# 目标与需求

目标：训练检测图片中是否有直立人体的检测器

Why choose this ： 想通过这个熟悉机器学习的训练过程，没有GPU只有CPU，--》简单模型

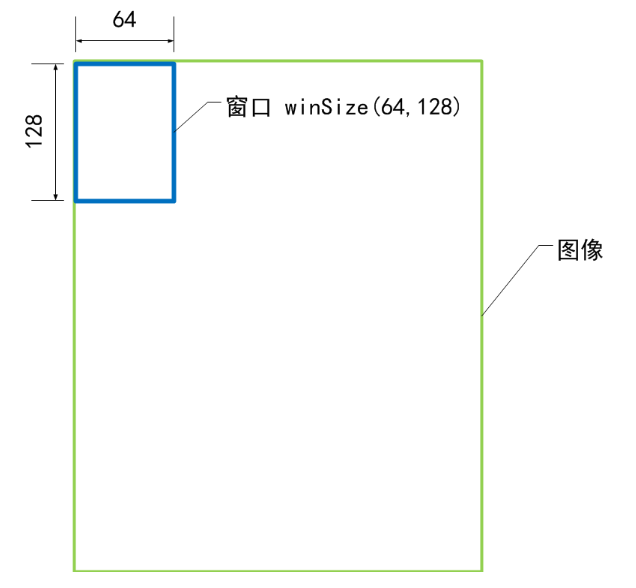
思路：二分类问题—》hog特征svm linear 模型—》训练出来的检测器只能检测和训练集一样大小的图片？--》滑动窗口 多尺度检测—》时间短，如何实现，语言，平台？--》只接触过一点点C++--》visual studio + opencv训练模型—》想学点新东西—》Qt使用训练好的模型，进行界面展示—》实干：学习svm与hog理论基础，学习opencv相关模块使用，训练自己的模型，学习Qt。

# HOG特征 与 提取

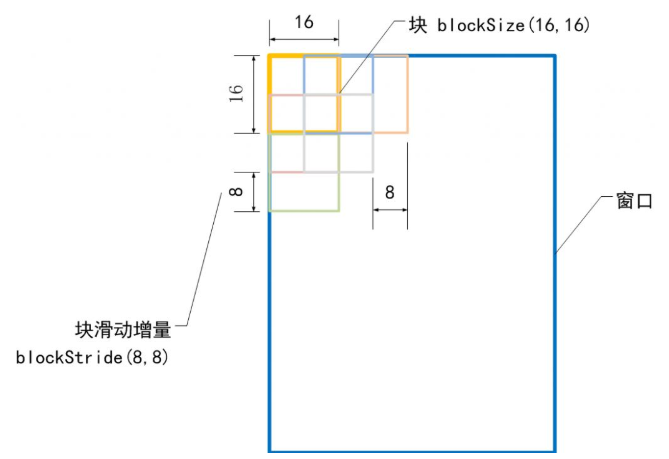
## Hog描述子 特征计算

1. HOGDescriptor() : winSize(64,128), blockSize(16,16), blockStride(8,8), cellSize(8,8), nbins(9)

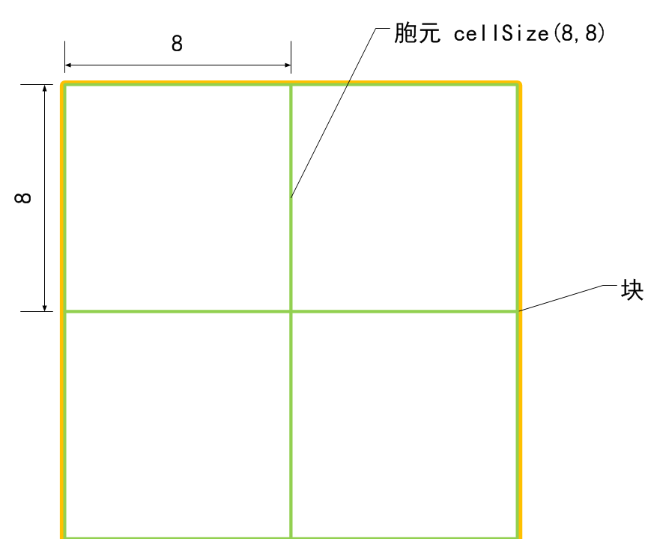
窗口大小 winSize



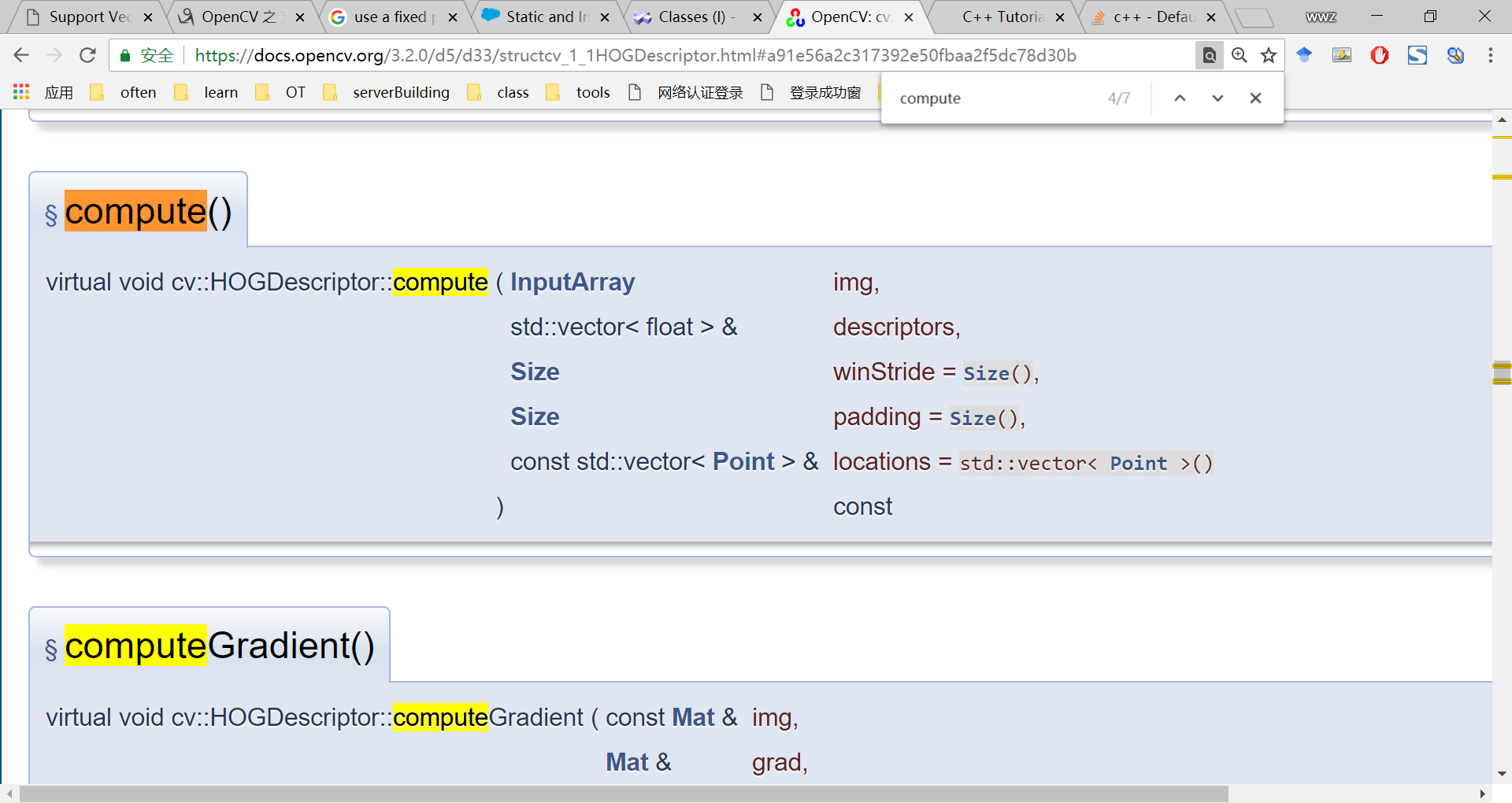
块大小 blockSize



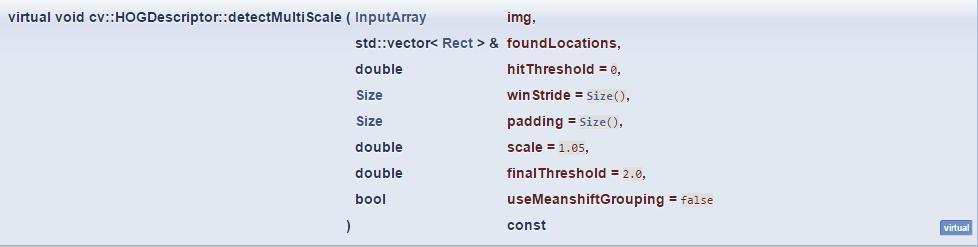
胞元大小 cellSize



## compute



## 多尺度检测



可以看到一共有8个参数。

1.img(必需)

这个不用多解释，显然是要输入的图像。图像可以是彩色也可以是灰度的。

2.foundLocations

存取检测到的目标位置

3.hitThreshold (可选)

opencv documents的解释是特征到SVM超平面的距离的阈值(*Threshold for the distance between features and SVM classifying plane)*

所以说这个参数可能是控制HOG特征与SVM最优超平面间的最大距离，当距离小于阈值时则判定为目标。

4.winStride(可选)

HoG检测窗口移动时的步长(水平及竖直)。

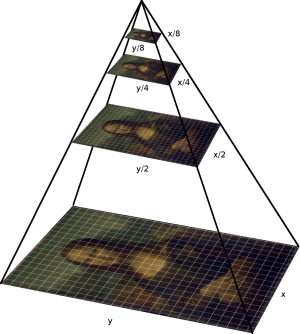
winStride和scale都是比较重要的参数，需要合理的设置。一个合适参数能够大大提升检测精确度，同时也不会使检测时间太长。

5.padding(可选)

在原图外围添加像素，作者在原文中提到，适当的pad可以提高检测的准确率（可能pad后能检测到边角的目标？）

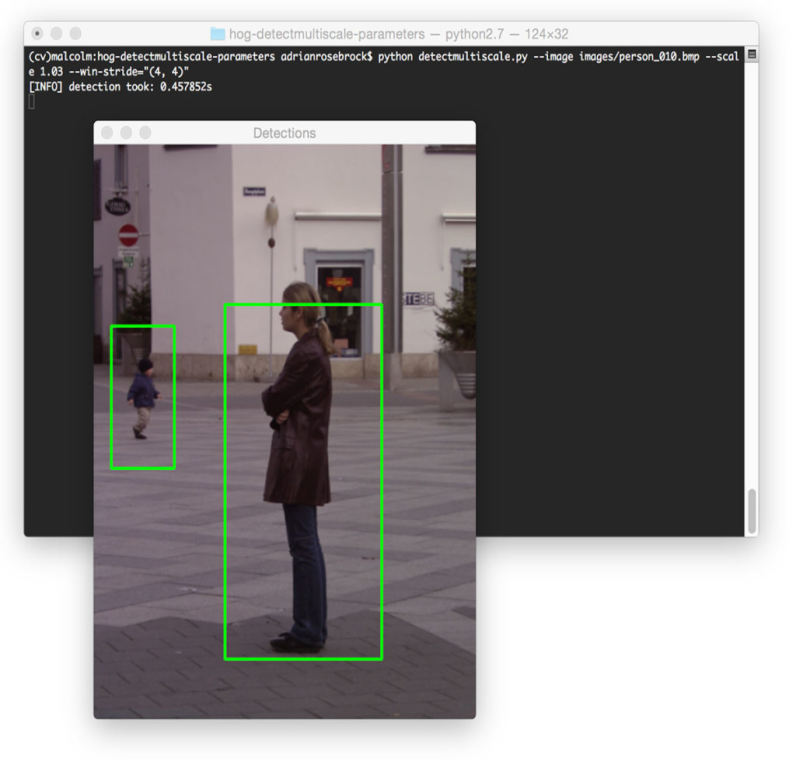
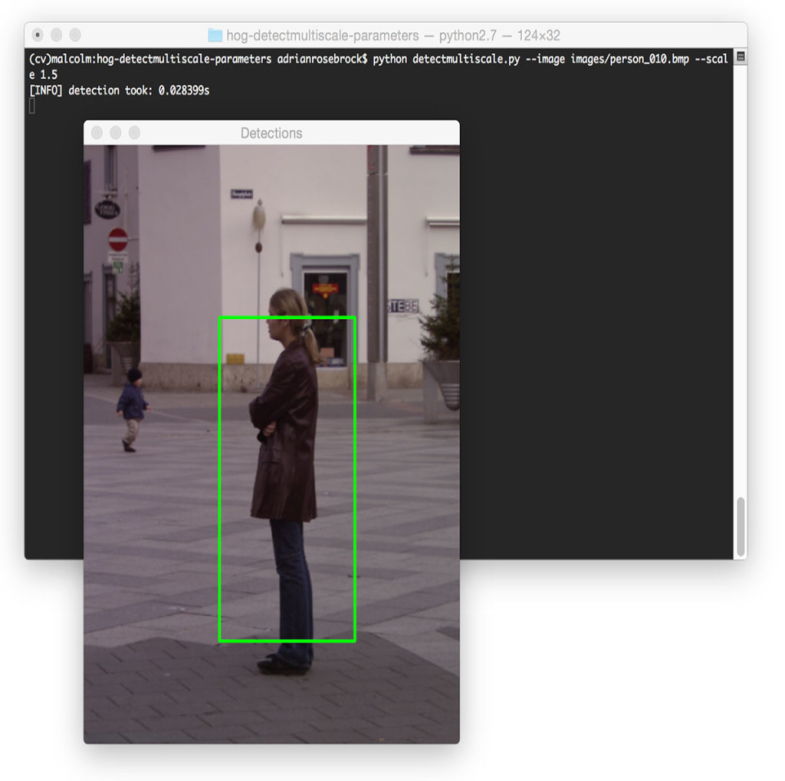
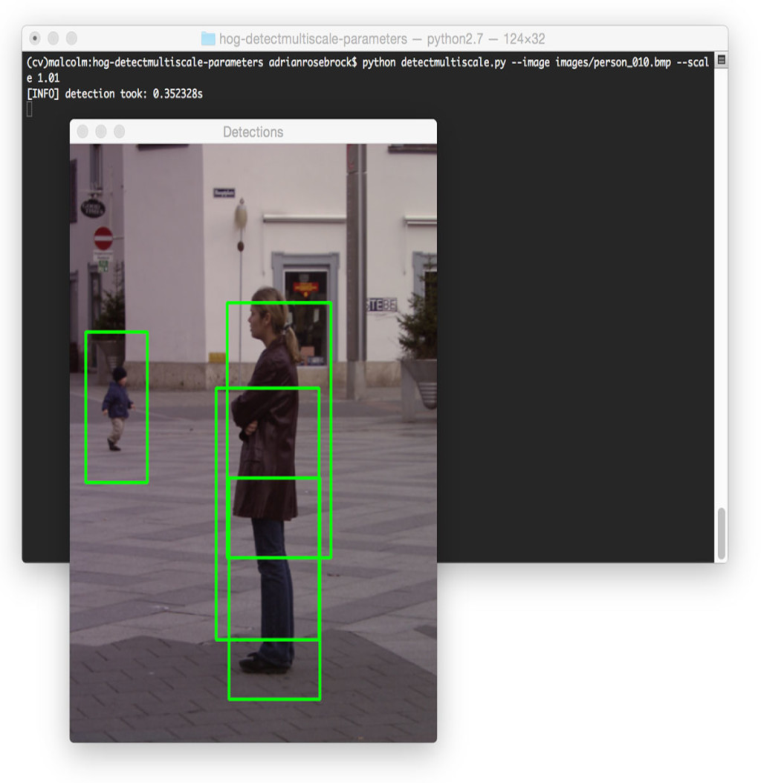
常见的pad size 有*(8, 8)*, *(16, 16)*, *(24, 24)*, *(32, 32)*.

6.scale(可选)

****

如图是一个图像金字塔，也就是图像的多尺度表示。每层图像都被缩小尺寸并用gaussian平滑。

scale参数可以具体控制金字塔的层数，参数越小，层数越多，检测时间也长。 一下分别是1.01  1.5 1.03 时检测到的目标。 通常scale在1.01-1.5这个区间



7.finalThreshold（可选）

这个参数不太清楚，有人说是为了优化最后的bounding box

8.useMeanShiftGrouping(可选)

bool 类型，决定是否应用meanshift 来消除重叠。

default为false，通常也设为false，另行应用non-maxima supperssion效果更好。

# SVM原理 与 实现

//计算-(alphaMat \* supportVectorMat),结果放到resultMat中，

注意因为svm.predict使用的是alpha\*sv\*another-rho，

如果为负的话则认为是正样本，

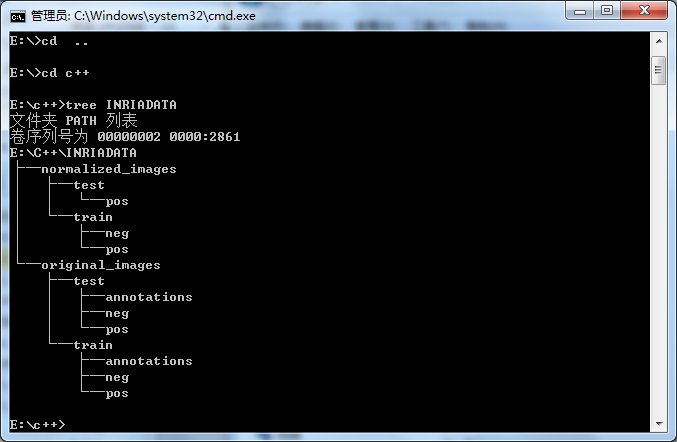
在HOG的检测函数中，使用rho+alpha\*sv\*another如果为正的话是正样本，

所以需要将后者变为负数之后保存起来

re = -1 \* alp \* sv;

# 数据集 与 处理

## INRIA数据集，目录结构和说明：



INRIADATA{

normalized\_images{

train{

pos:96x160大小，训练正样本，需要crop中间的64x128大小。已经做过flip，即包含左右对称的图

neg:大小不一，通常是几百乘几百，训练负样本，需要从每张图中随机crop 10个区域作为训练负样本

}

test{

pos: 70\*134,已经flip

}

}

original\_images{

train{

pos:训练正样本，大小不一

neg:训练负样本，大小不一

annotations:标注信息

}

test{

pos:大小不一

neg:大小不一

annotations:标注信息

}

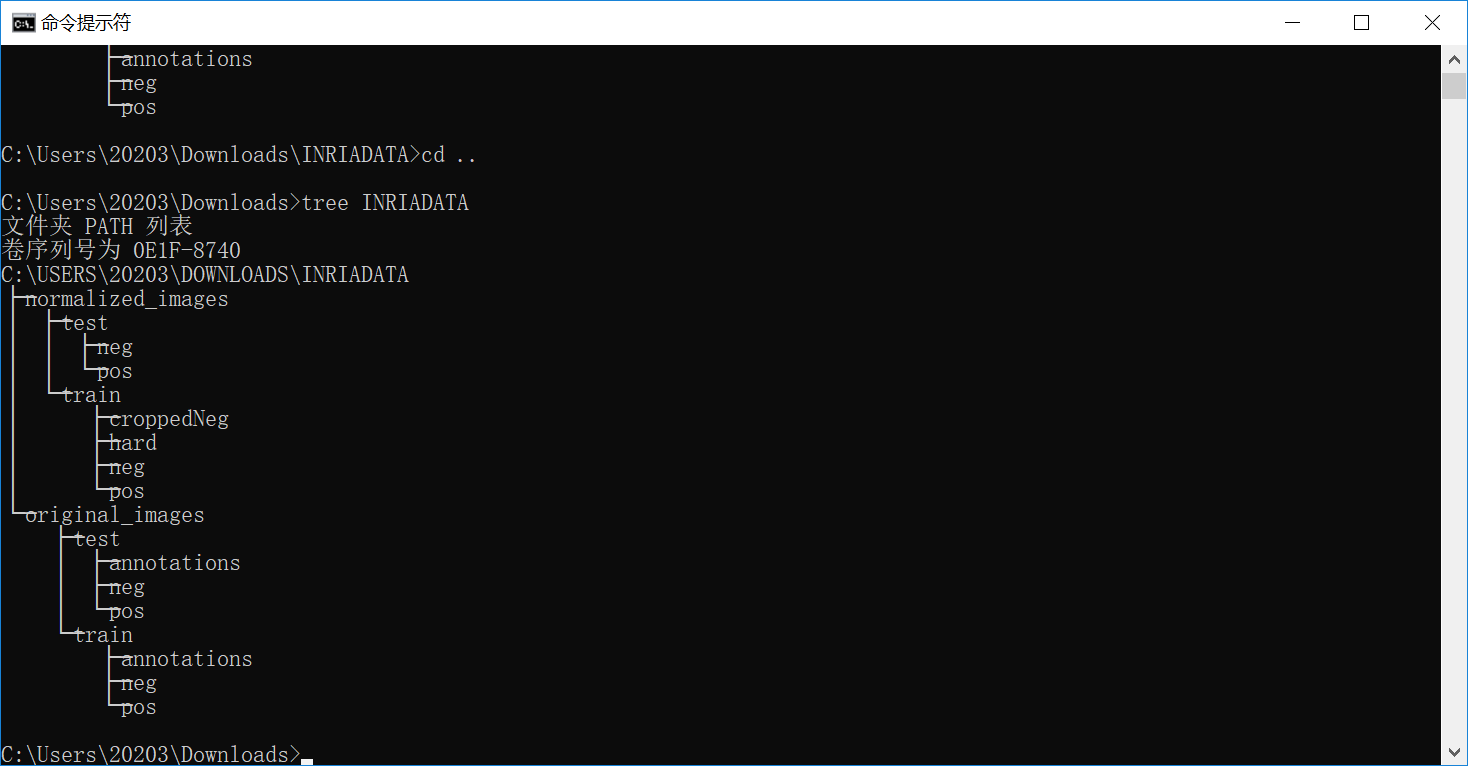
}

}

训练和测试的时候有几种选择方式：

用normalized\_images目录下的图片做训练，或者用original\_images目录下的图片+annotations获取行人区域做训练；测试则都在original\_images/test/pos上测试。

## 整理后的INRIA数据集，目录结构和说明：



# 训练过程

## 流程

– > Set the hog windowSize(64×128) and Compute the HOG features for the training image set containing vehicles and non-vehicles.

– > Train a SVM using these HOG features.

– > Test the trained SVM on the separate test imageset and evaluate the results and performance.

– > For the real world test, run a sliding window and use the SVM from above to predict vehicle/non-vehicle.

## 数据集DataSet类的构建

## HogDescriptor类 与 svm类

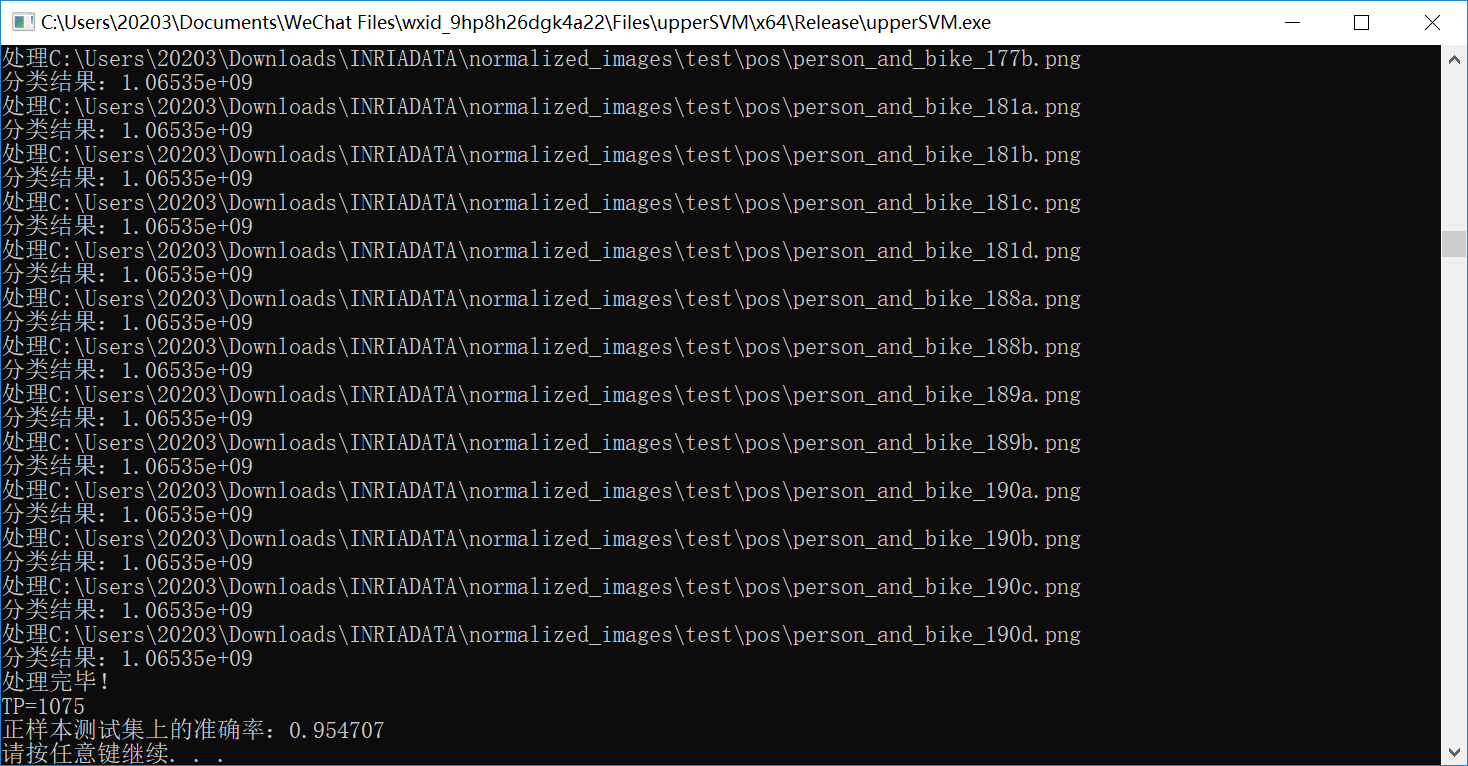
## 全部normalized train data + Linear kernel

|  |  |  |
| --- | --- | --- |
| 样本\预测 | Positive=1086 | Negative=4570 |
| Positive=1126 | TP=1075 | FN=51 |
| Negative=4530 | FP=11 | TN=4519 |

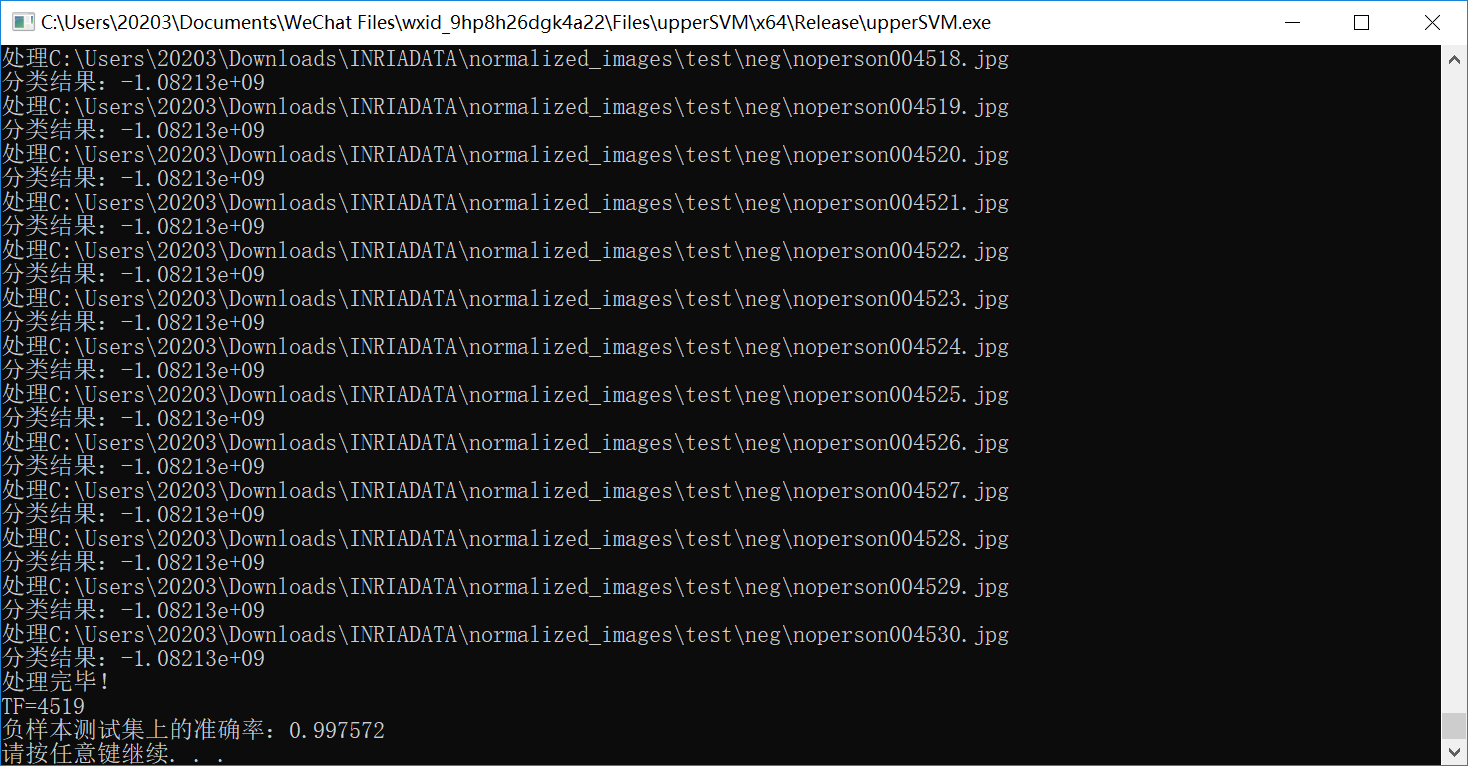
MultiScale检测结果



Test正样本集上的准确率



Test负样本集上的准确率



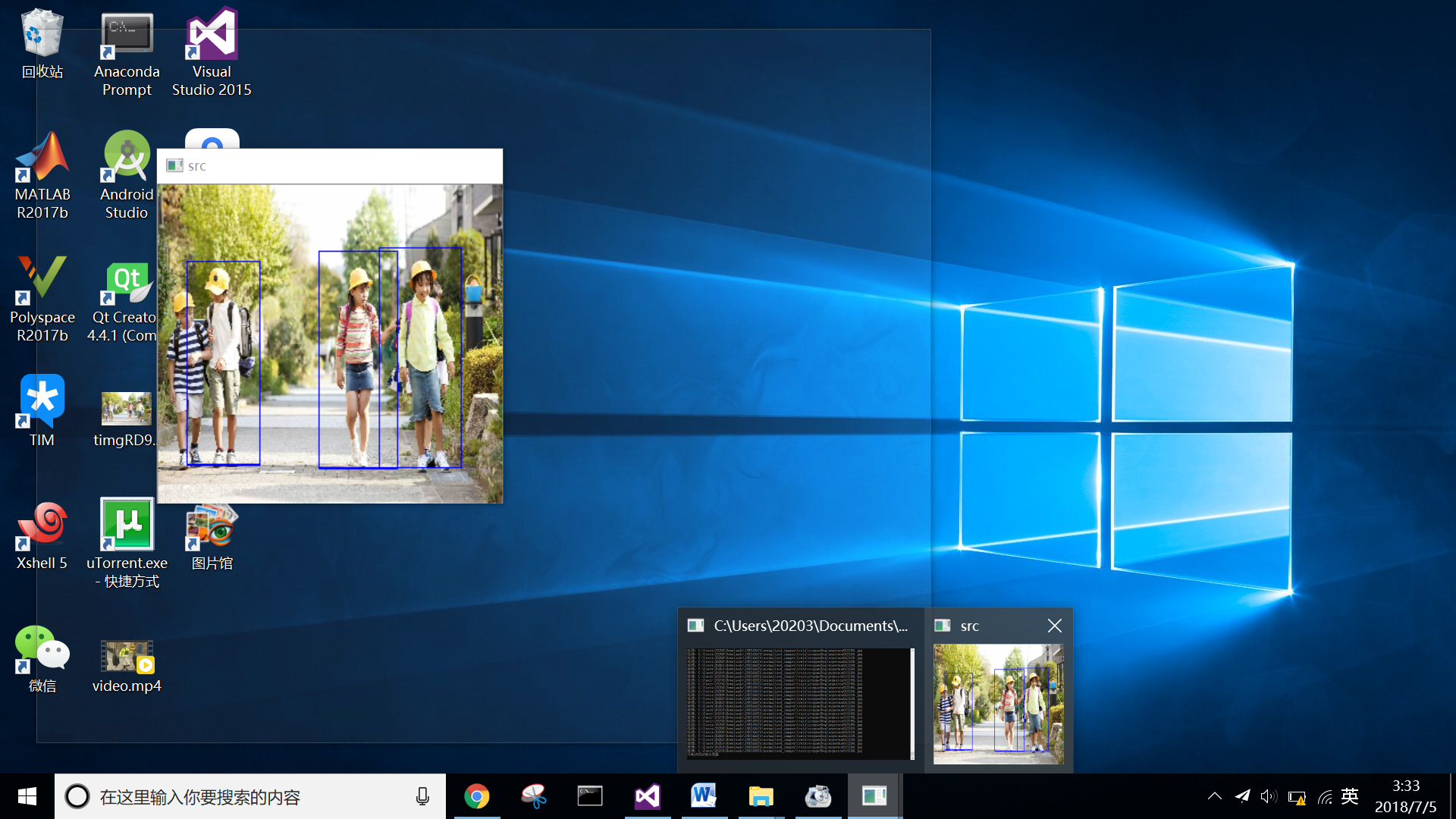
## hard example

### 难样本获取



### 加入难样本训练

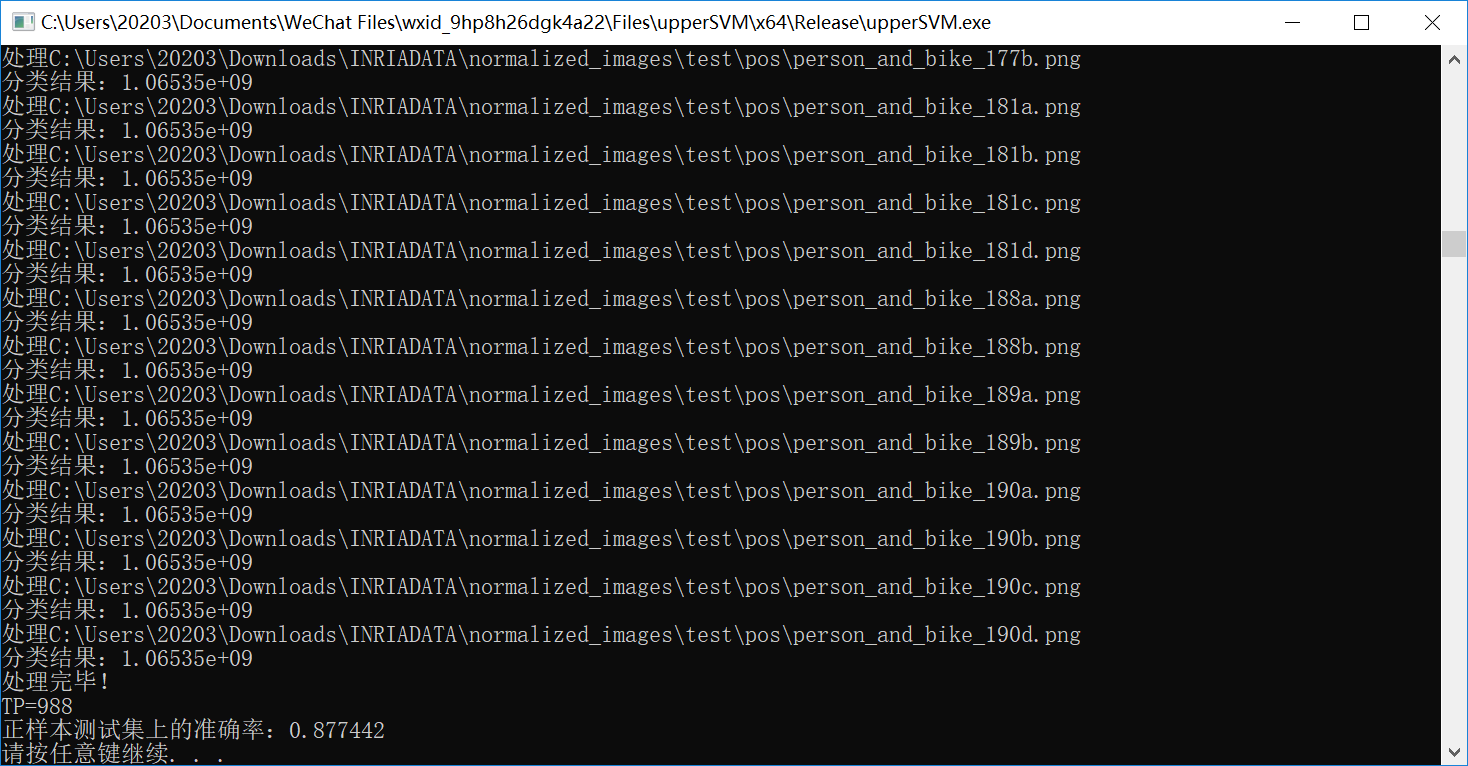
Add all hard example：



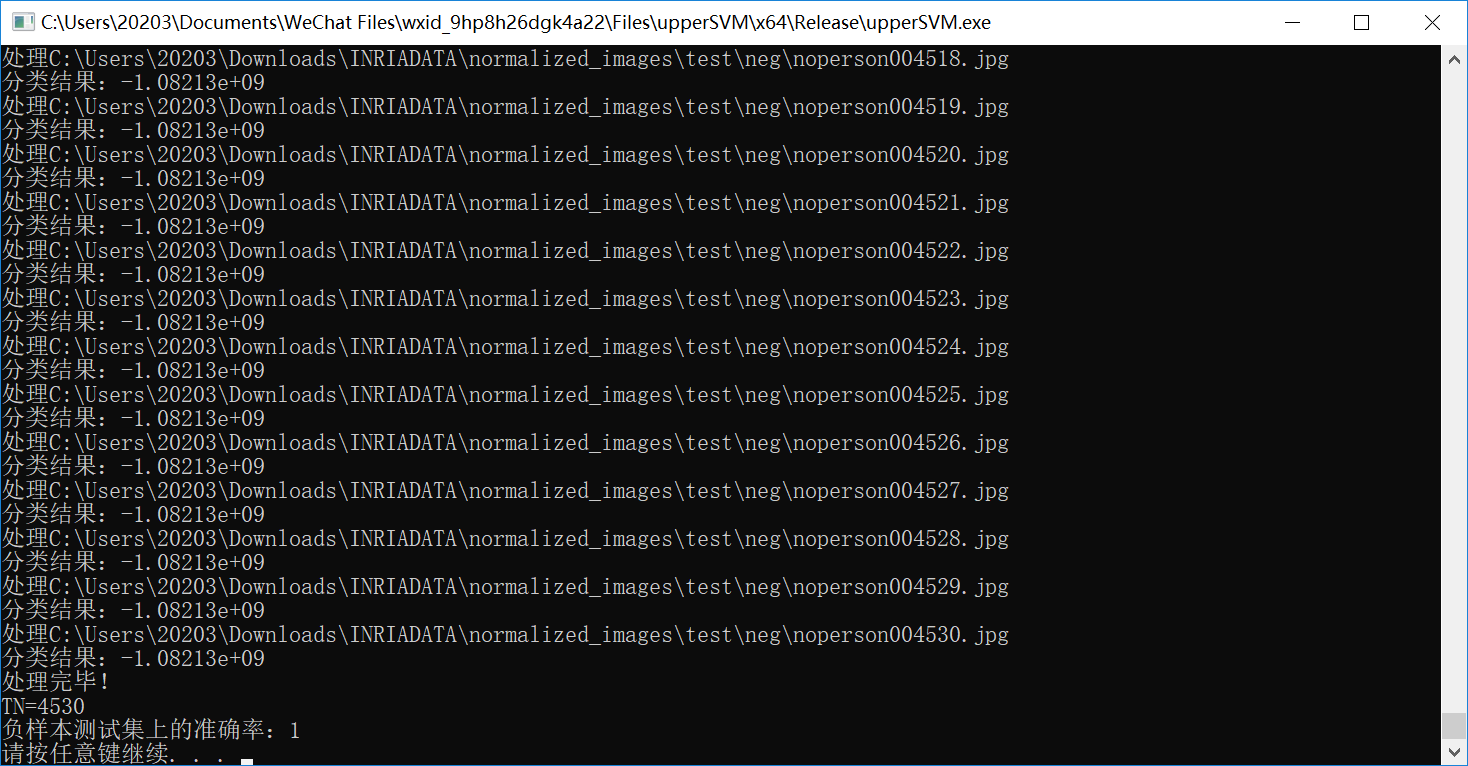
### All Hard+linear

|  |  |  |
| --- | --- | --- |
| 样本\预测 | Positive=988 | Negative=4668 |
| Positive=1126 | TP=988 | FN=138 |
| Negative=4530 | FP=0 | TN=4530 |

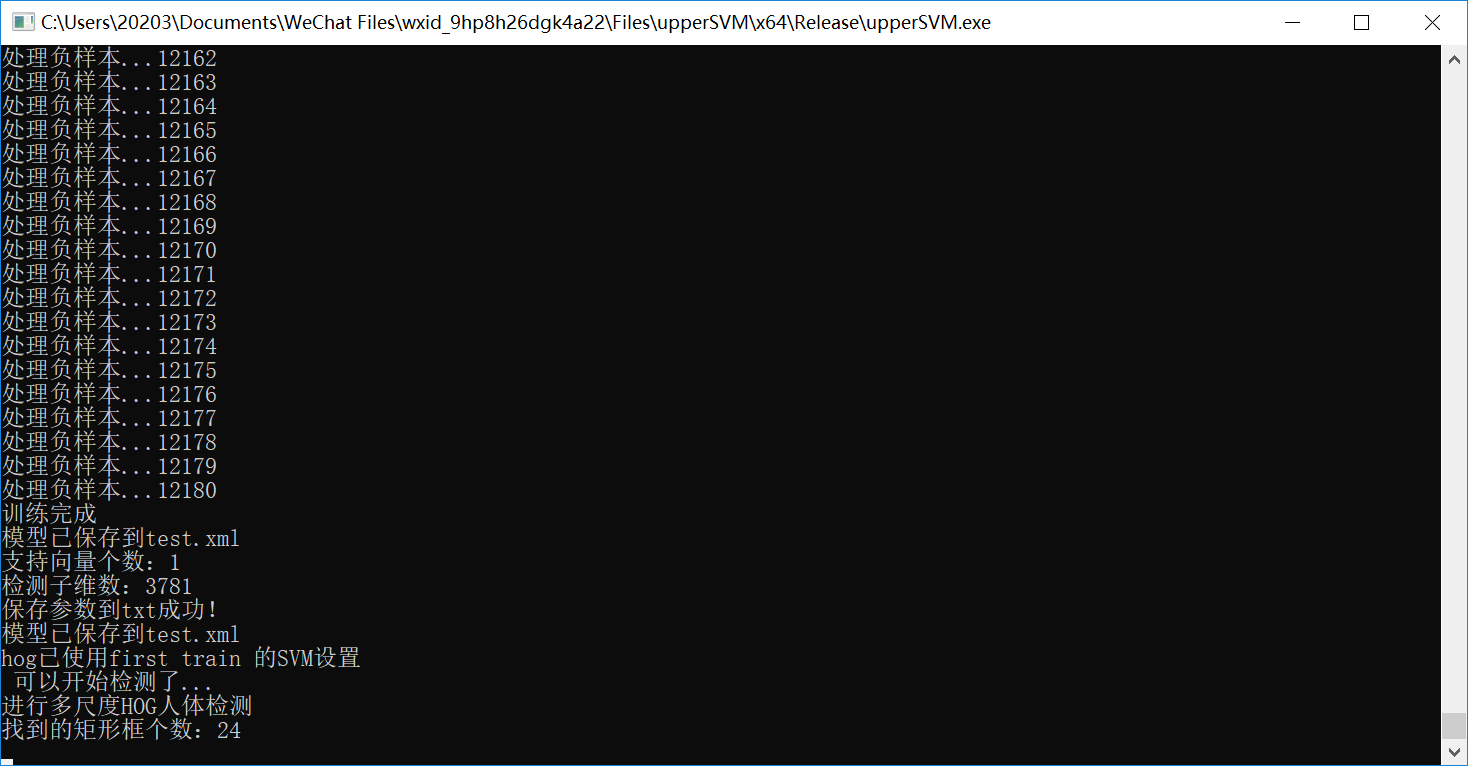
在测试正样本集上的结果：



在测试负样本集上的结果：



## 一些截图





# 开发平台 与 使用

## Visual Studio

检测器的训练是在visual studio IDE下完成的

## Different Compilers may not Interoperate

Object files and static libraries created with different compilers, or even with significantly different releases of the same compiler, often cannot be linked together. This issue is not specific to MinGW: many other compilers are mutually incompatible. Build everything from source with the same version of the same compiler if you can.

Dll's are slightly different. Sometimes you can link a DLL built with one compiler to an application compiled with another. This works well if the DLL is written in C, even if the application is written in C++. For example, MinGW C++ programs commonly link to the C runtime library provided with Windows. DLLs that are written in C++ work too, as long as you communicate with them only through a C interface declared with *extern "C"*. If you do otherwise, you will probably get linker errors because different compilers mangle C++ names differently.

## QT

### What is Qt?

Qt is a cross-platform application development framework for desktop, embedded and mobile. Supported Platforms include Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS and others.

Qt is not a programming language on its own. It is a framework written in C++. A preprocessor, the MOC (Meta-Object Compiler), is used to extend the C++ language with features like signals and slots. Before the compilation step, the MOC parses the source files written in Qt-extended C++ and generates standard compliant C++ sources from them. Thus the framework itself and applications/libraries using it can be compiled by any standard compliant C++ compiler like Clang, GCC, ICC, MinGW and MSVC.

### Widgets

With Qt, GUIs can be written directly in C++ using its Widgets module. Qt also comes with an interactive graphical tool called Qt Designer which functions as a code generator for Widgets based GUIs. Qt Designer can be used stand-alone but is also integrated into Qt Creator.

### Build System

Although any build system can be used with Qt, Qt brings its own qmake. It is a cross-platform frontend for platform-native build systems, like GNU Make, Visual Studio and Xcode.

CMake is also a popular alternative to build Qt projects, Qt 4 support has been integrated years ago and Qt 5 provided support early.

A new player entered the game recently: The Qt Build Suite a.k.a Qbs. Qbs is a QML based build system which also provides support for Javascript. This build system not only provides building capability but also packaging like cmake.

### STL 与 qt中的string转换

const char \* 数组和 STL的转换, 这两者的转换最为常见.

const char \* => STL 我一般直接使用

|  |  |
| --- | --- |
| 1  2 | const char \* s = "hello , world!";  string str(s); |

直接赋值完毕了.

如果是作为一个参数,还可以直接用调用string的构造函数.

|  |  |
| --- | --- |
| 1  2  3 | const char \* s = "hello const char";  string s1("hello , string1 ");  string s2 = s1 + string(s); |

s2就这样轻松搞定.

需要注意的是, string 的拼接不建议用上述的写法, 若条件允许, 写成这样会更好一些:

|  |  |
| --- | --- |
| 1  2  3 | const char \* s = "hello const char";  string s1("hello , string1 ");  s1 += string(s); |

string 到const char \*就更加简单了

|  |  |
| --- | --- |
| 1  2 | string str("hello world !");  const char \* s = str.c\_str(); |

不过需要注意的是,这个当前stack下只能被调用一次,就会被系统自动释放掉. 当然,你可以作为参数传入某个function,然后被调用很多遍,然后一return这个s就不能再用了. 所以这个 .c\_str() 似乎专门就是为了传参时候的类型转换用的.

QString这个第三者的加入 const char \* => QString, QString当然不敢不支持. 和STL的string一样,

|  |  |
| --- | --- |
| 1  2  3 | const char \* s = "hello , world !";  QString qs1(s);  QString qs2 = qs1.append(QString(s)); |

两种方法都是可以的.

如果需要特定编码的话,还可以

|  |  |
| --- | --- |
| 1  2 | QString qs3= QString(QLatin1String(s));  QString qs4 = QString::fromLocal8Bit(s); |

而QString=> \*有很多方法.最直白的是这样的

|  |  |
| --- | --- |
| 1  2  3  4 | QString qs1= "Hello , world !";  QByteArray ba = qs1.toLocal8Bit();  const char \*s = ba.data();  const char \*s2 = ba.constData(); |

当然还可以这样:

|  |  |
| --- | --- |
| 1  2  3 | QString qs1= "Hello , world !";  string str = qs1.toStdString();  const char \*s= str.c\_str(); |

其实就是先转为 STL 的 string 再转为 const char \*.

### QString

一、字符串连接函数。  
1、QString也重载的+和+=运算符。这两个运算符可以把两个字符串连接到一起。  
      
2、QString的append()函数则提供了类似的操作，例如：

   str = "User: ";    
   str.append(userName);    
   str.append("\n");

二、获取字符串某位置的值。

//

QString中有没有函数能截取出两个指定位置之间的字符串？

    QString x = "Nine pineapples";    
    QString y = x.mid(5, 4);            // y == "pine"    
    QString z = x.mid(5);               // z == "pineapples"

1、mid()函数接受两个参数，第一个是起始位置，第二个是取串的长度。如果省略第二个参数，则会从起始位置截取到末尾。正如上面的例子显示的那样

2、函数left()和rigt()类似，都接受一个int类型的参数n，都是对字符串进行截取。不同之处在于，left()函数从左侧截取n个字符，而right()从右侧开始截取。下面是left()的例子：

   1. QString x = "Pineapple";    
   2. QString y = x.left(4);      // y == "Pine"

三、获取字符在字符中的位置。  
     indexOf()  
   //存在lastIndexOf（）函数，返回字符串的最后一次出现的索引?    
   QString x = "sticky question";    
   QString y = "sti";    
   x.indexOf(y);               // returns 0    
   x.indexOf(y, 1);            // returns 10    
   x.indexOf(y, 10);           // returns 10    
   x.indexOf(y, 11);           // returns -1

四、可以检测字符串是不是以某个特定的串开始或结尾。  
    startsWith()    endsWith()  
      
      
    if (url.startsWith("http:") && url.endsWith(".png"))    
         {  }

    这段代码等价于

   if (url.left(5) == "http:" && url.right(4) == ".png")   
         {  }

五、

     字符串替换函数replace();  
     trimmed()函数去除字符串两侧的空白字符(注意，空白字符包括空格、Tab以及换行符，而不仅仅是空格);  
     toLower()和toUpper()函数会将字符串转换成小写大写字符串；  
     remove()和 insert()函数提供了删除和插入字符串的能力;  
     simplified()函数可以将串中的所有连续的空白字符替换成一个，并且把两端的空白字符去除，例如"   \t   ”会返回一个空格" "。

六、const char \*类型的C风格字符串与QString字符之间的转换。

     简单来说，QString的+=即可完成这个功能：  
     str += " (1870)";

     这里，我们将const char \* 类型的字符串" (1870)"转换成为QString类型。  
     如果需要显式的转换，可以使用QString的强制转换操作，或者是使用函数fromAscii()等。  
     为了将QString类型转成const char \*字符串，需要进行两步操作，一是使用toAscii()获得一个QByteArray类型对象，  
     然后调用它的data()或者constData()函数，  
     例如：

     printf("User: %s\n", str.toAscii().data());

     为了方便使用，Qt提供了一个宏qPrintable()，这个宏等价于toAscii().constData()，例如：

      printf("User: %s\n", qPrintable(str));   
        
      我们调用QByteArray类上面的data()或者constData()函数，将获得QByteArray 内部的一个const char\*类型的字符串，  
      因此，我们不需要担心内存泄漏等的问题，Qt会替我们管理好内存。不过这也暗示我们，注意不要使用这个指针太长时间，  
      因为如果 QByteArray被delete，那么这个指针也就成为野指针了。如果这个QByteArray对象没有被放在一个变量中，  
      那么当语句结束后，QbyteArray对象就会被delete，这个指针也就被delete 了。

七、字符串与其它类型的转换函数。  
     toInt()     转整型  
     toDouble()  转双精度型  
     toLong()    转长整型  
     这些函数都接受一个bool指针作为参数，函数结束之后将根据是否转换成功设置为true或者false：

     bool ok;    
     double d = str.toDouble(&ok);    
     if(ok)     
     {    
         // do something...    
     } else {    
          // do something...    
     }

     整型转字符串型：  
         1、使用static的函数number()可以把数字转换成字符串。例如：  
             QString str = QString::number(54.3);   
         2、 也可以使用非static函数setNum()来实现相同的目的：  
              QString str;    
              str.setNum(54.3);

八、QString则提供了一个sprintf()函数实现了与C语言中的printf函数相同的功能。  
      
    1. str.sprintf("%s %.1f%%", "perfect competition", 100.0);   
       这句代码将输出：perfect competition 100.0%  
         
    2、另一种格式化字符串输出的函数arg():  
   
        str = QString("%1 %2 (%3s-%4s)").arg("permissive").arg("society").arg(1950).arg(1970);

{

QString str;

str ="%1 %2";

str.arg("%1f","Hello"); // returns "%1f Hello"

str.arg("%1f").arg("Hello"); // returns "Hellof %2"

}

      这段代码中，%1, %2, %3, %4作为占位符，将被后面的arg()函数中的内容依次替换，比如%1将被替换成permissive，  
      %2将被替换成society，%3将被替换成 1950，%4将被替换曾1970，最后，  
      这句代码输出为：permissive society (1950s-1970s). arg()函数比起sprintf()来是类型安全的，  
      同时它也接受多种的数据类型作为参数，因此建议使用arg()函数而不是传统的 sprintf()。

九、求字符串的长度，返回值为INT型。  
     length();

十、如何正确显示中文字符

如果使QT根据Locale的环境变量取得字符集，可以使用如下命令：  
QString::fromLocal8Bit("你好，世界！");

使用QString的fromLocal8Bit()函数；  
  
  
QString str;  
  
  
str = str.fromLocal8Bit("哈哈哈");

## opencv

### Opencv中的String

According to the [document](http://docs.opencv.org/master/d1/d8f/classcv_1_1String.html#aba22f87619091a1b6a2899d0b45a3472), cv::String has a constructor taking one std::string as its parameter, which means std::string could be implicitly converted to cv::String.

std::string filename = "Path/to/img.jpg";

cv::Mat img = cv::imread(filename);

works fine