关于编译 sift 出现的问题

1.进行 sift 特征提取的实验

参考网上已经有的程序，进行编译和调试，对于遇到的问题进行解决和学习。

a.首先，原程序比较长直接运行出现了很多问题，所以先运行图像读取和显示的部分。如图一所示。

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdio.h>//标准输入输出头文件，编写C语言中，常用到printf()和scanf()函数.
#include <string.h>//是C版本的头文件，包含比如strcpy、strcat之类的字符串处理函数。

#define TRUE 1//define 用于简单的字符替换

/*extern "C"{
#include <vl/generic.h>
#include <vl/sift.h>
}*/

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{

    cout << "SIFT特征提取演示程序" << endl;
    //VL_PRINT("Hello world!\n");

    Mat img_1 = imread("1.jpg", 0);
    namedWindow("Source Image", 1);
    imshow("Source Image", img_1);
    waitKey(0);

    return 0;
}
```

图一

不调用 vlfeat 库时，正常编译 g++ s.cpp -o s，出现错误，如图二所示。

```
lujingyu@lujingyu-Inspiron-3521:~/sift$ g++ s.cpp -o s
/tmp/ccyTTGJM.o：在函数‘main’中：
s.cpp:(.text+0x6f)：对‘cv::imread(cv::String const&, int)’未定义的引用
s.cpp:(.text+0xa6)：对‘cv::namedWindow(cv::String const&, int)’未定义的引用
s.cpp:(.text+0xf8)：对‘cv::imshow(cv::String const&, cv::_InputArray const&
定义的引用
s.cpp:(.text+0x120)：对‘cv::waitKey(int)’未定义的引用
/tmp/ccyTTGJM.o：在函数‘cv::String::String(char const*)’中：
s.cpp:(.text._ZN2cv6StringC2EPKc[_ZN2cv6StringC5EPKc]+0x4d)：对‘cv::String:
cate(unsigned long)’未定义的引用
/tmp/ccyTTGJM.o：在函数‘cv::String::~String()’中：
s.cpp:(.text._ZN2cv6StringD2Ev[_ZN2cv6StringD5Ev]+0x14)：对‘cv::String::dea
te()’未定义的引用
/tmp/ccyTTGJM.o：在函数‘cv::Mat::~Mat()’中：
s.cpp:(.text._ZN2cv3MatD2Ev[_ZN2cv3MatD5Ev]+0x39)：对‘cv::fastFree(void*)’未
的引用
/tmp/ccyTTGJM.o：在函数‘cv::Mat::release()’中：
s.cpp:(.text._ZN2cv3Mat7releaseEv[_ZN2cv3Mat7releaseEv]+0x4b)：对‘cv::Mat::
ocate()’未定义的引用
collect2: error: ld returned 1 exit status
```

图二

原因是找不到 opencv 中我所需要的函数库。通过到 opencv 官网查询需要用 cmake 进行编译。

```
cmake_minimum_required(VERSION 2.8)
project( s )
find_package( OpenCV REQUIRED )
include_directories(${OpenCV_INCLUDE_DIRS})
add_executable( s s.cpp )
target_link_libraries( s ${OpenCV_LIBS})
```

进行在终端进行编译，就可以运行得到输出的图像没有错误。

之后，将关于 vlfeat 的注释去掉，再想用 cmake 编译就需要添加关于 vlfeat 的数据库链接参数。但是目前对于 cmake 的了解有限,还没有学会如何添加参数。

b.所以又需要在终端编译，那就要解决图二中的错误。参考 vlfeat 官网中的一个例子，编译时的命令为 g++ main.cpp -o vlfeat-test -IVLROOT -LVLROOT/bin/a64/ -lvl (vlfeat-test:生成文件名自己定义 -I：头文件名 -L：数据库的路径 -l：库的名字)，所以想到是否编译 opencv 时也需要输入数据库的链接参数，通过网上搜索，需要加`pkg-config --cflags --libs opencv`这条命令。但是执行 g++ s.cpp -o s -IVLROOT -LVLROOT/bin/a64/ -lvl `pkg-config --cflags --libs opencv` 后出现了另一个错误，找不到-lippicv。开始以是库名就是 lippicv，通过了解函数库的定义后得知其实是找不到 ippicv，经过手动添加到/usr/local/lib 后，再运行，就没有任何问题了，编译成功。

c.在终端输入 pkg-config --cflags --libs opencv，如图三所示。

```
lujingyu@lujingyu-Inspiron-3521:~/sift$ pkg-config --cflags --libs opencv
-I/usr/local/include/opencv -I/usr/local/include -L/usr/local/lib -lopencv_shape
 -lopencv_stitching -lopencv_objdetect -lopencv_superres -lopencv_videostab -lip
picv -lopencv_calib3d -lopencv_features2d -lopencv_highgui -lopencv_videoio -lop
encv_imgcodecs -lopencv_video -lopencv_photo -lopencv_ml -lopencv_imgproc -lopen
cv_flann -lopencv_core
```

图三

能够看到所有的数据库，但是我的这段程序只用到了 highgui 和 core 这两个数据库，所以是否可以只加-lopencv_highgui 和-lopencv_core 呢？结果如图四所示。

```
lujingyu@lujingyu-Inspiron-3521:~/sift$ g++ s.cpp -o s -IVLROOT -LVLROOT/bin/a64
/ -lvl -lopencv_highgui -lopencv_core
/usr/bin/ld: /tmp/ccM6jcPJ.o: undefined reference to symbol '_ZN2cv6imreadERKNS_
6StringEi'
//usr/local/lib/libopencv_imgcodecs.so.3.1: error adding symbols: DSO missing fr
om command line
collect2: error: ld returned 1 exit status
```

图四

根据错误提示发现少 libopencv_imgcodecs 这个动态库，有可能是 highgui.hpp 或者 core.hpp 中包含了 imgcodecs.hpp。通过查看这两个头文件发现 highgui.hpp 中包含了 imgcodecs.hpp 添加后再运行，编译成功。之后又想试试去掉-I 和-L 只输入-l 是否可以运行，发现也是可以编译成功的，换句话说就是可以自动找到相应的路径，因为之前已经把 vl 文件拷贝到/usr/include/下，库拷贝到/usr/lib 下，但所需要的数据库是必须要指明的。

d.综上，对于无法找到 ippicv 的问题，因为程序中没有用到这个文件，所以如果要用 pkg-config --cflags --libs opencv 这条命令，/usr/local/lib 下就需要拥有全部的函数库，无论缺少哪个都会出错。但是为什么用 cmake 编译是就没有出现错误，可能是自动可以找到我所需要的函数库，没有用的就略过。

e.接下来对整个程序进行编译，出现了一个错误 circle 没有定义。发现需要 imgproc.hpp 再进行编译，成功。图五为实验结果。

图五

2.对动态库和静态库的学习

函数库分为动态库和静态库两种。函数库：把一些共用的函数制作成函数库，供其他程序使用。

静态库在程序编译时会被链接到目标代码中，程序运行是将不需要该静态库。

动态库在程序编译时不会被链接到目标代码中，而是在程序运行时才被载入，因此在程序运行时还需要动态库的存在。

无论是静态库还是动态库,都由.o 文件创建。静态库文件的命名规范以 lib 为前缀，紧接着跟静态库名，扩展为.a。使用静态库内部函数，只需要在使用到这些公用函数的源程序中包含这些公用的函数原型声明，然后在用 gcc 命令生成目标文件时指明静态库名，gcc 将会从静态库中将公用的函数链接到目标文件中。动态库后缀为.so。使用时需要在/esr/lib 和/lib 等目录中查找需要的动态库文件。

3.老师 sift 部分这样就可以了嘛？开始 BoW?