show and Tell: A Neural Image Caption Generator" (http://arxiv.org/abs/1411.4555 (http://arxiv.org/abs/1411.4555))

This example consists of three parts:

1. COCO Preprocessing - prepare the dataset by precomputing image representations using GoogLeNet
2. COCO RNN Training - train a network to predict image captions
3. COCO Caption Generation - use the trained network to caption new images

In [1]:
```
import sklearn
import numpy as np
import lasagne
import skimage.transform

from lasagne.utils import floatX

import theano
import theano.tensor as T

import matplotlib.pyplot as plt
%matplotlib inline

import json
import pickle
```

```
Using gpu device 0: Tesla K80 (CNMeM is enabled with initial size: 91.0% of m
emory, cuDNN 5105)
```

In [2]:
```
import googlenet
```

In [3]:
```
cnn_layers = googlenet.build_model()
cnn_input_var = cnn_layers['input'].input_var
cnn_feature_layer = cnn_layers['loss3/classifier']
cnn_output_layer = cnn_layers['prob']

get_cnn_features = theano.function([cnn_input_var],
lasagne.layers.get_output(cnn_feature_layer))
```

In [4]:
```
model_param_values = pickle.load(open('blvc_googlenet.pkl'))['param values']
lasagne.layers.set_all_param_values(cnn_output_layer, model_param_values)
```

In [5]:

```
MEAN_VALUES = np.array([104, 117, 123]).reshape((3,1,1))

def prep_image(im):
    if len(im.shape) == 2:
        im = im[:, :, np.newaxis]
        im = np.repeat(im, 3, axis=2)
    # Resize so smallest dim = 224, preserving aspect ratio
    h, w, _ = im.shape
    if h < w:
        im = skimage.transform.resize(im, (224, w*224/h), preserve_range=True)
    else:
        im = skimage.transform.resize(im, (h*224/w, 224), preserve_range=True)

    # Central crop to 224x224
    h, w, _ = im.shape
    im = im[h//2-112:h//2+112, w//2-112:w//2+112]

    rawim = np.copy(im).astype('uint8')

    # Shuffle axes to c01
    im = np.swapaxes(np.swapaxes(im, 1, 2), 0, 1)

    # Convert to BGR
    im = im[::-1, :, :]

    im = im - MEAN_VALUES
    return rawim, floatX(im[np.newaxis])
```

In [6]:

```
SEQUENCE_LENGTH = 32
MAX_SENTENCE_LENGTH = SEQUENCE_LENGTH - 3 # 1 for image, 1 for start token, 1
 for end token
BATCH_SIZE = 1
CNN_FEATURE_SIZE = 1000
EMBEDDING_SIZE = 256

d = pickle.load(open('lstm_coco_trained.pkl'))
vocab = d['vocab']
word_to_index = d['word_to_index']
index_to_word = d['index_to_word']
```

In [7]:

```
l_input_sentence = lasagne.layers.InputLayer((BATCH_SIZE, SEQUENCE_LENGTH -
1))
l_sentence_embedding = lasagne.layers.EmbeddingLayer(l_input_sentence,
                                                     input_size=len(vocab),
                                                     output_size=EMBEDDING_SIZ
E,
                                                    )

l_input_cnn = lasagne.layers.InputLayer((BATCH_SIZE, CNN_FEATURE_SIZE))
l_cnn_embedding = lasagne.layers.DenseLayer(l_input_cnn, num_units=EMBEDDING_S
IZE,
                                            nonlinearity=lasagne.nonlinearitie
s.identity)

l_cnn_embedding = lasagne.layers.ReshapeLayer(l_cnn_embedding, ([0], 1, [1]))

l_rnn_input = lasagne.layers.ConcatLayer([l_cnn_embedding, l_sentence_embeddin
g])
l_dropout_input = lasagne.layers.DropoutLayer(l_rnn_input, p=0.5)
l_lstm = lasagne.layers.LSTMLayer(l_dropout_input,
                                  num_units=EMBEDDING_SIZE,
                                  unroll_scan=True,
                                  grad_clipping=5.)
l_dropout_output = lasagne.layers.DropoutLayer(l_lstm, p=0.5)
l_shp = lasagne.layers.ReshapeLayer(l_dropout_output, (-1, EMBEDDING_SIZE))
l_decoder = lasagne.layers.DenseLayer(l_shp, num_units=len(vocab), nonlinearit
y=lasagne.nonlinearities.softmax)

l_out = lasagne.layers.ReshapeLayer(l_decoder, (BATCH_SIZE, SEQUENCE_LENGTH, l
en(vocab)))
```

In [8]:

```
lasagne.layers.set_all_param_values(l_out, d['param values'])
```

In [9]:

```
x_cnn_sym = T.matrix()
x_sentence_sym = T.imatrix()

output = lasagne.layers.get_output(l_out, {
                l_input_sentence: x_sentence_sym,
                l_input_cnn: x_cnn_sym
                })

f = theano.function([x_cnn_sym, x_sentence_sym], output)
```

