DataLab Cup 3: Image Caption

Shan-Hung Wu & DataLab
Fall 2018

In [1]:
```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = ""
import tensorflow as tf
import pandas as pd
import numpy as np
import _pickle as cPickle

print("This notebook uses TensorFlow version {}".format(tf.__version__))
```

```
This notebook uses TensorFlow version 1.6.0
```

Task: Image Caption

Given a set of images, your task is to generate suitable sentences to describe each of the images.

You will compete on the modified release of 2014 Microsoft COCO dadtaset, which is the standard testbed for image caption.

- 102739 images for training set, where each images is annotated with 5 captions.
- 20548 images for testing (you must generate 1 caption for each image)

Model: Image Caption

Given an image, in order to be able to generate descriptive sentence for it, our model must meet several requirements:

1. our model should be able to extract high level concepts of images, such as the scene, the background, the color or positions of objects in that iamge
   => better use CNN to extract iamge features.
2. the generated caption must be grammatically correct
   => better use RNN to capture relationships of word sequences.
3. the generated caption must describes the image
   => RNN should learn the correspondence of image and words.
4. use RNN to generate next word based on current words, but the length of caption may vary
   => add special tokens `<ST>` and `<ED>` to each caption, so that our model knows when to start and stop.

Preprocess

Our model requires several preprocessing of inputs. Here we'll only give a quick summary.

Preprocess: Text

Since dealing with raw string is inefficient, we've

- encode each vocabulary in dataset/text/vocab.pkl
- append `<ST>` and `<ED>` to each caption
- represent captions by a sequence of integer IDs
- replace rare words by `<RARE>` token to reduce vocabulary size for more efficient training

By looking up the vocabulary dictionary, we can decode sequence vocabulary IDs back to original caption, as shown in following cell.

- dataset/text/train_enc_cap.csv is a dataframe containing `img_id` and `caption` of training data
- dataset/text/enc_map.pkl is a dictionary mapping word to id
- dataset/text/dec_map.pkl is a dictionary mapping id back to word

In [2]:
```
vocab = cPickle.load(open('dataset/text/vocab.pkl', 'rb'))
print('total {} vocabularies'.format(len(vocab)))
```

```
total 26900 vocabularies
```

In [3]:

```python
def count_vocab_occurance(vocab, df):
    voc_cnt = {v: 0 for v in vocab}
    for img_id, row in df.iterrows():
        for w in row['caption'].split(' '):
            voc_cnt[w] += 1
    return voc_cnt


df_train = pd.read_csv(os.path.join('dataset', 'train.csv'))

print('count vocabulary occurances...')
voc_cnt = count_vocab_occurance(vocab, df_train)

# remove words appear < 50 times
thrhd = 50
x = np.array(list(voc_cnt.values()))
print('{} words appear >= 50 times'.format(np.sum(x[(-x).argsort()] >= thrhd)))
```

```
count vocabulary occurances...
3153 words appear >= 50 times
```

In [4]:

```python
def build_voc_mapping(voc_cnt, thrhd):
    """
    enc_map: voc --encode--> id
    dec_map: id --decode--> voc
    """

    def add(enc_map, dec_map, voc):
        enc_map[voc] = len(dec_map)
        dec_map[len(dec_map)] = voc
        return enc_map, dec_map

    # add <ST>, <ED>, <RARE>
    enc_map, dec_map = {}, {}
    for voc in ['<ST>', '<ED>', '<RARE>']:
        enc_map, dec_map = add(enc_map, dec_map, voc)
    for voc, cnt in voc_cnt.items():
        if cnt < thrhd:  # rare words => <RARE>
            enc_map[voc] = enc_map['<RARE>']
        else:
            enc_map, dec_map = add(enc_map, dec_map, voc)
    return enc_map, dec_map


enc_map, dec_map = build_voc_mapping(voc_cnt, thrhd)
# save enc/decoding map to disk
cPickle.dump(enc_map, open('dataset/text/enc_map.pkl', 'wb'))
cPickle.dump(dec_map, open('dataset/text/dec_map.pkl', 'wb'))
```

