RAPIDS 预装镜像使用文档

本文档以 RAPIDS 用于加速图像搜索为例，介绍了如何在预装镜像中使用 RAPIDS 加速库。本文档示例是在阿里云 GN6V(Tesla V100)实例上完成的。

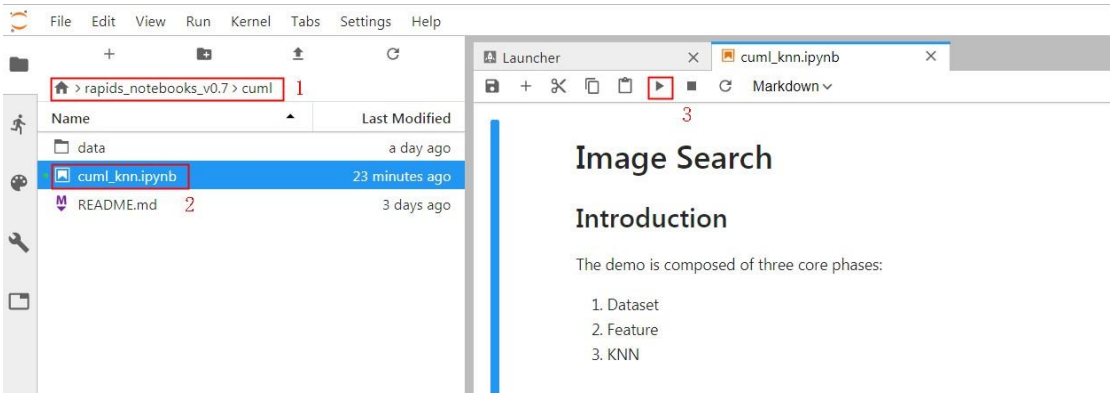
启动 JupyterLab

```
# Go to the notebooks directory.
cd /rapids

# Run the following command to start JupyterLab and set the logon password:
jupyter-lab --allow-root --ip=0.0.0.0 --no-browser --NotebookApp.token='your logon password'

# Exit jupyterlab: press Ctrl+C twice.
```

打开浏览器输入 http://(IP address of your GPU instance):8888（需要在安全组里开放 TCP 8888 端口）进入 jupyterlab 密码登录界面，输入启动命令中设定的密码即可成功登录。

打开图像搜索案例

1. 打开案例所在目录：rapids_notebooks_v0.7/cuml。
2. 双击打开 cuml_knn.ipynb 文件。
3. 点击 Run 按钮进行代码执行。



案例执行过程

图像搜索案例主要有三个部分：数据集处理，图片特征提取，相似图片搜索。该案例结果展示了使用 cuML 的 KNN 与 sklearn 的 KNN 的性能对比。

数据集处理

数据集处理包括数据集下载和解压，图片读取，图片展示和数据集分割。

1. 数据集下载和解压

该案例使用的是 stl10 数据集，该数据集中包含 10 万张未打标的图片，每张图片的尺寸为(96 x 96 x 3)，此处用户可以将其替换成其他数据集(为便于进行图片特征提取，所有图片的尺寸需相同)。案例提供了 download_and_extract(data_dir)方法，该方法可以进行 stl10 数据集的下载和解压(rapids 镜像中已经将数据集下载到./data 目录，执行 download_and_extract()方法可以直接进行解压操作)。

2. 图片读取

解压后的数据格式为二进制格式，使用 read_all_images(path_to_data)方法进行数据加载并转成 NHWC(N, Height, width, channel) 格式，以便使用 tensorflow 进行特征提取。

Read Data

[3]:
```
# the path of unlabeled data
path_unlabeled = os.path.join(data_dir, 'stl10_binary/unlabeled_X.bin')
# get images from binary
images = read_all_images(path_unlabeled)
print('>>> images shape: ', images.shape)
```

```
>>> images shape:  (100000, 96, 96, 3)
```

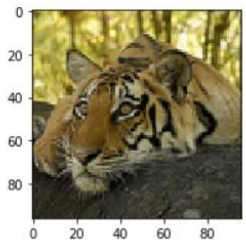
3. 图片展示

执行 show_image(image)方法，随机展示一张数据集中的图片。

Show Image

[4]:
```
import random
import matplotlib.pyplot as plt
%matplotlib inline

def show_image(image):
    """show image"""
    fig = plt.figure(figsize=(3, 3))
    plt.imshow(image)
    plt.show()
    fig.clear()
```

