第 三 次 实 验 报 告

课程名称	内容安全实验				
学生姓名	陈曦	学号	2020302181081	指导老师	张典
专业	网络安全	班级	2020 级 3 班	实验时间	2023.4.10

一、实验内容

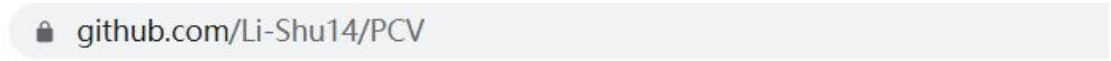
1. 任选图片，实现仿射变换。
2. 任选图片，提取颜色直方图特征，完成图像的匹配、查找。
3. 理解 HOG 算法，使用 SVM\KNN 完成目标行人检测，并阐述算法原理。
4. 理解 SIFT 算法，完成图片匹配任务，并阐述算法原理。
5. 在本地虚拟环境中部署 yolo 模型（v3-v8 任选），完成图像目标检测任务，并阐述所选用的模型原理。
6. 在实验 5 的基础上，添加可视化 UI，做成一个小工具，开源。

二、实验原理

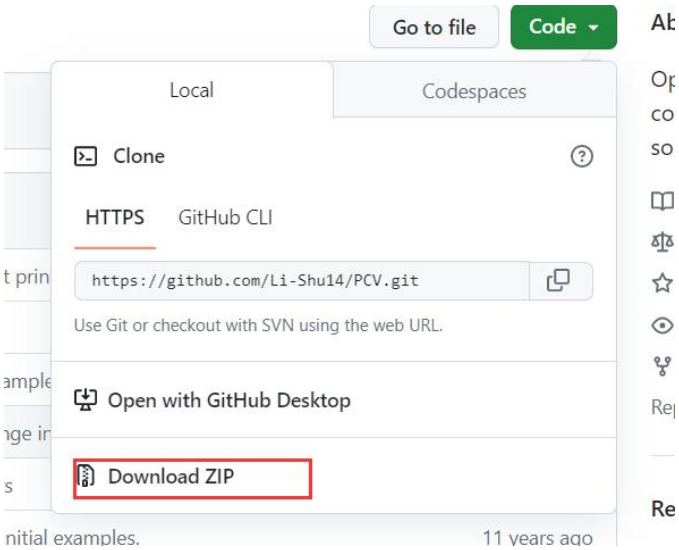
使用 opencv 工具来对图像进行各种操作和检测。

1. Opencv 库

正常 pip 安装失败。从 github 中安装。



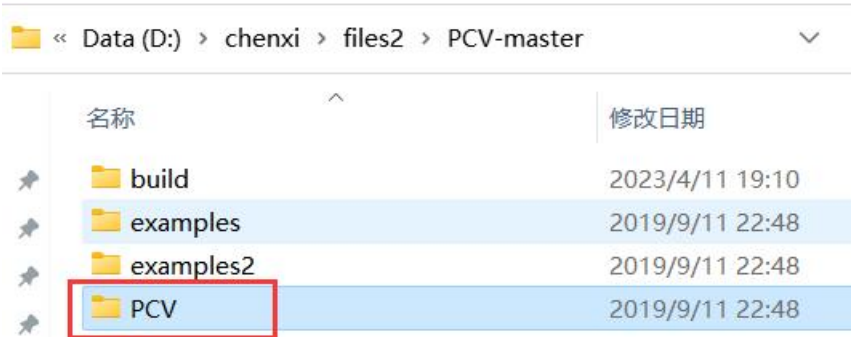
下载工具压缩包。



解压后，在命令行中执行安装程序。

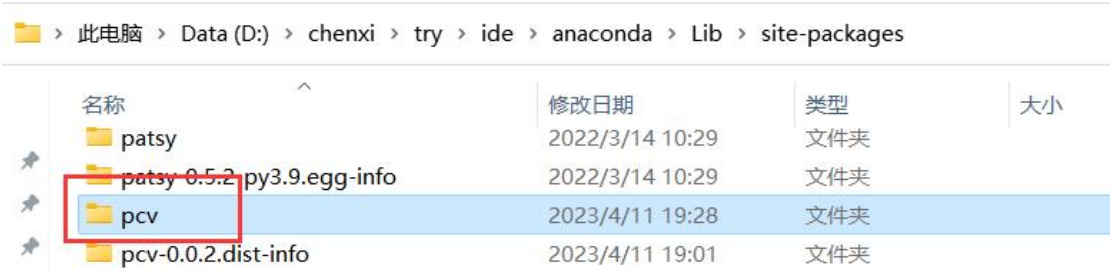
```
D:\chenxi\files2\PCV-master>python setup.py install
running install
running build
running build_py
creating build
creating build\lib
creating build\lib\PCV
copying PCV\__init__.py -> build\lib\PCV
creating build\lib\PCV\classifiers
copying PCV\classifiers\bayes.py -> build\lib\PCV\classifiers
copying PCV\classifiers\knn.py -> build\lib\PCV\classifiers
copying PCV\classifiers\__init__.py -> build\lib\PCV\classifiers
creating build\lib\PCV\clustering
copying PCV\clustering\hcluster.py -> build\lib\PCV\clustering
copying PCV\clustering\__init__.py -> build\lib\PCV\clustering
```

可以看到安装成功。



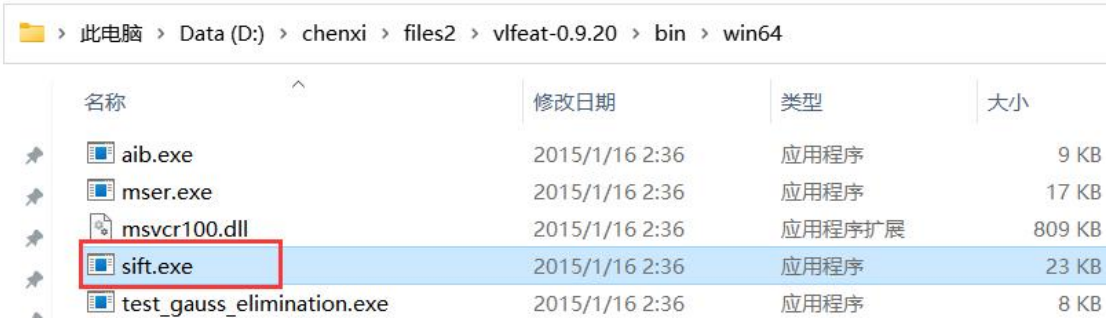
将 PCV 文件夹复制，准备安装到 anaconda 库文件夹中。

安装到 anaconda 库文件夹。路径为 anaconda/Lib/site-pachages。

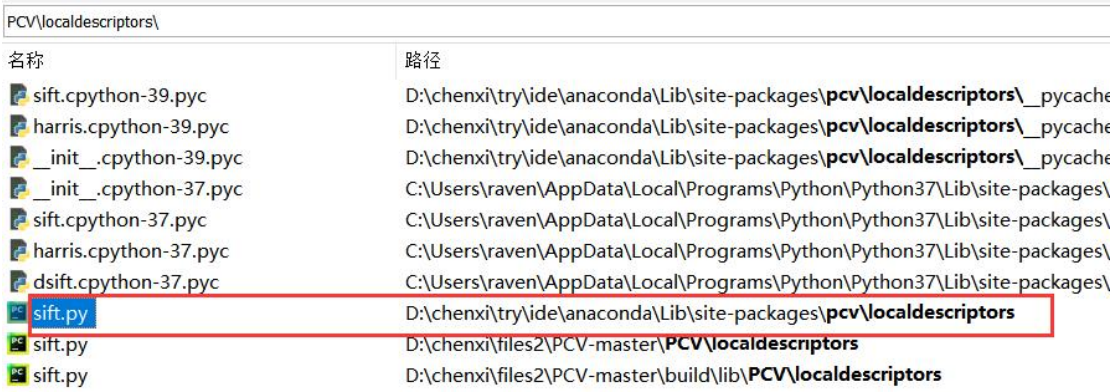


还缺乏 vlfeat 可执行文件。在官网中下载该工具压缩包。

解压，并将它复制到 pcv\localdescriptors 文件夹中。



查看 sift.py 程序所在的路径。



打开文件并编辑，将查询到的路径放入 cmmd 参数后面的路径中。

```
cmmd = str("D:\chenxi\files2\vlfeat-0.9.20\bin\win64\sift.exe "+imagename+" --output="+resultname+
       " "+params)
```

2. 基本思路

OpenCV 是一种开源计算机视觉库，它提供了一些用于处理图像和视频的工具和算法。OpenCV 最初是由英特尔公司开发的，现在已经成为了一种广泛使用的开源工具，它被用于各种各样的计算机视觉应用程序中，包括人脸检测、目标识别、图像分割、机器人视觉、运动跟踪、手势识别等。

OpenCV 提供了许多用于图像处理和计算机视觉的功能，包括图像滤波、特征检测、特征匹配、图像分割、摄像机校准、三维重建等。它还提供了 C++、Python 和 Java 等编程语言的接口，方便用户在各种平台上进行开发。

Opencv 用途举例：

图像处理：OpenCV 提供了一系列用于处理图像的函数和算法，包括图像滤波、图像变换、图像分割、形态学操作等。

特征检测与描述：OpenCV 提供了用于检测和描述图像中的特征的函数和算法，例如 SIFT、SURF、ORB、FAST 等。

目标检测与识别：OpenCV 提供了用于目标检测和识别的函数和算法，例如 Haar Cascade 分类器、HOG+SVM 分类器等。

视频处理：OpenCV 提供了一些用于处理视频的函数和算法，包括视频捕捉、视频压缩、视频分析等。

机器学习：OpenCV 提供了一些用于机器学习的函数和算法，例如支持向量机、神经网络、K 均值聚类等。

三、实验步骤

1. 任选图片，实现仿射变换

导入工具包 cv2，numpy。

```
import numpy as np
import cv2 as cv
```

调整目的图片格式。将目的图片命名为 titanic.jpg 存入和代码同一个目录下，使用 cv 工具将图片灰化并存入变量 img0，再将此图片缩小。

```
img0 = cv.imread('titanic.jpg', 0)
img = cv.resize(img0, None, fx=0.5, fy=0.5, interpolation=cv.INTER_CUBIC)
rows, cols = img.shape
```

将图片进行仿射变化。M 为变化矩阵，使用 warpAffine 函数利用变化矩阵变化图片。此变换图片存为 dst1。

```
M = np.float32([[1, 0, 100], [0, 1, 50]])
dst1 = cv.warpAffine(img, M, (cols, rows))
```

使用 getRotationMatrix2D 函数可以获取旋转变换矩阵。再利用这个变换矩阵对图像进行放射变换。此变换图片存为 dst2。

```
M = cv.getRotationMatrix2D(((cols-1)/2.0,(rows-1)/2.0),90,1)
dst2 = cv.warpAffine(img, M,(cols,rows))
```

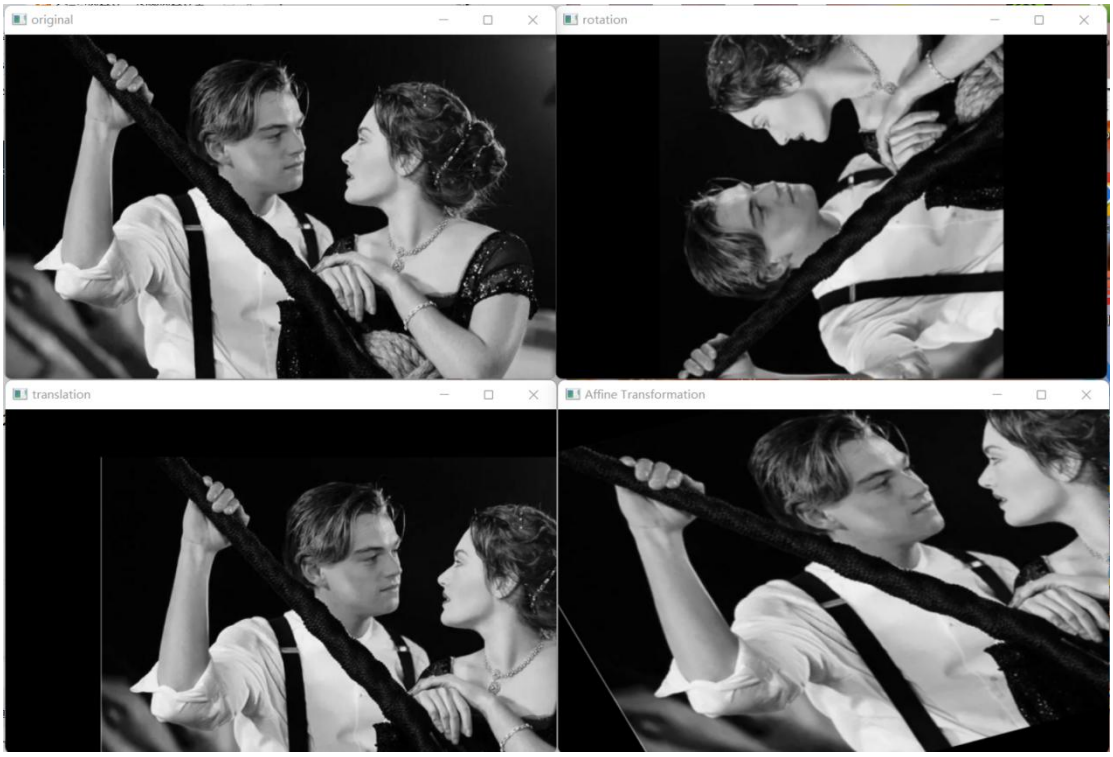
getAffineTransform 函数通过确认源图像中不在同一直线的三个点对应的目标图像的位置，来获取对方仿射变换矩阵，从而用该仿射变换矩阵对图像进行统一的仿射变换。

```
pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])
M = cv.getAffineTransform(pts1,pts2)
dst3 = cv.warpAffine(img, M, (cols, rows))
```

使用 imshow 函数将图片展示。包括原图和 3 次仿射变换。waitKey 和 destroyAllWindows 是使图片停留直到图片被手动关闭。

```
cv.imshow('original',img)
cv.imshow('rotation',dst2)
cv.imshow('translation',dst1)
cv.imshow('Affine Transformation', dst3)

cv.waitKey(0)
cv.destroyAllWindows()
```

实验结果如下所示。



2. 任选图片，提取颜色直方图特征，完成图像的匹配、查找

【任选图片，提取颜色直方图特征】

选择图片保存为 vol.png 在代码文件夹下。原图如下图所示。

代码 main2.py 完成了提取图片直方图特征。导入工具 cv2 和 numpy。

```
1.  import cv2
2.  import numpy as np
```

用函数实现提取图片的 RGB 颜色特征数据。Opencv 提供了 calcHist 函数来计算图像直方图。calcHist 函数说明如下：

```
void calcHist( const Mat* images, int nimages,
                           const int* channels, InputArray mask,
                           OutputArray hist, int dims, const int*
histSize,
                           const float** ranges, bool uniform=true, bool
accumulate=false );
```

参数	参数解释
images	输入的图像的指针
nimages	输入图像个数
channels	需要统计直方图的第几通道
mask	必须是一个 8 位数组并且和 images 的数组大小相同
hist	直方图计算的输出值
dims	输出直方图的维度
histSize	直方图中每个 dims 维度需要分成多少个区间
ranges	统计像素值的区间
uniform = true	是否对得到的直方图数组进行归一化处理
accumulate = false	在多个图像时，是否累积计算像素值的个数

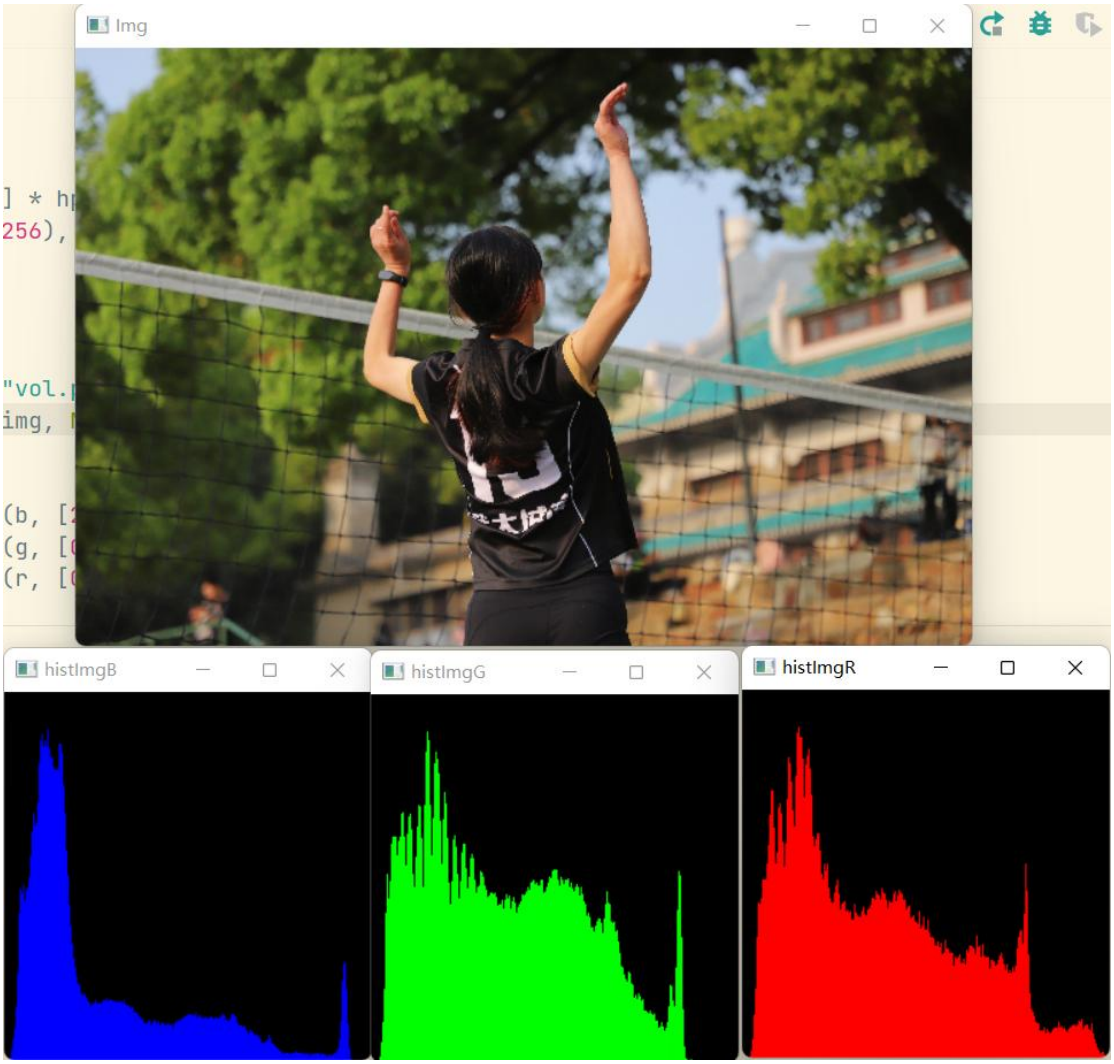
```
1.  def calcAndDrawHist(image, color):
2.      hist = cv2.calcHist([image], [0], None, [256], [0.0, 255.0])
3.      minVal, maxVal, minLoc, maxLoc = cv2.minMaxLoc(hist)
4.      histImg = np.zeros([256, 256, 3], np.uint8)
5.      hpt = int(0.9 * 256)
6.
7.      for h in range(256):
8.          intensity = int(hist[h] * hpt / maxVal)
9.          cv2.line(histImg, (h, 256), (h, 256 - intensity), color)
10.     return histImg
```

程序运行入口，将 vol.png 导入，使用 opencv 的 split 函数，将一个图像数组拆分为 RGB 三个通道。并将三个通道使用上方的函数生成直方图。最后输出原图和三个直方图。

```
1.  if __name__ == '__main__':
2.      original_img = cv2.imread("vol.png")
3.      img = cv2.resize(original_img, None, fx=0.15, fy=0.15, interpol
    ation=cv2.INTER_CUBIC)
```

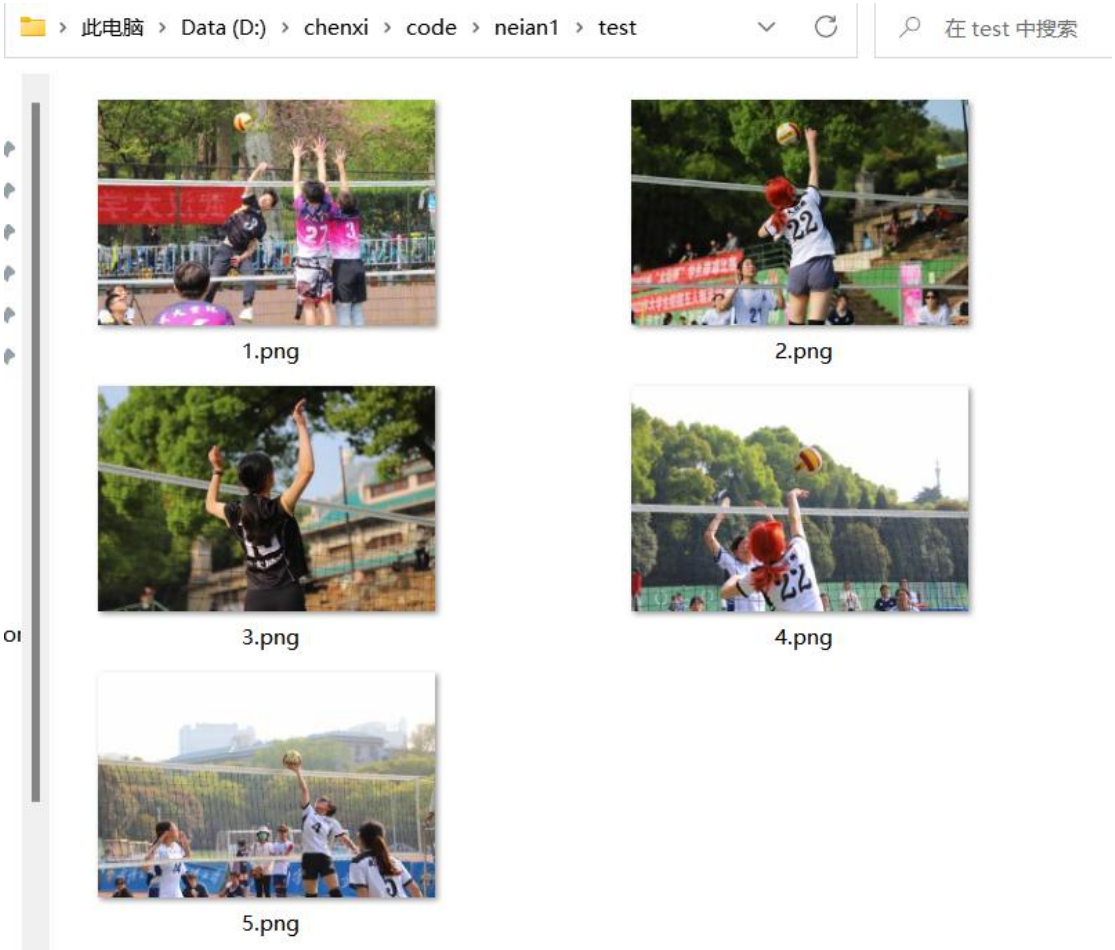
```
    b, g, r = cv2.split(img)

    histImgB = calcAndDrawHist(b, [255, 0, 0])
    histImgG = calcAndDrawHist(g, [0, 255, 0])
    histImgR = calcAndDrawHist(r, [0, 0, 255])

    cv2.imshow("histImgB", histImgB)
    cv2.imshow("histImgG", histImgG)
    cv2.imshow("histImgR", histImgR)
    cv2.imshow("Img", img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

代码运行结果：



【根据直方图特征完成图片的匹配、查找】

代码 main2.2py 实现了根据直方图来在一个图片文件夹中寻找目标图片，即通过遍历文件夹中的每一个图片，计算和目标图片的直方图数据相似度，来判断哪一个图片是目标图片，或者与目标图片最为类似。
在和代码同一个文件夹下创建文件夹 test，其中放置了 5 张图片。代码将从这个文件夹中，查找与目标照片相同的照片。



程序编写如下。
导入工具 cv2，os，matplotlib。需要加上 matplotlib.usee（‘TkAgg’）。

```
import cv2
import os
import numpy as np
from PIL import Image
import requests
from io import BytesIO
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
```

编写计算照片相似度的函数 calculate。使用 calcHist 函数计算两个图片的相似度。

```
def calculate(img1, img2):
    hist1 = cv2.calcHist([img1], [0], None, [256], [0.0, 255.0])
    hist2 = cv2.calcHist([img2], [0], None, [256], [0.0, 255.0])
    degree = 0
    for i in range(len(hist1)):
        if hist1[i] != hist2[i]:
            degree = degree + \
                 (1 - abs(hist1[i] - hist2[i]) / max(hist1[i], hist2[i])
    )
        else:
            degree = degree + 1
    degree = degree / len(hist1)
    return degree
```

使用 cmp 函数比较两个图片数组的相似度，得出数值。

```
def cmp(img1, img2):
    p1 = cv2.split(img1)
    p2 = cv2.split(img2)
    delta = 0
    for i, j in zip(p1, p2):
        delta += calculate(i, j)
    delta = delta / 3
    return delta
```

执行函数入口。在文件夹里遍历，计算每一个图片和目标图片的相似度。最后展示处相似度最大的图片。并打印相似数值。

```
if __name__ == "__main__":
    res = []
    mid = 0
    file_list = os.listdir('test')
    for file in file_list:
        p1 = 'vol.png'
        p2 = 'test' + '/' + file
        img1 = cv2.imread(p1)
        img2 = cv2.imread(p2)
        mid = cmp(img1, img2)
        if mid==1:
            img = cv2.resize(img2, None, fx=0.15, fy=0.15, interpolatio
n=cv2.INTER_CUBIC)
            cv2.imshow('the same', img)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
```

```
            res.append(mid)
        print(res)
```

执行函数结果，跳出与目标文件相似的图片。

可以看到终端中，第三种图片的相似度为 1.0。

```
D:\chenxi\try\ide\anaconda\python.exe D:/chenxi/code/neian1/main2.2.py
[array([0.49169385], dtype=float32), array([0.6021939], dtype=float32), 1.0, array([0.5536775], dtype=float32), array([0.42397285], dtype=float32)]

进程已结束,退出代码0
```

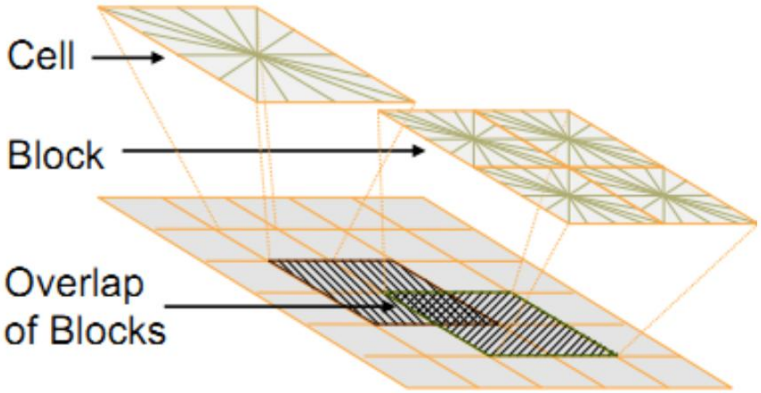
3. 理解 HOG 算法，使用 SVM\KNN 完成目标行人检测

【HOG 算法理解】

HOG（方向梯度直方图）是用于在计算机视觉和图像处理领域，目标检测的特征描述子。该项技术是用来计算图像局部出现的方向梯度次数或信息进行计数。此种方法跟边缘方向直方图、尺度不变特征变换以及形状上下文方法有很多相似。但与它们的不同点是：HOG 的计算基于一致空间的密度矩阵来提高准确率。即：在一个网格密集的大小统一的细胞单元上计算，而且为了提高性能，还采用了重叠的局部对比度归一化技术。HoG 特征与 SVM 分类器结合，已经被广泛应用于图像识别中，尤其在行人检测。

HOG 的核心思想是所检测的局部物体外形能够被光强梯度或边缘方向的分布所描述。通过将整幅图像分割成小的连接区域称为 cells，每个 cell 生成一个方向梯度直方图或者 cell 中 pixel 的边缘方向，这些直方图的组合可表示出所检测目标的目标）描述子。为改善准确率，局部直方图可以通过计算图像中一个

较大区域称为 block 的光强作为 measure 被对比标准化，然后用这个 measure 归一化这个 block 中的所有 cells. 这个归一化过程完成了更好的照射/阴影不变性。与其他描述子相比，HOG 得到的描述子保持了几何和光学转化不变性除非物体方向改变。而 Block 与 Cells 关系如下图所示。



使用 SRM 进行行人检测。代码如下。
下载行人监测数据集 INRIA 。

首先，INRIA 数据集的原始图片，来自 GRAZ 01 数据集和网络上的一些图片。这些图片相比于 MIT 行人数据集，人的姿态和光照条件啥的都更加全面，适合做行人检测。每张图片中都进行了行人区域的标定，也就是画了一个矩形框，把矩形框的左上定点坐标和矩形长度、宽度记录下来。
数据集结构如下所示（注意下面不是代码，是结构示意图）。

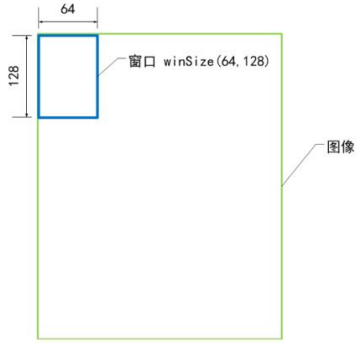
```
INRIADATA{
    normalized_images{
        train{
            pos:96x160大小，训练正样本，需要crop中间的64x128大小。已经做过flip，即包含左右对称的图
            neg:大小不一，通常是几百乘几百，训练负样本，需要从每张图中随机crop 10个区域作为训练负样本
        }
    }
    original_images{
        train{
            pos:训练正样本，大小不一
            neg:训练负样本，大小不一
            annotations:标注信息
        }
        test{
            pos:大小不一
            neg:大小不一
            annotations:标注信息
        }
    }
}
```

【使用 SVM 完成目标行人检测】

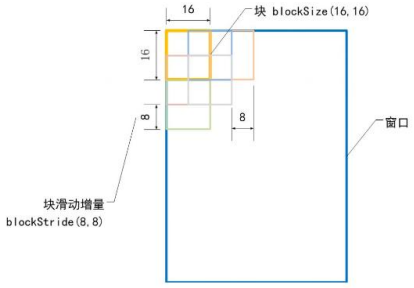
编写 hog_descriptor 函数，提取图片的 HOG 特征。
使用 opencv 的 HOGDescriptor 的函数。

```
1.   CV_WRAP HOGDescriptor() : winSize(64,128), blockSize(16,16), bloc
   kStride(8,8),
2.       cellSize(8,8), nbins(9), derivAperture(1), winSigma(-1),
3.       histogramNormType(HOGDescriptor::L2Hys), L2HysThreshold(0.2),
   gammaCorrection(true)
4.       nlevels(HOGDescriptor::DEFAULT_NLEVELS)
5.   {}
```

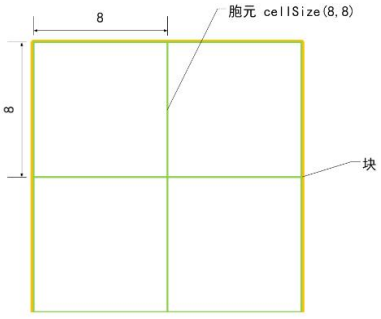
几个重要参数：
winSize(64,128)， blockSize(16,16)， blockStride(8,8)， cellSize(8,8)，nbins(9)
窗口大小 winSize：



块大小 blockSize：



块移动的步长：blockStride(8,8)
这里是指的块移动的步长，括号中分别为 X 方向和 Y 方向块移动的步长。
胞元大小：cellSize



梯度方向数 nbins:
表示在一个胞元（cell）中统计梯度的方向数目，例如 nBins=9 时，在一个胞元内统计 9 个方向的梯度直方图，每个方向为 180/9=20 度。

```
def hog_descriptor(image):
    if (image.max()-image.min()) != 0:
        image = (image - image.min()) / (image.max() - image.min())
        image *= 255
        image = image.astype(np.uint8)
    hog = cv2.HOGDescriptor((64, 128), (16, 16), (8, 8), (8, 8), 9)
    hog_feature = hog.compute(image)
    return hog_feature
```

接下来展示具体的算法如何实现。编写函数 mynms，这里阈值选择的是 0.8。

```
def mynms(box_list,prob_list,threshold=0.8):
    x1 = box_list[:,0]
    y1 = box_list[:,1]
    x2 = box_list[:,2]
    y2 = box_list[:,3]
    areas = (x2-x1+1)*(y2-y1+1)
    box_result = []
    flag = []
    index = prob_list.argsort()[::-1] #想要从大到小排序
    while index.size>0:
        i = index[0]
        flag.append(i)
        x11 = np.maximum(x1[i], x1[index[1:]]) # calculate the points of overlap
        y11 = np.maximum(y1[i], y1[index[1:]])
        x22 = np.minimum(x2[i], x2[index[1:]])
        y22 = np.minimum(y2[i], y2[index[1:]])
        w = np.maximum(0, x22 - x11 + 1)
        h = np.maximum(0, y22 - y11 + 1)
        overlaps = w * h
        ious = overlaps / (areas[i] + areas[index[1:]] - overlaps)
        #idx = np.where(ious < threshold)[0]
        index = np.delete(index,np.concatenate(([0],np.where(ious < threshold)[0])))
        #index = index[idx + 1]
    return box_list[flag].astype("int")
```

通过训练集中的标准的 box 大小计算出适合的 box 大小。首先是编写用于提取 HOG 特征的函数，这个已经在前文中讲述过了。再就是要通过原数据集的 annotations 文件夹中的关于训练集正样本图像中行人的大小计算适合的 window 大小。

```
box_list = []
ANN = os.listdir('D:/chenxi/files2/pedestrian/INRIADATA/original_images/train/annotations')
flag = "(Xmax, Ymax)"
for i in range(len(ANN)):
    for line in open('D:/chenxi/files2/pedestrian/INRIADATA/original_images/train/annotations/'+ANN[i], encoding="GBK"):
        if flag in line:
            boxsize = line.split(flag)
            boxsize = str(boxsize[1])
            boxsize = boxsize.replace("(", "")
            boxsize = boxsize.replace(",", "")
            boxsize = boxsize.replace(")", "")
            boxsize = boxsize.replace("-", "")
            boxsize = boxsize.replace(":", "")
            boxsize = boxsize.split()
            box = (float(boxsize[2])-float(boxsize[0]) ,float(boxsize[3])-float(boxsize[1]))
            box_list.append(box)
box_list = np.array(box_list)
minlist = np.min(box_list,axis=0)
maxlist = np.max(box_list,axis=0)
minscale = min(minlist[0]/64,minlist[1]/128)
maxscale = min(maxlist[0]/64,maxlist[1]/128)
minscale = math.ceil(minscale)
maxscale = math.ceil(maxscale)
print(minscale)
print(maxscale)
```

接下来这段代码使用了一个支持向量机（SVM）模型来检测图像中的行人。首先，通过 joblib.load 函数从磁盘加载已经训练好的 SVM 模型，并将其存储在变量 clf 中。

```
imglist = os.listdir('D:/chenxi/files2/pedestrian/INRIADATA/original_images/test/pos')
```

然后，代码使用 os.listdir 函数读取一个文件夹中的所有文件名，并且循环遍历这些文件。在每个循环迭代中，代码使用 io.imread 读取一个图像，然后使用 cv2.cvtColor 将其从 RGBA 颜色空间转换为 BGR 颜色空间。

```
for i in range(101,len(imglist)):
    img = io.imread(osp.join('D:/chenxi/files2/pedestrian/INRIADATA/original_images/test/pos', imglist[i]))
    img = cv2.cvtColor(img, cv2.COLOR_RGBA2BGR)
    h,w,c= img.shape
```

```
    patch_list = []
    hog_feature = []
    box_list = []
    for j in range(minscale,maxscale+1,1):
      winsize = [j*64,j*128]
      for m in range(0,h-winsize[1],20):
        for n in range(0,w-winsize[0],20):
          patch = img[m:m+winsize[1],n:n+winsize[0]]
          patch = cv2.resize(patch, (64,128), interpolation = cv2.INTER_NEAREST)
          boxcoord = (m,n,m+winsize[1],n+winsize[0])
          hogfea = hog_descriptor(patch)
          hog_feature.append(hogfea)
          box_list.append(boxcoord)
          patch_list.append(patch)
```

接下来，代码将图像分割为一系列大小不同的矩形块，每个矩形块都被缩放为相同的大小（64x128 像素），并计算其 HOG 特征描述符。这些特征描述符被收集到一个 hog_feature 数组中，同时每个矩形块的坐标被收集到一个 box_list 数组中。

```
    hog_feature = np.array(hog_feature).squeeze()
    box_list = np.array(box_list)
```

然后，代码使用 clf.predict_proba 函数对这些 HOG 特征进行分类，并保留正类别的预测概率。只有预测概率大于等于 0.99 的矩形块才会被保留，这些矩形块的坐标被收集到一个 boxzhong 数组中，并且通过 mynms 函数进行非极大值抑制（NMS）处理。

```
    prob = clf.predict_proba(hog_feature)[:, 1]
    mask = (prob>=0.99)
    box_list = box_list[mask]
    prob = prob[mask]
    boxzhong = mynms(box_list,prob)
```

最后，代码使用 cv2.rectangle 函数在原始图像上绘制矩形框来标记被检测到的行人，并将结果保存到磁盘中。在每张图像处理完成后，代码会输出当前处理的图像编号，并且在处理完前 150 张图像后停止循环。
实验结果保存在路径 D:/chenxi/files2/namodel/result/下。

```
    for k in range(len(boxzhong)):
      cv2.rectangle(img, (boxzhong[k][1], boxzhong[k][0]), (boxzhong[k][3], boxzhong[k][2]),(0, 0, 255),3)
    cv2.imwrite('D:/chenxi/files2/namodel/result/'+imglist[i]+".jpg",img)
    print(str(i))
    if i == 150:
```

```
        break

print("结束")
```

编写 main3.2，这段代码是一个基于 HOG 特征和 SVM 分类器的行人检测模型的训练和测试代码。对数据进行处理，提取正负样本的 Hog 特征。计算传入图像的 HOG 特征。

导入工具。

```
import cv2
import os
import numpy as np
import os.path as osp
from skimage import io
import random
from sklearn import metrics
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import joblib
```

导入图像数据。代码首先导入训练和测试数据集中的图像。

同样编写函数提取 HOG 特征。

```
def hog_descriptor(image):
    if (image.max()-image.min()) != 0:
        image = (image - image.min()) / (image.max() - image.min())
        image *= 255
        image = image.astype(np.uint8)
    hog = cv2.HOGDescriptor((64, 128), (16, 16), (8, 8), (8, 8), 9)
    hog_feature = hog.compute(image)
    return hog_feature
```

导入图像，获得正样本和负样本的 HOG 特征，并标记。

```
poslist = os.listdir('D:/chenxi/files2/pedestrian/INRIADATA/normalized_images/train/pos')
neglist = os.listdir('D:/chenxi/files2/pedestrian/INRIADATA/normalized_images/train/neg')
testlist = os.listdir('D:/chenxi/files2/pedestrian/INRIADATA/normalized_images/test/pos')
testnlist = os.listdir('D:/chenxi/files2/pedestrian/INRIADATA/original_images/test/neg')
```

设置显示正样本图像和负样本原始图像的信息。

```
hog_list = []
label_list = []
print("正样本图像有"+str(len(poslist)))
```

```
print("负样本原始图像有"+str(len(neglist))+",每个原始图像提供十个负样本")
```

这段代码使用 Python 语言编写，它的作用是对一组图像进行 HOG 特征提取，并将特征和标签存储到两个不同的列表中，以便后续的机器学习模型训练和测试。

```
for i in range(len(poslist)):
    posimg = io.imread(osp.join('D:/chenxi/files2/pedestrian/INRIADATA/normalized_images/train/pos',poslist[i]))
    posimg = cv2.cvtColor(posimg,cv2.COLOR_RGBA2BGR)
    #所用图像已经经过标准化
    posimg = cv2.resize(posimg, (64, 128), interpolation=cv2.INTER_NEAREST)
    pos_hog = hog_descriptor(posimg)
    hog_list.append(pos_hog)
    label_list.append(1)
```

从一组负样本图像中随机截取多个小图像，并对每个小图像进行 HOG 特征提取，并将特征和标签存储到两个不同的列表中，以便后续的机器学习模型训练和测试。

```
for i in range(len(neglist)):
    negimg = io.imread(osp.join('D:/chenxi/files2/pedestrian/INRIADATA/normalized_images/train/neg',neglist[i]))
    negimg = cv2.cvtColor(negimg, cv2.COLOR_RGBA2BGR)

    #在每张 negimg 图像中截取 10 张标准大小的图片作为负样本
    for j in range(10):
        y = int(random.random() * (negimg.shape[0] - 128))
        x = int(random.random() * (negimg.shape[1] - 64))
        negimgs = negimg[y:y + 128, x:x + 64]
        negimgs = cv2.resize(negimgs, (64, 128), interpolation=cv2.INTER_NEAREST)
        neg_hog = hog_descriptor(negimgs)
        hog_list.append(neg_hog)
        label_list.append(0)
print(type(hog_list[10]))
print(type(hog_list[-10]))
hog_list = np.float32(hog_list)
label_list = np.int32(label_list).reshape(len(label_list),1)
```

将列表 hog_list 和 label_list 转换为 numpy 数组，并将它们的数据类型和形状调整为适合机器学习模型训练和测试的格式。

```
print(type(hog_list[10]))
print(type(hog_list[-10]))
hog_list = np.float32(hog_list)
label_list = np.int32(label_list).reshape(len(label_list),1)
```

训练 SVM，并在 Test 上测试。

```
clf = SVC(C=1.0, gamma='auto', kernel='rbf', probability=True)
clf.fit(hog_list.squeeze(), label_list.squeeze())
joblib.dump(clf, "D:/chenxi/files2/namodel/trained_svm.m")
```

对一组测试集正样本图像进行 HOG 特征提取，并将提取到的特征和标签存储到两个不同的列表中，以便后续的机器学习模型测试。

```
test_hog = []
test_label = []
for i in range(len(testlist)):
    testimg = io.imread(osp.join('D:/chenxi/files2/pedestrian/INRIADATA/normalized_images/test/pos', testlist[i]))
    testimg = cv2.cvtColor(testimg, cv2.COLOR_RGBA2BGR)
    testimg = cv2.resize(testimg, (64, 128), interpolation=cv2.INTER_NEAREST)
    testhog = hog_descriptor(testimg)
    test_hog.append(testhog)
    test_label.append(1)
```

对一组测试集负样本图像进行 HOG 特征提取，并将提取到的特征存储到一个列表中。具体来说，代码的第一行使用 for 循环遍历一个列表 testnlist，这个列表包含了一组测试集负样本图像文件的文件名。第二行代码使用 imread 函数从文件系统中读取一个图像，并将其存储到变量 testnegimg 中。第三行代码使用 cv2.cvtColor 函数将图像从 RGBA 格式转换为 BGR 格式，这是因为 HOG 特征提取函数 hog_descriptor 需要接受 BGR 格式的图像。

```
for i in range(len(testnlist)):
    testnegimg = io.imread(osp.join('D:/chenxi/files2/pedestrian/INRIADATA/original_images/test/neg',testnlist[i]))
    testnegimg = cv2.cvtColor(testnegimg, cv2.COLOR_RGBA2BGR)

    #在每张 negimg 图像中截取 10 张标准大小的图片作为负样本
    for j in range(10):
        y = int(random.random() * (testnegimg.shape[0] - 128))
        x = int(random.random() * (testnegimg.shape[1] - 64))
        testnegimgs = testnegimg[y:y + 128, x:x + 64]
        testnegimgs = cv2.resize(testnegimgs, (64, 128), interpolation=cv2.INTER_NEAREST)
        testneg_hog = hog_descriptor(testnegimgs)
        test_hog.append(testneg_hog)
        test_label.append(0)
test_hog = np.float32(test_hog)
test_label = np.int32(test_label).reshape(len(test_label),1)
#可以导入训练后的 SVM
clf = joblib.load("D:/chenxi/files2/namodel/trained_svm.m")
```

使用已经训练好的 SVM 分类器对测试集进行分类，并计算分类结果的精确率、召回率和阈值。具体来说，代码首先使用 predict_proba 函数对测试集进行分类，并将结果存储到 prob 变量中。predict_proba 函数返回一个二维数组，其中每行代表一个测试样本，第一列代表预测结果为负样本的概率，第二列代表预测结果为正样本的概率。由于我们只关心预测结果为正样本的概率，因此代码使用 [:, 1] 索引提取了第二列数据，并将其存储到 prob 变量中。
使用 precision_recall_curve 函数计算分类结果的精确率、召回率和阈值，并将结果分别存储到 precision、recall 和 thresholds_1 变量中。precision_recall_curve 函数的第一个参数是测试集的真实标签，第二个参数是分类器的预测概率。函数将返回三个数组，分别代表不同阈值下的精确率、召回率和阈值。

```
prob = clf.predict_proba(test_hog.squeeze())[:, 1]
precision, recall, thresholds_1 = metrics.precision_recall_curve(test_label.squeeze(), prob)
```

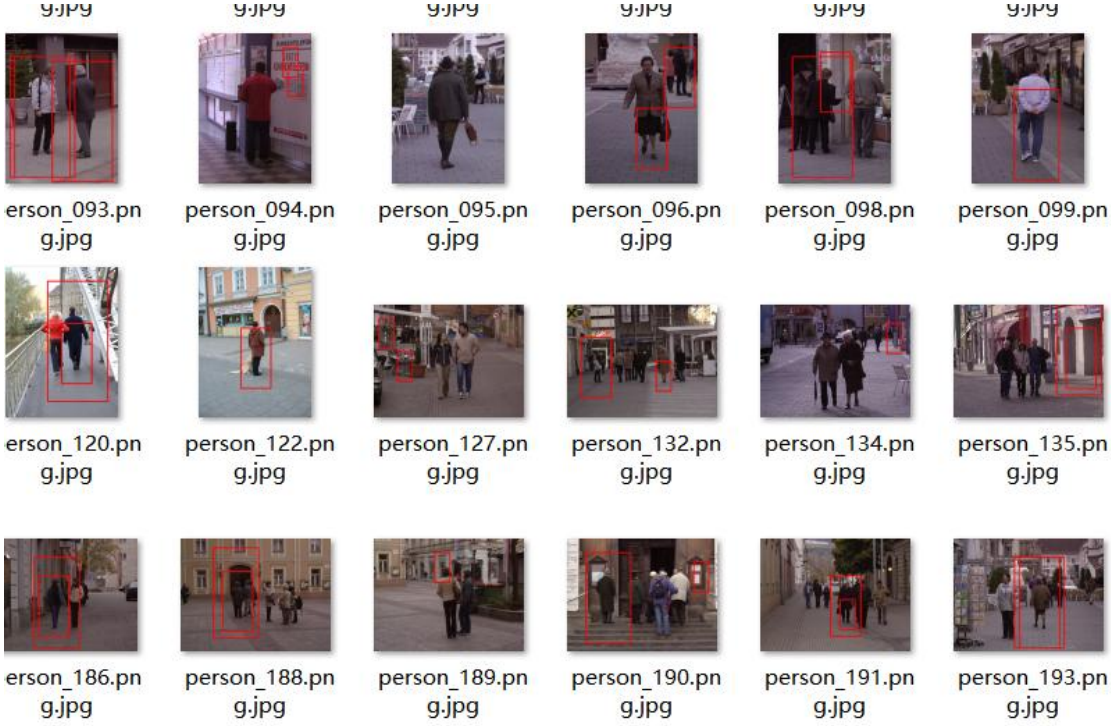
使用 Matplotlib 库绘制分类器的 PR 曲线和 ROC 曲线，并将结果保存到本地。使用 average_precision_score 函数计算 PR 曲线下的面积（Average Precision，简称 AP），并将结果存储到 Ap 变量中。

```
plt.figure(figsize=(20, 20), dpi=100)
plt.plot(precision, recall, c='red')
plt.scatter(precision, recall, c='blue')
plt.xlabel("precision", fontdict={'size': 16})
plt.ylabel("recall", fontdict={'size': 16})
plt.title("PR_curve", fontdict={'size': 20})
plt.savefig('D:/chenxi/files2/namodel/savefig/PR.png',dpi=300)
Ap=metrics.average_precision_score(test_label.squeeze(), prob)

fpr, tpr, thresholds_2 = metrics.roc_curve(test_label.squeeze(), prob, pos_label=1)

plt.figure(figsize=(20, 20), dpi=100)
plt.plot(fpr, tpr, c='red')
plt.scatter(fpr, tpr, c='blue')
plt.xlabel("FPR", fontdict={'size': 16})
plt.ylabel("TPR", fontdict={'size': 16})
plt.title("ROC_curve", fontdict={'size': 20})
plt.savefig('D:/chenxi/files2/namodel/savefig/ROC.png', dpi=300)
```

使用 roc_auc_score 函数计算 ROC 曲线下的面积，并将结果存储到 AUC 变量中。接着，打印 AUC 和 AP 的值。

```
AUC=metrics.roc_auc_score(test_label.squeeze(), prob)
print(AUC)
print(Ap)
```

执行代码，查看实验结果目录下的实验结果。可以看到行人被顺利检测。



4. 理解 SIFT 算法，完成图片匹配任务，并阐述算法原理

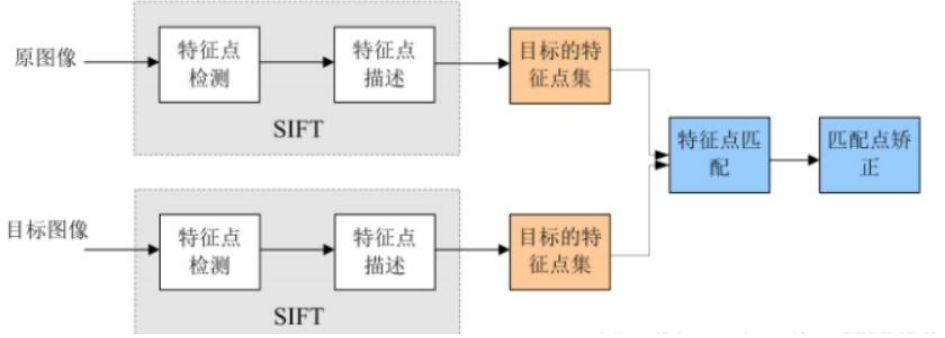
【SIFT 算法理解】

SIFT 算法实现流程：

1、提取关键点：关键点是一些十分突出的不会因光照、尺度、旋转等因素而消失的点，比如角点、边缘点、暗区域的亮点以及亮区域的暗点。此步骤是搜索所有尺度空间上的图像位置。通过高斯微分函数来识别潜在的具有尺度和旋转不变的兴趣点。

2、定位关键点并确定特征方向：在每个候选的位置上，通过一个拟合精细的模型来确定位置和尺度。关键点的选择依据于它们的稳定程度。然后基于图像局部的梯度方向，分配给每个关键点位置一个或多个方向。所有后面的对图像数据的操作都相对于关键点的方向、尺度和位置进行变换，从而提供对于这些变换的不变性。

3. 通过各关键点的特征向量，进行两两比较找出相互匹配的若干对特征点，建立景物间的对应关系。



【提取图像的 SIFT 特征和角点特征】

编写函数 main4.py 来对使用 SIFT 提取特征。该函数可以提取 SIFT 特征以及 harris 角点特征。

导入工具包。

```
from PIL import Image
from pylab import *
from pcv.localdescriptors import sift
from pcv.localdescriptors import harris
```

读入文件并将文件转换为 pmg 格式，进行特征特区。

```
imname = 'vol.png'
im = array(Image.open(imname).convert('L'))
sift.process_image(imname, 'empire.sift')
l1, d1 = sift.read_features_from_file('empire.sift')
```

在图像左侧绘制原始图像和 SIFT 特征点。

```
figure()
gray()
subplot(121)
sift.plot_features(im, l1, circle=False)
title(u'SIFT*')
```

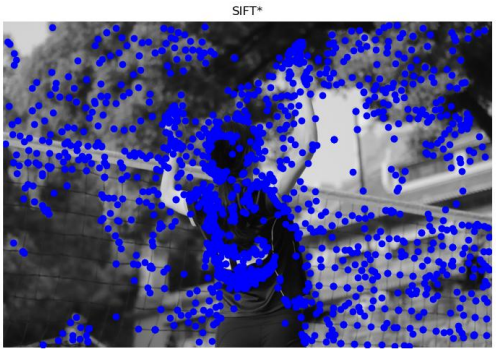
检测 harris 角点。

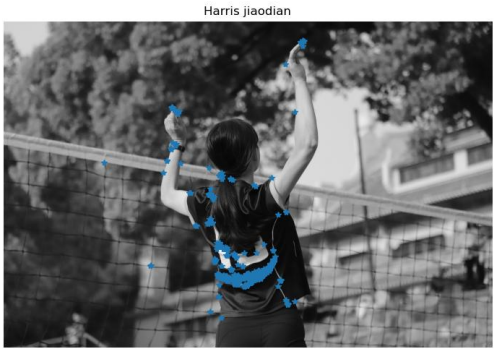
使用 Harris 角点检测算法计算原始图像的 Harris 响应值，并将结果存储到 harrisim 变量中，使用 get_harris_points 函数从 Harris 响应值中提取角点坐标，其中第二个参数为角点邻域的大小，第三个参数为角点阈值，返回的是一个列表，每个元素是一个包含角点坐标和 Harris 响应值的元组。使用 plot 函数在原始图像上绘制角点，显示图像。

在图像右侧绘制原始图像中的 Harris 角点，以便与 SIFT 特征点进行比较。

```
harrisim = harris.compute_harris_response(im)
subplot(122)
filtered_coords = harris.get_harris_points(harrisim, 6, 0.1)
imshow(im)
plot([p[1] for p in filtered_coords], [p[0] for p in filtered_coords], '*')
axis('off')
title(u'Harris jiaodian')
show()
```

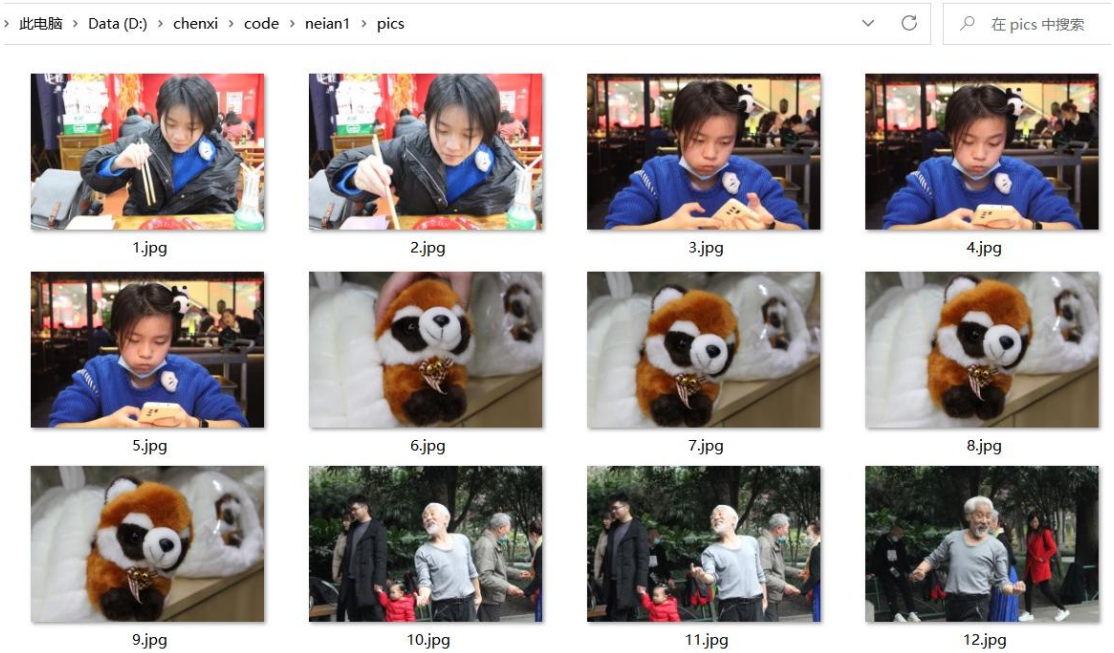
程序执行结果如下：左侧是图像提取 SIFT 特征，右侧事图像提取 Harris 角点。





【完成图片匹配任务】

编写程序 main4.2.py，通过 SIFT 特征，来识别一个文件夹中与目标图片最为类似的三张图片。
图片文件夹如下所示。



将文件夹 pics 的第三张图片设为目标图片。

导入工具包。

```
from PIL import Image
from pylab import *
from pcv.localdescriptors import sift
import matplotlib.pyplot as plt
```

将目标文件存为变量 im1，使用 SIFT 类中的 process_image 函数对图片进行 SIFT 特征点提取。特征点和描述子分别存储在 l1 和 d1 变量中。

```
im1f = 'pics/3.jpg'
im1 = array(Image.open(im1f))
sift.process_image(im1f, 'pics_res/out_sift_1.txt')
l1, d1 = sift.read_features_from_file('pics_res/out_sift_1.txt')
```

计算一张图片和多张图片之间的 SIFT 特征点匹配，并记录匹配点对的数量和对应的图片路径。匹配点对的数量将以单维链表数组 arr 的形式存储，匹配点对数量和图片路径之间的映射关系将以字典型数组 arrHash 的形式存储。

```
arrHash = {}
for i in range(1, 12):
    im2f = 'pics/'+str(i)+'.jpg'
    im2 = array(Image.open(im2f))
    sift.process_image(im2f, 'pics_res/out_sift_2.txt')
    l2, d2 = sift.read_features_from_file('pics_res/out_sift_2.txt')
    matches = sift.match_twosided(d1, d2)
    length=len(matches.nonzero()[0])
    length=int(length)
    arr.append(length)
    arrHash[length]=im2f
```

使用 sort 函数对 arr 数组进行排序操作。切片操作[::-1]将数组反转，使得数组中最大的元素排在最前面，并且切片操作[:3]将数组截取到前三个元素，即保留匹配点对数量最大的三张图片，使用 Matplotlib 库中的 figure 函数创建一个大小为 6×12 的图像窗口，以便绘制输出图像。

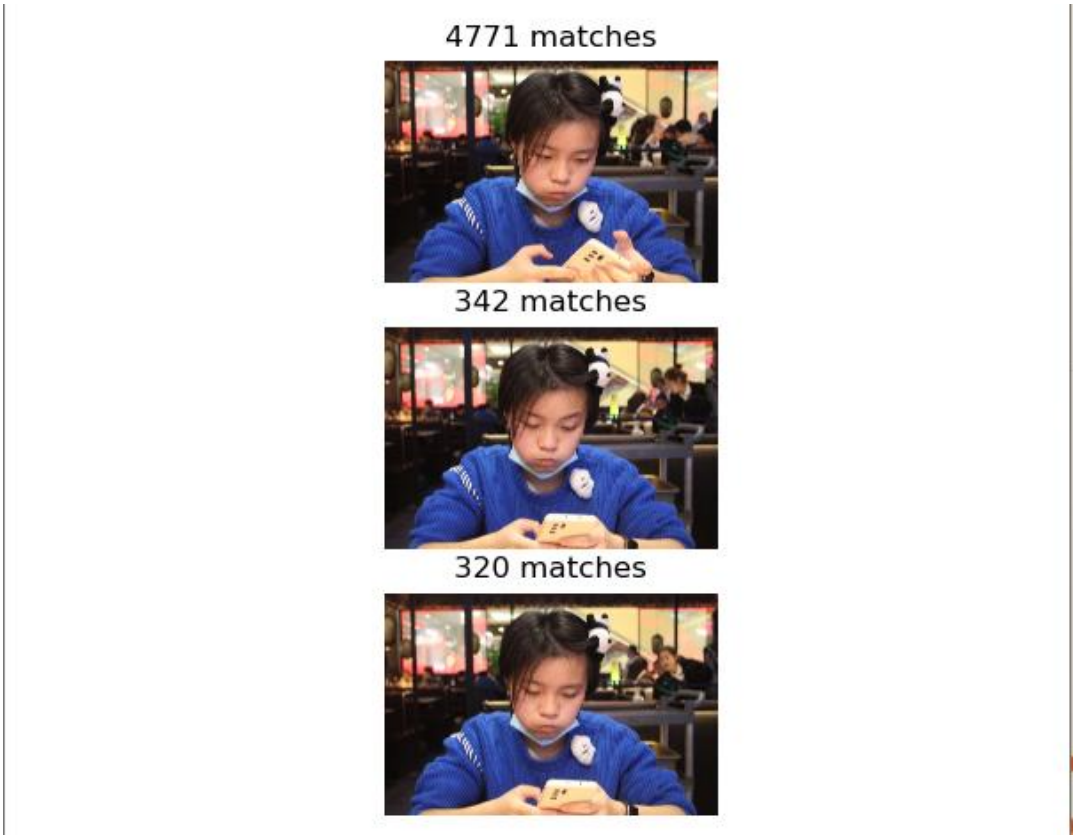
```
arr.sort()
arr=arr[::-1]
arr=arr[:3]
i=0
plt.figure(figsize=(6,12))
```

使用 for 循环遍历 arr 数组中的元素。PIL 库中的 Image.open 函数读取图片，并将其转换为 NumPy 数组。借助 Matplotlib 库中的 subplot 函数创建一个子图，并设置子图的位置。set_title 函数设置子图的标题，其中包括匹配点对数

量。 axis 函数设置子图不显示坐标轴，并使用 imshow 函数显示图片，并通过 i 变量更新子图的位置。最后使用 plt.show 函数显示输出图像。

```
    for item in arr:
        if(arrHash.get(item)!=None):
            img=arrHash.get(item)
            im1 = array(Image.open(img))
            ax=plt.subplot(511 + i)
            ax.set_title('{} matches'.format(item))
            plt.axis('off')
            imshow(im1)
            i = i + 1

    plt.show()
```

程序运行结果，根据特征点匹配数量大小，找出三张与目标图片相似的图片。



5. 在本地虚拟环境中部署 yolo 模型（v3-v8 任选），完成图像目标检测任务,并阐述所选用的模型原理

【yolov5 模型理解】

Yolov5 是一种目标检测的工具。

【目标检测】

目标检测是模式识别问题的一种，是计算机视觉领域中的一个重要研究方向，也是其他复杂视觉任务的基础。 作为图像理解和计算机视觉的基石，目标检测是解决分割、场景理解、目标跟踪、图像描述和事件检测等更高层次 视觉任务的基础。其本质就是回归问题和聚类问题。

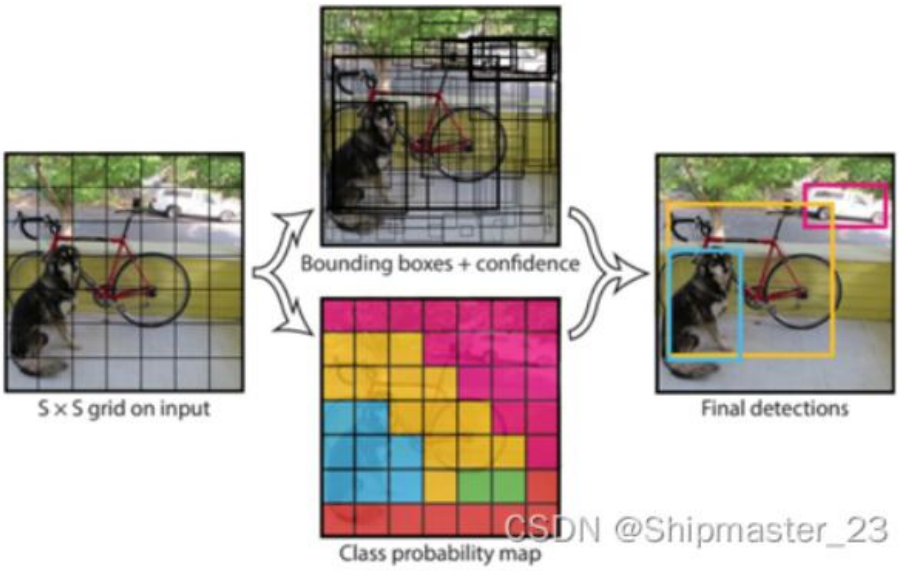
【yolov5 原理】

yolov5 首先会将一幅图像分成 S*S 个网格，如果某个 object 的中心落在这个网格内，则这个网格就负责预测这个 object。

每个网格要预测 B 个 bounding box，每个 bounding box 除了要回归自身的位置之外(x,y,w,h)，还要附带预测一个 confidence 值，共 5 个值。

每个网格还要预测 C 个类别信息，记为 C 类。则 S*S 个网格，每个网格要预测 B 个 bounding box 还要预测 C 个 categories。输出就是 S*S(5*B+C)的一个张量。

最后去除可能性较低的目标窗口，用 NMS 非极大值抑制去除冗余窗口。

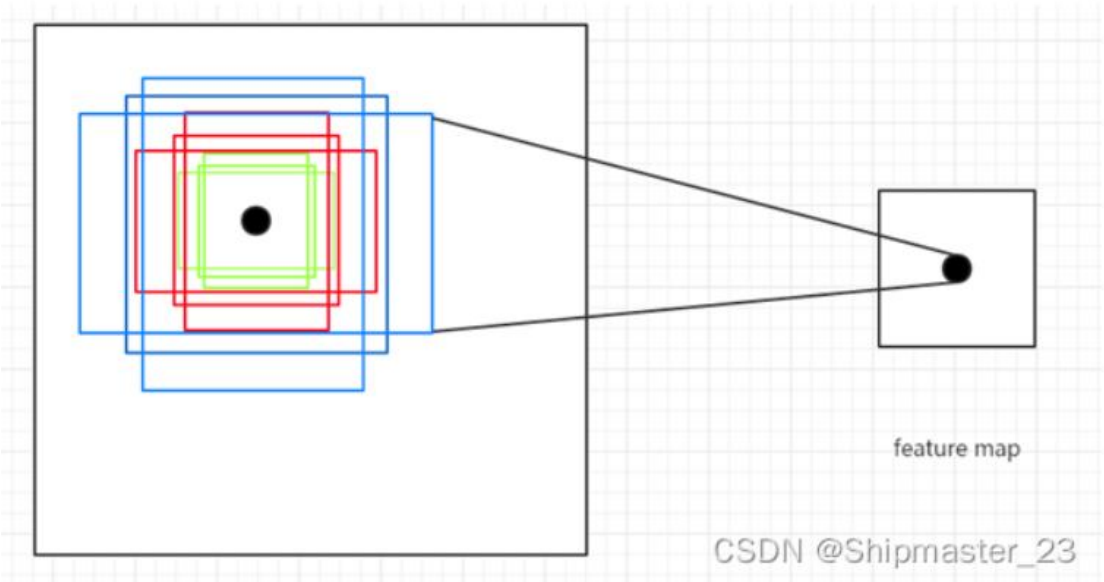


【自适应锚框计算】

锚框就是 bounding box 在进行非极大值抑制之前的名字，其本质上是同一个东西。

YOLOv5 对于不同的数据集，都会计算先验框 anchor。然后在网络训练时，网络会在 anchor 的基础上进行预测，然后输出预测框，再和标签框进行对比，最后就进行梯度地反向传播。在 YOLOv3、YOLOv4 中，训练不同的数据集时，是使用单独的脚本进行初始锚框的计算，在 YOLOv5 中，则是将此功能嵌入到整个训练代码里中。所以在每次训练开始之前，它都会根据不同的数据集来自适应计算 anchor。

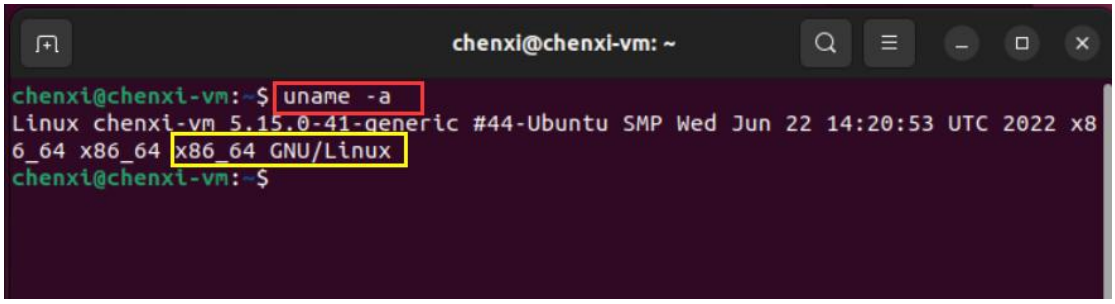
【自适应锚框计算步骤】
Step1：读取训练集中所有图片的 w、h 以及检测框的 w、h
Step2：将读取的坐标修正为绝对坐标
Step3：使用 Kmeans 算法对训练集中所有的检测框进行聚类，得到 k 个 anchors
Step4：通过遗传算法对得到的 anchors 进行变异，如果变异后效果好将其保留，否则跳过
Step5：将最终得到的最优 anchors 按照面积返回



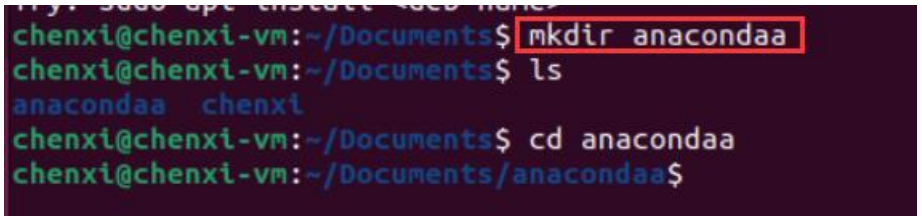
【自适应图片缩放】
自适应图片缩放针对不同的目标检测算法而言，我们通常需要执行图片缩放操作，即将原始的输入图片缩放到一个固定的尺寸，再将其送入检测网络中。yolov5 处理的图片的大小是 640*640，如果输入的不是 640*640 的话，会按照比例进行缩放，将不够的部分用黑色来填充。因为网络里定义的步长是 32，而图片的大小必须是 32 的整数倍，所以输入的图片大小是 640*640。
原始的缩放方法存在着一些问题，因为在实际的使用中的很多图片的长宽比不同，所以在进行缩放填充之后，两端的黑边大小都不相同，但是如果填充过多，则会存在大量的信息冗余，从而影响整体的推理速度。
为了进一步提升推理速度，YOLOv5 提出一种方法能够自适应的添加最少的黑边到缩放之后的图片中。也就是通过调用 letterbox 这个类实现自适应图片缩放。其核心思想是根据原始图片大小与输入到网络图片大小计算缩放比例，根据原始图片大小与缩放比例计算缩放后的图片大小，计算黑边填充数值。这样，图像高度上两端的黑边变少了，在推理时，计算量也会减少，目标检测速度得到提升。

【在本地虚拟环境中部署 yolov5 模型，完成图像目标检测任务】

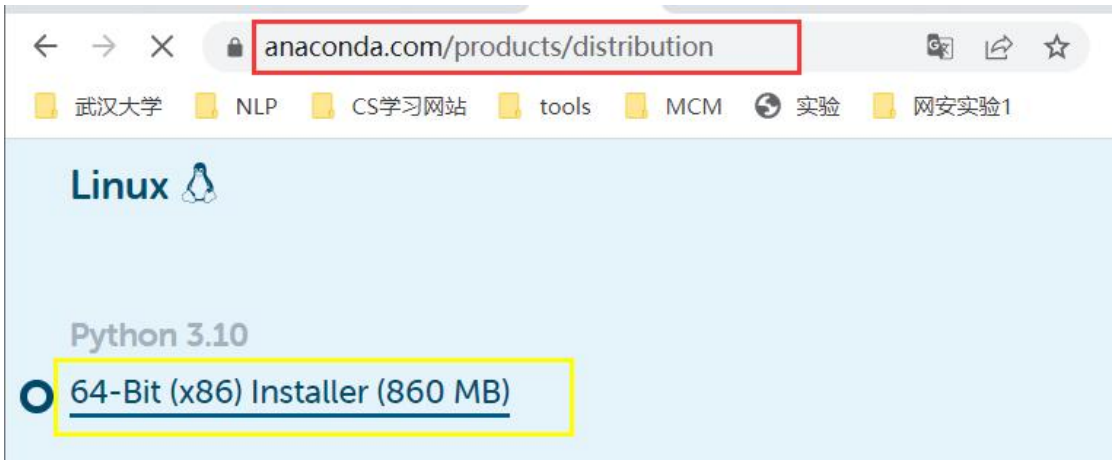
查看系统架构。为 64 位的 Linux。



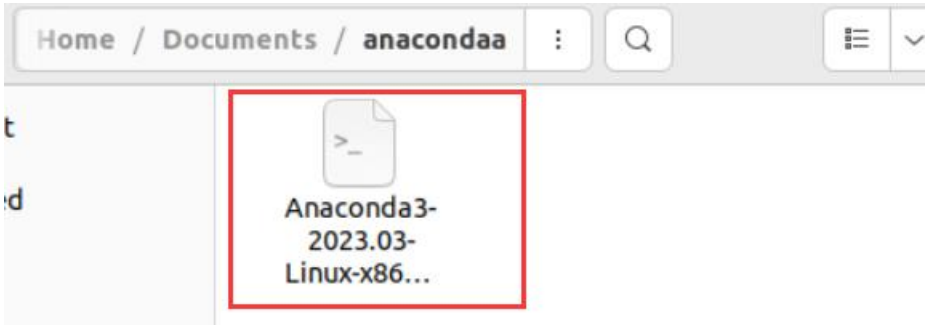
新建一个 anacondaa 文件夹，用来安装 Anaconda。



在官网上下载 64 位的 Linux 环境 Anaconda。

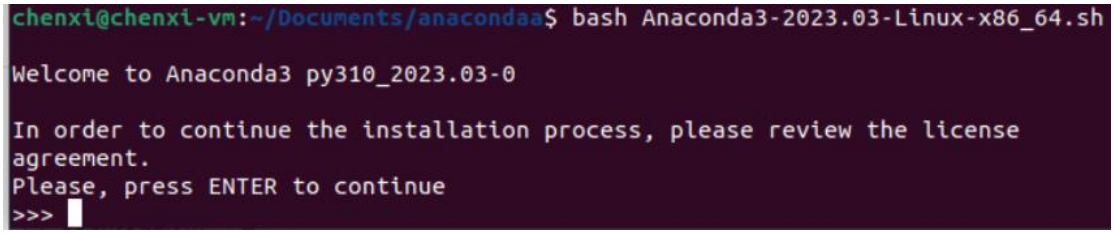


安装到 Anacondaa 文件夹下。

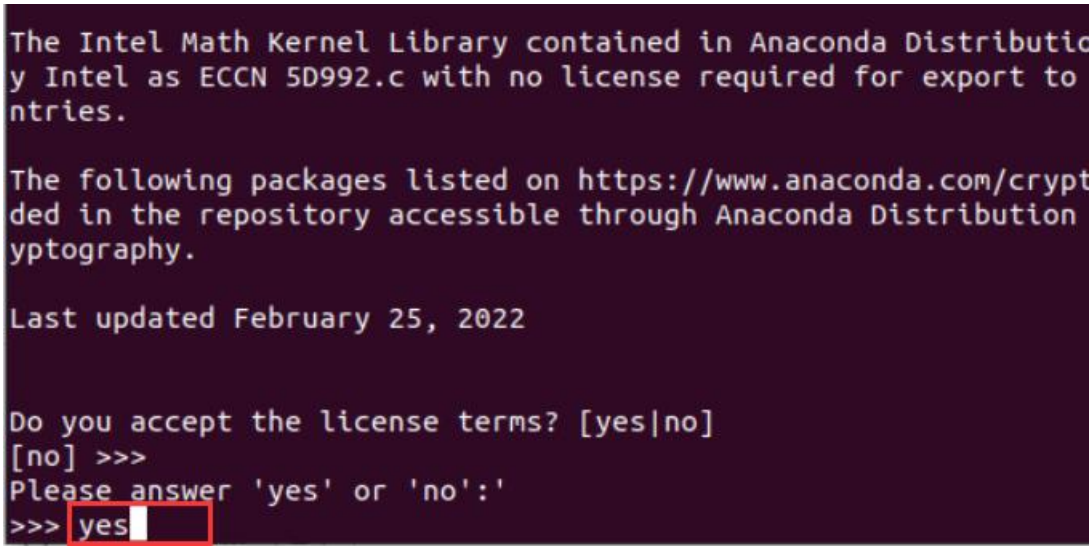


使用命令 bash 进行安装。

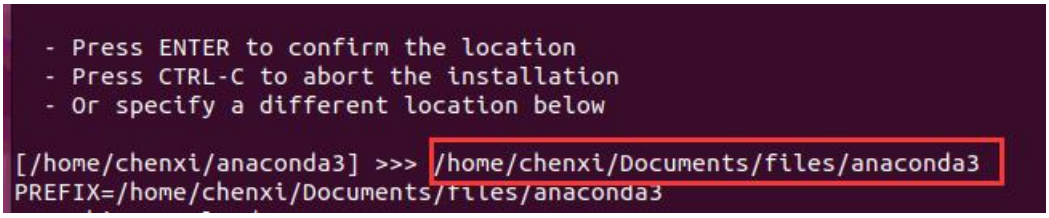
安装过程：摁 Enter 键。



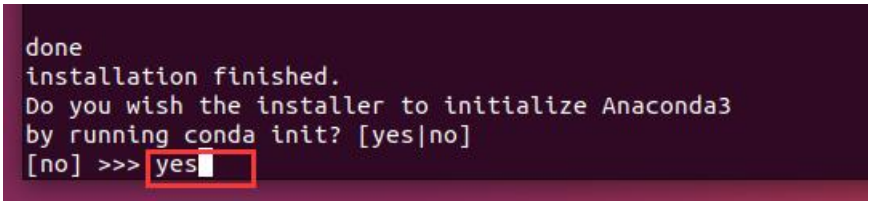
输入 yes。



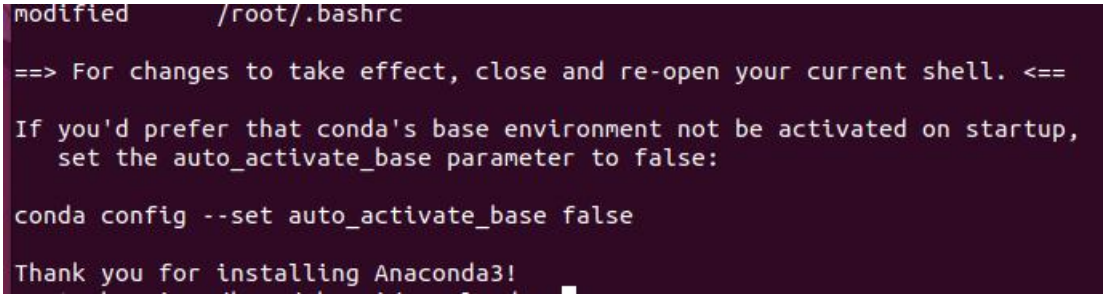
设置 anaconda 安装的目录。



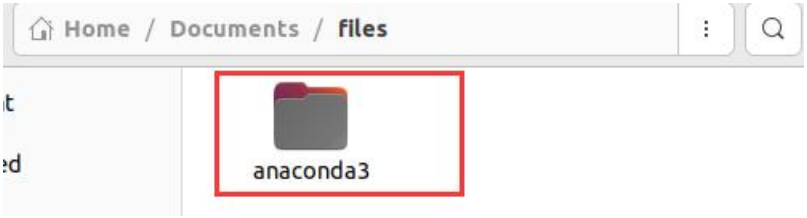
继续输入 yes。



安装成功。



可以在文件夹里查看。



使用 source ~/.bashrc 命令,将新的 PATH 环境变量加载到当前的 shell 会话中。

```
chenxi@chenxivm:~/Documents/files/anaconda3$ source ~/.bashrc
(base) chenxi@chenxivm:~/Documents/files/anaconda3$
```

创建一个 python3.8 的名称为 yolo 的环境。

```
(base) chenxi@chenxivm:~/Documents/files$ conda create -n yolo python=3.8
```

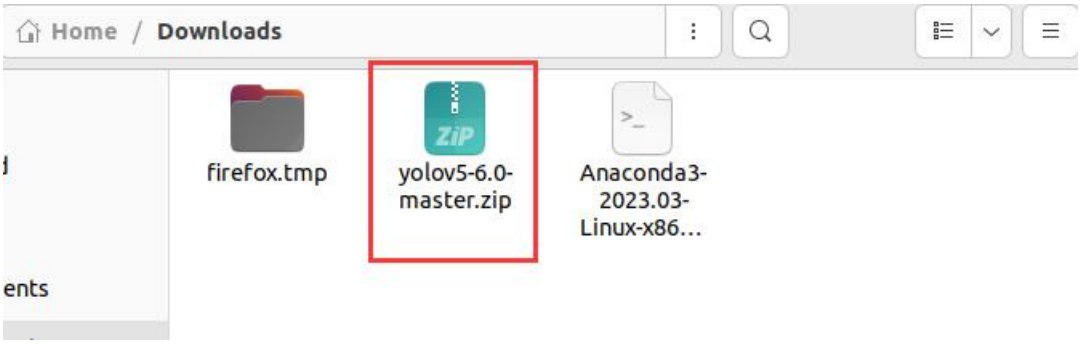
输入 y 确定。

```
Proceed ([y]/n)? y

Downloading and Extracting Packages

Preparing transaction: done
Verifying transaction: -
```

创建环境成功。

```
#
# To activate this environment, use
#
#     $ conda activate yolo
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

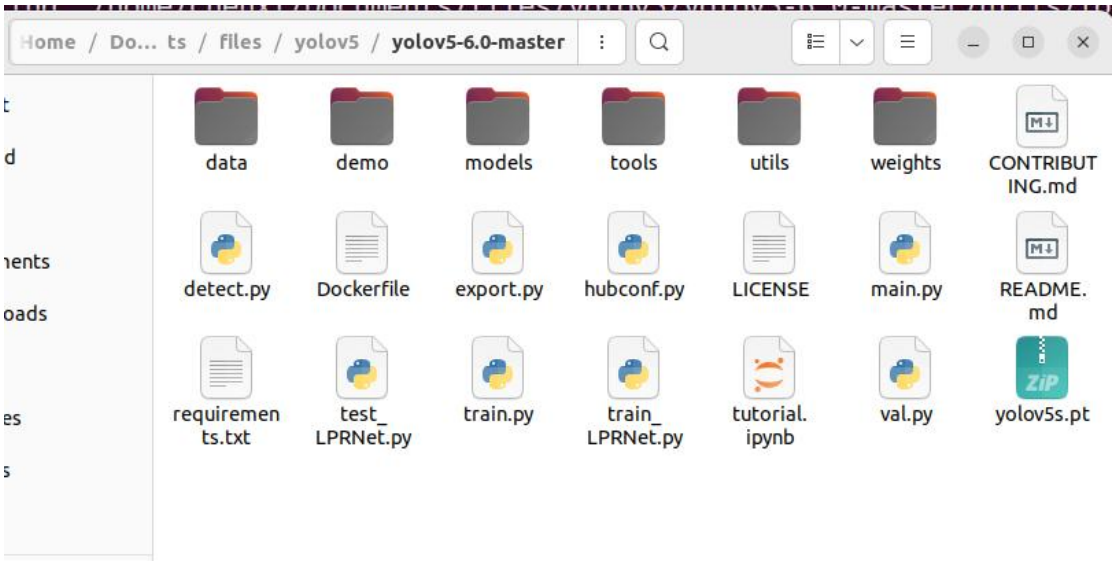
从 github 官网上下载 yolov5-version6.0 版本的代码。



使用 unzip 命令解压压缩包到创建的 yolov5 文件夹中。

```
(base) chenxi@chenxivm:~/Downloads$ ls
Anaconda3-2023.03-Linux-x86_64.sh  firefox.tmp  yolov5-6.0-master.zip
(base) chenxi@chenxivm:~/Downloads$ unzip yolov5-6.0-master.zip -d ~/Documents/files/yolov5/
Archive:  yolov5-6.0-master.zip
e4c05be7ad2d16d927bf8b8cf1c6590a487bf471
   creating: /home/chenxi/Documents/files/yolov5/yolov5-6.0-master/
```

可以在 yolov5 文件夹中看到解压缩的文件。



在已经安装好的 anaconda 环境中，输入下面命令创建名为 yolo 的 python3.8 环境。

```
(base) chenxi@chenxivm:~/Documents/files/anaconda3$ conda create -n yolo python=3.8
```

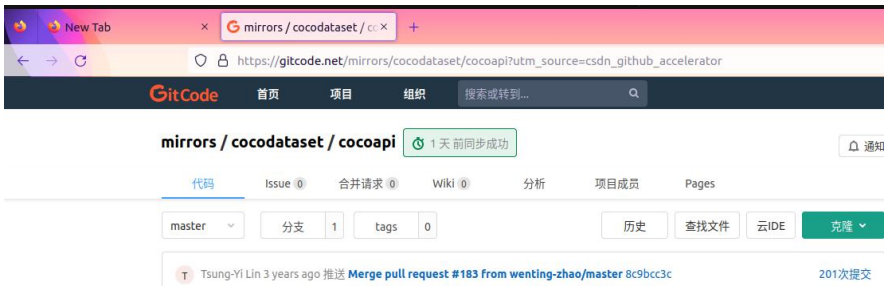
创建完成之后，输入命令激活环境。

```
(base) chenxi@chenxivm:~/Documents/files/anaconda3$ conda create -n yolo python=3.8
WARNING: A conda environment already exists at '/home/chenxi/Documents/files/anaconda3/envs/yolo'
Remove existing environment (y/[n])? n


CondaSystemExit: Exiting.

(base) chenxi@chenxivm:~/Documents/files/anaconda3$ conda activate yolo
(yolo) chenxi@chenxivm:~/Documents/files/anaconda3$
```

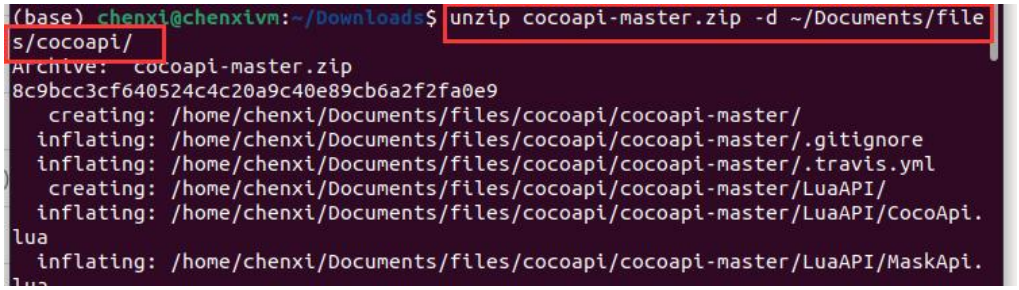
使用命令和指定的清华源，下载 requirements.txt 中所需要的工具包。

```
(yolo) chenxi@chenxivm:~/Documents/files/yolov5/yolov5-6.0-master$ pip install -U -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting matplotlib>=3.2.2
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/5d/22/f55638bea4af17edf23e1c919ad5d256141bbeec0196c450be9785f1dcb6/matplotlib-3.7.1-cp38-cp38-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (9.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.2/9.2 MB 18.8 MB/s eta 0:00:00
Collecting numpy>=1.18.5
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/9c/ee/77768cade9607687fadbcc1dcbb82dba0554154b3aa641f9c17233ffabe8/numpy-1.24.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 25.4 MB/s eta 0:00:00
```

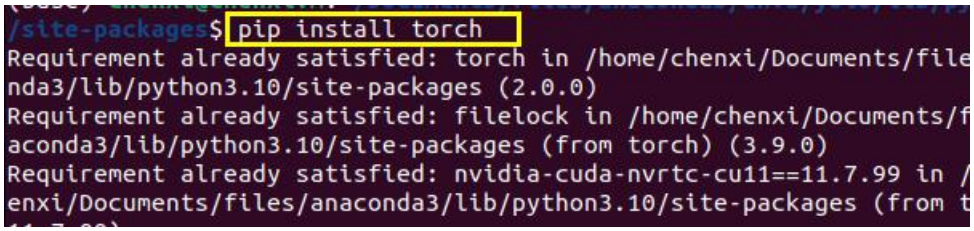
进入 github，下载 cocoapi。



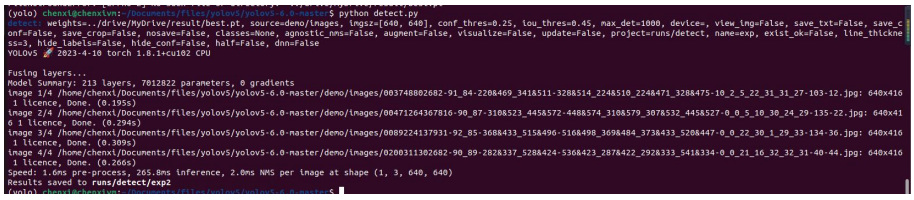
解压缩该工具到指定文件夹。并安装该工具。



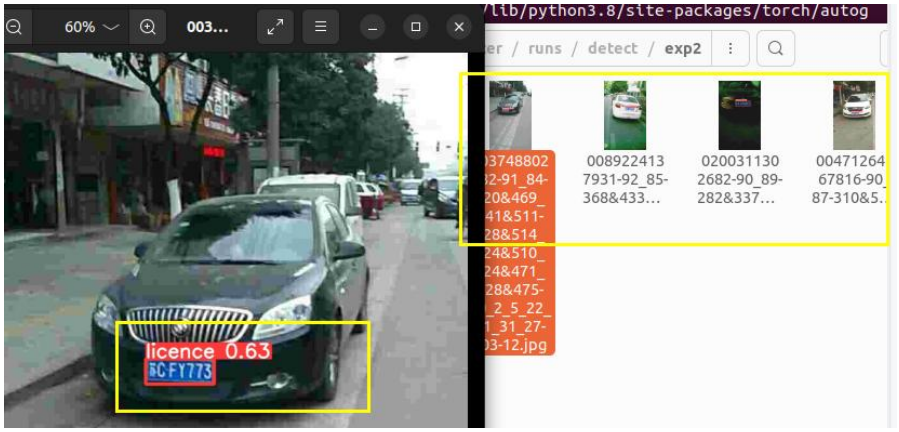
这时候发现工具还没有办法成功运行。检查发现有些工具包没有成功被下载。下载所需的工具包。



进入 yolo 环境，使用命令 python detect.py 运行该工具。结果保存在文件夹 runs/detect/exp2 中。



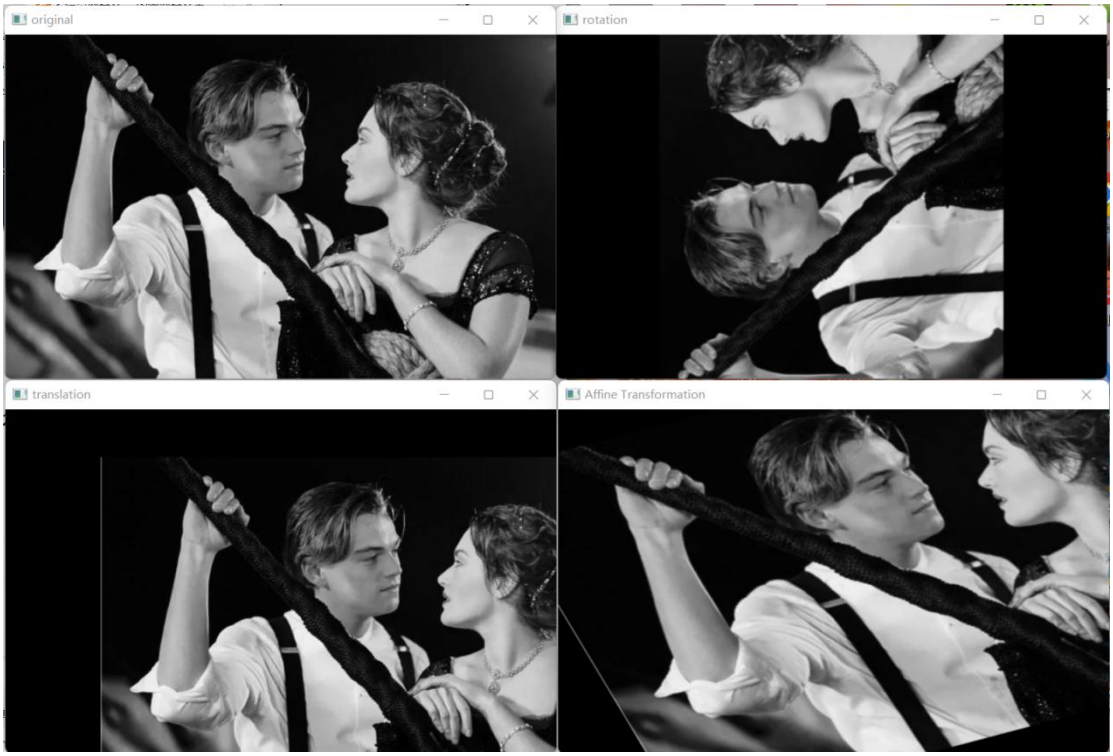
查看结果文件夹。发现车牌并成功识别并标注。



四、实验结果

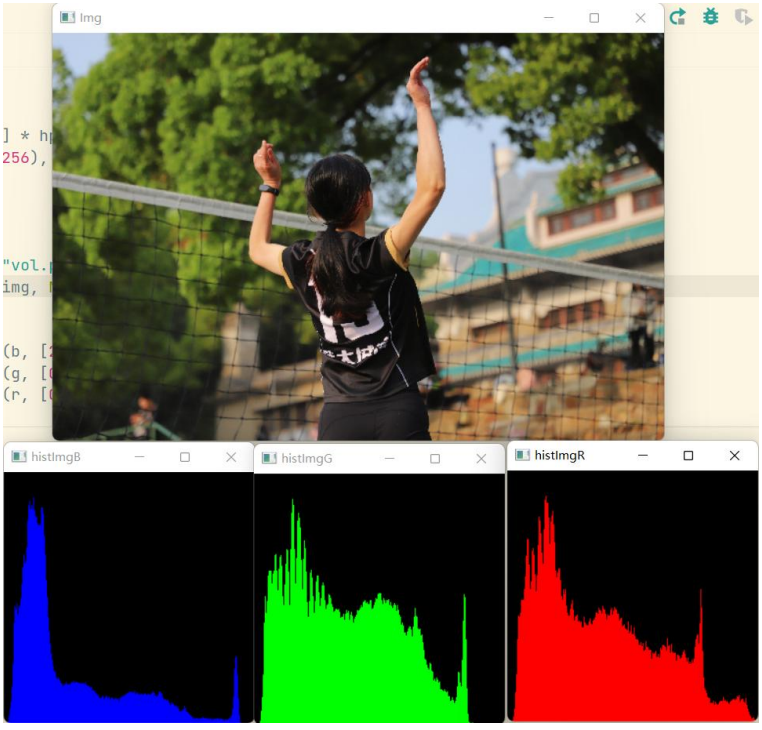
1. 任选图片，实现仿射变换。

仿射变换成功。

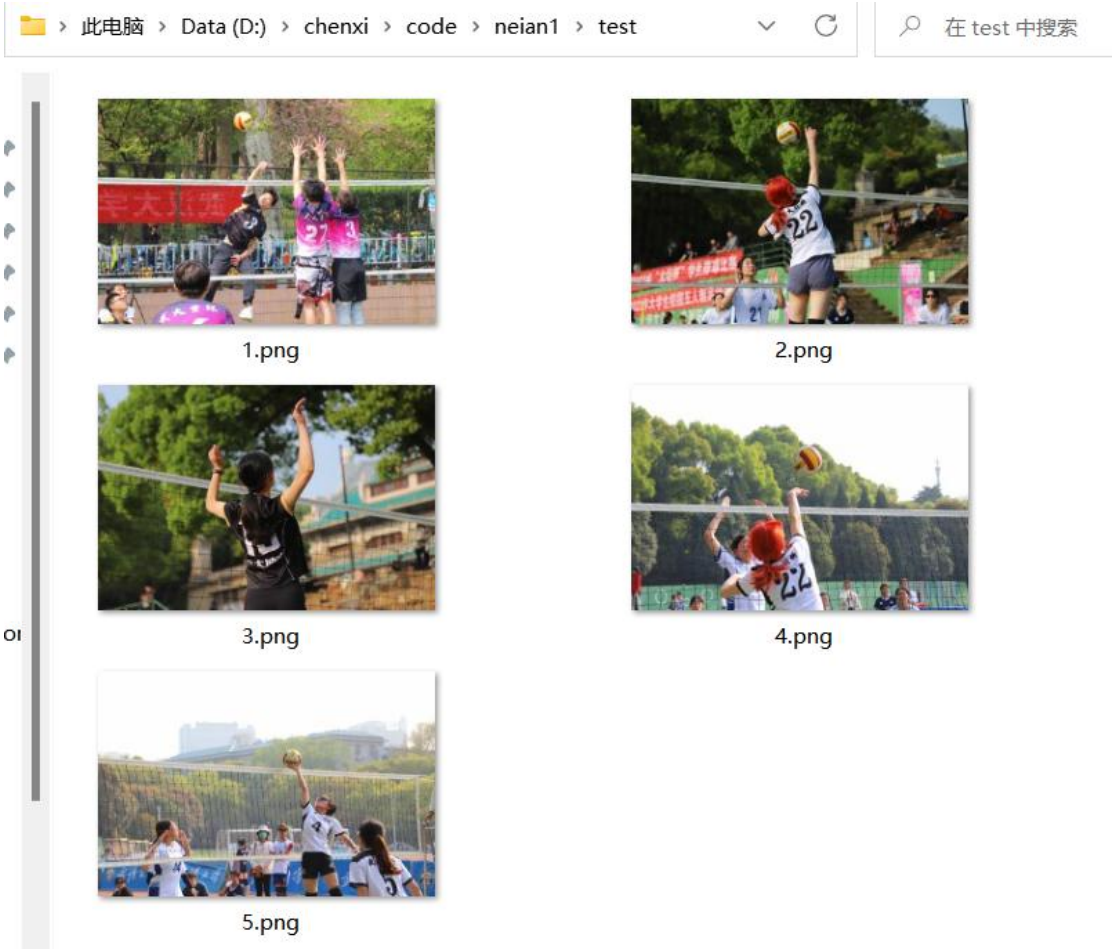


2. 任选图片，提取颜色直方图特征，完成图像的匹配、查找。

提取目标图片的直方图特征：



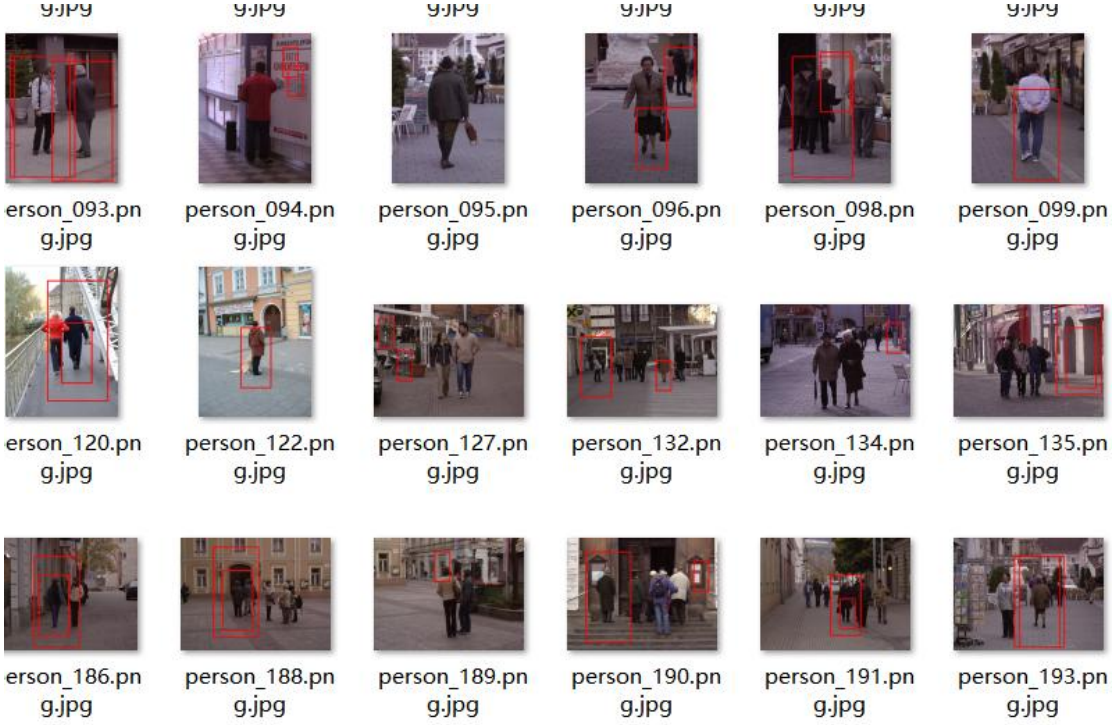
通过直方图相似度，在下面这 5 个图片中，找出与目标图片最相似的图片。
5 个图片如下：



最相似的图片：

3. 理解 HOG 算法，使用 SVM\KNN 完成目标行人检测，并阐述算法原理。

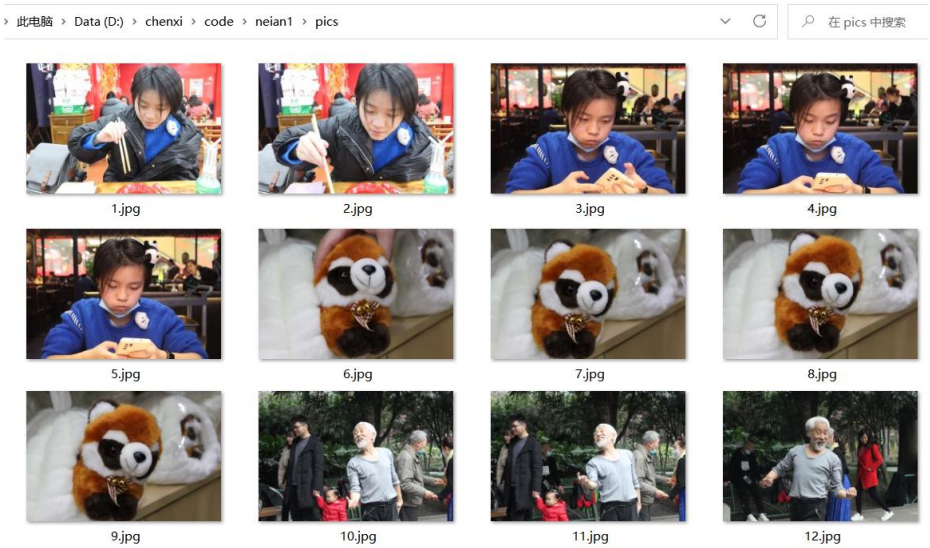
完成行人检测：并用红框标出。



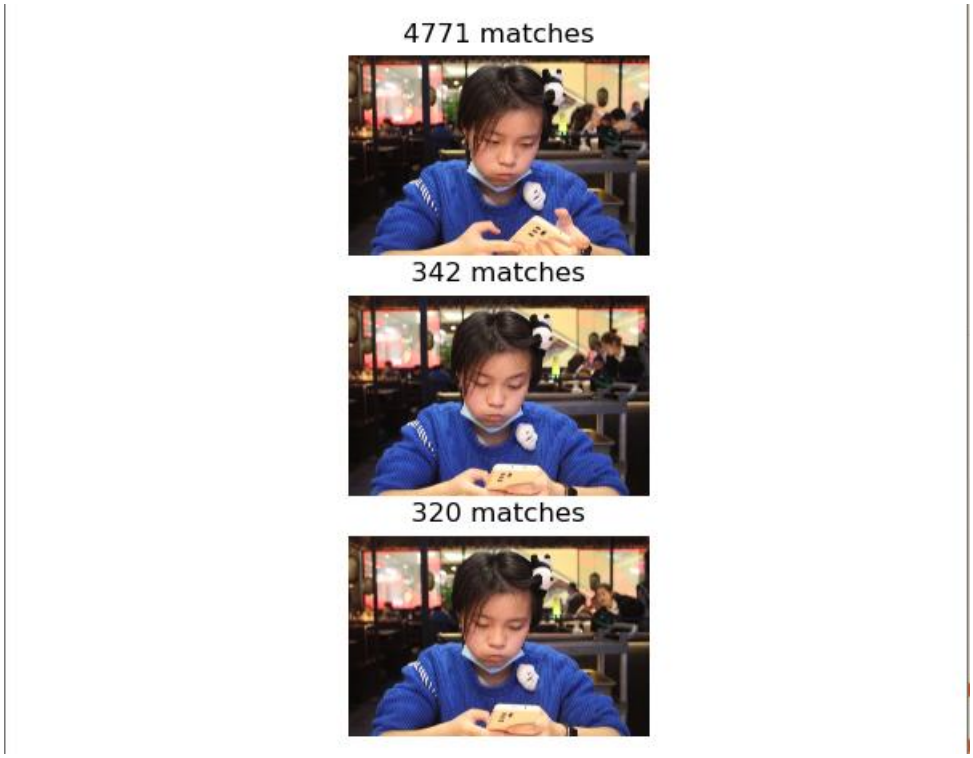
4. 理解 SIFT 算法，完成图片匹配任务，并阐述算法原理。

目标图片：

根据 SIFT 特征在下面这 12 张图片里找出与目标图片最相似的图片：

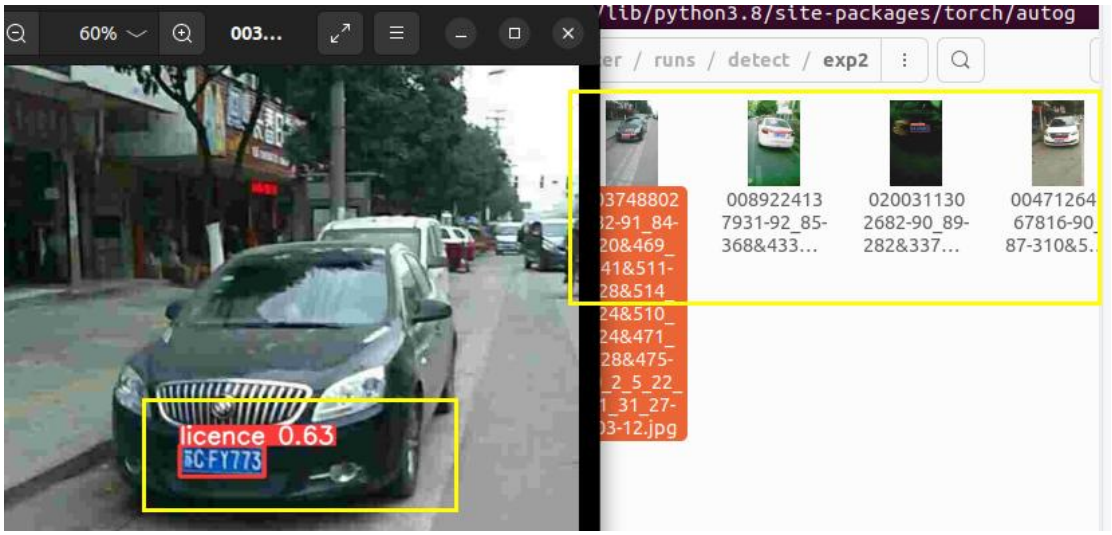


找出的 3 张最相似的图片：



5. 在本地虚拟环境中部署 yolo 模型（v3-v8 任选），完成图像目标检测任务，并阐述所选用的模型原理。

使用 yolov5，检测图像结果：



五．实验心得

在本次实验中，我学习了 OpenCV 工具。首先我对图片进行了仿射变换，这是一种常用的图像变换方法，它可以将图像进行平移、旋转、缩放等操作。在实验中，我尝试了不同的变换方式，比如平移、旋转、缩放和剪切等，并观察了它们对图像的影响。

接着我学习了直方图，并使用其中的函数对图像进行了直方图提取。直方图是一种常用的图像处理方法，可以用于图像增强、图像分割、目标检测等任务。

使用 opencv 工具进行了两个应用：HOG+SVM 图像特征提取，完成了目标行人的检测；学习了 SIFT 算法，可以对目标图像进行 SIFT 的图像特征提取，并利用此特征完成图像的匹配。

这是一种目标检测算法，可以识别图像中的不同物体，并框定它们的位置和大小。相比于传统的目标检测算法，YOLOv5 具有更高的检测精度和更快的检测速度。

最后我在本地虚拟环境中部署了 yolov5 模型，了解了 YOLOv5 的基本原理和算法流程,然后使用它对图像进行了目标检测。我尝试了不同的模型和参数设置，比如模型大小、学习率、迭代次数等，并对它们进行了比较和分析。

通过本次实验，我不仅学习了 OpenCV 工具的基本使用，还对计算机视觉和图像处理有了更深入的了解。我相信这些知识将对我的未来研究和工作产生积极的影响。