**LSB卡方分析**

## 1. 卡方分析简介

卡方分析是一种统计检验方法，用于评估观察到的数据与期望数据之间的差异是否显著。在隐写分析中，卡方检验常用于检测图像中隐写操作引入的统计异常，特别是针对像素值或变换域系数的分布变化。

### 1.1 卡方分析在隐写分析中的应用

在隐写分析中，卡方检验可以用于：

（1）空间域检测：

单比特替换检测：分析图像中 LSB（最低有效位）替换后的像素值分布是否与原始图像的分布有显著差异。

多比特替换检测：扩展至多位替换，检测更复杂的隐写操作。

（2）变换域检测：

频域系数检测：如 DCT（离散余弦变换）系数的统计分析，用于检测 JPEG 图像中的隐写操作（如 EzStego）。

### 1.2 卡方分析原理

卡方分析的步骤是：图像中像素灰度值为i，其中0≤ i ≤255。如果载体图像未经隐写，像素灰度值为2j的数目和像素灰度值为（2j+1）的数目会相差得很远。秘密信息在嵌入之前往往经过加密，可以看作是0、1 随机分布的比特流，而且值为0与1 的可能性大致都是1/2。如果秘密信息完全替代载体图像的最低位，那么两个的值会比较接近，可以根据这个性质判断图像是否经过隐写。下图1显示了在隐写率为1时，图片像素值频率直方图的变化。可以看到，在经过LSB隐写后，像素灰度值频率分布直方图会有较大变化。像素灰度值为2j的数目和像素灰度值为（2j+1）的数目之间的差距会缩小，大致趋于相等。

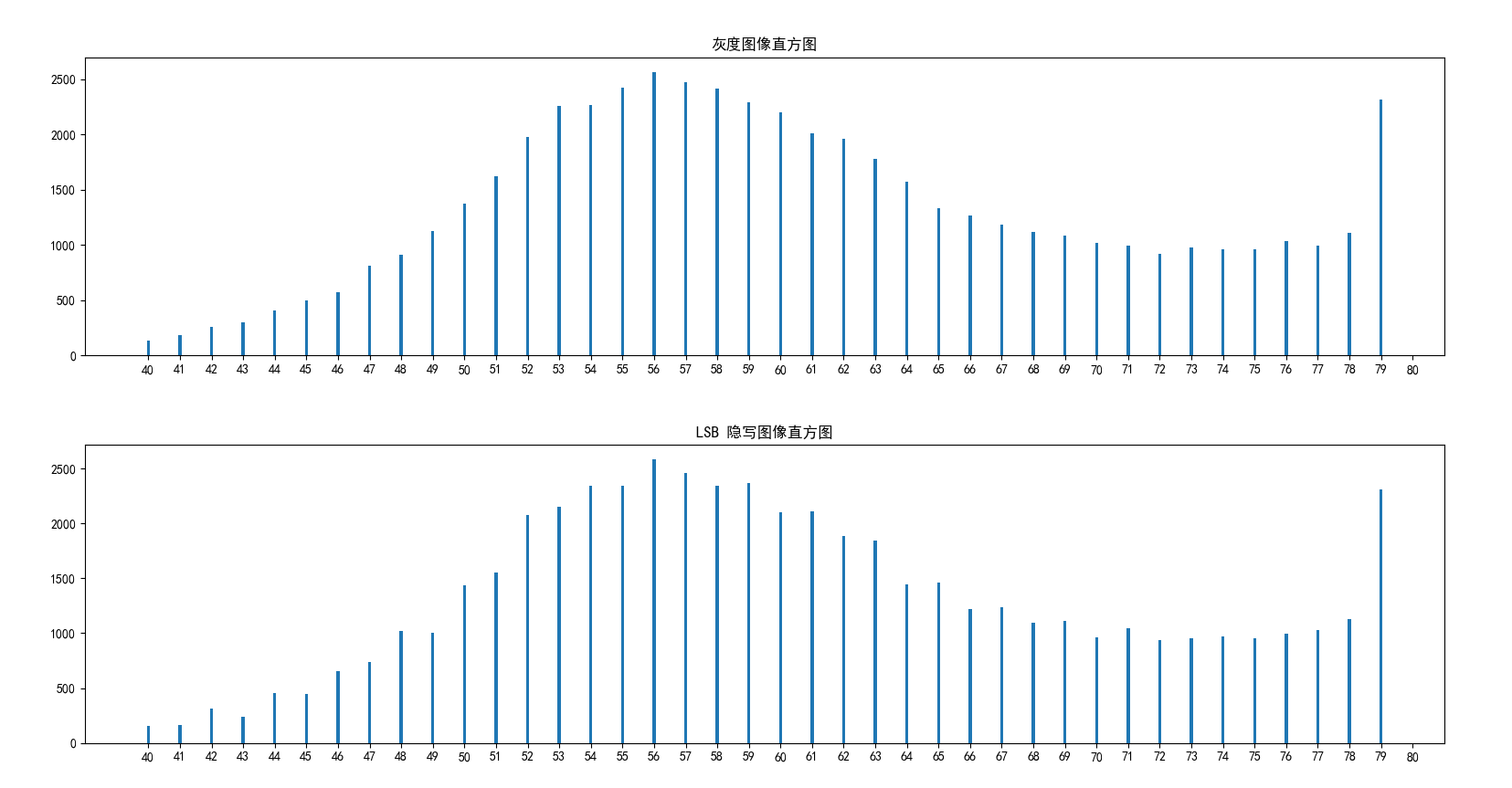


图1 LSB隐写前后图片像素值频率直方图对比

我们假设q是一个像素被选中用于隐藏信息的概率；表示载体图像中，值为j的像素个数，表示隐写图像中，值为j的像素个数。我们可以提出下列假设：对于，有个像素携带秘密信息，其中大约个像素的最低比特与消息相同，不需要修改，剩下的个像素变为2i+1；对于，大约个像素的最低比特变为2i+1。由此我们得到携密图像中的和期望值：