In [5]:

```python
def caption_to_ids(enc_map, df):
    img_ids, caps = [], []
    for idx, row in df.iterrows():
        icap = [enc_map[x] for x in row['caption'].split(' ')]
        icap.insert(0, enc_map['<ST>'])
        icap.append(enc_map['<ED>'])
        img_ids.append(row['img_id'])
        caps.append(icap)
    return pd.DataFrame({
              'img_id': img_ids,
              'caption': caps
            }).set_index(['img_id'])


enc_map = cPickle.load(open('dataset/text/enc_map.pkl', 'rb'))
print('[transform captions into sequences of IDs]...')
```

```
df_proc = caption_to_ids(enc_map, df_train)
df_proc.to_csv('dataset/text/train_enc_cap.csv')
```

After preprocessing text, we can load what we need in training and testing.

In [6]:

```
df_cap = pd.read_csv(
    'dataset/text/train_enc_cap.csv')  # a dataframe - 'img_id', 'cpation'
enc_map = cPickle.load(
    open('dataset/text/enc_map.pkl', 'rb'))  # token => id
dec_map = cPickle.load(
    open('dataset/text/dec_map.pkl', 'rb'))  # id => token
vocab_size = len(dec_map)


def decode(dec_map, ids):
    """decode IDs back to origin caption string"""
    return ' '.join([dec_map[x] for x in ids])


print('decoding the encoded captions back...\n')
for idx, row in df_cap.iloc[:8].iterrows():
    print('{}: {}'.format(idx, decode(dec_map, eval(row['caption']))))
```

```
decoding the encoded captions back...

0: <ST> a group of three women sitting at a table sharing a cup of tea <ED>
1: <ST> three women wearing hats at a table together <ED>
2: <ST> three women with hats at a table having a tea party <ED>
3: <ST> several woman dressed up with fancy hats at a tea party <ED>
4: <ST> three women wearing large hats at a fancy tea event <ED>
5: <ST> a twin door refrigerator in a kitchen next to cabinets <ED>
6: <ST> a black refrigerator freezer sitting inside of a kitchen <ED>
7: <ST> black refrigerator in messy kitchen of residential home <ED>
```

Transfer Learning: pretrained word embedding

Since image-caption requires good understanding of word meanings, you can use pretrained word embedding model to do word embedding. Word embedding model can be either fine-tuned or fixed.

Preprocess: Image

Since the raw image takes about 20GB and may take some time to download all of them. It's not included in the released file. But if you'd like to download original image, you can request MS-COCO on-the-fly: MS-COCO

Transfer Learning: pretrained CNN

Our task, image caption, requires good understandings of images, like

- objects appeared in the image
- relative positions of objects
- colors, sizes, etc.

Training a good CNN from scratch is challenging and time-consuming, so we'll use existing pretrained CNN model. The one we've prepared for you is the winner of 2012-ILSVRC model - VGG-16 in pretrained/cnn.py. We use VGG-16 to extract image features and then apply PCA to reduce the dimension of image features. In summary, for each image, we

1. feed the raw image into VGG-16
2. take the output of second last layer
3. apply PCA to reduce dimension to 256

The resulting 256-dimensional image feature is saved as dataset/train_img256.pkl and dataset/test_img256.pkl and the transformed factor in PCA is saved in dataset/U.pkl so that we can process new images for our model.

In [7]:

```
img_train = cPickle.load(open('dataset/train_img256.pkl', 'rb'))
# transform img_dict to dataframe
img_train_df = pd.DataFrame(list(img_train.items()), columns=['img_id', 'img'])
print('Images for training: {}'.format(img_train_df.shape[0]))
```

```
Images for training: 102739
```

Training

We have preprocessed text and image for this task. In this section, we'll go through necessary steps to successfully train an image-caption model.

Create tfrecord dataset

All training data will be stored in `.tfrecords` file which is TensorFlow recommended file format. A `.tfrecords` file represents a sequence of (binary) strings. The format is not random access, so it is suitable for streaming large amounts of data but not suitable if fast sharding or other non-sequential access is desired.
Note: You can use either `.tfrecords` as input format or other format you want. Here demonstrate how to create `.tfrecords` for training.