[10]:
```
# random show a image
rand_image_index = random.randint(0, images.shape[0])
show_image(images[rand_image_index])
```



4. 数据集分割

按照 9:1 的比例把数据集分为两部分，分别用于图片索引库的创建和图片搜索。

Split Dataset

```
from sklearn.model_selection import train_test_split

train_images, query_images = train_test_split(images, test_size=0.1, random_state=123)
print('train_images shape: ', train_images.shape)
print('query_images shape: ', query_images.shape)
```

```
train_images shape:  (90000, 96, 96, 3)
query_images shape:  (10000, 96, 96, 3)
```

图片特征提取

使用 Tensorflow 和 Keras 进行图片特征提取，模型使用已训练的 ResNet50(notop, 基于 ImageNet 数据集)。该阶段包括 tensorflow 参数设定，ResNet50(notop)模型下载，图片特征提取。

1. Tensorflow 参数设定

Tensorflow 默认使用所有 GPU 内存，我们需要留出部分 GPU 内存给 cuml。这里提供了两种 GPU 内存参数设定方法，方法 1：config.gpu_options.allow_growth = True 依据运行需求进行内存分配。方法 2：设定可以使用的 GPU 内存比例 config.gpu_options.per_process_gpu_memory_fraction = 0.3，此处默认设定了 0.3，即 Tensorflow 可以使用整块 GPU 内存的 0.3 倍，此值用户可以依据应用场景进行修改。案例使用方法 2。

Image Features

```
# set tensorflow params to adjust GPU memory usage, if use default params, tensorflow would use
# nearly all of the gpu memory, we need reserve some gpu memory for cuml.
import os
# only use device 0
os.environ["CUDA_VISIBLE_DEVICES"] = "0"

import tensorflow as tf
from keras.backend.tensorflow_backend import set_session
config = tf.ConfigProto()
# method 1: allocate gpu memory base on runtime allocations
# config.gpu_options.allow_growth = True
# method 2: determines the fraction of the onerall amount of memory
# that each visibel GPU should be allocated.
config.gpu_options.per_process_gpu_memory_fraction = 0.3
set_session(tf.Session(config=config))
```

```
Using TensorFlow backend.
```

2. ResNet50(notop)预训练模型下载

使用基于 ImageNet 数据集的 ResNet50(notop)预训练模型。该过程需要联通公网进行模型下载(模型约 91M)，下载后的模型默认保存到/root/.keras/models/目录。

- Weights:可以选择 None(随机初始化权重值)或者 imagenet(pre-training on ImageNet)，此处选择 imagenet。
- include_top:是否包含整个 ResNet50 网络结构的最后一个全链接层，本案例使用神经网络模型的主要目的是进行特征提取而非图片分类，所以此处设定为 False。
- input_shape:图片的输入 shape，为可选参数，该参数只有当 include_top 设定为 False 时才可以使用，图片必须有 3 个 inputs channels，且宽和高不应低于 32。此处设为(96, 96, 3)。

- pooling:当 inculde_top 设为 False 时，需要设定池化层模式，可以设为 None 或 avg 或 max，如果设定为 None 输出为 4D tensor，avg 和 max 输出为 2D tensor，建议选择 avg 或 max。此处选择 max。

```
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input
```

```
# download resnet50(notop) model(first running) and load model
model = ResNet50(weights='imagenet', include_top=False, input_shape=(96, 96, 3), pooling='max')
```

```
WARNING:tensorflow:From /root/anaconda3/envs/rapids/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:263:
colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.2/resnet50_weights_tf_dim_ordering_tf_k
ernels_notop.h5
94658560/94653016 [==============================] - 9s 0us/step
```

使用 model.summary()查看模型的网络结构。

```
[10]: # network summary
      model.summary()
```

```
add_16 (Add)                    (None, 3, 3, 2048)   0           bn5c_branch2c[0][0]
                                                                 activation_46[0][0]
activation_49 (Activation)      (None, 3, 3, 2048)   0           add_16[0][0]
global_max_pooling2d_1 (GlobalM (None, 2048)         0           activation_49[0][0]
==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
```