In [10]:

```
def predict(x_cnn):
    x_sentence = np.zeros((BATCH_SIZE, SEQUENCE_LENGTH - 1), dtype='int32')
    words = []
    i = 0
    while True:
        i += 1
        p0 = f(x_cnn, x_sentence)
        pa = p0.argmax(-1)
        tok = pa[0][i]
        word = index_to_word[tok]
        if word == '#END#' or i >= SEQUENCE_LENGTH - 1:
            return ' '.join(words)
        else:
            x_sentence[0][i] = tok
            if word != '#START#':
                words.append(word)
```

Grab a random photo (not from ImageNet or MSCOCO as far as I know)

In [11]:

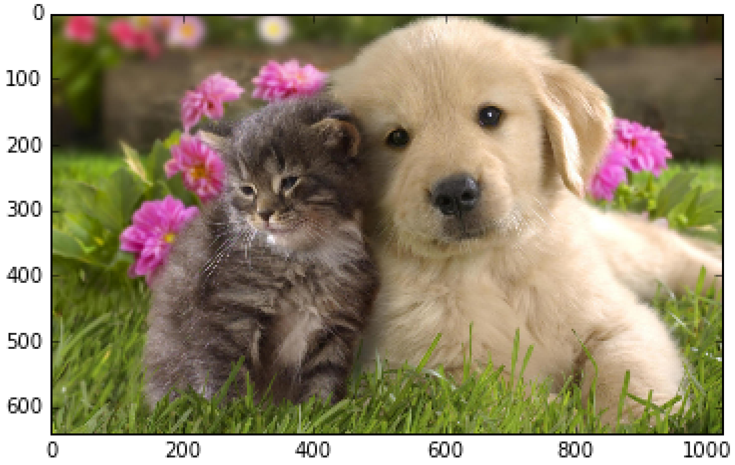
```
im1 = plt.imread('Dog-and-Cat-Wallpaper-teddybear64-16834786-1280-800-1024x64
0.jpg')
```

In [12]:

```
plt.imshow(im1)
```

Out[12]:

```
<matplotlib.image.AxesImage at 0x7fc716983dd0>
```

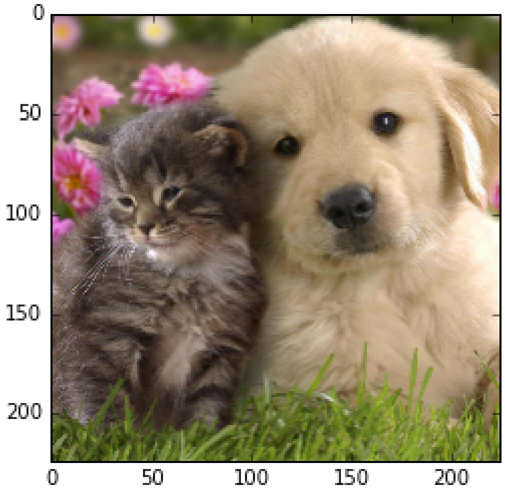


In [13]:

```
rawim, cnn_im = prep_image(im1)
```

```
In [14]: plt.imshow(rawim)
```

```
Out[14]: <matplotlib.image.AxesImage at 0x7fc7108110d0>
```



```
In [15]: p = get_cnn_features(cnn_im)
         CLASSES = pickle.load(open('blvc_googlenet.pkl'))['synset words']
         print(CLASSES[p.argmax()])
```

```
golden retriever
```

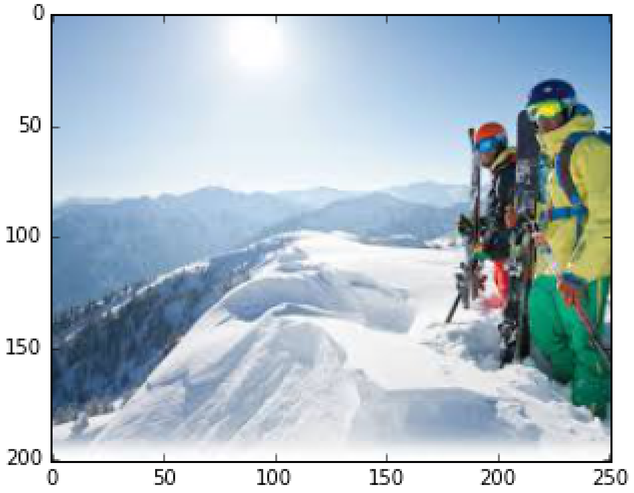
```
In [16]: x_cnn = get_cnn_features(cnn_im)
```

```
In [17]: # Sample some predictions
         for _ in range(5):
             print(predict(x_cnn))
```

```
a brown and white dog sitting on a bench
a brown dog is sitting on a bed
a cat is laying on a bed with a dog
a dog laying on a bench in front of a car
a dog is laying on a bed with a dog
```

```
In [18]: im2 = plt.imread('image2.jpeg')
In [19]: plt.imshow(im2)
```

