信息安全基础课程实验报告

1813023 谭智元

1 引言

基于图像内容的复制检测 (*copy detection*) 已成为在版权保护领域最为重要的技术手段之一，该技术主要包括两段过程：基于内容的特征提取和图像匹配。但是，建立为匹配使用的特征库需要足量的存储空间，这不仅极大地增加了时间与空间的开销，而且缺乏灵活性。幸运的是，感知图像哈希（*perceptual image hashing*）是一种解决上述问题的良好手段，其提取基于图像内容的特征并将其编码为哈希值。一方面，基于内容的特征保证了更高的复制检测准确性；另一方面，使用哈希值取代特征库减少了空间消耗并提高了效率。同时，在鲁棒性与分辨能力之间寻求一种更好的平衡是图像哈希的主要研究目标之一，这将有助于图像哈希在多媒体管理和信息安全领域的应用。

为上述研究目标，刘世光等人[1] 为复制检测提出了一种有效的图像哈希方法。具体而言，为获取针对拷贝攻击（*copy attack*）的鲁棒性，该方法首先在用以显示图像纹理变化的灰度共生矩阵（*gray-level co-occurrence matrix, **GLCM***）上提取全局统计特征；之后，为提高分辨能力的上限，该方法对子图中首行及首列的局域 **DCT**（*discrete cosine transform*）系数稍加改造，用以计算向量距离（*vector distance*）；最后，这两种互补的信息（从纹理中获取的全局特征和从向量距离中获取的局部特征）被同时用来生成哈希值。

本文是对上述工作进行的**复现**。本文依据论文中提供的原理使用 ***Python*** 语言实现了该方法，并重现了论文中绝大部分的实验。特别需要指出的是，本文并未重现该方法与其他方法进行比较的实验，这是由时间精力等方面的限制以及复现其他方法技术上的困难所导致。基于实验结果，本文于末尾提出了对本次复现实验的总结及的展望。

2 原理

方法的主要框架如图 1所示。其主要包括三个部分（为了方便展示使用不同颜色标出）：基于内容的特征提取，哈希值生成，相似度计算。为了能稳定地提取特征，方法首先将对给定的图像进行预处理操作，包括双线性插值（*bilinear interpolation*）、高斯滤波（*Gaussian filtering*）和颜色空间转换（*color space conversion*）。这些都是非常基础的图像预处理技术，被广泛地应用在许多算法[2][3][4][5] 中。然后，分别提取由 **GCLM** 包含的全局纹理特征和由 **DCT** 包含的局部距离特征，这一部分将在 2.1节中详细介绍。接着，数字化提取的特征，并根据预先设定的规则将其转化成哈希值，这一部分将在 2.2节中阐释。最后，通过比较数据库中目标图像与待测图像哈希值之间的相似度，目标图像的所有拷贝版本将会被检测出来，这一部分将在 2.3节中展示。

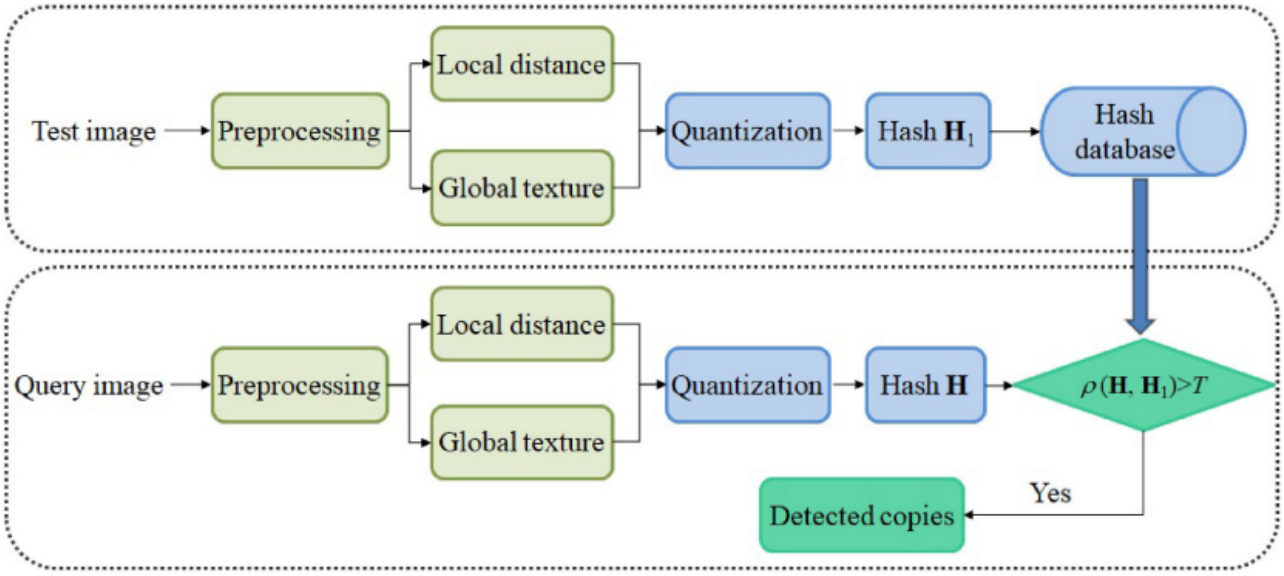


图 1: 方法框架

2.1 基于内容的特征提取

为减轻对图像内容的数字攻击带来的影响，方法在预处理阶段利用三项操作生成一幅标准化的图像，以便之后进行特征提取。原始图像的尺寸通过双线性插值变为 $B \times B$，以保证不同的图像拥有相同的哈希长度，也初步修正了像素间的几何失真。之后，为减小轻微噪声和由插值操作对图像质量引起的误差的影响，方法利用高斯低通滤波（*Gaussian low-pass filtering*）对标准化大小后的图像进行平滑。其卷积核可定义如下：

$$T_G = \frac{T^{(1)}(i,j)}{\sum_i \sum_j T^{(1)}(i,j)} \tag{1}$$

$T^{(1)}$ 由下式计算得到：

$$T^{(1)} = e^{\frac{-(i^2+j^2)}{2\sigma^2}} \tag{2}$$

其中，σ 是给定的高斯分布的标准差，i 和 j 表示卷积核中元素的下标。例如，若卷积核大小为 3×3，则 $-1 \leqslant i \leqslant 1$ 且 $-1 \leqslant j \leqslant 1$。

此外，方法选择 **HSI**（*hue, saturation, intensity*）颜色空间中的强度通道（*intensity component*）进行稳定特征的提取。这是因为 **HSI** 颜色空间更接近于人类的视觉感知。因此，对于使用 **RGB**（*red, green, blue*）颜色空间的图像，需要利用以下公式获取其强度通道：

$$I = \frac{1}{3}(R + G + B) \tag{3}$$

最后，经过上述预处理之后，方法将从标准化的强度图像中提取两种特征，即全局纹理特征和局部向量距离特征：

1. 全局纹理特征：纹理是一类具有鲁棒性的全局视觉特征，它能反映图像的同质性。纹理可通过一系列拥有特定灰度分布和空间结构的像素来表征，灰度共生矩阵（*gray-level co-occurrence matrix, **GLCM***）就是一类通过研究像素的空间关系来描述纹理的方法。**GLCM** 中定义了 14 项用以分析纹理的参数，其中只有 4 项参数互不相关且能使用较小的时间开销获得较高的分类准确性，即对比度（*contrast*），相关性（*correlation*），能量（*energy*）和同质性（*homogeneity*）。其

计算公式如下：

$$contrast = \sum_x \sum_y |x-y|^2 p(x,y) \tag{4}$$

$$correlation = \sum_x \sum_y \frac{(x-u_x)(y-u_y)p(x,y)}{\sigma_x \sigma_y} \tag{5}$$

$$energy = \sum_x \sum_y p(x,y)^2 \tag{6}$$

$$homogeneity = \sum_x \sum_y \frac{1}{1+|x-y|^2} p(x,y) \tag{7}$$

其中，$P(x,y)$ 是 $P(x,y|d,\theta)$ 的简化表示，$P(x,y|d,\theta)$ 表示在距值为 y 的像素的距离为 d，角度为 θ 的方向上，值为 x 的像素出现的次数。参考一个著名方法[6]，$d=1$ 且 $\theta = 0°, 45°, 90°, 135°$ 在权衡准确性和复杂度之间做得较好，因而使用这套参数。于是，对于每一个方向，方法都会计算出上述 4 项特征，因而在标准化后得到长度为 $4 \times 4 = 16$ 的全局纹理特征 **T**。

2. 局部向量距离特征：尽管纹理能提供一种全局的稳定特征，但其并不能反映图像的基本属性。与之相对，二维 **DCT**（***2D-DCT***）能将图像的能量集中到不同的频率上，其中能反映图像结构信息的低频系数主要集中在矩阵的左上部分。因此，方法选择 **DCT** 变换后矩阵首行与首列的元素构造向量特征，这样就能保留图像的低频特征（结构信息）。

具体而言，强度图像被切分成大小为 $b \times b$ 的子图，子图的数量为 $N = (B/b)^2$，其中 B 代表标准化后图片的大小。令 $\mathbf{B}_i$ 是从左至右及从上至小数的第 i 幅子图（$1 \leqslant i \leqslant N$），$B_i(j,k)$ 则表示 $\mathbf{B}_i$ 中位于第 $(j+1)$ 行和第 $(k+1)$ 列的像素。将 **2D-DCT** 作用于子图 $\mathbf{B}_i$，则首行 $C_i(0,v)$ 与首列 $C_i(u,0)$ 的系数由下式给出：

$$C_i(0,v) = \frac{\sqrt{2}}{b} \sum_{j=0}^{b-1} \sum_{k=0}^{b-1} B_i(j,k) \cos\left[\frac{(2k+1)v\pi}{2b}\right] \tag{8}$$

$$C_i(u,0) = \frac{\sqrt{2}}{b} \sum_{j=0}^{b-1} \sum_{k=0}^{b-1} B_i(j,k) \cos\left[\frac{(2j+1)u\pi}{2b}\right] \tag{9}$$

由于行 $C_i(0,v)$ 与 $C_i(u,0)$ 的末尾部分对于压缩和噪声敏感，方法只选取每行/列的第 2 至第 $n+1$ 个元素（$n = b/2$）构建向量 $Q_i = [C_i(0,1), C_i(0,2), \ldots, C_i(0,n), C_i(1,0), C_i(2,0), \ldots, C_i(n,0)]^T$。因此，基于 **DCT** 低频获取的特征矩阵 **Q**（大小为 $b \times N$）可以表示如下：

$$\mathbf{Q} = [\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_N] \tag{10}$$

之后，对 **Q** 的每一行进行标准化[7]，用 **U** 用来表示标准化后的 **Q**。同时，构造向量 $\mathbf{U}_0 = [u_0(1), u_0(2), \ldots, u_0(b)]^T$ 用来计算向量距离，其中 $u_0(l)(1 \leqslant l \leqslant b)$ 表示 **U** 中每一行的均值。令 $\mathbf{U}_i = [u_i(1), u_i(2), \ldots, u_i(b)]^T$ 表示矩阵 **U** 的第 i 列，则 $\mathbf{U}_i$ 和 $\mathbf{U}_0$ 间的欧拉距离（*Euclidean distance*）$d(i)$ 为：

$$d(i) = \sqrt{\sum_{l=1}^{b} (u_i(l) - u_0(l))^2} \tag{11}$$

至此，我们得到了 **DCT** 低频系数间的向量距离矩阵，表示为 $D = [d(1), d(2), \ldots, d(N)]$。

2.2 哈希值生成

上述两种特征（即全局纹理特征 **T** 和局部向量距离特征 **D**）提取完成后，可将这两种向量拼接生成哈希值，即 $\mathbf{F} = [\mathbf{TD}]$，其中 **F** 的长度为 $(16 + N)$。假定标准化后图像尺寸为 512×512，子图大小为 64×64（$N = (512/64)^2 = 64$）。显然，哈希序列 **F** 中元素数量为 80。

接着，为提高安全性，方法将利用秘钥生成的伪随机流应用于 **F** 之上，以构建具有安全性的哈希序列。具体而言，方法首先输入秘钥作为随机生成器的种子，便可生成长度为 80 的伪随机流（序列）。之后，对伪随机序列中的元素进行排序，使用向量 **Z** 记录排序后的元素在原始序列中的下标值。最后，利用向量 **Z** 对哈希序列 **H** 进行置乱，由以下等式表示：

$$h(q) = f(z(q)) \tag{12}$$

其中 $h(q)$ 和 $z(q)$ 分别表示 **H** 和 **Q** 中的第 q 个元素（$1 \leqslant q \leqslant 80$）。显然，哈希序列的长度决定了排列的数量，即 $80! = 7.16 \times 10^{118}$，这意味着在不知道秘钥的情况下，很难猜测出正确的哈希值。

最后，由于小数会占据许多位的存储空间，加密后的哈希序列中的元素将通过舍入操作转化成整型。为方便表示，仍使用 **H** 表示最终的图像哈希序列，即：

$$\mathbf{H} = [h(1), h(2), \ldots, h(80)] \tag{13}$$

2.3 相似度计算

拷贝图像是通过比较一对哈希值间的相似度（*similarity*）而被检测出来。即，给定一幅待测图像，其首先被编码成定长哈希序列 **H**，之后计算其与数据库中的哈希序列的相似度。若相似度大于阈值 T，则数据库中该哈希序列对应的图片就被判定为待测图片的拷贝版，否则不认为这两幅图片具有复制关系。相似度的计算公式如下：

$$\rho(\mathbf{H}, \mathbf{H}_1) = \frac{\sum_q^{80}[h(q) - u)][h_1(q) - u_1]}{\sqrt{\sum_q^{80}[h(q) - u)]^2 \sum_q^{80}[h_1(q) - u_1)]^2} + \Delta s} \tag{14}$$

其中，$h(q)$ 和 $h_1(q)$ 分别是 **H** 和 $\mathbf{H}_1$ 的第 q 个元素，u 和 u_1 分别为 **H** 和 $\mathbf{H}_1$ 的平均值。Δs 为一个极小常量，以避免分母为 0 的情况。相似度 $\rho(\mathbf{H}, \mathbf{H}_1)$ 越高，越能准确地检测出拷贝图像，反之亦然。

操作	参数值	数量
JPEG 压缩（*JPEG compression*）	$30, 40, \ldots, 100$	8
斑纹噪声（*Speckle noise*）	$0.001, 0.002, \ldots, 0.01$	10
椒盐噪声（*Salt and Pepper noise*）	$0.001, 0.002, \ldots, 0.01$	10
亮度调整（*Brightness adjustment*）	$\pm 10, \pm 20$	4
对比度调整（*Contrast adjustment*）	$\pm 10, \pm 20$	4
伽马修正（*Gamma correction*）	$0.7, 0.9, 1.1, 1.2$	4
高斯过滤（*Gaussian filtering*）	$0.3, 0.4, \ldots, 1.0$	8
图像缩放（*Image scaling*）	$0.5, 0.75, 0.9, 1.1, 1.5, 2.0$	6
旋转，裁剪，调整大小（*Rotation, cropping, rescaling*）	$\pm 1, \pm 2, \pm 3, \pm 4, \pm 5$	10
总数		64

表 1: 数字攻击操作与其参数

(a) 原始图像

(b) 旋转后图像

(c) 裁剪后图像

(d) 调整大小后图像

图 2: 旋转，裁剪，调整大小

3 实现与实验

3.1 实验配置与数据集

在具体实现上，输入图片的大小将被标准化为 512×512，切割的子图大小为 64×64，因而子图的总数为 64。另外，我们选择每张子图上每行/列的第 2 至第 33 个元素构造特征矩阵。也即，我们的参数选择为：$B = 512, b = 64, N = (B/b)^2 = 64, n = b/2 = 32$。因此，只包含整型元素的哈希序列长度为 80。

所有的复现代码均使用 Pycharm 2019.3.3(Community Edition) 软件编写，使用 Python 3.7 解释器执行。另外，部分测试数据的生成利用了 MATLAB R2014a 与 Adobe Photoshop CC 2018 软件。所有软件与代码运行在处理器为 Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz 和 8.0 GB RAM 的个人电脑上。实验使用了三个被广泛用在图像哈希和复制检测的数据集：USC-SIPI 数据集，Copydays 数据集和 UCID。在 USC-SIPI 数据集上，笔者选取其中具有代表性的 4 张图片进行实验

为了之后实验的需要，我们需要对图像进行某些数字攻击，以生成视觉上保留了原图内容的复制图像作为测试数据。我们所采用的操作如表 1所示。由于原论文中“水印攻击（*watermarking*）”这种数字攻击的方法并不明确，实现起来具有一定难度，且其对算法性能的影响不明显，因而在此抛弃 10 种水印攻击，使得数字攻击的总数只剩 64 种。其中，斑纹噪声和椒盐噪声通过 **MATLAB** 脚本添加，亮度调整和对比度调整通过编写 ***Javascript*** 脚本在 Photoshop 中批处理得到，其余操作均利用 ***Python*** 开源库 **OpneCV** 和 **PIL** 完成。

特别地，对于表中最后一行展示的“旋转，裁剪，调整大小”操作，具体流程如图 2所示。该操

作首先对输入的原始图像（如图 2a）进行旋转，得到如图 2b所示的图像。接着对旋转后的图像进行裁剪，去除那些由旋转引入的用于填充的像素（图 2a中的黑色像素块），得到的图像如图 2c所示。最后将裁剪后的图像重新缩放至原始图像的大小（如图 2d）。该操作对应的参数中“+”表示逆时针旋转，而“-”表示顺时针旋转。由于各软件并未显式提供上述操作，因此笔者手动实现了该部分。由图 2b所示，若要去除所有的填充像素，只需得到点 A 与点 B 相对于原图矩阵的坐标即可。显然，点的坐标可以通过计算 x 和 y 的值得到。由几何关系可以推出，x 和 y 的值可由以下公式计算得到：

$$x = \frac{H\sin|\theta| + (\cos|\theta| - 1)W}{2\cos|\theta|} \tag{15}$$

$$y = \frac{W\sin|\theta| + (\cos|\theta| - 1)H}{2\cos|\theta|} \tag{16}$$

其中，W 和 H 分别表示原图的宽和高，θ 表示旋转角度。因此，只需截取旋转后 (x, y)（左上角）至 $(W - x, H - y)$（右下角）的图像，再放大至原图尺寸即可。

3.2 感知鲁棒性的实验分析

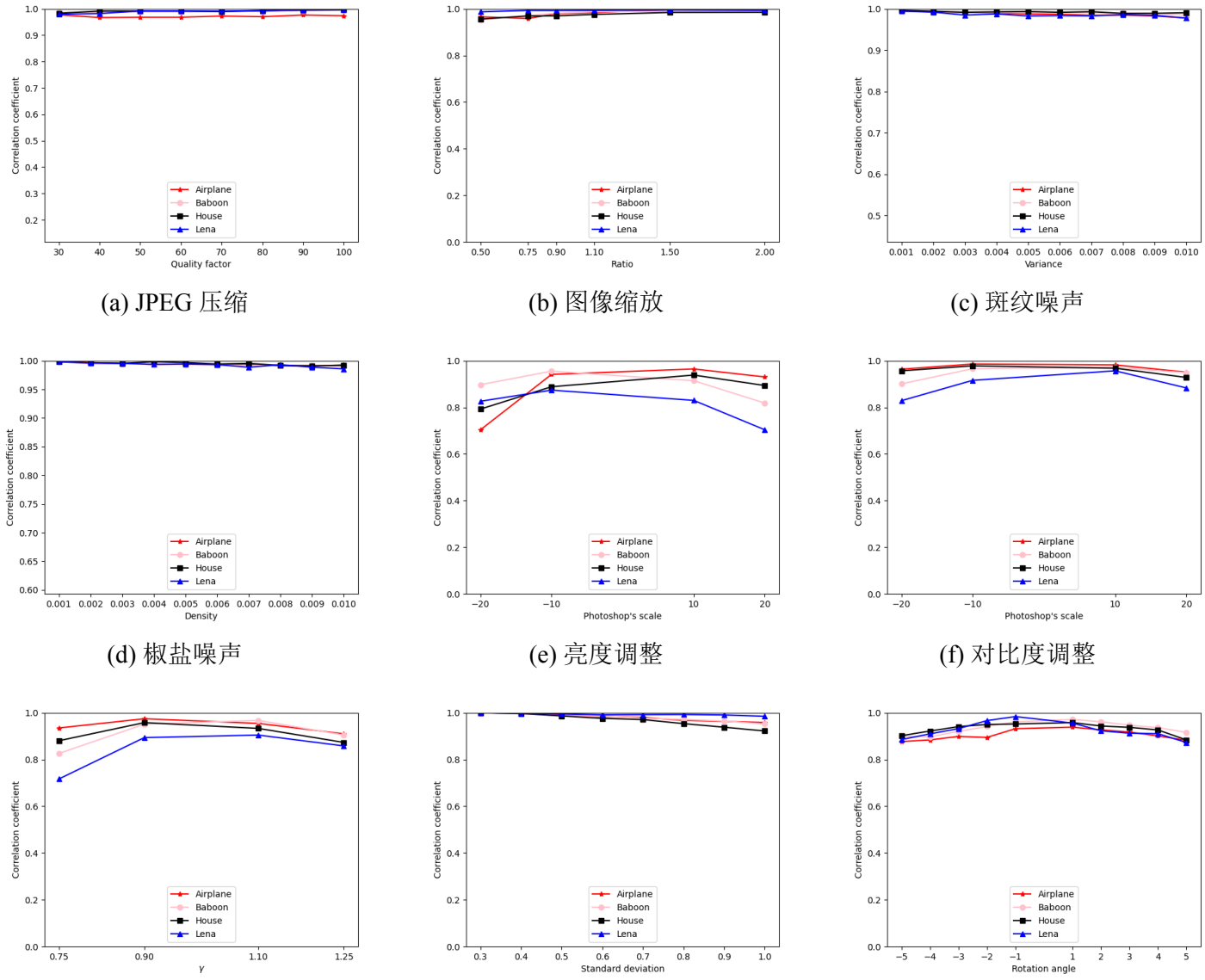
本节主要通过实验检验方法的感知鲁棒性，即方法生成的图像哈希值是否具有抵御数字攻击干扰的能力。因此，笔者分别在 USC-SIPI 数据集与 Copydays 数据集上进行了相关实验。具体而言，笔者分别对这两个数据集中的所有图片实施表 1中所有 64 种操作，以产生与原图视觉相似的拷贝图像。因此，在 UCS-SIPI 数据集上（只选取了 4 张图像）得到了 $4 \times 64 = 256$ 张拷贝图像，在 Copydays 数据集上（共有 157 张图像）得到了 $157 \times 64 = 10048$ 张拷贝图像。理论上来看，原图与其拷贝的版本相似度应接近于 1，方可说明方法的感知鲁棒性。

在 USC-SIPI 数据集上的实验结果如图 3所示。其中，每张折线图的的横坐标表示数字攻击的参数值，纵坐标表示相似度（相关系数）。从图 3可以看出，所有原图与其进行修改后的复制图之间的相关系数均大于 0.7035，绝大部分接近于 1，这说明笔者较好地实现了该方法，且对拷贝攻击具有良好的鲁棒性。

需要特别说明的是，图 3i是进行“旋转，裁剪，调整大小”操作的实验结果。在原论文中，进行该操作后，相关系数的值不再接近于 1，在部分图像上甚至骤降到 0.65 左右。但在笔者的实现方案下，该操作对相关系数的影响较小，其值均在 0.873 以上。这可能是由实现方案的不同导致。笔者采用自定义的裁剪方法“精准”剔除了多余的像素块，使得大部分的图像特征得以保留。而原论文可能采取在旋转后的图像中截取定长的矩阵块的策略，去除了更多的图像特征，因而得到了较差的实验结果。图 3i所示的结果是在旋转角度不大、像素矩阵为方阵（512×512）的情况下得到，对于图像长与宽不

操作	最小值	最大值	中位数	平均值
JPEG 压缩（*JPEG compression*）	0.2407	0.9997	0.9950	0.9613
斑纹噪声（*Speckle noise*）	0.2534	0.9990	0.9914	0.9568
椒盐噪声（*Salt and Pepper noise*）	0.2533	0.9988	0.9916	0.9576
亮度调整（*Brightness adjustment*）	0.0282	0.9882	0.8932	0.8492
对比度调整（*Contrast adjustment*）	0.4125	0.9978	0.9591	0.9343
伽马修正（*Gamma correction*	0.0607	0.9886	0.9054	0.8603
高斯过滤（*Gaussian filtering*）	0.9791	0.9997	0.9955	0.9950
图像缩放（*Image scaling*）	0.9727	0.9993	0.9953	0.9948
旋转，裁剪，调整大小（*Rotation, cropping, rescaling*）	0.2296	0.9914	0.9447	0.9066

表 2: 相似度的统计特征（Copydays 数据集）



(a) JPEG 压缩　(b) 图像缩放　(c) 斑纹噪声

(d) 椒盐噪声　(e) 亮度调整　(f) 对比度调整

(g) 伽马修正　(h) 高斯过滤　(i) 旋转，裁剪，重调

图 3: 相似度结果（USC-SIPI 中 4 幅图像与其被数字修改的图像计算得到）

等的情形可能有更为真实的表现，这将在之后的实验结果中进行说明。

另外，在原论文中，“亮度调整”、“对比度调整”和“伽马修正”的结果均特别好（相关系数均接近与 1），在笔者实验中的表现却不尽如人意。这可能是因为这三种操作均显著影响了 **GLCM** 中的 *contrast* 统计量。该统计量的范围位于 $[0, (row-1)^2]$（其中 row 表示矩阵的行数），而在标准化后其取值范围锐减，且在最后存储时还需进行四舍五入的操作，导致该特征最后的取值仅有接近 0 的几个整数。这导致与 *contrast* 相关的信息丢失严重，难以据此判断是否为拷贝图像，因而出现如图所示的情况。

由于 Copydays 数据集上的实验结果众多，笔者不选择使用折线图一一对比展示，而是选取结果的某些统计特征放在表中进行展示，即最大值，最小值，中位数和平均值，结果如表 2所示。可以发现，相关系数的平均值均在 0.8492 之上），且中位数均大于 0.8932，说明得到的相关系数大部分位于 $[0.8932, 1.0000]$ 区间内，反映出实现的方法总体表现优秀，能较好地检测出拷贝攻击。值得注意的是，除去“高斯过滤”和“图像缩放”操作外，笔者的实现方法表现出相当低的最小值（甚至达到 0.0282），说明笔者的实现方案并不具备良好的鲁棒性。其原因已在前文论述，但相较于 USC-SIPI 数据集的 4 张图像的结果仍有明显降低，一定程度上反映了多重数字攻击的真实影响。然而，表中“亮度调整”

和“伽马修正”操作的结果却出人意料地低（最小值分别为 0.0282 和 0.0607），这可能是由于伽马修正可以作为调整亮度的一种手段，而基于 ***Python*** 的实现（使用的库和解释器等）有可能对图像亮度相关操作较为敏感，笔者也鉴于二者结果的相似性推测 Photoshop 中对亮度的调整部分利用了伽马修正。

3.3 分辨能力的实验分析

我们选择使用公开的图像数据集 **UCID** 进行算法分辨能力的测试。**UCID** 数据集中包含许多具有高度视觉相似性的图像（某些图像的相似程度甚至高于基于原图修改后的图像与原图的相似程度），但因其是在不同的拍摄条件下（例如不同的拍照角度等）获取的，它们应被认为是不同的图像。因此，对该算法而言，能否将该数据集上视觉相似的图像判定为不同的图像是极大的挑战，由此获得的实验结果也将更具有说服力。

首先，笔者提取出 **UCID** 中 1338 幅图像的哈希值，之后计算其两两之间的相关系数，最后得到 $C_{1338}^2 = 1338 \times (1338-1)/2 = 894453$ 个结果。其最小值、最大值和平均值分别为 -0.2502、0.9647 和 0.5558。注意到，相关系数的平均值 0.5558 显著小于拷贝图像间的相关系数（表 2中的最小平均值为 0.8603），说明复现的算法具有有效区分不同图像的能力。

为了度量算法的鲁棒性和分辨能力，这里引入两种衡量标准：真阳率（*true positive rate*, P_{TPR}）和假阳率（*false positive rate*, P_{FPR}）。P_{TPR} 代表拷贝图像被成功检出的比例，该值越高说明算法鲁棒性越强。P_{TPR} 代表不同图像被错误检测为拷贝图像的比例，该值越小说明算法分辨能力越强。其可由以下公式计算得到：

$$P_{TPR}(\rho \geqslant T) = \frac{n_1}{N_1} \tag{17}$$

$$P_{FPR}(\rho \geqslant T) = \frac{n_2}{N_2} \tag{18}$$

在式(17)中，n_1 表示被正确检测为拷贝图像的数量，N_1 表示拷贝图像的总数（即 $N_1 = 10048$）。在式(18)中，n_2 表示被错误检测为拷贝图像的数量，N_2 表示 **UCID** 数据集上图像对的总数（即 $N_2 = 894453$）。T 表示用来区分图像的阈值。使用不同的阈值计算得到的 P_{FPR} 和 P_{TPR} 如表 3所示。显然，随着阈值 T 的减小，P_TPR 逐渐增大，说明正确识别的拷贝图像增多，意味着算法的良好鲁棒性；同时，被错误识别为拷贝图像的图像数目也逐渐增大，导致 P_FPR 的递增。可以看到，在 $T = 0.86$ 时，$P_{FPR} = 1.96 \times 10^{-3}$ 且 $P_{TPR} = 90.06\%$。也就是说，在 10^3 的图片中仅有 1 张被错误地检出，而与此同时保持着 90% 以上的识别准确率。尽管笔者复现的算法未能达到原论文中优秀的实验结果（$T = 0.90 : P_{FPR} = 1.34 \times 10^{-5}, P_{TPR} = 96.81\%$），但其依旧具备良好的鲁棒性和区分能力。

阈值	P_{FPR}(**UCID**)	P_{TPR}(**Copydays**)	阈值	P_{FPR}(**UCID**)	P_{TPR}(**Copydays**)
$T = 0.97$	0	67.80%	$T = 0.90$	1.80×10^{-4}	86.05%
$T = 0.96$	2.24×10^{-6}	72.24%	$T = 0.89$	3.33×10^{-4}	87.45%
$T = 0.95$	2.24×10^{-6}	75.68%	$T = 0.88$	6.46×10^{-4}	88.56%
$T = 0.94$	6.71×10^{-6}	78.83%	$T = 0.87$	1.17×10^{-3}	89.35%
$T = 0.93$	1.45×10^{-5}	81.31%	$T = 0.86$	1.96×10^{-3}	90.06%
$T = 0.92$	3.47×10^{-5}	83.03%	$T = 0.85$	3.09×10^{-3}	90.51%
$T = 0.91$	8.72×10^{-5}	84.59%	$T = 0.84$	4.75×10^{-3}	91.06%

表 3: 不同阈值下 P_{FPR} 与 P_{TPR} 的值

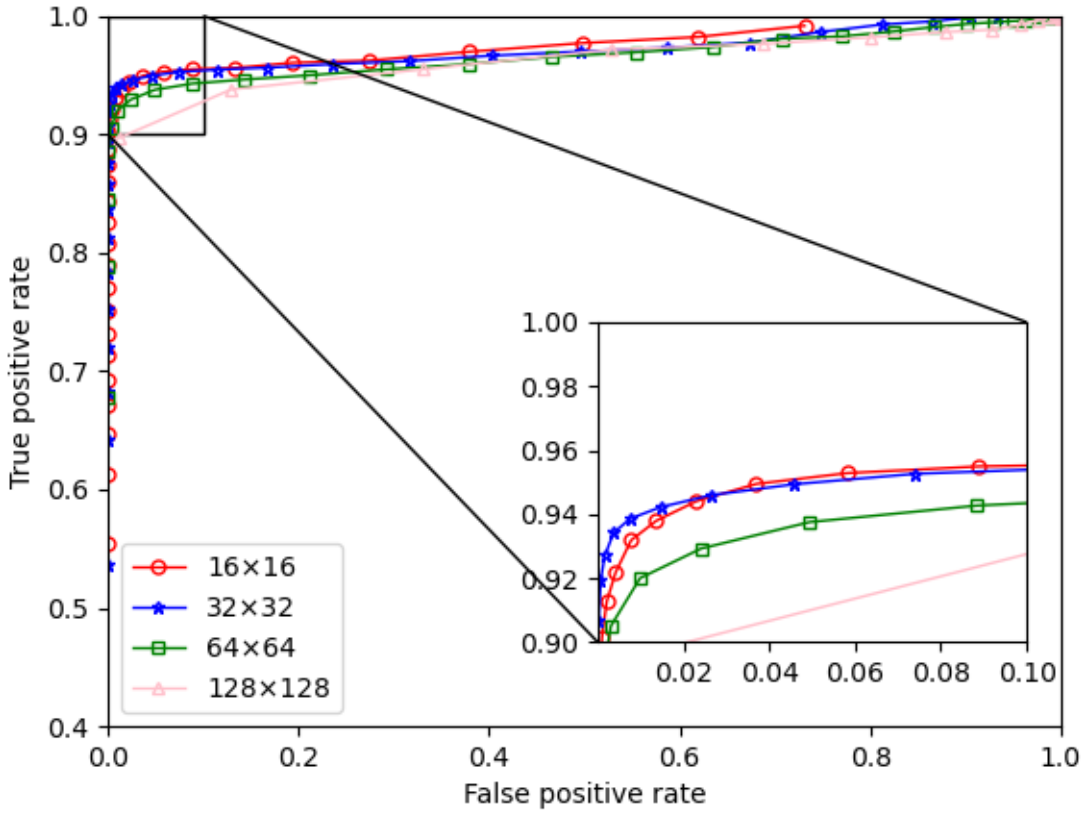


图 4: 改变子图大小得到的 **ROC** 曲线图

3.4 参数的影响分析

这一小节通过分析受试者工作特征曲线（*Receiver Operating Characteristic*, ***ROC***）探究不同子图大小对哈希算法的影响。**ROC** 曲线是由在不同阈值下计算得到的一系列点 (P_{FPR}, P_{TPR}) 组成，其被用来衡量算法鲁棒性和分辨能力之间的平衡性。在 **ROC** 曲线中，x 轴代表 P_{FPR}，y 轴代表 P_{TPR}。越靠近左上角的曲线代表的算法具有在鲁棒性和分辨能力间更好的平衡性。

本节实验均在只改变子图大小而保持其他参数不变的条件下进行，使用到的子图大小为 16×16、32×32、64×64 和 128×128。实验仍计算出 10048 对拷贝图像的相关系数与 89443 对不同图像的相关系数，然后使用不同的阈值得到 **ROC** 曲线，如图 4所示。可以看到，所有 **ROC** 曲线均距左上角较近，证明算法的有效性不随参数的改变而剧烈波动。在笔者的实现方法中，16×16 与 32×32 参数下的曲线更接近左上角（两者相差无几），优于 64×64 参数下的算法，与原论文不一致（原论文 64×64 参数为最优）。这可能是因为更小的子图包含着更丰富的细节信息，使得算法在进行分辨时拥有更坚实的依据。但如表 4所示，64×64 参数下 **AUC**（*Area Under ROC Curve*）的值最大，说明在该参数下算法具有最佳的鲁棒性与分辨能力的平衡。

子图大小	AUC	哈希长度
16×16	0.70787	1040 位
32×32	0.90660	272 位
64×64	0.96591	80 位
128×128	0.95110	32 位

表 4: 相似度的统计特征（Copydays 数据集）

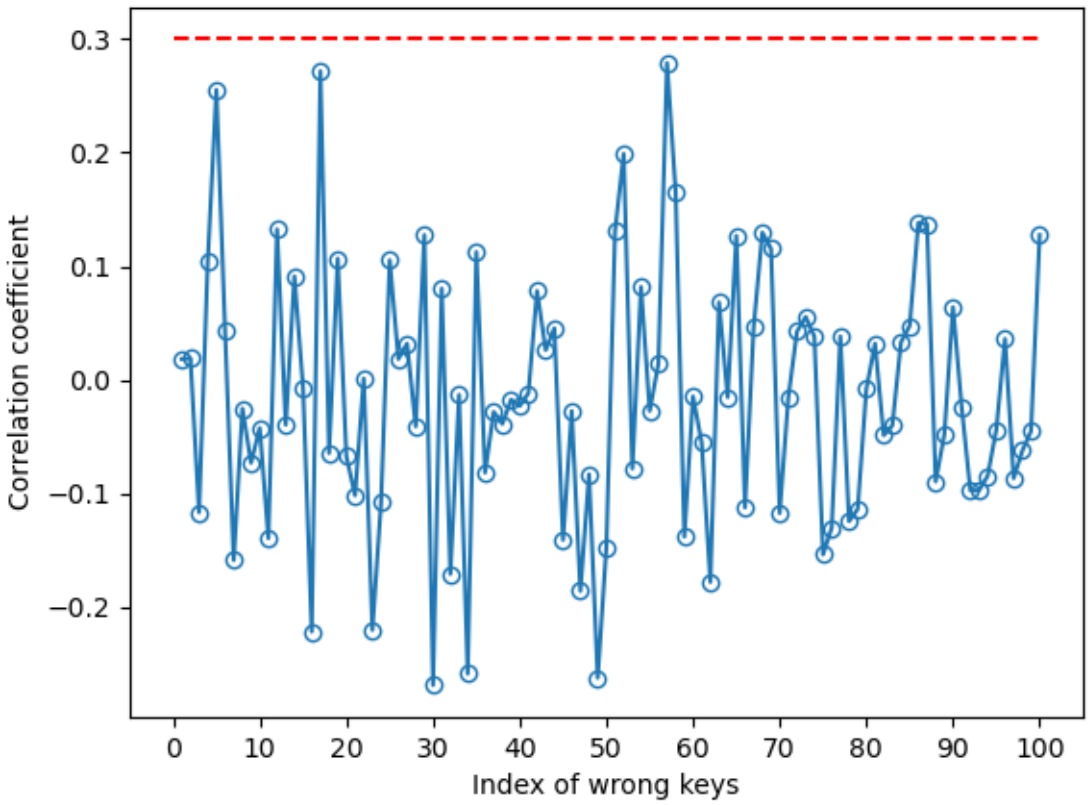


图 5: 使用正确的秘钥和其余 100 个不同秘钥生成的哈希值间的相关系数

3.5 秘钥敏感性分析

秘钥敏感性表示，对于使用不同的秘钥生成的哈希值，其相关系数应尽可能地小。针对 USC-SIPI 数据集上的 *Airplane* 图像，笔者在每次实验时随机生成 100 个由 $ASCII$ 码位于 $[33, 126]$ 区间的字符组成，长度在 $[1, 100]$ 之间的字符串作为错误秘钥，并计算错误秘钥生成的哈希值与正确秘钥生成的哈希值间的相关系数。其中某次实验结果如图 5所示。

可以看到，大部分的相关系数在 ± 0.3 间波动，所有相关系数均小于 0.3，这说明对于同一幅图片，在更换秘钥之后生成的哈希值与原来的哈希值间的相似度很低，即相同的图片利用不同的秘钥生成哈希值就会被判定为不相关的图片，因而论证了算法的秘钥依赖性。但是，笔者的实现方案很可能导致不稳定的秘钥依赖性。在笔者的实现方案中，秘钥字符串先通过哈希函数映射到某 16 位比特的值，然后其再作为种子生成秘钥流。由于使用的流生成函数只接受至多 16 位比特作为随机种子，因而可能的种子只有 $2^{16} = 65536$ 个。不同长度与组成的秘钥字符串在被映射到 2^{16} 比特空间时极有可能出现映射到相同值的情形，这就造成不同的秘钥可能生成相同流的情况。笔者将这作为未来的改进方向之一，通过自己实现或寻找新的函数扩充比特空间，从而减小碰撞几率。

3.6 颜色空间分析

在数字图像处理领域，在不同颜色空间中进行转换是一种常用的技术手段。特别对于基于图像内容的特征提取而言，不同的颜色空间对算法的性能均有一定的影响。因此，本节实验将在不同的颜色空间（**HSI**、**CIE Lab**、**YCbCr**）提取特征，并进行对比实验。

具体而言，**HSI** 空间的 **I** 通道、**CIE Lab** 空间的 **L** 通道和 **YCbCr** 空间的 **Y** 通道将被当做强度组件（*intensity component*）进行特征提取。在其余参数保持不变的情况下，运行哈希算法并采取与 3.4节

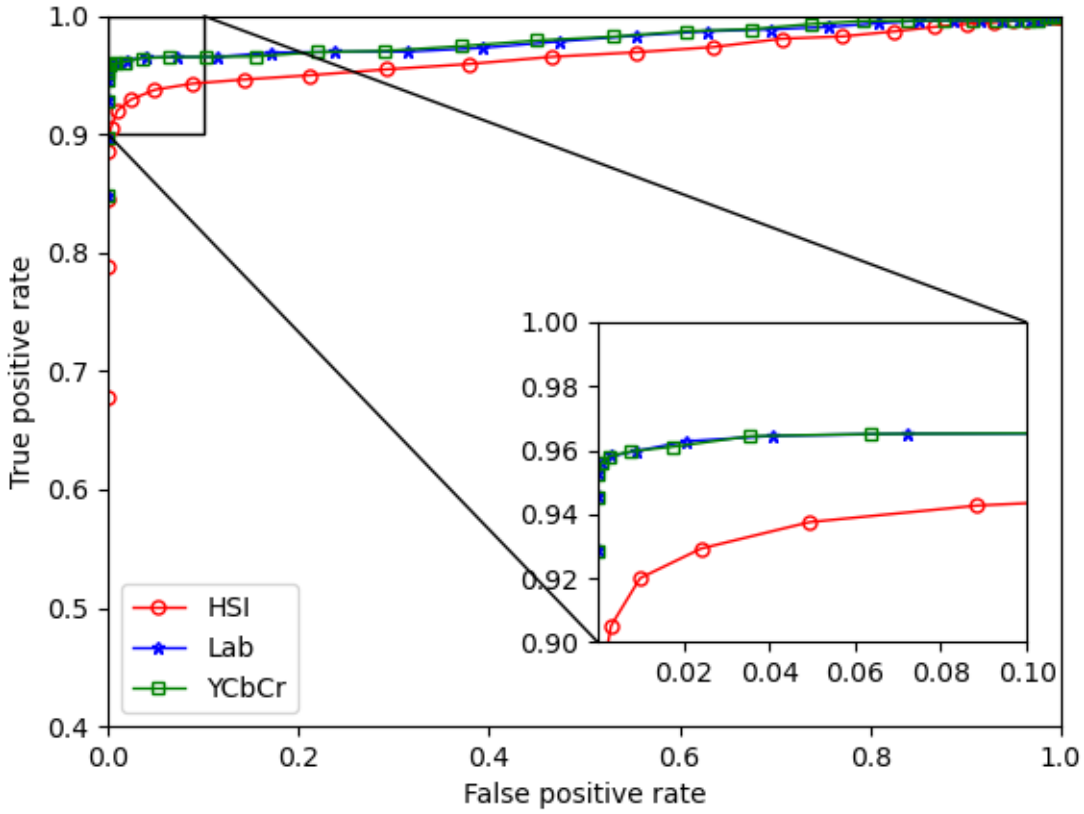


图 6: 不同颜色空间下的 **ROC** 曲线

相同的方法获取 **ROC** 曲线。结果如图 6所示。可以看到，在 **CIE Lab** 和 **YCbCr** 空间下的曲线（二者相差无几）更靠近左上角，这意味使用在 **CIE Lab** 和 **YCbCr** 空间中提取的特征能使算法具有更好的平衡性。同时，在 **HSI**、**CIE Lab** 和 **YCbCr** 颜色空间下的 **AUC** 分别为 0.9659，0.9757 和 0.9761，这进一步证明笔者复现的算法更适合在 **CIE Lab** 和 **YCbCr** 颜色空间中提取特征。这与原论文的主张相违背，目前笔者想不到合理的解释。

3.7 拷贝检测性能

本节的实验用来测试算法拷贝检测的准确性。本节使用的数据集[8] 包含 10 种（非洲，沙滩，建筑，公交车，恐龙，大象，花，马，山和食物）类别，每种类别包含 100 张尺寸为 256×384 或 384×256 的图像（共计 $10 \times 100 = 1000$ 张）。每一种类别里的图像在视觉上极为相似，但均为在不同的拍摄条件下得到。构造数据集的过程如下：首先，从原数据集的每个类别均抽取一张图像构造查询数据库（存储这 10 张图像的哈希值在数据库中）；之后，对查询数据库的每张图像进行 15 种拷贝攻击，因而得到 150 张拷贝图像；最后，将这些拷贝图像与原数据集混合，构成测试数据集（共有 $1000 + 150$ 张图像）。这 15 种拷贝攻击如下所示：

- 参数为 50 的亮度调整
- 参数为 50 的对比度调整
- 参数为 1.5 的伽马修正
- 参数为 0.6 的高斯过滤
- 参数为 0.05 的斑纹噪声
- 参数为 0.05 的椒盐噪声

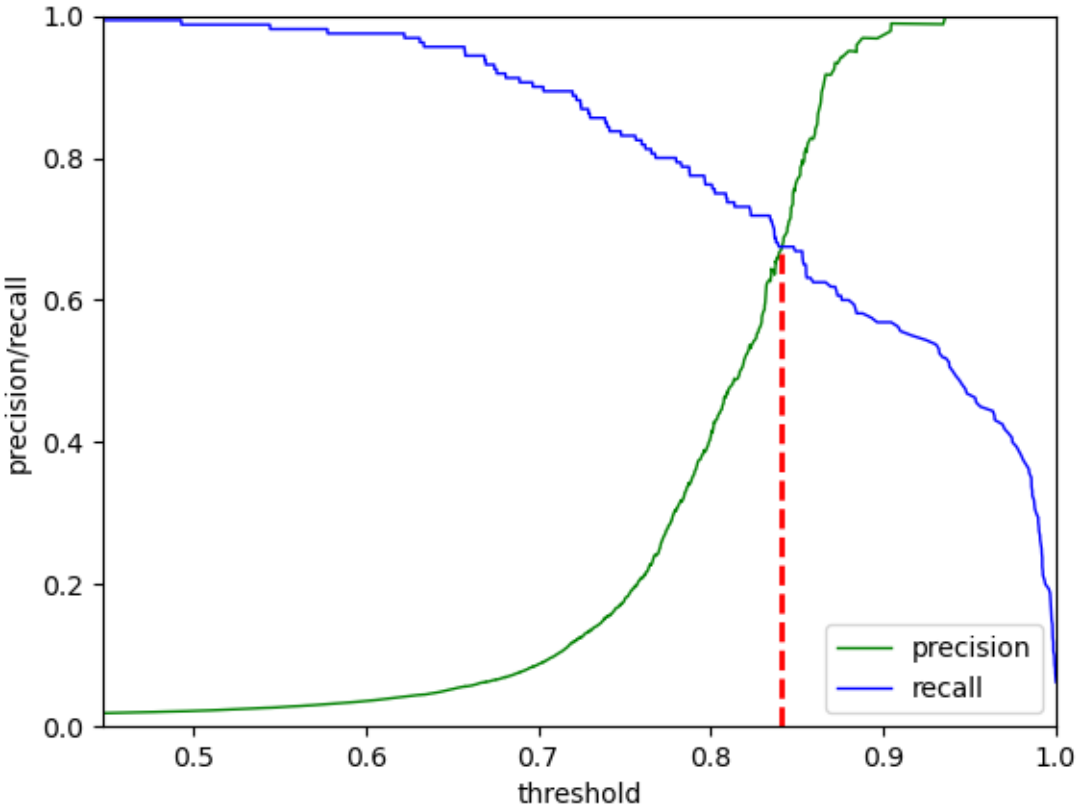


图 7: 查准率（*precision*）和查全率（*recall*）随不同阈值变化的曲线

- 参数为 0.05 的白噪声（高斯噪声）
- 参数为 30，50 的 JPEG 压缩
- 插入文本“Copyright in 2019”
- 参数为 50，75 的图像缩放
- 参数为 5°，8°，10° 的旋转，裁剪，调整大小

实验采用查准率（*precision*）和查全率（*recall*）随不同阈值变化的曲线展示实验结果。在给定的阈值下，查准率（*precision*）和查全率（*recall*）通过以下公式计算：

$$precision = \frac{numberofcorrectlydetectedcopies}{numberofallreturnedresults} \tag{19}$$

$$precision = \frac{numberofcorrectlydetectedcopies}{numberofallcopies} \tag{20}$$

实验结果如图 7所示。当阈值从 0 增长至 1 时，查全率降低而查准率升高。于是，在区间 $[0.8, 0.9]$ 之间，查全率与查准率在某个阈值下近似相等，此时认为算法达到最佳的性能，定义其为算法的最优阈值。在笔者复现的算法中最优阈值约为 0.841，此时查全率和查准率均达到 0.671 左右。相较于原论文的结果（约 0.83）具有显著降低，说明笔者的实现方案未能达到论文作者实现方案相同的检测性能。

4 总结与展望

在本课程中，笔者相对广泛性地了解了信息安全领域各个方向正在进行的工作，并选择图像哈希领域作为本次的复现工作的展开。笔者采用纯 **Python** 语言按照论文[1] 思路重现其工作，除去多种算法之间的比较实验未能完成外（受时间与精力的限制），其余实验均在保证与原论文尽可能一致的条

件下完成。从结果上来看，可能由于是实现方法的不同，多数实验的结果与原论文有出入，整体工作并不算效果出众，在特征提取和序列加密方面仍有改进的空间。但更重要的是，在复现的过程中学习到了图像哈希算法的基本思想，了解了数字图像的相关处理方法，探寻了获取图像特征的多种途径，还尝试使用 LaTeX 进行本文的书写，对以后类似工作的开展有极大帮助。笔者复现的代码、使用的数据集、实验结果等物料均已开源[9]。

参考文献

[1] HUANG Z, LIU S. Perceptual Image Hashing With Texture and Invariant Vector Distance for Copy Detection[J]. IEEE Transactions on Multimedia, 2020, 23: 1516-1529.

[2] KHELIFI F, JIANG J. Perceptual image hashing based on virtual watermark detection[J]. IEEE Transactions on Image Processing, 2009, 19(4): 981-994.

[3] TANG Z, ZHANG X, ZHANG S. Robust perceptual image hashing based on ring partition and NMF[J]. IEEE transactions on knowledge and data engineering, 2013, 26(3): 711-724.

[4] TANG Z, RUAN L, QIN C, et al. Robust image hashing with embedding vector variance of LLE[J]. Digital Signal Processing, 2015, 43: 17-27.

[5] LIN C Y, CHANG S F. A robust image authentication method distinguishing JPEG compression from malicious manipulation[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2001, 11(2): 153-168.

[6] HARALICK R M, SHANMUGAM K, DINSTEIN I H. Textural features for image classification[J]. IEEE Transactions on systems, man, and cybernetics, 1973(6): 610-621.

[7] TANG Z, HUANG Z, ZHANG X, et al. Robust image hashing with multidimensional scaling[J]. Signal processing, 2017, 137: 240-250.

[8] WANG J Z, LI J, WIEDERHOLD G. SIMPLIcity: Semantics-sensitive integrated matching for picture libraries[J]. IEEE Transactions on pattern analysis and machine intelligence, 2001, 23(9): 947-963.

[9] https://github.com/BHbean/Papers/ImageHashing.