In [8]:

```python
def create_tfrecords(df_cap, img_df, filename, num_files=5):
    ''' create tfrecords for dataset '''

    def _float_feature(value):
        return tf.train.Feature(
            float_list=tf.train.FloatList(value=value))

    def _int64_feature(value):
        return tf.train.Feature(
            int64_list=tf.train.Int64List(value=value))

    num_records_per_file = img_df.shape[0] // num_files

    total_count = 0

    print("create training dataset....")
    for i in range(num_files):
        # tfrecord writer: write record into files
        count = 0
        writer = tf.python_io.TFRecordWriter(
            filename + '-' + str(i + 1) +'.tfrecords')

        # start point (inclusive)
        st = i * num_records_per_file
        # end point (exclusive)
        ed = (i + 1) * num_records_per_file if i != num_files - 1 else img_df.shape[0]

        for idx, row in img_df.iloc[st:ed].iterrows():

            # img representation in 256-d array format
            img_representation = row['img']

            # each image has some captions describing it.
            for _, inner_row in df_cap[df_cap['img_id'] == row['img_id']].iterrows():
                # caption in different sequence length list format
                caption = eval(inner_row['caption'])

                # construct 'example' object containing 'img', 'caption'
                example = tf.train.Example(features=tf.train.Features(
                    feature={
                        'img': _float_feature(img_representation),
                        'caption': _int64_feature(caption)
                    }))

                count += 1
                writer.write(example.SerializeToString())
        print("create {}-{}.tfrecords -- contains {} records".format(
                                    filename, str(i + 1), count))
        total_count += count
        writer.close()
    print("Total records: {}".format(total_count))
```

Note: this cell will take about 30 minutes to create all training examples into `tfrecords`. Suggest that you can run create_tfrecord.py in the background.

In [9]:

```
# uncomment next line to create tfrecords file
# create_tfrecords(df_cap, img_train_df, 'dataset/tfrecords/train', 10)
```

In [10]:

```
import glob
training_filenames = glob.glob('dataset/tfrecords/train-*')

# get the number of records in training files
def get_num_records(files):
    count = 0
    for fn in files:
        for record in tf.python_io.tf_record_iterator(fn):
            count += 1
    return count

num_train_records = get_num_records(training_filenames)
print('Number of training records in all training file: {}'.format(
    num_train_records))
```

```
Number of training records in all training file: 513969
```

We need to use a parser to parse what is in `.tfrecords`

In [11]:

```
def training_parser(record):
    ''' parse record from .tfrecords file and create training record

    :args
      record - each record extracted from .tfrecords
    :return
      a dictionary contains {
          'img': image array extracted from vgg16 (256-dim),
          'input_seq': a list of word id
                     which describes input caption sequence (Tensor),
          'output_seq': a list of word id
                     which describes output caption sequence (Tensor),
          'mask': a list of one which describe
                     the length of input caption sequence (Tensor)
      }
    '''

    keys_to_features = {
      "img": tf.FixedLenFeature([256], dtype=tf.float32),
      "caption": tf.VarLenFeature(dtype=tf.int64)
    }

    # features contains - 'img', 'caption'
    features = tf.parse_single_example(record, features=keys_to_features)

    img = features['img']
    caption = features['caption'].values
    caption = tf.cast(caption, tf.int32)

    # create input and output sequence for each training example
    # e.g. caption :   [0 2 5 7 9 1]
    #      input_seq:  [0 2 5 7 9]
    #      output_seq: [2 5 7 9 1]
    #      mask:       [1 1 1 1 1]
    caption_len = tf.shape(caption)[0]
    input_len = tf.expand_dims(tf.subtract(caption_len, 1), 0)
    input_seq = tf.slice(caption, [0], input_len)
    output_seq = tf.slice(caption, [1], input_len)
    mask = tf.ones(input_len, dtype=tf.int32)

    records = {
      'img': img,
      'input_seq': input_seq,
      'output_seq': output_seq,
      'mask': mask
    }

    return records
```

Consume tfrecord dataset

The `Dataset` API in TensorFlow supports a variety of file formats so that you can process large datasets that do not fit in memory. The `tf.data.TFRecordDataset` class enables you to stream over the contents of one or more TFRecord files as part of an input pipeline. The great thing among it is that it can dynamically pad to the equal length of sequence in each batch. As in previous Lab taught, we can use `Iterator` to consume data.