当时有（因为LSB只会导致变为或者变为，以及像素值不改变，因此有）

由此可以将计算所得的期望值与实际2i像素的值进行比较，随着隐写率增加，期望值与实际值趋近相等。由此，统计量s符合自由度为k-1的卡方分布。若s小于等于我们设定的阈值，即判定图像隐写。

## 2. 实现卡方分析算法

卡方的实现主要分为空间域检测和变换域检测，本次实验以空间域检测为主。

### 2.1 空间域卡方分析

在空间域卡方分析中，我们主要关注图像像素值的分布。隐写操作通常会打乱 LSB 的分布，导致统计特性发生变化。卡方检验可以衡量这种变化是否显著。

### 2.2实现步骤

1. **计算图像的像素值分布**。
2. **计算卡方统计量**：
3. **判断统计显著性**：较高的卡方值表明分布差异显著，可能不存在隐写操作，反之可能存在隐写操作。我们使用自由度为(k-1)的卡方分布的概率分布函数，代入统计量，计算概率值。

### 2.3 实现代码

见附件chi\_square.py

### 2.4 主要函数说明

**LSB隐写：**

LSB\_embed\_random函数用于在图像中随机嵌入LSB隐写。随机选取嵌入的索引，并生成随机秘密消息进行嵌入。

LSB\_embed\_continuous函数用于在图像中连续地以行优先嵌入秘密信息，根据选择模式不同可以分为随机选择开始索引以及选择图片第一个像素作为开始索引。

**图像分块：**

image\_partition函数用于将图像按指定块大小分割，大小不足的会进行补全。

**卡方检验：**

chi\_square函数用于计算卡方统计量，以及根据统计量和卡方分布的概率分布函数计算概率。

**位分布直方图绘制：**

plot\_bit\_histogram函数用于绘制原始图像和隐写图像在指定LSB位数下的像素值分布直方图。

**性能评估相关：**

embed\_evalution函数执行以下步骤：

1. **读取图像**：读取并转换为灰度图像。
2. **定义隐写率：**定义从0.0到1.0的隐写率，步长为0.1。
3. **进行多次实验**：对于每个隐写率，进行M=5次实验。每次实验中，嵌入隐写信息，并计算卡方统计量。根据传入的参数，我们会选择连续嵌入或随机嵌入
4. **结果统计与输出**：计算每个隐写率下卡方统计量及预测概率值，并打印结果。
5. **绘图**：绘制隐写率与卡方统计量及p值的关系图。

evalution\_sample\_size函数执行以下步骤：

1.**隐写：**分别以0，10%，30%，50%，70%的隐写率，按连续嵌入的方式对图片进行隐写。

2.**进行实验：**从10%开始，逐步扩大卡方分析的范围，并计算卡方分析的统计值。

3.**绘图：**绘制不同隐写率下，选取图像部分的比例与卡方分析所得的预测值的曲线。

## **3.**实验结果及分析

### 3.1 关键参数

在卡方分析中，以下参数会对卡方分析的检测效果产生影响。

1. **嵌入方式**

对于LSB，存在随机嵌入和连续嵌入两种方式。

随机嵌入指数据随机选择嵌入位置，不按照连续的序列进行嵌入。而连续嵌入数据指从某个位置开始，按照顺序连续地嵌入到载体中。这可能会对载体的统计特性有影响，尤其是在分块的情况下。在随机插入时，隐写嵌入对载体频率分布的改变被分配到各个分块中，这与连续嵌入对载体频率分布的改变可能存在差异。

1. **隐写率**

隐写率指的是嵌入到介质中的数据相对于其容量的比例。它对卡方测试的敏感性和有效性有显著影响。因为隐写率不同，隐写操作对载体统计特性的破坏程度也不同，会影响卡方分析的结果。

1. **分块大小**

分块大小通常也会对卡方分析的结果产生影响。因为不同大小的分块，统计特性可能会有较大差异。

### 3.2 性能指标

我们令隐写图像为P，正常图像为N，T表示预测与实际一致，N表示预测与实际不一致。以下定义准确率、虚警率和漏检率

**准确率（Accuracy）**：

**定义**：分类正确的比例，包括正确识别的正常图像和隐写图像。

**计算公式**：

**虚警率**：

**定义**：将正常图像错误地判定为隐写图像的比例。

**计算公式**：

**漏检率**：

**定义**：将隐写图像错误地判定为正常图像的比例。

**计算公式**：

### 3.3 实验设置

1. **嵌入方式**

为了检测嵌入方式对卡方检测效果的影响，我们定义从0.0到1.0的隐写率，我们生成随机数据，并分别采用随机嵌入和连续嵌入两种方法将数据嵌入图片，并使用卡方检测检测图片存在隐写的概率。图片分块我们设置列数和原始图片相同，行数为原始图片的一半。