3. 图片特征提取

使用 model.predict()方法分别对分割后的两个图片数据集进行特征提取。

```
[10]: %%time
      train_features = model.predict(train_images)
      print('train features shape: ', train_features.shape)
```

```
train features shape:  (90000, 2048)
CPU times: user 33.6 s, sys: 7.94 s, total: 41.5 s
Wall time: 36.3 s
```

```
[11]: %%time
      query_features = model.predict(query_images)
      print('query features shape: ', query_features.shape)
```

```
query features shape:  (10000, 2048)
CPU times: user 3.64 s, sys: 704 ms, total: 4.34 s
Wall time: 3.76 s
```

相似图片搜索

1. 使用 cuml KNN 进行相似图片搜索

设定 K 值为 3(k=3)，即查找最相似的 3 张图片，该值用户可以依据使用场景自定义设定，cuml 的 KNN 的创建过程分为两步，第一步为创建索引阶段：fit，第二步为近邻搜索阶段：kneighbros()。

cuml KNN

```
[12]: from cuml.neighbors import NearestNeighbors

[13]: %%time
      knn_cuml = NearestNeighbors()
      knn_cuml.fit(train_features)

      CPU times: user 888 ms, sys: 60 ms, total: 948 ms
      Wall time: 192 ms

[14]: %%time
      distances_cuml, indices_cuml = knn_cuml.kneighbors(query_features, k=3)

      CPU times: user 1.59 s, sys: 492 ms, total: 2.08 s
      Wall time: 791 ms
```

2. 使用 sklearn KNN 进行相似图片搜索

同 cuml KNN 设定 K 值为 3(n_neighbors=3)，且使用所有 CPU(n_jobs=-1)进行近邻搜索，该测试环境为 8 核 CPU。

sklearn KNN

```
[15]: from sklearn.neighbors import NearestNeighbors

[16]: %%time
      knn_sk = NearestNeighbors(n_neighbors=3, metric='sqeuclidean', n_jobs=-1)
      knn_sk.fit(train_features)

      CPU times: user 856 ms, sys: 36 ms, total: 892 ms
      Wall time: 114 ms

[17]: %%time
      distances_sk, indices_sk = knn_sk.kneighbors(query_features, 3)

      CPU times: user 18.2 s, sys: 29.9 s, total: 48.1 s
      Wall time: 7min 34s
```

3. 搜索结果对比

对比 cuml KNN 和 sklearn KNN 的搜索结果，验证两种实现方式的输出结果是否相同。输出结果为两个数组：

- distance:最近的 K 个距离值，本案例搜索了 10000 张图片，K 值为 3，所以 distance.shape=(10000, 3)。
- indices: 对应的图片索引，indices.shape=(10000, 3)。

由于该数据集中存在重复的图片，即容易出现相同图片不同索引的情况，所以不使用 indices 做结果对比，这里仅对比 distances 结果。考虑计算误差的因素，假定当 cuml KNN 和 sklearn KNN 的 10000 张图片的 3 个最小距离值误差都在 1 以内则认为结果相同。

Compare

```
# compare the distance obtained while using sklearn and cuml models
(np.abs(distances_cuml - distances_sk) < 1).all()
```

```
True
```

4. 相似图片搜索结果展示

案例默认从 1 万张搜索图片中，随机选择 5 张用于展示，最终显示的图片有 5 行 4 列，其中第一列为搜索图片，第二列为图片索引库中最相似的图片，第三列为第二相似，第四列为第三相似。每张相似图片的标题为计算的距离。

```
# get random indices
random_show_index = np.random.randint(0, query_images.shape[0], size=5)
random_query = query_images[random_show_index]
random_indices = indices_cuml[random_show_index].astype(np.int)
random_distances = distances_cuml[random_show_index]

# show result images
for query_image, sim_indices, sim_dists in zip(random_query, random_indices, random_distances):
    sim_images = train_images[sim_indices]
    show_images(query_image, sim_images, sim_dists)
```