In [12]:

```
def tfrecord_iterator(filenames, batch_size, record_parser):
    ''' create iterator to eat tfrecord dataset

    :args
        filenames     - a list of filenames (string)
        batch_size    - batch size (positive int)
        record_parser - a parser that read tfrecord
                        and create example record (function)

    :return
        iterator      - an Iterator providing a way
                        to extract elements from the created dataset.
        output_types  - the output types of the created dataset.
        output_shapes - the output shapes of the created dataset.
    '''
    dataset = tf.data.TFRecordDataset(filenames)
    dataset = dataset.map(record_parser, num_parallel_calls=16)

    # padded into equal length in each batch
    dataset = dataset.padded_batch(
      batch_size=batch_size,
      padded_shapes={
          'img': [None],
          'input_seq': [None],
          'output_seq': [None],
          'mask': [None]
      },
      padding_values={
          'img': 1.0,       # needless, for completeness
          'input_seq': 1,    # padding input sequence in this batch
          'output_seq': 1,  # padding output sequence in this batch
          'mask': 0         # padding 0 means no words in this position
      })

    dataset = dataset.repeat()             # repeat dataset infinitely
    dataset = dataset.shuffle(3*batch_size)  # shuffle the dataset

    iterator = dataset.make_initializable_iterator()
    output_types = dataset.output_types
    output_shapes = dataset.output_shapes

    return iterator, output_types, output_shapes
```

Build input tensor

Build input for training

- image_embed - image embedding array in 256-dimension `(shape=[batch_size, 256])`
- input_seq - a list of word id describing input sequence `(shape=[batch_size, padded_length])`
- target_seq - a list of word id describing output sequence `(shape=[batch_size, padded_length])`
- input_mask - a list of 1/0 to indicate whether it is a word `(shape=[batch_size, padded_length])`

Get Sequence embeddings

We have a list of sequence id, but we need to embed each word to a embedding vector. You can either train a word_embedding or use pre-trained word embedding model.
Note: TensorFlow provides a very efficient implementation to do lookup embedding.

> `seq_embeddings = tf.nn.embedding_lookup(embedding_matrix, input_seq)`

In [13]:

```
def get_seq_embeddings(input_seq, vocab_size, word_embedding_size):
    with tf.variable_scope('seq_embedding'), tf.device("/cpu:0"):
        embedding_matrix = tf.get_variable(
            name='embedding_matrix',
            shape=[vocab_size, word_embedding_size],
            initializer=tf.random_uniform_initializer(minval=-1, maxval=1))
        # [batch_size, padded_length, embedding_size]
        seq_embeddings = tf.nn.embedding_lookup(embedding_matrix, input_seq)
    return seq_embeddings
```

Build the model

A thing to note is that the input/outputs fed at training is slightly different from those at testing:

- training: we have a pair (caption and image) of example, then feed image representation into initial state of rnn and caption embeddings into rnn inputs.
- testing: we start generating the caption by providing `<ST>` and image as input, then we sample a word as next word, and use the sampled word and rnn state as input for next timestep to generate sequential words until the token `<ED>` is sampled as next word

In [14]:

```
class ImageCaptionModel(object):
    ''' simple image caption model '''

    def __init__(self, hparams):
        self.hps = hparams

    def _build_inputs(self):
        """ construct the inputs for model """
        self.filenames = tf.placeholder(tf.string,
                                        shape=[None], name='filenames')
        self.training_iterator, types, shapes = tfrecord_iterator(
          self.filenames, self.hps.batch_size, training_parser)

        self.handle = tf.placeholder(tf.string, shape=[], name='handle')
        iterator = tf.data.Iterator.from_string_handle(self.handle,
                                                       types, shapes)
        records = iterator.get_next()

        image_embed = records['img']
        image_embed.set_shape([None, self.hps.image_embedding_size])
        input_seq = records['input_seq']
        target_seq = records['output_seq']
        input_mask = records['mask']

        self.image_embed = image_embed # (batch_size, img_dim)
        self.input_seq = input_seq # (batch_size, seqlen)
        self.target_seq = target_seq # (batch_size, seqlen)
        self.input_mask = input_mask # (batch_size, seqlen)

        # convert sequence of index to sequence of embedding
        with tf.variable_scope('seq_embedding'), tf.device('/cpu:0'):
            self.embedding_matrix = tf.get_variable(
                    name='embedding_matrix',
                    shape=[self.hps.vocab_size,
                           self.hps.word_embedding_size],
                    initializer=tf.random_uniform_initializer(
                        minval=-1, maxval=1))
            # [batch_size, seqlen, embedding_size]
            seq_embeddings = tf.nn.embedding_lookup(
                self.embedding_matrix, self.input_seq)

    def _build_model(self):
        """ Build your image caption model """
        pass

    def build(self):
        """ call this function to build the inputs and model """
        self._build_inputs()
        self._build_model()

    def train(self, sess, training_filenames, num_train_records):
        """ write a training function for your model """
        pass

    def predict(self, sess, img_vec, dec_map):
        """ generate the caption given an image """
        pass
```

Setting hyperparameters

You can set all hyperparameters here.

In [15]:

```
def get_hparams():
    hparams = tf.contrib.training.HParams(
      vocab_size=vocab_size,
      batch_size=64,
      rnn_units=100,
      image_embedding_size=256,
      word_embedding_size=256,
      drop_keep_prob=0.7,
      lr=1e-3,
      training_epochs=1,
      max_caption_len=15,
      ckpt_dir='model_ckpt/')
    return hparams
```

In [16]:

```
# get hperparameters
hparams = get_hparams()
# create model
model = ImageCaptionModel(hparams)
model.build()
```

In [17]:

```
# start training
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
model.train(sess, training_filenames, num_train_records)
```

Inference

The behavior of training and inferencing a RNN model is different. At inference time, we could only take the partial captions generated by the model, which is possible not perfect, and use it as input to generate next word. In fact, how to effectively improve the quality of RNN model is an active research problem.

Inference: Simple Caption Generation

The simplest inference process would be just generate text word by taking the most likely one, and feed this chosen word as input to get following words until we've generated enough length caption or hit the `<ED>` token.

input to get following words until we've generated enough length caption or hit the <ED> token.

In [20]:

```
tf.reset_default_graph()
model = ImageCaptionModel(hparams)
model.build()

# sample one image in training data and generate caption
testimg = img_train_df.iloc[9]['img']
testimg = np.expand_dims(testimg, axis=0)

with tf.Session(config=config) as sess:
    saver = tf.train.Saver()
    # restore variables from disk.
    ckpt = tf.train.get_checkpoint_state(hparams.ckpt_dir)
    if ckpt and ckpt.model_checkpoint_path:
        saver.restore(sess,
                      tf.train.latest_checkpoint(hparams.ckpt_dir))
        caption = model.predict(sess, testimg, dec_map)
        print(caption)
    else:
        print("No checkpoint found.")
```

```
INFO:tensorflow:Restoring parameters from model_ckpt/model.ckpt-12253780
others band interesting bushes narrow morning lots band interesting bushes narrow morning lots band inter
esting
```