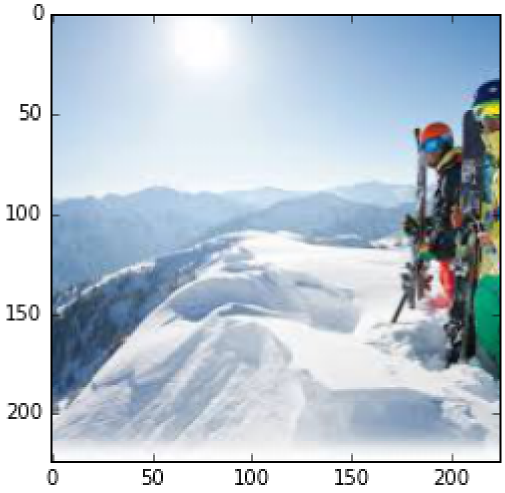
```
Out[19]: <matplotlib.image.AxesImage at 0x7fc7108c6490>
```



```
In [20]: rawim, cnn_im = prep_image(im2)
```

```
In [21]: plt.imshow(rawim)
```

```
Out[21]: <matplotlib.image.AxesImage at 0x7fc70ef3a090>
```



```
In [22]: p = get_cnn_features(cnn_im)
         CLASSES = pickle.load(open('blvc_googlenet.pkl'))['synset words']
         print(CLASSES[p.argmax()])
```

```
alp
```

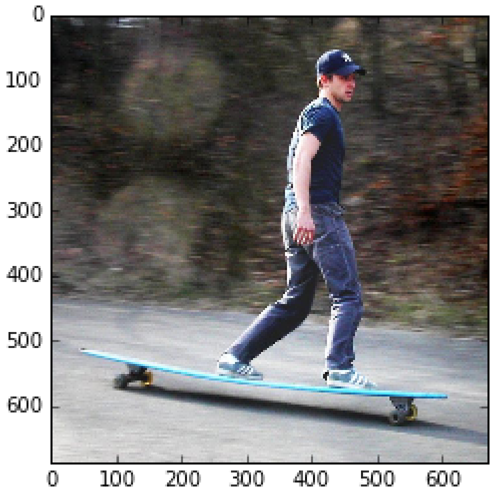
```
In [23]: x_cnn = get_cnn_features(cnn_im)
```

In [24]:
```
# Sample some predictions
for _ in range(5):
    print(predict(x_cnn))
```

```
a man is standing on a snow covered slope
a man riding a snowboard down a snowy hill
a man riding a snowboard down a slope
a man in a snow jacket riding a snowboard down the snow
a man in a blue ski jacket skiing in the snow
```

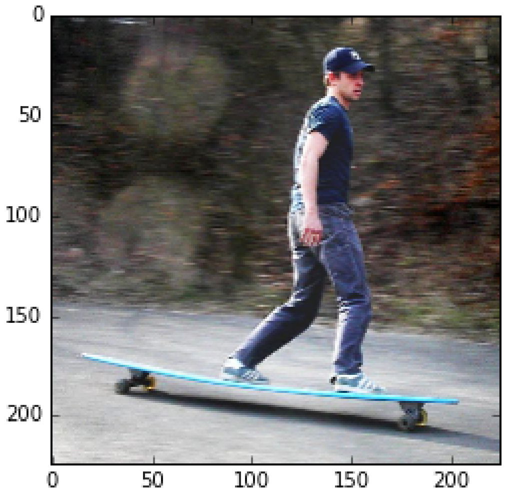
In [25]:
```
im2 = plt.imread('image3.jpg')
plt.imshow(im2)
```

Out[25]:
```
<matplotlib.image.AxesImage at 0x7fc70ee48450>
```



In [26]:
```
rawim, cnn_im = prep_image(im2)
plt.imshow(rawim)
```

Out[26]:
```
<matplotlib.image.AxesImage at 0x7fc70ed91250>
```



In [27]:

```
p = get_cnn_features(cnn_im)

CLASSES = pickle.load(open('blvc_googlenet.pkl'))['synset words']

print(CLASSES[p.argmax()])
```

```
ski
```

In [28]:

```
x_cnn = get_cnn_features(cnn_im)

# Sample some predictions

for _ in range(5):

    print(predict(x_cnn))
```

```
a man in a black jacket skis on a snowy slope
a man in a blue shirt and a surfboard in a snow
a man riding a snowboard on a snowy hill
a man in a black jacket is on a snowboard
a man on a snowboard is standing in the snow
```

In [ ]:

In [ ]: