第 二 次 实 验 报 告

课程名称	内容安全实验				
学生姓名		学号		指导老师	熊翘楚
专业	网络空间安全	班级		实验时间	2022. 03. 28

一、实验内容

1、基于 word2vec 模型的文本分类任务：文本素材可使用爬虫爬取或使用网上现成语料库。

注：实验报告中应写出所使用的算法基本原理

2、调用百度 AI 开放平台的文本内容审核 API，完成自定义数据集的内容审核。

二、实验原理

1. 仿射变换

一个任意的仿射变换都能表示为乘以一个矩阵再加上一个向量。一般使用 2x3 矩阵 M 来表示仿射变换，其中向量 B 起着平移的作用，矩阵 A 中的对角线决定缩放，反对角线决定旋转或错切。因此通过矩阵可以确定仿射变换的具体方式，从而实现平移、缩放、旋转、错切、翻转等操作。

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}_{2\times 2}, \quad B = \begin{bmatrix} b_{00} \\ b_{10} \end{bmatrix}_{2\times 1}$$

$$M = [A \quad B] = \begin{bmatrix} a_{00} & a_{01} & b_{00} \\ a_{10} & a_{11} & b_{10} \end{bmatrix}_{2\times 3}$$

2. 基于颜色直方图特征的图像匹配或查找

颜色直方图所描述的是不同色彩在整幅图像中所占的比例，而并不关心每种色彩所处的空间位置，即无法描述图像中的对象和物体。然而颜色直方图适合用

于图像匹配或查找，因为一般而言越相似的图像，其颜色直方图的相似度也会越高，所以反过来说，两个图像的颜色直方图相似度越高，那么这两个图像本身也是大概率相似的。颜色直方图可以是基于不同的颜色空间和坐标系，常用的颜色空间有 RGB、HSV、HLS、CMYK、Lab 等。本实验中使用的颜色空间是 RGB 颜色空间。

本实验中最关键的两个函数分别是 cv2 库中的 calcHist 和 cv2 库中的 compareHist。calcHist 用于提取颜色直方图特征，常用参数是前五个参数，其中第四个参数 histSize 比较关键。第一个参数是图像信息，一般传入 imread 函数的返回值；第二个参数是用于计算颜色直方图的通道，可选[0, 1, 2]；第三个参数表示是否使用 mask，mask 可以选择图片中需要处理的部分并且忽略图片中不需要处理的部分，设置为 None 则表示不使用 mask；第四个参数 histSize 表示将每个通道的颜色值分为多少份，因为 RGB 三个通道各有 256 个颜色，全部计算会使得计算量较大，并且相似图片检索的任务需要一定的模糊度，所以可以将每个通道的颜色值分为 histSize 个部分；第五个参数是像素值的范围。

```
cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate ]]) #返回hist
```

compareHist 函数所需要的系数相对较少，前两个参数分别是需要比较的两个图像的颜色直方图，第三个参数是使用的比较方法。常用的比较方法是皮尔逊相关系数比较、卡方比较、相交比较和巴氏距离比较。在 cv2 库的定义中，皮尔逊相关系数比较和相交比较得到的值越大则相似度越高，卡方比较和巴氏距离比较得到的值越小则相似度越高。

方法名	标识符	计算公式
相关 Correlation	HISTCMP_CORREL	$d(H_1,H_2)=\frac{\sum_I(H_1(I)-\bar{H_1})(H_2(I)-\bar{H_2})}{\sqrt{\sum_I(H_1(I)-\bar{H_1})^2\sum_I(H_2(I)-\bar{H_2})^2}}$ $\bar{H_k}=\frac{1}{N}\sum_J H_k(J)$ 的数目。
卡方 Chi-square	HISTCMP_CHISQR	$d(H_1,H_2)=\sum_I\frac{(H_1(I)-H_2(I))^2}{H_1(I)+H_2(I)}$
相交 Intersection	HISTCMP_INTERSECT	$d(H_1,H_2)=\sum_I\min(H_1(I),H_2(I))$
巴氏距离 Bhattacharyya	HISTCMP_BHATTACHARYYA	$d(H_1,H_2)=\sqrt{1-\frac{1}{\sqrt{\bar{H_1}\bar{H_2}N^2}}\sum_I\sqrt{H_1(I)\cdot H_2(I)}}$

3.基于 HOG 特征，使用 SVM\KNN 完成目标行人检测

HOG 特征，也就是方向梯度直方图特征是一种在计算机视觉和图像处理中用来进行物体检测的特征描述子。HOG 特征通过计算和统计图像局部区域的梯度方向直方图来构成特征。该特征的主要思想是在图像中，局部目标的表象和形状能够被梯度很好地描述。HOG 特征关注梯度的统计信息，而梯度主要出现在边缘的地方，因此该特征可以用于定位局部目标。

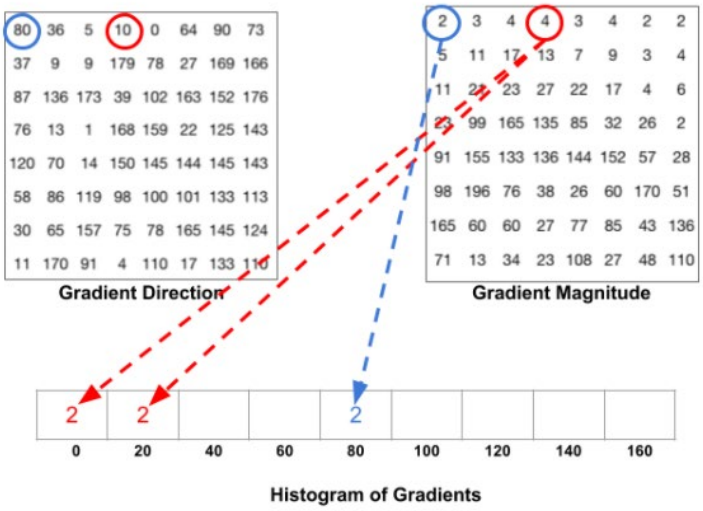
HOG 优点在于因为该特征提取方法作用于图像的局部方格单元上，所以对于图像几何和光学的形变都能保持很好的不变性。并且 HOG 特征非常适合用于图像中的人体检测，因为该特征可以忽略一些人体细微的动作而不影响效果。因此本实验基于 HOG 特征完成目标行人检测任务。

HOG 特征的具体应用步骤有五个，第一步是图像预处理，这是一个可选的步骤。其中图像预处理包括伽马校正和灰度化，伽马校正可以减少光度对实验的影响；灰度化是将输入的彩色图像变为灰度图，不过彩色图像也可以用于 HOG 特征的提取，做法是分别计算三个通道的梯度值，然后选择梯度最大的通道。第二步是计算每一个像素点的梯度值，得到和原图像规模一样的梯度图。先计算每个像素点的水平方向梯度和垂直方向梯度，然后使用如下公式计算总的梯度值和梯度方向。

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = arctan\frac{g_x}{g_y}$$

第三步是计算梯度直方图。经过以上两步后每个像素点都拥有梯度值和梯度方向。计算一般在 8x8 的 cell 中进行，如果每个值都考虑进去那么计算量将会比较大。HOG 使用的方法是将 0-180 度的梯度方向划分为 9 个 bins，每 20 度为一个 bin，将每个 bin 中像素对应的梯度值进行累加。这种计算方式得到的值会少很多，因此使得 HOG 对于环境光照等因素不那么敏感。



第四步是对 block 进行归一化，block 是通过 2x2 个 cell 使用滑动窗口划分而成，将每个 cell 的直方图进行拼接，得到 block 的特征值。归一化的目的是降低光照的影响，方法是用向量中的每个值除以向量的模长。第五步是计算 HOG 特征向量，将一定范围内的 block 拼接成 vector，再整合所有 block 的 vector，从而得到整个图像的 HOG 特征描述符。

本实验代码中使用到了 skimage 库中的 hog 函数，该函数存在以下主要参数。第一个参数表示输入的灰度图像，orientations 表示 bins 的数量，pixels_per_cell 表示 cell 的大小，visualize 表示是否返回 HOG 的图像，False 表示不返回，

cells_per_block 表示每个 block 中的 cell 数量。

```
fd = hog(img, orientations=9, pixels_per_cell=(8, 8), visualize=False, cells_per_block=(3, 3))
```

除此之外，SVM 和 KNN 都是常用的分类模型。SVM(Support Vector Machine)，即支持向量机是在特征空间中最大化间隔的线性分类器。KNN(k-Nearest Neighbor)，即 k 近邻算法的核心思想是如果一个样本在特征空间中的 k 个最相邻的样本的大多数属于某一种类别时，那么该样本也属于这个类别。

4.基于 SIFT 算法的图片匹配

SIFT，即尺度不变特征变换，用于图像处理领域，这种算法可在图像中检测出关键点，是一种局部特征描述子。SIFT 算法非常稳定，对旋转、尺度缩放、亮度变化保持不变，对仿射变换、噪声等也保持一定的稳定性。

SIFT 特征检测可以分为四个步骤，第一步是尺度空间的极值检测，该步骤搜索所有尺度空间上的图像，通过高斯微分函数识别潜在的对尺度不变的特征点。第二步是特征点定位，通过一个拟合精细模型确定每个侯选位置的尺度，依据他们的稳定程度选取关键点。第三步是特征方向赋值，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，后续的所有操作都是对于关键点的方向、尺度和位置进行变换，从而提供这些特征的不变性。第四步是特征点描述，在每个特征点的邻域内，在选定的尺度上测量图像的局部梯度，这些梯度被变换为 SIFT 特征向量。

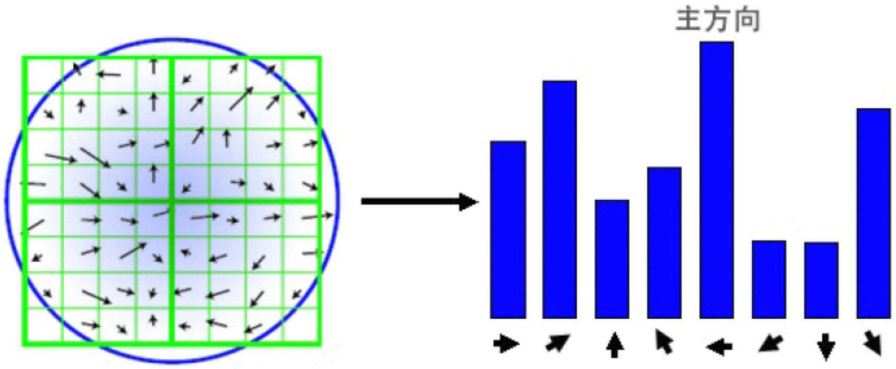
其中高斯尺度空间通过图像的模糊程度模拟人在距离物体由远到近时物体在视网膜上成像过程，距离物体越近图像越模糊。对图像和高斯函数进行卷积运算能够模糊图像，使用不同的高斯核可以得到模糊程度不同的图像。构建尺度空间的目的是为了检测出在不同的尺度下都存在的特征点，而检测特征点较好的算子是 LoG，然而 LoG 运算量过大，因此一般使用 DoG 近似计算 LoG。DoG 的定义如下图所示，其中 G 是高斯核函数，L 是图像的高斯尺度空间。

$$\begin{aligned} D(x,y,\sigma) &= [G(x,y,k\sigma) - G(x,y,\sigma)] * I(x,y) \\ &= L(x,y,k\sigma) - L(x,y,\sigma) \end{aligned}$$

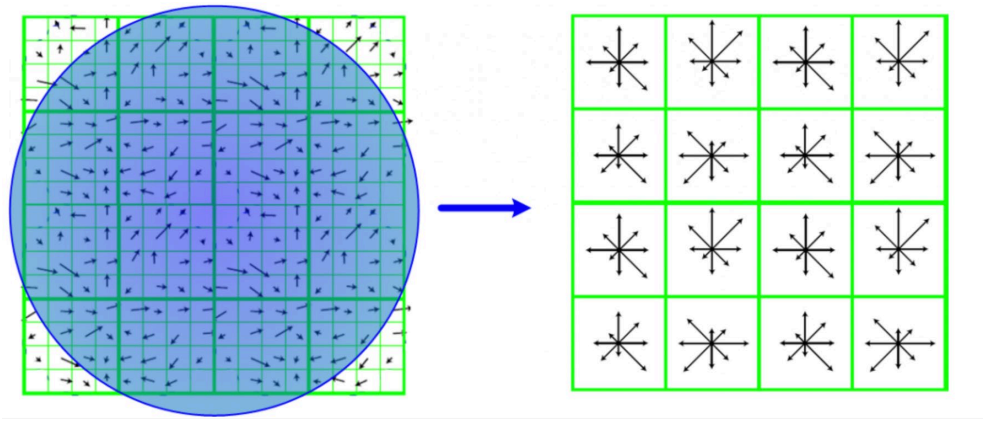
为了寻找尺度空间的极值点，每个像素点要和其同一尺度空间和相邻的尺度

空间的所有相邻点进行比较，当其大于或者小于所有相邻点时，这个点就是极值点。然而通过比较检测得到的极值点因为是在离散空间中搜索得到的，不一定是真正的极值点，因此需要剔除两种不符合要求的点。这两种点分别是低对比度的特征点和不稳定的边缘响应点。

在删除这些特征点后，需要确定特征点的主方向。方法是计算以特征点为中心的一定领域内的所有像素点的梯度的幅值和方向，并统计这些数据得到直方图，直方图横轴是方向，纵轴是该方向上梯度的幅值的累加值，直方图中最高所对应的方向就是特征点的主方向。



最后生成特征点的描述子。做法是先将坐标轴旋转为特征点的方向，然后以特征点为中心取 16x16 的窗口，计算窗口内的像素的梯度的幅值和方向，并将窗口内的像素分为 16 块，在每块中是其像素的 8 个方向的直方图统计，从而形成 128 维的特征向量。



以上是生成 SIFT 特征向量的过程，接下来介绍基于 SIFT 算法的图片匹配

过程。首先生成两幅图像的 SIFT 特征向量，使用欧氏距离作为两幅图像的关键点的相似度判定度量。再取一张图中的关键点，遍历另一张图得到与该关键点距离最近的两个关键点，如果其中最近距离除以次近距离得到的值小于某个阈值，则判定为一对匹配点。

三、实验步骤

1.仿射变换

首先使用 cv.imread 方法读取图片，第一个参数是图片路径，第二个参数如果取 0,则表示读入灰度图片;如果取-1 或者默认情况下,则表示读入完整原图。本实验选择读入灰度图片。

然后使用 cv.warpAffine 方法实现图片的平移，cv.warpAffine 方法的重点在于第二个参数 M，这个参数表示仿射变换的矩阵，平移使用到的变换矩阵在 python 代码中可以这样定义：np.float32([[1, 0, X], [0, 1, Y]])，其中 X 和 Y 是图片在 x 轴和 y 轴上平移的距离。因此，图片的平移使用到的代码如下所示。

```
#Load a image in grayscale
img = cv.imread('cat.jpeg', 0)
rows, cols, dim = img.shape

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

缩放图片所使用的变换矩阵 M 如下图中代码所示,即 np.float32([[W, 0, 0], [0, H, 0]])，可以改变矩阵中 W 和 H 的值，从而改变图片在 x 轴和 y 轴上的缩放倍数。

```
#Scale
M = np.float32([[0.6, 0, 0], [0, 1.4, 0]])
dst4 = cv.warpAffine(img, M, (cols, rows))
```

旋转图片需要使用 cv.getRotationMatrix2D 方法得到变换矩阵 M，第一个参数 center 表示旋转中心，第二个参数 angle 表示旋转角度，第三个参数 scale 表示缩放尺度。得到变换矩阵 M 后再使用 cv.warpAffine 方法完成图片的旋转。使用代码如下所示。

```
#Rotation
#cols-1 and rows-1 are the coordinate limits.
M = cv.getRotationMatrix2D(((cols - 1) / 2.0, (rows - 1) / 2.0), 90, 1)
dst2 = cv.warpAffine(img, M, (cols, rows))
```

X 方向的错切使用到的变换矩阵 M 为 np.float32([[1, A, 0], [0, 1, 0]])，其中 A 的值表示错切的距离大小。Y 方向的错切使用到的变换矩阵 M 为 np.float32([[1, 0, 0], [B, 1, 0]])，其中 B 的值表示错切的距离大小。

```
#Shear in x direction
M = np.float32([[1, 0.4, 0], [0, 1, 0]])
dst5 = cv.warpAffine(img, M, (cols, rows))

#Shear in y direction
M = np.float32([[1, 0, 0], [0.4, 1, 0]])
dst6 = cv.warpAffine(img, M, (cols, rows))
```

图片的原点翻转可以使用变换矩阵 np.float32([[-1, 0, 0], [0, -1, 0]])实现，X 轴翻转可以使用变换矩阵 np.float32([[1, 0, 0], [0, -1, 0]])实现，Y 轴翻转可以使用变换矩阵 np.float32([[-1, 0, 0], [0, 1, 0]])实现。

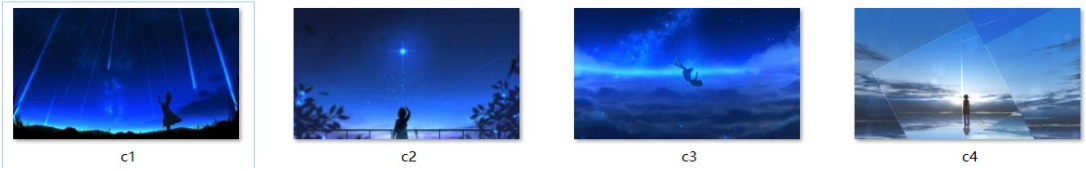
```
#Reflect about origin
dst7 = cv.flip(img, -1)

#Reflect about x-axis
dst8 = cv.flip(img, 0)

#Reflect about y-axis
dst9 = cv.flip(img, 1)
```

实现图片的仿射变换需要确定两组三点，一组三点表示输入图片的三角形顶点坐标，另一组三点表示输出图片的相应的三角形顶点坐标。通过确定两组三点，可以使用 cv.getAffineTransform 方法得到变换矩阵 M，再使用 cv.warpAffine 方法实现图片的仿射变换。

```
#AffineTransform
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))
```

2.基于颜色直方图特征的图像匹配或查找

首先下载得到四张图片，以 c1.jpg 为源图，将所有包括源图在内的四张图片分别与源图进行基于颜色直方图特征的图像匹配，即比较源图与目标图之间的相似度。在相似度的计算这个步骤中，本实验选择了四种比较方法，分别是皮尔逊相关系数比较、卡方比较、相交比较和巴氏距离比较。



使用 cv.imread 函数分别读取每张图片。因为 cv2 库默认以 BGR 的像素排列顺序读取图片，matplotlib.pyplot 库默认以 RGB 的像素排列顺序读取图片，在比较颜色直方图时需要使用 cv2 库，在比较相似度结果以及显示图片时需要使用 matplotlib.pyplot 库，因此为了正常显示图片选择使用 cv.cvtColor 函数进行从 BGR 到 RGB 的转换。转换后再使用 cv.calcHist 函数计算该图片的颜色直方图，并使用 cv.normalize 函数归一化和均衡化颜色直方图，从而提高图片的对比度。再调用 flatten 方法将数据降为一维后存入 hists 列表中，为后续的颜色直方图比较做好准备。

```
# 存储颜色直方图
hists = []
# 存储图像
images = []
# 本地加载的图像
filenames = ['c1.jpg', 'c2.jpg', 'c3.jpg', 'c4.jpg']
# 提取每个图像的颜色直方图
for i, fileName in enumerate(filenames):
    # 读取图像
    image = cv.imread(fileName)
    # 转换成RGB
    images.append(cv.cvtColor(image, cv.COLOR_BGR2RGB))
    # 提取颜色直方图
    hist = cv.calcHist([image], [0, 1, 2], None, [16, 16, 16], [0, 256, 0, 256, 0, 256])
    # 归一均衡化
    hists.append(cv.normalize(hist, hist).flatten())
```

然后调用 cv.compareHist 函数将源图分别与不同的目标图之间计算相似度，使用的相似度计算方法有四种，四种方法得到的结果都会记录在实验结果部分中。最后使用 matplotlib.pyplot 库将所有目标图以及计算得到的值全部打印出来。

```
results = []
# 将源直方图与目标直方图进行比较
for hist in hists:
    # 以第一张图片作为源图
    d = cv.compareHist(hists[0], hist, method)
    if method == cv.HISTCMP_BHATTACHARYYA:
        results.append(1 - (d ** 2))
    else:
        results.append(d)

# 绘图
fig = plt.figure('Bhattacharyya')
fig.suptitle('Bhattacharyya', y=0.8, fontsize=16)
for i, result in enumerate(results):
    ax = fig.add_subplot(1, len(images), i+1)
    ax.set_title('{:.6f}'.format(result))
    plt.imshow(images[i])
    plt.axis('off')
plt.show()
```

3.基于 HOG 特征，使用 SVM\KNN 完成目标行人检测

首先需要读取图像，因为图像尺寸可能各不相同，所以需要统一图像尺寸。然后使用 hog 函数提取 HOG 特征值，hog 函数的具体说明可见上述实验原理部分。原始数据集在使用时可能会存在一些问题，因此这里使用的是整理后的数据集，

数据集分为 POS 类和 NEG 类，分别读取两个类别的图像并计算 HOG 特征值。POS 类用 1 表示，NEG 类用 0 表示。

```
#读取POS类数据集
for filename in glob.glob(os.path.join(pos_im_path, "*.png")):
    #读取图像
    img = cv2.imread(filename, 0)
    #统一图像尺寸
    img = cv2.resize(img, (64, 128))
    #提取HOG特征
    fd = hog(img, orientations=9, pixels_per_cell=(8, 8), visualize=False, cells_per_block=(3, 3))
    #保存特征信息
    data.append(fd)
    labels.append(1)

#读取NEG类数据集
for filename in glob.glob(os.path.join(neg_im_path, "*.jpg")):
    #读取图像
    img = cv2.imread(filename, 0)
    #统一图像尺寸
    img = cv2.resize(img, (64, 128))
    #提取HOG特征
    fd = hog(img, orientations=9, pixels_per_cell=(8, 8), visualize=False, cells_per_block=(3, 3))
    #保存特征信息
    data.append(fd)
    labels.append(0)
data = np.float32(data)
labels = np.array(labels)
```

因为上述处理将数据集中所有图像进行读取，所以需要处理数据集，将数据集中的 70%分为训练集，30%分为测试集。

```
#将数据集中的70%分为训练集，30%分为测试集
num = int(len(data) * 0.7)
train_data = data[:num]
train_labels = labels[:num]
test_data = data[num:]
test_labels = labels[num:]
```

然后定义 SVM 或 KNN 模型，使用模型在训练集上训练，并在测试集上得到预测结果，计算模型在测试集上的准确率。

```
#使用SVM\KNN模型
model = LinearSVC()
#model = KNeighborsClassifier(n_neighbors=5)
model.fit(train_data, train_labels)
joblib.dump(model,  model_path)

#model = joblib.load(model_path)

#在测试集上计算模型准确率
predicted = model.predict(test_data)
mask = predicted == test_labels
correct = np.count_nonzero(mask)
result = (float(correct) / len(test_labels)) * 100
print('测试集准确率：%f%%' % (result))
```

模型训练后可以保存在本地，模型文件保存路径由源代码中的 model_path 变量指定，也可以通过 predict_path 变量指定待检测的图像文件路径。预测会使用滑动窗口，提取滑动窗口的 HOG 特征值，并输入至模型中，根据模型预测该图像区域中是否存在行人，然后输出指定图像的行人检测结果。本实验的具体代码请见 project3.py 文件。

4.基于 SIFT 算法的图片匹配

首先分别读取两幅图像，调整两幅图像的尺寸，并分别计算得到两幅图像的 SIFT 特征向量。

```
#读取图像
origimg = plt.imread('./SIFTimg/1.jpeg')
if len(origimg.shape) == 3:
    img = origimg.mean(axis=-1)
else:
    img = origimg
#计算得到SIFT特征向量
keyPoints,discriptors = SIFT(img)

origimg2 = plt.imread('./SIFTimg/01.jpeg')
if len(origimg.shape) == 3:
    img2 = origimg2.mean(axis=-1)
else:
    img2 = origimg2
ScaleRatio = img.shape[0]*1.0/img2.shape[0]

img2 = np.array(Image.fromarray(img2).resize((int(round(ScaleRatio * img2.shape[1])),img.shape[0]), Image.BICUBIC))
keyPoints2, discriptors2 = SIFT(img2,True)
```

在 SIFT 函数中，对于输入图像先使用 DoG 算子生成高斯金字塔，然后再根据上述实验原理部分所述的计算方法得到 SIFT 特征描述子。

```python
def SIFT(img,showDoGimgs = False):
    SIFT_SIGMA = 1.6
    SIFT_INIT_SIGMA = 0.5 # 假设的摄像头的尺度
    sigma0 = np.sqrt(SIFT_SIGMA**2-SIFT_INIT_SIGMA**2)

    n = 3

    #使用DoG算子生成高斯金字塔
    DoG,GuassianPyramid = getDoG(img, n,sigma0)
    if showDoGimgs:
        for i in DoG:
            for j in i:
                plt.imshow(j.astype(np.uint8), cmap='gray')
                plt.axis('off')
                plt.show()

    KeyPoints = LocateKeyPoint(DoG, SIFT_SIGMA, GuassianPyramid, n)
    #计算SIFT特征描述子
    discriptors = calcDescriptors(GuassianPyramid,KeyPoints)

    return KeyPoints,discriptors
```

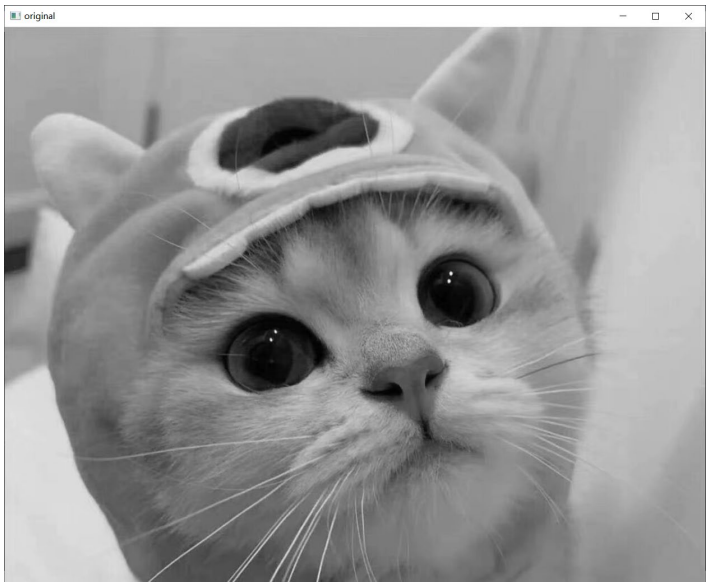
然后定义了 KNN 模型，使用 KNN 模型寻找两幅图像的匹配点，最后将这些匹配点在两幅图像上绘制出来。

```python
#使用KNN模型寻找两幅图像的匹配点
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(discriptors,[0]*len(discriptors))
match = knn.kneighbors(discriptors2,n_neighbors=1,return_distance=True)

keyPoints = np.array(keyPoints)[:,:2]
keyPoints2 = np.array(keyPoints2)[:,:2]

keyPoints2[:, 1] = img.shape[1] + keyPoints2[:, 1]

origimg2 = np.array(Image.fromarray(origimg2).resize((img2.shape[1],img2.shape[0]), Image.BICUBIC))
result = np.hstack((origimg,origimg2))

keyPoints = keyPoints[match[1][:,0]]

X1 = keyPoints[:, 1]
X2 = keyPoints2[:, 1]
Y1 = keyPoints[:, 0]
Y2 = keyPoints2[:, 0]

#绘制两幅图像的匹配点
drawLines(X1,X2,Y1,Y2,match[0][:,0],result)
```

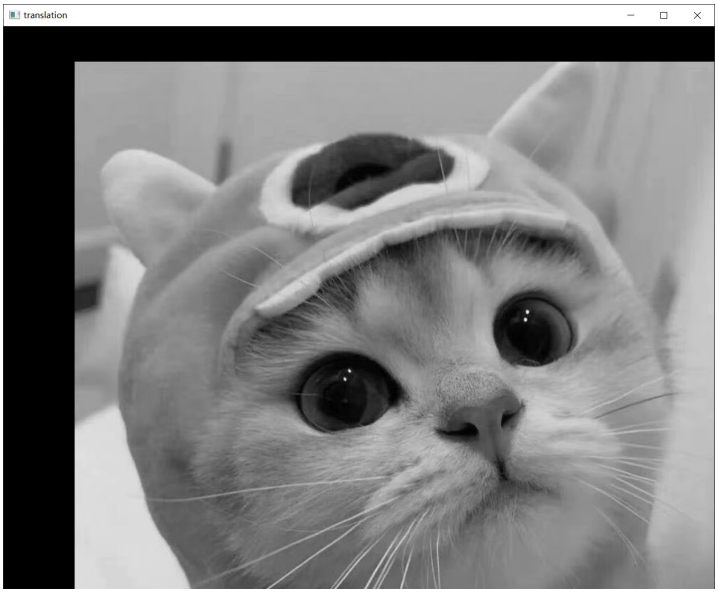
四、实验结果

1. 仿射变换

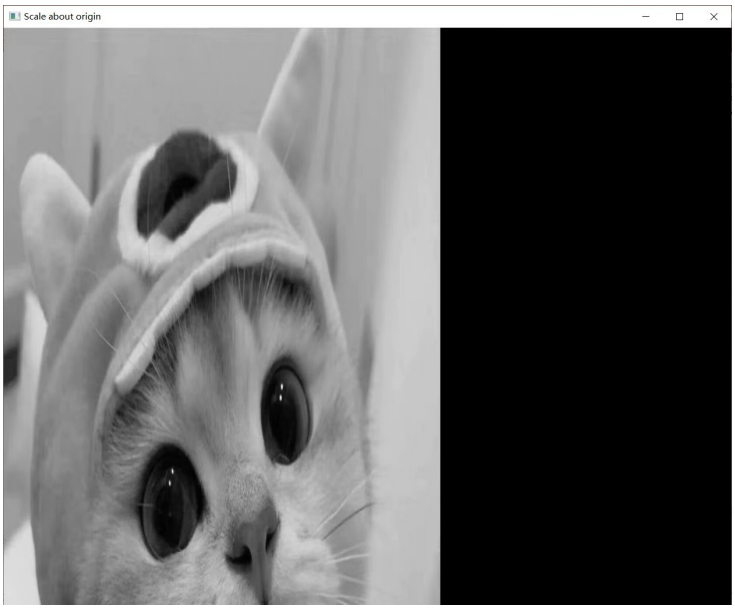
原灰度图



平移后的图片

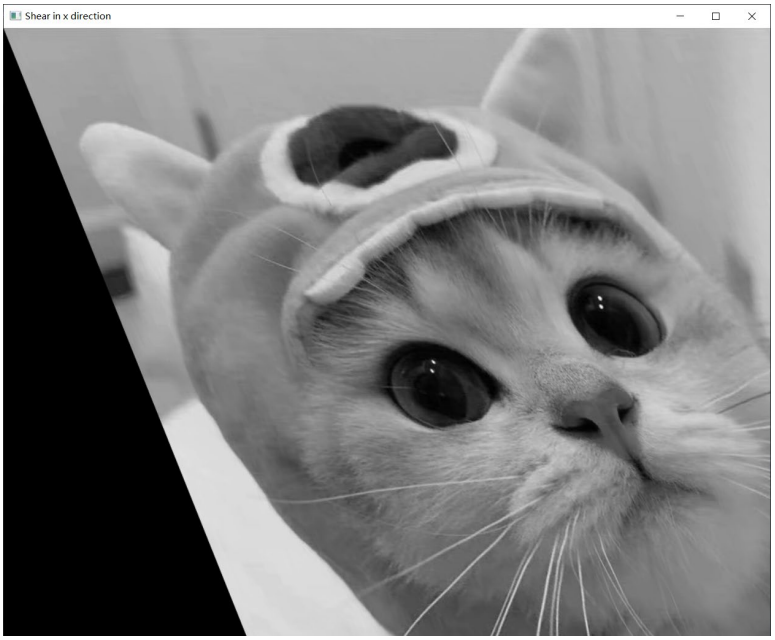


缩放后的图片

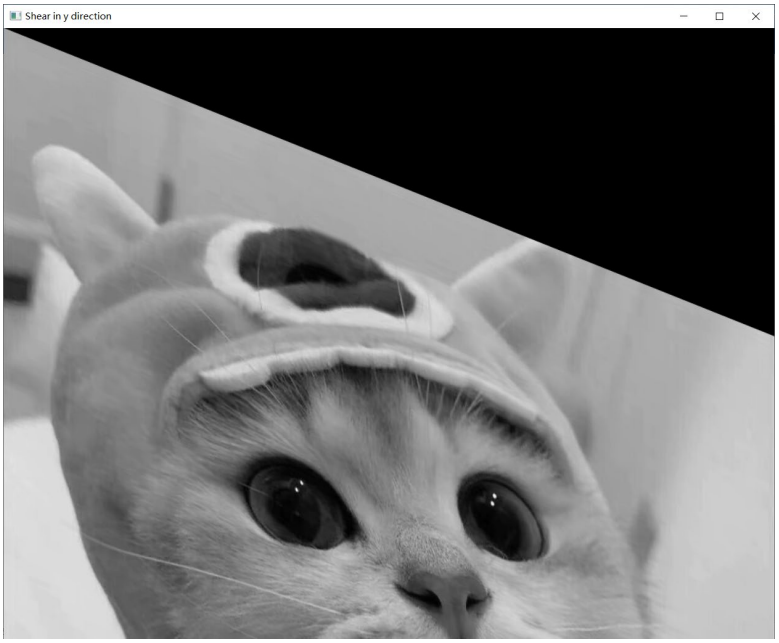


旋转后的图片

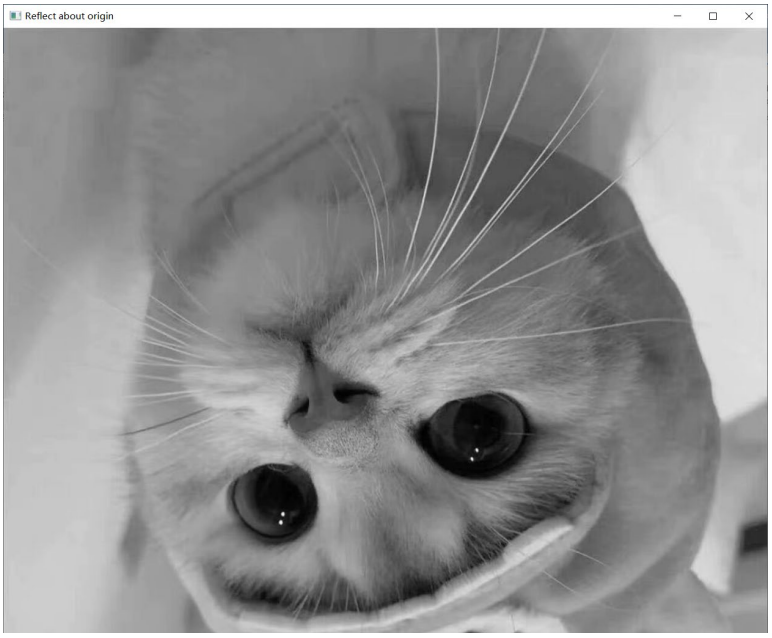
X 方向错切后的图片



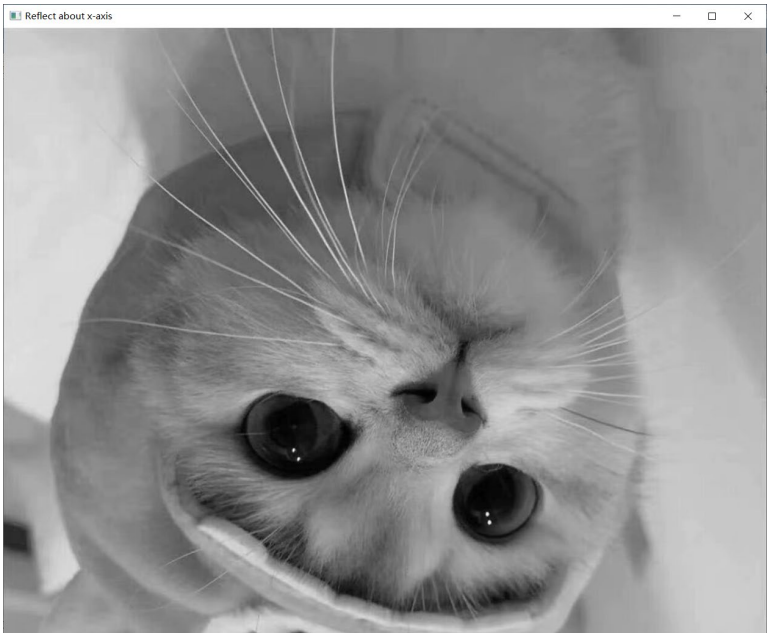
Y 方向错切后的图片



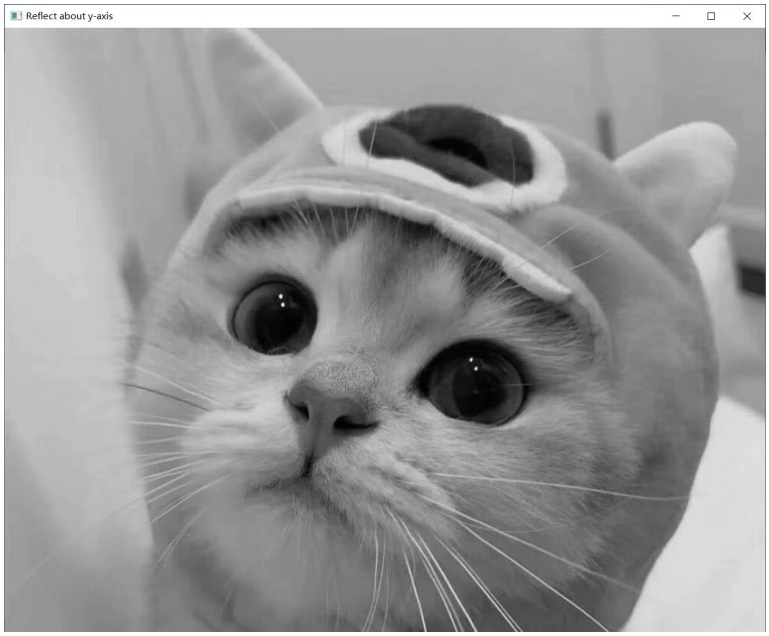
原点翻转后的图片



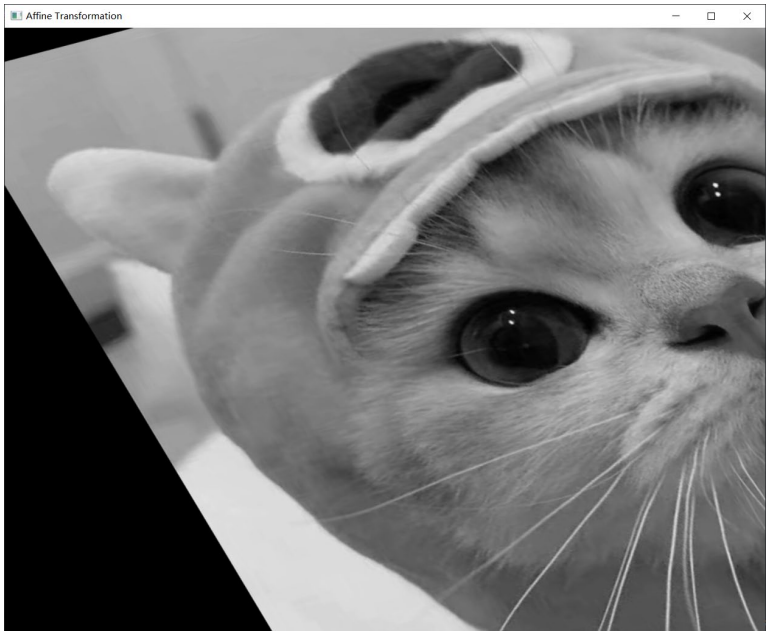
X 轴翻转后的图片



Y 轴翻转后的图片



仿射变换后的图片

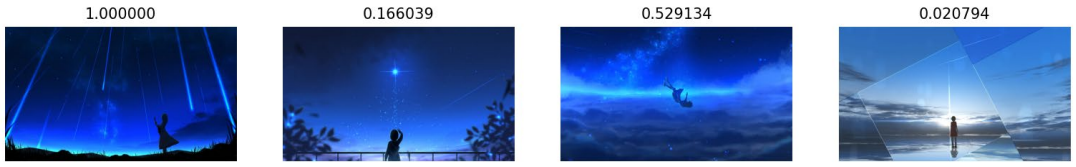


2.基于颜色直方图特征的图像匹配或查找

源图如图所示。

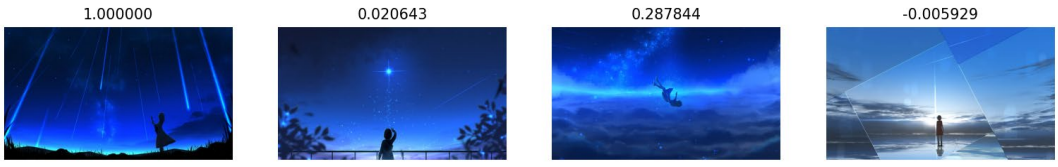
使用巴氏距离进行比较，范围为 0 到 1 之间，计算得到的值越大，相似度越高。

Bhattacharyya



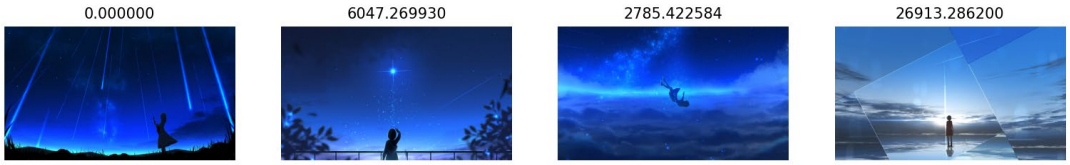
使用皮尔逊相关系数进行比较，范围为-1 到 1 之间，计算得到的值的绝对值越接近 1，相似度越高。

Correl



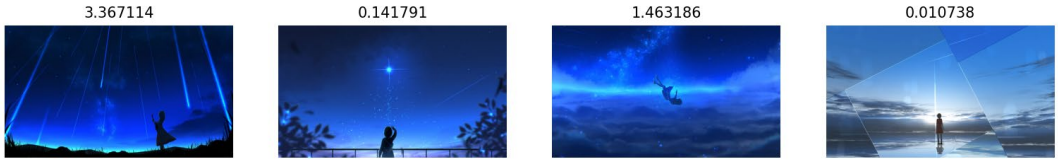
使用卡方比较，最小值为 0，最大值无上限，计算得到的值越小，相似度越高。

Chisquare



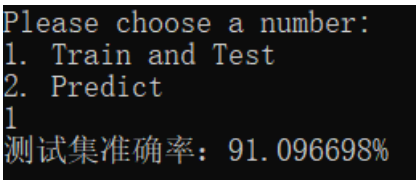
使用相交系数进行比较，计算得到的值越大，相似度越高。

Intersect



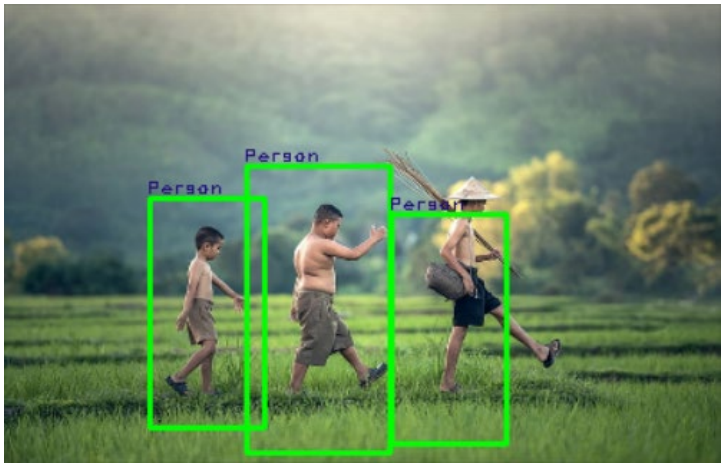
3.基于 HOG 特征，使用 SVM\KNN 完成目标行人检测

模型在测试集上的准确率



待检测的图像

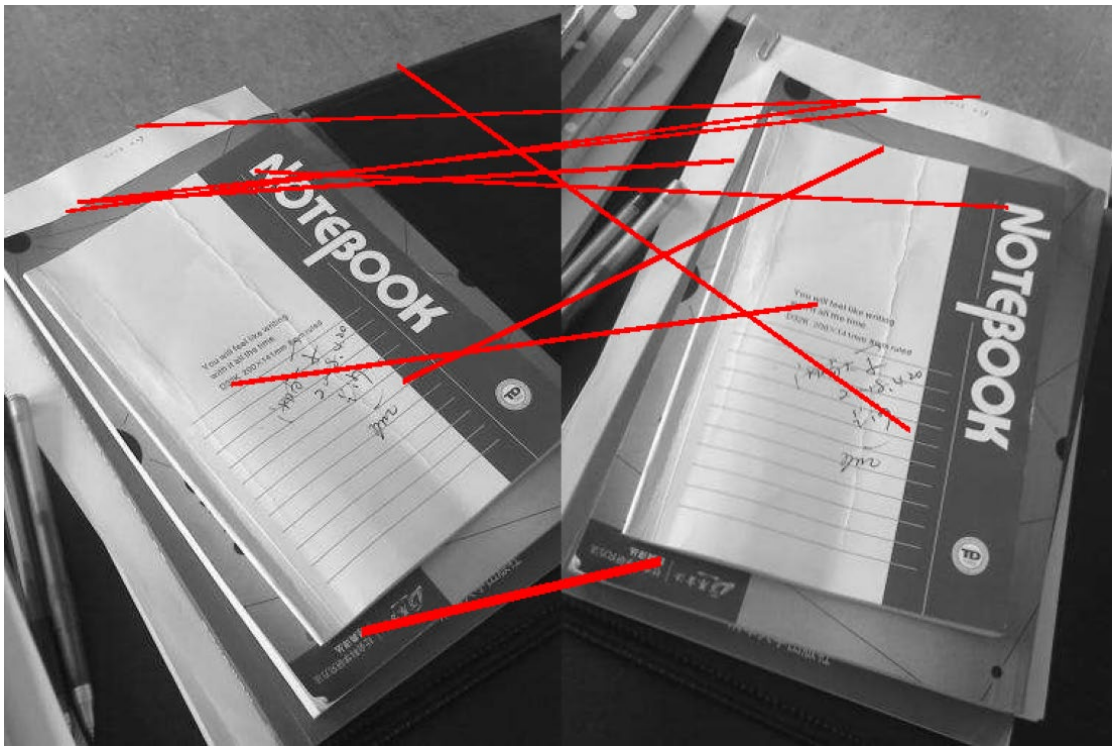
模型检测行人结果



4.基于 SIFT 算法的图片匹配

两幅经过视角平移的图像的 SIFT 算法匹配结果

两幅经过视角旋转的图像的 SIFT 算法匹配结果



两幅经过视角缩放的图像的 SIFT 算法匹配结果