Captioning other images

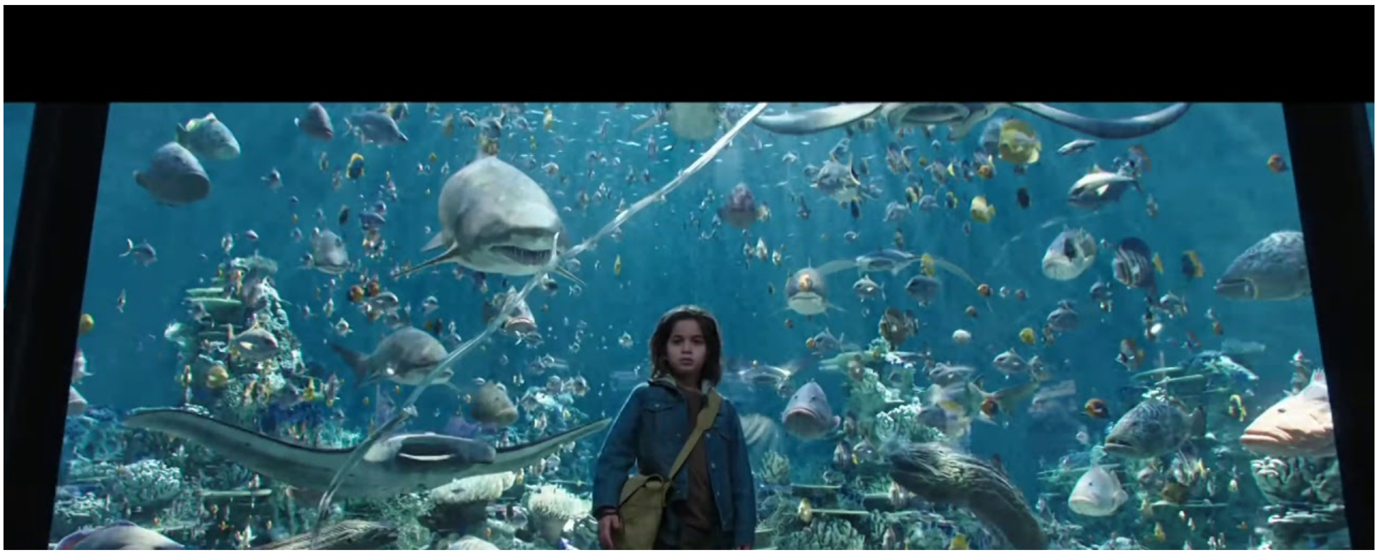
We can use the trained model to do captioning on our images.

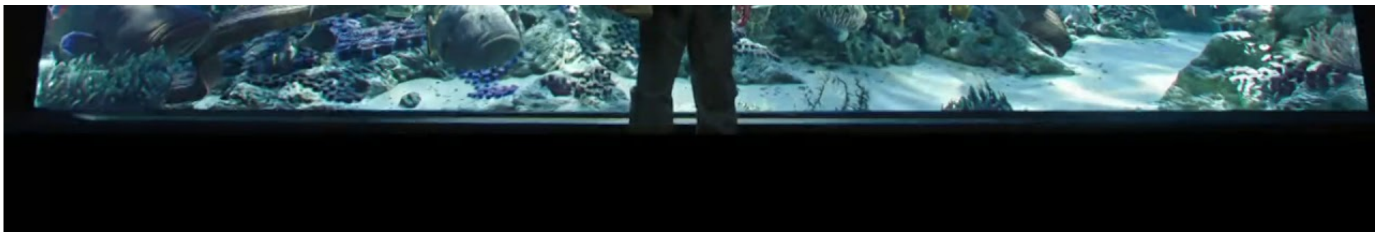
In [18]:

```
%matplotlib inline
import matplotlib.pyplot as plt
from IPython.display import Image, display
from pretrained.cnn import PretrainedCNN
import imageio
import skimage.transform
import numpy as np
import scipy

def demo(img_path, cnn_mdl, U, dec_map, hparams, max_len=15):
    """
    displays the caption generated for the image
    -------------------------------
    img_path: image to be captioned
    cnn_mdl: path of the image feature extractor
    U: transform matrix to perform PCA
    dec_map: mapping of vocabulary ID => token string
    hparams: hyperparams for model
    """

    def process_image(img, crop=True, submean=True):
        """
        implements the image preprocess required by VGG-16
        -------------------------------
        resize image to 224 x 224
        crop: do center-crop [skipped by default]
        submean: substracts mean image of ImageNet [skipped by default]
        """
        MEAN = np.array([103.939, 116.779, 123.68]).astype(np.float32) # BGR
        # center crop
        short_edge = min(img.shape[:2])
        yy = int((img.shape[0] - short_edge) / 2)
        xx = int((img.shape[1] - short_edge) / 2)
        crop_img = img[yy: yy + short_edge, xx: xx + short_edge]
        img = scipy.misc.imresize(crop_img, [224, 224, 3])
        img = scipy.misc.imresize(crop_img, [224, 224, 3])
        img = img.reshape((224,224,1)) if len(img.shape) < 3 else img

        if img.shape[2] < 3:
            print('dimension insufficient')
            img = img.reshape((224*224,
                               img.shape[2])).T.reshape((img.shape[2],
                                                                 224*224))
            for i in range(img.shape[0], 3):
                img = np.vstack([img, img[0,:]])
            img = img.reshape((3,224*224)).T.reshape((224,224,3))
        img = img.astype(np.float32)
        img = img[:,:,::-1]
        # RGB => BGR
        for i in range(3):
            img[:,:,i] -= MEAN[i]
        return img.reshape((224,224,3))

    display(Image(img_path))
    img = imageio.imread(img_path)

    # load pretrained cnn model
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    with tf.Session(config=config) as sess:
        sess.run(tf.global_variables_initializer())
        img_feature = np.dot(
            cnn_mdl.get_output(sess, [process_image(img)])[0].reshape((-1)), U)

    # reset graph for image caption model
    tf.reset_default_graph()
    model = ImageCaptionModel(hparams)
    model.build()
    with tf.Session(config=config) as sess:
        saver = tf.train.Saver()
        # restore variables from disk.
        ckpt = tf.train.get_checkpoint_state(hparams.ckpt_dir)
        if ckpt and ckpt.model_checkpoint_path:
            saver.restore(sess, tf.train.latest_checkpoint(hparams.ckpt_dir))
            caption = model.predict(sess, img_feature, dec_map)
            print(' '.join(caption))
        else:
            print("No checkpoint found.")
```

In [52]:

```
tf.reset_default_graph()  # reset graph for cnn model
U = cPickle.load(open('dataset/U.pkl', 'rb'))  # PCA transforming matrix
vgg = PretrainedCNN('pretrained/vgg16_mat.pkl')
demo('demo/example1.jpg', vgg, U, dec_map, hparams)
```





```
INFO:tensorflow:Restoring parameters from model_ckpt/model.ckpt-803000
A man and some fish swim in water
```

Evaluation

CIDErD is proposed on 2015 CVPR and is designed for image captioning task, which is adopted as one of evaluation metrics in MS-COCO competition.
To automatically evaluate quality of a caption, there are 2 main goals:

1. evaluate correct keywords related to that image
2. evalute the grammar quality of generated caption

Basically, CIDEr-D achieves the goals by first, construct the n-gram token dictionary (without stemming), and then compare the similarity of TF-IDF score between ground-truth caption and generated caption. The order is consider by using larger n of n-gram, it's practical since our caption is only a sentence.

However, since Kaggle-InClass donnot accept custom evaluation metric, we require you to compute your CIDEr-D score locally and submit to our competition page. Please run the executable - `CIDErD/gen_score` to generate CIDEr-D score. The followings are example steps to generate your submission:

1. Generate All captions of Testing images

In [19]:

```
def generate_captions(model, dec_map, img_test, max_len=15):
    img_ids, caps = [], []

    with tf.Session() as sess:
        saver = tf.train.Saver()
        # restore variables from disk.
        ckpt = tf.train.get_checkpoint_state(hparams.ckpt_dir)
        if ckpt and ckpt.model_checkpoint_path:
            saver.restore(sess,
                          tf.train.latest_checkpoint(hparams.ckpt_dir))
            for img_id, img in img_test.items():
                img_ids.append(img_id)
                caps.append(model.predict(sess, img, dec_map))
        else:
            print("No checkpoint found.")

    return pd.DataFrame({
            'img_id': img_ids,
            'caption': caps
        }).set_index(['img_id'])
```

In [23]:

```
# load test image  size=20548
img_test = cPickle.load(open('dataset/test_img256.pkl', 'rb'))

# create model
tf.reset_default_graph()
model = ImageCaptionModel(hparams)
model.build()

# generate caption to csv file
df_predict = generate_captions(model, dec_map, img_test)
df_predict.to_csv('generated/demo.csv')
```

You can quickly take a look at the generated caption generated/demo.csv to see how models learns about grammars, semantics, ...etc. However, please strictly follow our rule: **it's forbidden to do any manual modification to generated captions.**

2. Execute CIDEr-D executable to generate score.csv
Important: Download corresponding CIDEr-D version of your operating system,

- `CIDErD_macos` : for macos.
- `CIDErD_linux` : for linux.
- `CIDErD_win`: for windows.

because some path dapendence issue, you must change your directory to CIDErD, then execute `./gen_score`.

- `-i`: your generated captions in `csv` format
- `-r`: your evaluated CIDErD score, submit this file to Kaggle-InClass

You can see help manual by argument `-h`, for example, `./gen_score -h`

In [24]:
```
os.system('cd CIDErD && ./gen_score -i ../generated/demo.csv -r ../generated/score.csv')
```

```
0
```

3. Submit generated `score.csv` to DataLabCup: Image Caption

Hints

Training: Gradient-Clipping

When training RNN, the gradient easily explodes on the cliff-like error surface. To prevent gradient explosion problem, we'll do gradient-clipping to truncate large gradient updates.

In [ ]:
```
# create an optimizer
optimizer = tf.train.AdamOptimizer(learning_rate=lr)
# compute the gradients of a list of variables
grads_and_vars = optimizer.compute_gradients(total_loss,
                                             tf.trainable_variables())
# grads_and_vars is a list of tuple (gradient, variable)
# do whatever you need to the 'gradients' part
clipped_grads_and_vars = [(tf.clip_by_norm(gv[0], 1.0), gv[1])
                          for gv in grads_and_vars]
# apply gradient and variables to optimizer
train_op = optimizer.apply_gradients(
    clipped_grads_and_vars, global_step=global_step)
```

Training: Curricular Learning

Start training from easy examples, you can define captions with short length as easy examples, then gradually add longer captions to training examples. Curriculum learning has been shown to be a very useful technique when training RNN.

Training: Attention

Add a special attention layer to enable the network to focus on more important objects. With an attention mechanism, we allow the rnn to “attend” to different parts of the images at each step of the output generation. Importantly, we let the model learn what to attend to based on the input images and what it has produced so far.

Show, Attend and Tell: Neural Image Caption

Inference: Beam Search

The example code above generates caption by a locally greedy algorithm, which only samples a word with highest probability at each timestep. However, it doesn't necessarily going to give the best caption. The ideal caption should maximize the joint probability of all words at each timestep.

There's a commonly trick, called beam search, which has been empirically observed to improve testing performance by doing Breadth-First-Search over top k possible next word at each timestep, where k is called beam-size. We could rank the candidate captions by taking negative log likelihood(NLL) of joint probability of all words, then the above objective becomes

which can be easily summed up by taking negative log of softmax score at each timestep. After generating several possible captions, we could choose the one with least NLL as our final caption.

Sequence-to-Sequence Learning as Beam-Search Optimization

Other reference

Training: Scheduled Sampling

Curriculum learning strategy to gently change the training process from a fully guided scheme using the true previous token, towards a less guided scheme which mostly uses the generated token instead.

Scheduled Sampling for Sequence Prediction with RNN

Drawing

Training: Professor Forcing

Use adversarial domain adaptation to encourage the dynamics of the rnn to be the same when training the network and when sampling from the network over multiple time steps.

Professor Forcing

Drawing

```
In [ ]:
```