1. **隐写率**

与上一部分相同，不再赘述。我们会在检测嵌入方式对卡法检测效果的影响时，同步检测隐写率对卡方分析的影响。

1. **分块大小**

对于lena图，我们分别采用0、10%、30%、50%、70%这五个隐写率，将分块大小设置为列数和原始图片相同，行数与原始图片的比例从1.0逐步增加到1.0。

### 3.4 连续插入

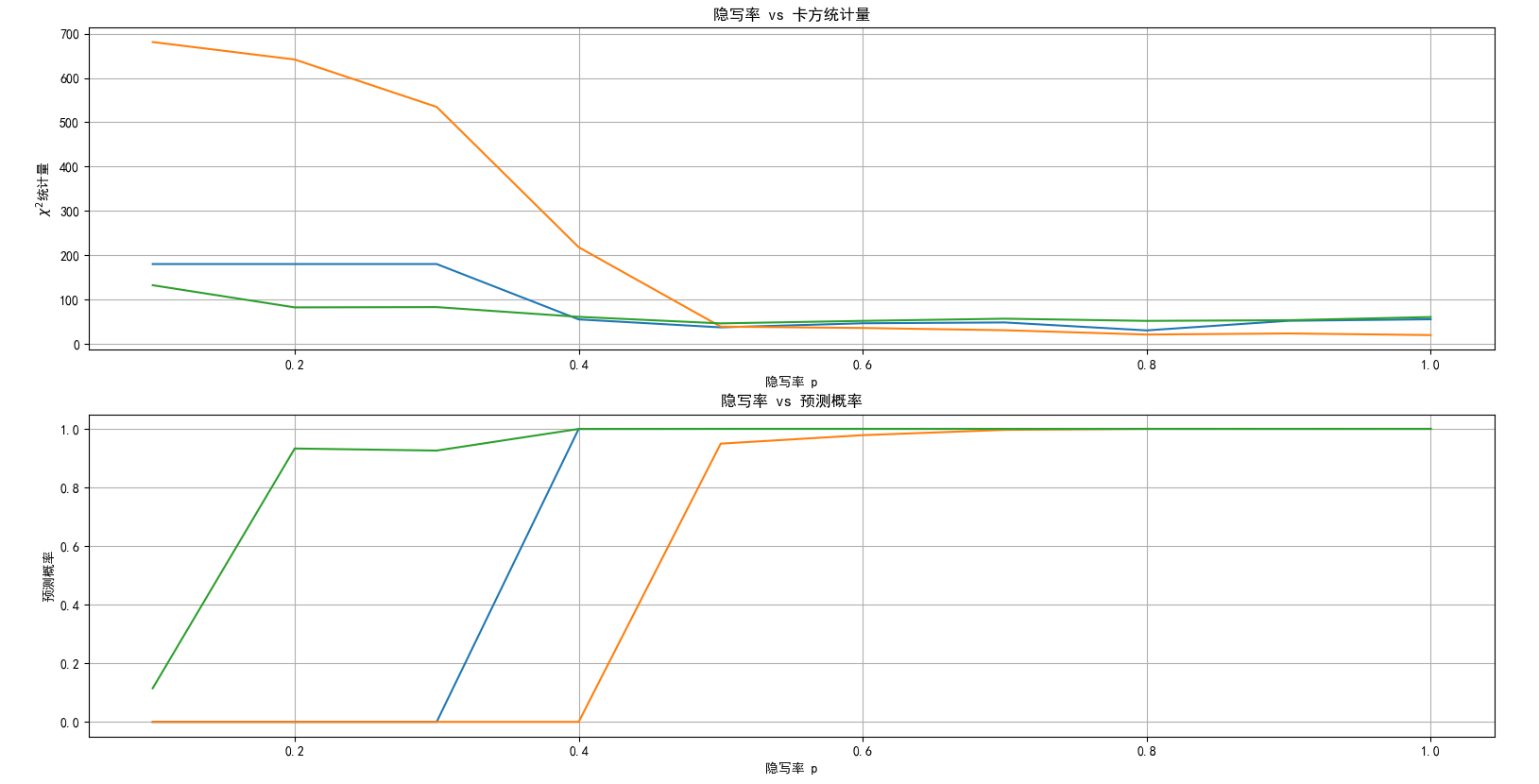


图2 连续嵌入时卡方统计量和预测存在隐写的概率随隐写率的变化情况

上图2为连续进行LSB嵌入时卡方统计量和预测存在隐写的概率随隐写率的变化情况。在连续插入时，随着隐写率增加，卡方分析的卡方统计量下降，预测存在隐写的概率上升，直到概率为1。这是因为随着隐写率增加，图片像素值频率分布被破坏地越严重，携密图像与原始图像的差异越明显。

### 3.5 随机插入

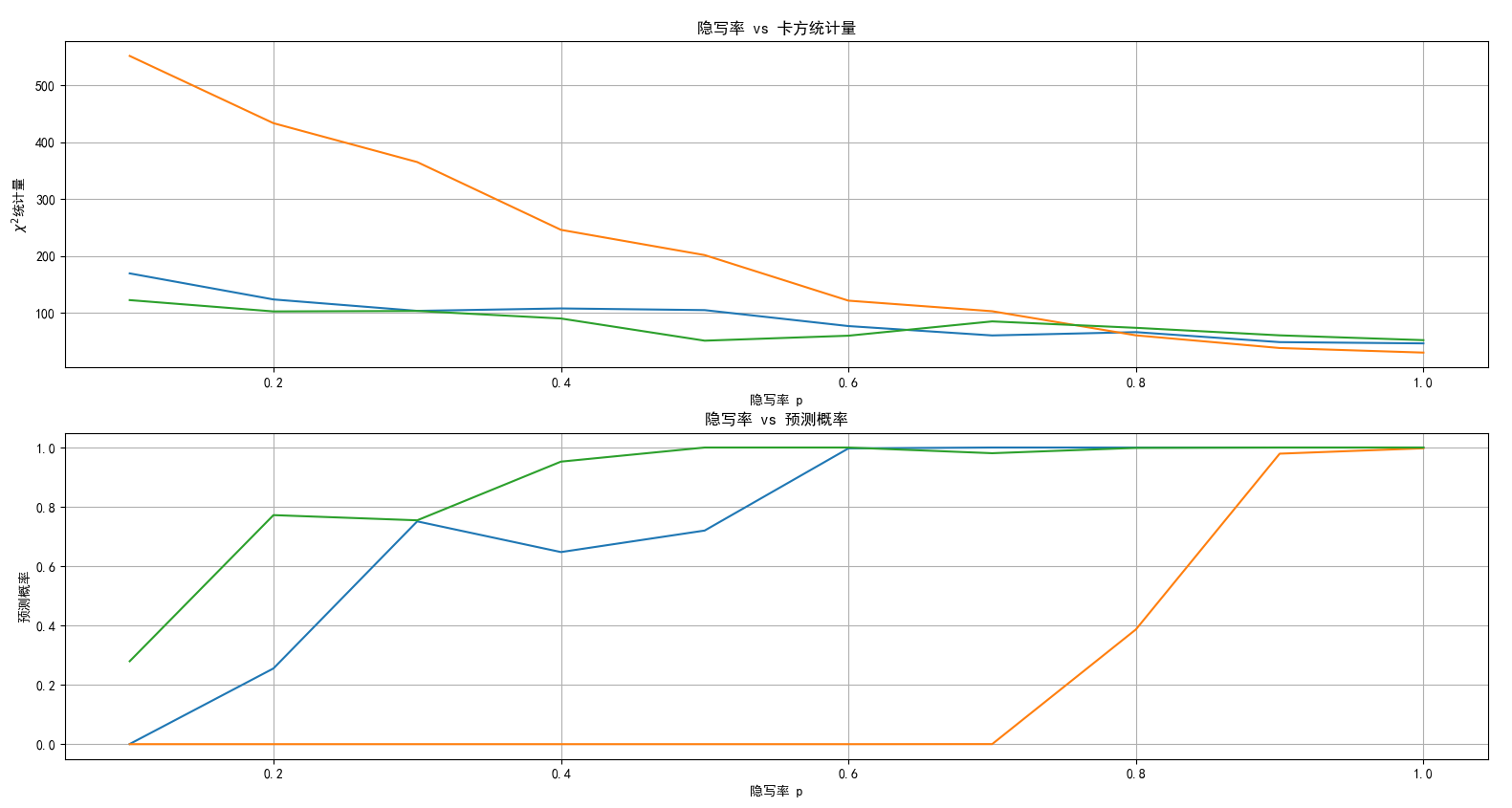


图3 随机嵌入时卡方统计量和预测存在隐写的概率随隐写率的变化情况

上图3为随机进行LSB嵌入时卡方统计量和预测存在隐写的概率随隐写率的变化情况。在随机插入时，随着隐写率增加，卡方分析的卡方统计量下降，预测存在隐写的概率上升，直到概率为1。这是因为随着隐写率增加，图片像素值频率分布被破坏地越严重，携密图像与原始图像的差异越明显。相比较于连续插入，随机插入在低隐写率时预测的概率较低，这是因为在随机插入时，隐写嵌入对图片像素值频率分布的改变被分配到各个图片分块中，导致在单个图片分块中，统计特性的变化不明显。而连续嵌入对某一个图片分块统计特性的影响更加显著。

图4卡方分析连续嵌入和随机嵌入的性能指标随隐写率的变化情况。可以看到，随着隐写率上升，卡方分析的准确率总体呈上升趋势，漏检率总体呈下降趋势，且在连续嵌入的情况下，收敛到稳定值的速度要快于随即嵌入。

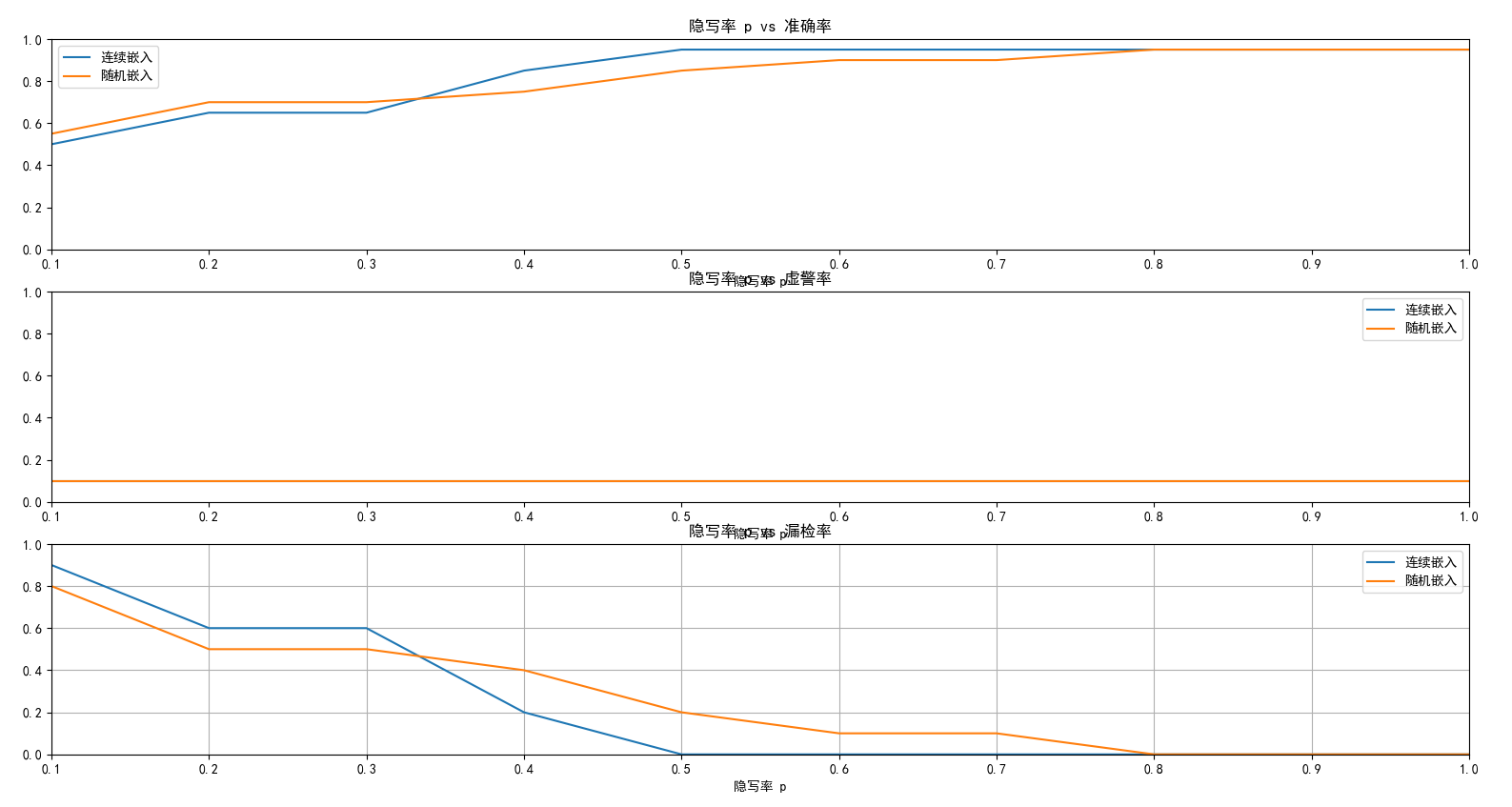


图4 连续嵌入和随机嵌入的性能指标随隐写率的变化情况

### 3.6 修改图片分块大小

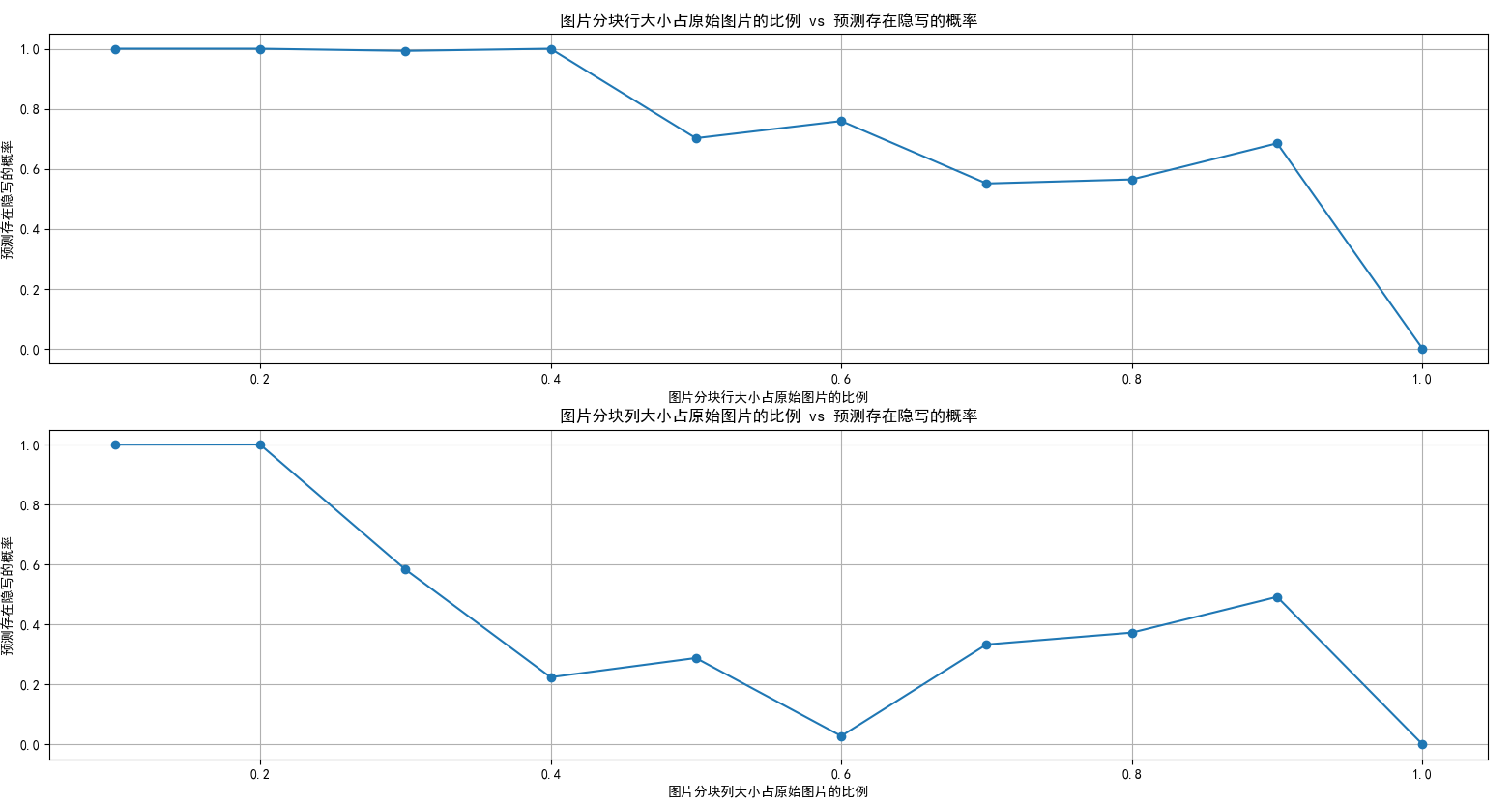


图5 分别修改列大小和行大小的结果

在隐写率为0.5，分别修改列大小和行大小的结果如上图5所示。

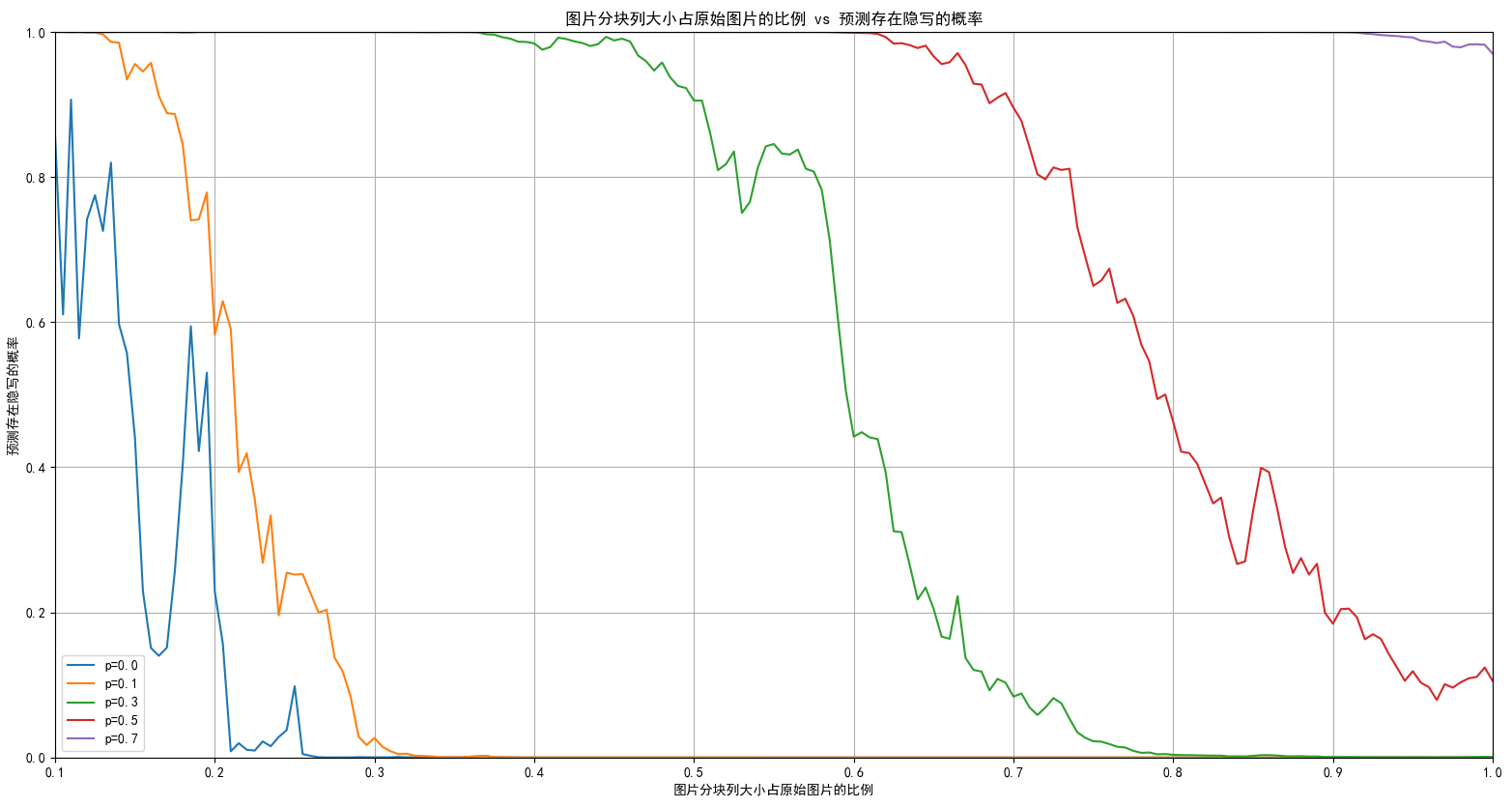


图6 在不同隐写率下，预测概率随图片分块大小变化情况

在不同隐写率下，修改分块大小得到的结果如上图6。可以看到预测存在隐写的概率随分块大小上升，总体呈下降趋势。这是因为随着分块大小的不断增加，隐写造成的统计特性改变更不显著。当隐写率增大时，增大分块大小对预测概率的影响会减小，也是因为隐写造成的统计特性改变范围更大。

在性能指标方面，我们分别考察10%、30%、50%、70%这四个隐写率下的性能指标。结果如下图7。可以看到，随着隐写率上升，卡方分析的准确率总体呈下降趋势，漏检率总体呈上升趋势。说明分块大小选取不当，会导致检测性能降低。

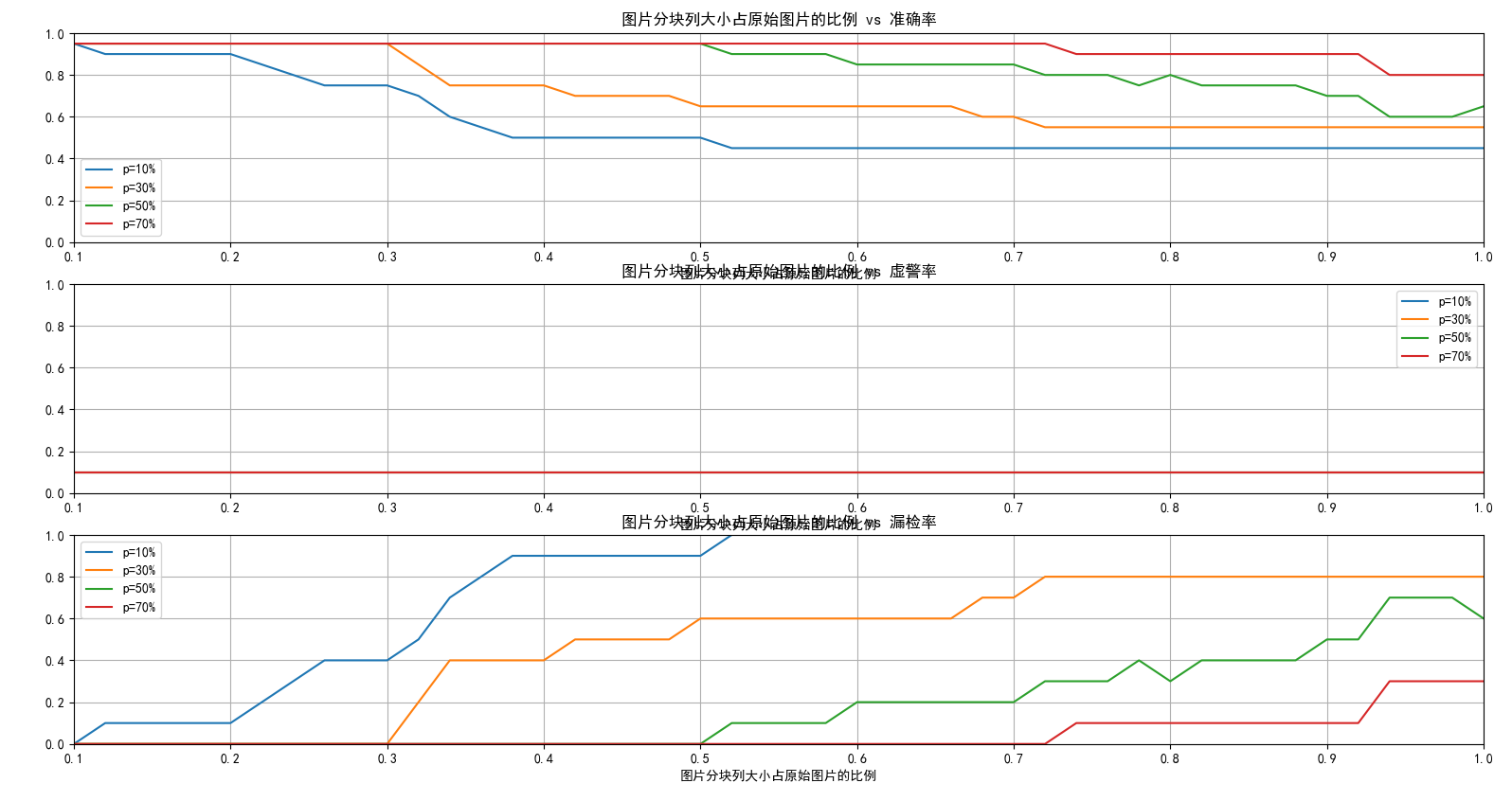


图7 在不同隐写率下，性能指标随图片分块大小变化情况

### 3.7 总结

根据上述结果可以得出以下结论：

1. 卡方分析在低隐写率时效果不佳；
2. 卡方分析在分析随机隐写时效果要弱于连续隐写；
3. 随着分块大小增大，卡方分析预测的概率结果总体呈下降趋势。

因此，随机隐写和降低隐写率，有利于降低隐写信息被卡方分析检测出的概率。

## 卡方分析与其他算法效率对比

### 6.1 检测效率：

卡方分析：卡方分析是一种基于统计的检测方法，它通过比较图像像素值的分布来检测隐写。根据搜索结果，卡方分析可以判定载体图像中是否含有秘密信息，但这要求秘密信息必须嵌满所有像素。实验结果表明，卡方分析在针对原始无损存储图像时可以获得优于RS隐写分析方法的性能，且计算速度显著高于RS隐写分析方法，有利于实现实时检测。

RS隐写检测：RS隐写检测算法是一种基于图像空域相关性的检测法。该方法将图像分为多个块，对每个块进行定制的翻转操作，根据构造的差别函数将图像中的组分为常规类、奇异类、不可用类。当信息嵌入后，会引起三个类别中元素个数的改变。RS检测算法具有较高检测效率，并且通过实验比较，指明5个对检测效率产生影响的关键因素，并且分析了产生影响的根本原因。

### 6.2准确性：

卡方分析：卡方分析的准确性取决于秘密信息是否嵌满所有像素。如果秘密信息没有嵌满所有像素，卡方分析的准确性可能会受到影响。

RS隐写检测：RS隐写检测算法不仅能检测图像是否含有隐藏信息，还能对嵌入的隐藏信息百分比有更加准确的估计，从而在准确性方面具有一定的优势。

### 6.3 适用性：

卡方分析：卡方分析适用于满嵌图像的检测，即所有像素都参与隐写的情况。

RS隐写检测：RS隐写检测算法适用于更广泛的隐写情况，不仅限于满嵌图像，因此具有更广的适用性。

### 6.4 计算复杂度：

卡方分析：卡方分析的计算复杂度相对较低，因为它主要涉及直方图的计算和比较。

RS隐写检测：RS隐写检测算法的计算复杂度可能更高，因为它涉及到图像块的划分和翻转操作，以及差别函数的计算。

综上所述，卡方分析在计算速度上可能优于RS隐写检测，特别是在针对原始无损存储图像时，而RS隐写检测在准确性和适用性方面可能更具优势。选择哪种方法取决于具体的应用场景和需求。

## 成员及分工

2024111059 邢耀正：报告撰写，代码实现，实验； 工作占比：50%

2024110940 林羽喆：报告撰写，代码实现，实验； 工作占比：50%

## 3.多比特位分析实现

多比特位分析（Multi-bit Level Analysis）在隐写分析中是指对图像中最低的多个有效位的分布进行统计分析，以检测潜在的隐写操作。相比于单比特位分析，多比特位分析能够捕捉更多的隐写信息，从而提高隐写检测的敏感性和准确性。

在代码中，多比特位分析主要通过chi\_square\_multi\_bits函数实现。以下是对该函数及其相关部分的详细解释，以及多比特位分析的实现原理。

### 3.1 多比特位分析的实现原理

在LSB隐写技术中，隐写信息通常嵌入到图像像素的最低有效位中。例如，1比特LSB隐写将信息嵌入到每个像素的最低位，而2比特LSB隐写则将信息嵌入到最低两位。

多比特位分析旨在检测图像中多个LSB位的统计分布是否与原始图像一致。通过计算图像的各个像素值上的频数分布，可以判断是否存在隐写操作。

与单比特位卡方分析类似，多比特位卡方分析原理是：图像中像素灰度值为i，其中0≤ i ≤255。如果载体图像未隐写，像素值为的数目和像素值为的数目会相差得很远，其中n是每一像素值嵌入的比特数，。秘密信息在嵌入之前往往经过加密，可以看作是0、1 随机分布的比特流，而且值为0与1 的可能性大致都是1/2。如果秘密信息完全替代载体图像的低位，那么这个值会比较接近，可以根据这个性质判断图像是否经过隐写。

我们假设q是一个像素被选中用于隐藏信息的概率；表示载体图像中，值为j的像素个数，表示隐写图像中，值为j的像素个数。我们可以提出下列假设：对于，有个像素携带秘密信息，其中大约个像素的最低比特与消息相同，不需要修改，剩下的个像素变为其他的像素；由此我们得到携密图像中的期望值为

当q=1时，有

在多比特位的卡方分析中，主要的操作与单比特位的卡方法分析一致：

**计算频数分布：**统计每种可能的比特组合出现的次数。

**计算卡方统计量：**衡量观察频数与期望频数之间的差异。

**计算p值：**判断卡方统计量的显著性，统计量越小，差异越显著，表明可能存在隐写操作。

### 3.2 代码实现

def **chi\_square\_test**(original\_fig, stego\_fig, bit\_level=2):

    """

    对比原始图像和隐写图像的像素值分布，计算卡方统计量

    :param original\_fig: 原始图像（二维numpy数组）

    :param stego\_fig: 隐写图像（二维numpy数组）

    :param bit\_level: LSB位数（默认为2）

    :return: 卡方统计量

    """

*# 提取指定位的LSB*

    mask = (1 << bit\_level) - 1  *# 创建掩码，如bit\_level=2时mask=0b11*

    original\_bits = original\_fig & mask  *# 提取原始图像的LSB位*

    stego\_bits = stego\_fig & mask      *# 提取隐写图像的LSB位*

*# 计算频数*

    original\_counts = np.bincount(original\_bits.flatten(), minlength=2 \*\* bit\_level)

    stego\_counts = np.bincount(stego\_bits.flatten(), minlength=2 \*\* bit\_level)

*# 期望频数（假设无隐写，即 stego\_counts = original\_counts）*

    expected = original\_counts

    observed = stego\_counts

*# 避免期望频数为0*

    nonzero = expected != 0

    chi2 = np.sum(((observed[nonzero] - expected[nonzero]) \*\* 2) / expected[nonzero])

    return chi2

提取多比特位：使用位掩码mask = (1 << bit\_level) - 1生成一个掩码。例如，bit\_level=2时，mask=0b11。使用按位与操作original\_fig & mask和stego\_fig & mask提取原始图像和隐写图像的指定LSB位。

计算频数分布：np.bincount函数用于统计每种可能的比特组合出现的次数。minlength=2 \*\* bit\_level确保所有可能的比特组合都有对应的计数。

例如，bit\_level=2时，有4种可能的比特组合00, 01, 10, 11，对应的计数分别存储在original\_counts和stego\_counts中。

计算卡方统计量：

期望频数expected假设隐写操作未进行，即隐写图像的频数应与原始图像一致。卡方统计量chi2通过公式计算。为避免除以零错误，首先筛选出期望频数不为零的类别。

def **chi\_square\_pvalue**(chi2, bit\_level=2):

    """

    计算卡方统计量对应的p值

    :param chi2: 卡方统计量

    :param bit\_level: LSB位数

    :return: p值

    """

    df = (2 \*\* bit\_level) - 1  *# 自由度*

    p\_value = 1 - chi2\_dist.cdf(chi2, df)

    return p\_value

计算p值：使用scipy.stats库中的卡方分布累积分布函数（CDF）计算p值。p值表示观察到的卡方统计量在卡方分布中的位置，p值越小，说明差异越显著，可能存在隐写操作。

自由度计算：自由度df为(2 \*\* bit\_level) - 1。例如，bit\_level=2时，df=3。

def **plot\_bit\_histogram**(original\_fig, stego\_fig, bit\_level=2):

    """

    绘制原始图像和隐写图像的LSB位分布直方图

    :param original\_fig: 原始图像

    :param stego\_fig: 隐写图像

    :param bit\_level: LSB位数

    """

    mask = (1 << bit\_level) - 1

    original\_bits = original\_fig & mask

    stego\_bits = stego\_fig & mask

    original\_counts = np.bincount(original\_bits.flatten(), minlength=2 \*\* bit\_level)

    stego\_counts = np.bincount(stego\_bits.flatten(), minlength=2 \*\* bit\_level)

    labels = [f"{i:0{bit\_level}b}" for i in **range**(2 \*\* bit\_level)]

    x = np.arange(**len**(labels))  *# 标签位置*

    width = 0.35  *# 柱状图宽度*

    fig, ax = plt.subplots(figsize=(12, 6))

    rects1 = ax.bar(x - width/2, original\_counts, width, label='原始图像')

    rects2 = ax.bar(x + width/2, stego\_counts, width, label='隐写图像')

    ax.set\_xlabel('LSB位值')

    ax.set\_ylabel('频数')

    ax.set\_title(f'LSB{bit\_level}位分布比较')

    ax.set\_xticks(x)

    ax.set\_xticklabels(labels)

    ax.legend()

    plt.show()

**绘制直方图**：对比原始图像和隐写图像在指定LSB位数下的频数分布。

**标签生成**：根据bit\_level生成相应的二进制标签。例如，bit\_level=2时，标签为'00', '01', '10', '11'。

**直方图展示**：使用双柱状图对比两者的频数分布，便于直观观察差异