* ImageNetについて

　学習データを作るために必要な画像のビックデータは、ImageNetの画像を使用した。

　ImageNetはWordNet階層（名詞のみ）に基づいて構成され、ImageNet階層の各ノードが平均500枚以上の画像で表示されている画像データベースである。

　WordNetではsynsetと呼ばれる同義語のグループに分類されている。ImageNetの画像データは、WordNetの”WordNet ID” (wnid)によってファイル名が付けられており、このwnidをWordNetに照らし合わせることで、どういう画像なのかがわかるようになっている。

* ImageNet APIについて

　一つのsynsetのwordnetを取扱いときは下位語のsynsetsのwordnetが以下のURLに得る。

<http://www.image-net.org/archive/wordnet.is_a.txt>

　全部の下位語（全部のsubtreeのsynsetがwordnetから始める）を取扱いときは以下のURLに得る。

[http://www.image-net.org/api/text/wordnet.structure.hyponym?wnid=[wnid]&full=1](http://www.image-net.org/api/text/wordnet.structure.hyponym?wnid=%5bwnid%5d&full=1)

　一つのsynsetのワルドは以下のURLに得る。

[http://www.image-net.org/api/text/wordnet.synset.getwords?wnid=[wnid](http://www.image-net.org/api/text/wordnet.synset.getwords?wnid=%5bwnid)]

このURLをクリクしたら、全部のsynsetに関するwordnet IDとwords（余白注）の対照をもらえる。

　以下のURLにAPIを提供した。全部のsynsetのリストをもらえる。

<http://www.image-net.org/api/text/imagenet.synset.obtain_synset_list>

　wordnetの階層

以下のファイルはWordNet3.0が提供した全部の“is-a”関係を提供する。各行は親—子供のペア。

ダウンロード方法

GitHub(<https://github.com/beniz)より転用>

DepDetact <https://deepdetect.com/tutorials/train-imagenet/>

|  |
| --- |
| #!/usr/bin/env python  # Copyright (c) 2014 Seiya Tokui  #  # Permission is hereby granted, free of charge, to any person obtaining a copy  # of this software and associated documentation files (the "Software"), to deal  # in the Software without restriction, including without limitation the rights  # to use, copy, modify, merge, publish, distribute, sublicense, and/or sell  # copies of the Software, and to permit persons to whom the Software is  # furnished to do so, subject to the following conditions:  #  # The above copyright notice and this permission notice shall be included in  # all copies or substantial portions of the Software.  #  # THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR  # IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,  # FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE  # AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER  # LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,  # OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN  # THE SOFTWARE.  import argparse  import imghdr  import Queue  import os  import socket  import sys  import tempfile  import threading  import time  import urllib2  import glob  def download(url, timeout, retry, sleep, verbose=False):  """Downloads a file at given URL."""  count = 0  while True:  try:  f = urllib2.urlopen(url, timeout=timeout)  if f is None:  raise Exception('Cannot open URL {0}'.format(url))  content = f.read()  f.close()  break  except urllib2.HTTPError as e:  if 500 <= e.code < 600:  if verbose:  sys.stderr.write('Error: HTTP with code {0}\n'.format(e.code))  count += 1  if count > retry:  if verbose:  sys.stderr.write('Error: too many retries on {0}\n'.format(url))  raise  else:  if verbose:  sys.stderr.write('Error: HTTP with code {0}\n'.format(e.code))  raise  except urllib2.URLError as e:  if isinstance(e.reason, socket.gaierror):  count += 1  time.sleep(sleep)  if count > retry:  if verbose:  sys.stderr.write('Error: too many retries on {0}\n'.format(url))  raise  else:  if verbose:  sys.stderr.write('Error: URLError {0}\n'.format(e))  raise  #except Exception as e:  # if verbose:  # sys.stderr.write('Error: unknown during download: {0}\n'.format(e))  return content  def imgtype2ext(typ):  """Converts an image type given by imghdr.what() to a file extension."""  if typ == 'jpeg':  return 'jpg'  if typ is None:  raise Exception('Cannot detect image type')  return typ  def make\_directory(path):  if not os.path.isdir(path):  os.makedirs(path)  def download\_imagenet(list\_filename,  out\_dir,  timeout=10,  retry=10,  num\_jobs=1,  sleep\_after\_dl=1,  verbose=False,  offset=0,  msg=1):  """Downloads to out\_dir all images whose names and URLs are written in file  of name list\_filename.  """  make\_directory(out\_dir)  count\_total = 0  with open(list\_filename) as list\_in:  for i, l in enumerate(list\_in):  pass  count\_total = i + 1  count\_total -= offset  sys.stderr.write('Total: {0}\n'.format(count\_total))    num\_jobs = max(num\_jobs, 1)  entries = Queue.Queue(num\_jobs)  done = [False]  counts\_fail = [0 for i in xrange(num\_jobs)]  counts\_success = [0 for i in xrange(num\_jobs)]  def producer():  count = 0  with open(list\_filename) as list\_in:  for line in list\_in:  if count >= offset:  name, url = line.strip().split(None, 1)  entries.put((name, url), block=True)  count += 1  entries.join()  done[0] = True  def consumer(i):  while not done[0]:  try:  name, url = entries.get(timeout=1)  except:  continue  try:  if name is None:  if verbose:  sys.stderr.write('Error: Invalid line: {0}\n'.line)  counts\_fail[i] += 1  continue  directory = os.path.join(out\_dir, name.split('\_')[0])  rpath = os.path.join(directory, '{0}.\*'.format(name))  lf = glob.glob(rpath)  if lf:  print "skipping: already have", lf[0]  counts\_success[i] += 1  entries.task\_done()  continue  content = download(url, timeout, retry, sleep\_after\_dl)  ext = imgtype2ext(imghdr.what('', content))  try:  make\_directory(directory)  except:  pass  path = os.path.join(directory, '{0}.{1}'.format(name, ext))  with open(path, 'w') as f:  f.write(content)  counts\_success[i] += 1  time.sleep(sleep\_after\_dl)  except Exception as e:  counts\_fail[i] += 1  if verbose:  sys.stderr.write('Error: {0} / {1}: {2}\n'.format(name, url, e))    entries.task\_done()  def message\_loop():  if verbose:  delim = '\n'  else:  delim = '\r'  while not done[0]:  count\_success = sum(counts\_success)  count = count\_success + sum(counts\_fail)  rate\_done = count \* 100.0 / count\_total  if count == 0:  rate\_success = 0  else:  rate\_success = count\_success \* 100.0 / count  sys.stderr.write(  '{0} / {1} ({2}%) done, {3} / {0} ({4}%) succeeded {5}'.format(  count, count\_total, rate\_done, count\_success, rate\_success, delim))  time.sleep(msg)  sys.stderr.write('done')  producer\_thread = threading.Thread(target=producer)  consumer\_threads = [threading.Thread(target=consumer, args=(i,)) for i in xrange(num\_jobs)]  message\_thread = threading.Thread(target=message\_loop)  producer\_thread.start()  for t in consumer\_threads:  t.start()  message\_thread.start()  # Explicitly wait to accept SIGINT  try:  while producer\_thread.isAlive():  time.sleep(1)  except:  sys.exit(1)  producer\_thread.join()  for t in consumer\_threads:  t.join()  message\_thread.join()  sys.stderr.write('\ndone\n')  if \_\_name\_\_ == '\_\_main\_\_':  p = argparse.ArgumentParser()  p.add\_argument('list', help='Imagenet list file')  p.add\_argument('outdir', help='Output directory')  p.add\_argument('--jobs', '-j', type=int, default=1,  help='Number of parallel threads to download')  p.add\_argument('--timeout', '-t', type=int, default=10,  help='Timeout per image in seconds')  p.add\_argument('--retry', '-r', type=int, default=10,  help='Max count of retry for each image')  p.add\_argument('--sleep', '-s', type=float, default=1,  help='Sleep after download each image in second')  p.add\_argument('--verbose', '-v', action='store\_true',  help='Enable verbose messages')  p.add\_argument('--offset', '-o', type=int, default=0,  help='Offset to where to start in Imagenet list file')  p.add\_argument('--msg', '-m', type=int, default=1,  help='Logging message every x seconds')  args = p.parse\_args()  download\_imagenet(args.list, args.outdir,  timeout=args.timeout, retry=args.retry,  num\_jobs=args.jobs, verbose=args.verbose,  offset=args.offset, msg=args.msg) |