

图像复原

实验目的

1. 使用霍夫曼编码对图像进行压缩。
2. 使用离散余弦变换（DCT）对图像进行压缩。

实验环境

- Matplotlib
- OpenCV
- NumPy

实验原理

霍夫曼编码原理

霍夫曼编码是一种常见的无损压缩算法。它基于字符（或像素值）出现的频率，将频率较高的字符分配短编码，频率较低的字符分配较长编码，从而降低整体的编码长度。具体步骤如下：

- 首先统计图像中每个像素值的出现频率，形成频率表；
- 根据频率表构建霍夫曼树。霍夫曼树是一棵二叉树，每个叶子节点表示一个像素值，路径长短与频率成反比；
- 最后，沿霍夫曼树生成每个像素值的霍夫曼编码。

我们构建霍夫曼树的方式是将每个像素频率视作权重，每次从节点集中选择两个权重最小的节点合并，直至只剩下一个根节点。

```
In [3]: import cv2
import numpy as np
import heapq
from collections import Counter, defaultdict

# 定义霍夫曼树节点类
class HuffmanNode:
    def __init__(self, symbol, freq):
        self.symbol = symbol
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq

# 霍夫曼编码类
class HuffmanCoding:
    def __init__(self):
```

```

self.codes = {}
self.reverse_codes = {}

# 构建霍夫曼树
def build_huffman_tree(self, frequency):
    heap = [HuffmanNode(symbol, freq) for symbol, freq in frequency.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        node1 = heapq.heappop(heap)
        node2 = heapq.heappop(heap)
        merged = HuffmanNode(None, node1.freq + node2.freq)
        merged.left = node1
        merged.right = node2
        heapq.heappush(heap, merged)

    return heap[0] # 返回霍夫曼树的根节点

# 生成编码
def generate_codes(self, node, current_code=""):
    if node is None:
        return

    if node.symbol is not None:
        self.codes[node.symbol] = current_code
        self.reverse_codes[current_code] = node.symbol
        return

    self.generate_codes(node.left, current_code + "0")
    self.generate_codes(node.right, current_code + "1")

# 压缩函数
def compress(self, image):
    # 转换图像为一维数组
    data = image.flatten()

    # 统计频率
    frequency = Counter(data)

    # 构建霍夫曼树
    root = self.build_huffman_tree(frequency)

    # 生成编码
    self.generate_codes(root)

    # 压缩数据
    compressed_data = "".join([self.codes[symbol] for symbol in data])
    return compressed_data

# 解压缩函数
def decompress(self, compressed_data, image_shape):
    current_code = ""
    decoded_data = []

    for bit in compressed_data:
        current_code += bit
        if current_code in self.reverse_codes:
            symbol = self.reverse_codes[current_code]
            decoded_data.append(symbol)
            current_code = ""

```

```
# 将解码数据转回原图形状
return np.array(decoded_data).reshape(image_shape)
```

假设像素集为 $\{x_1, x_2, \dots, x_n\}$, 其对应的概率为 $\{p(x_1), p(x_2), \dots, p(x_n)\}$, 则构建霍夫曼编码树的过程可以保证最优的平均编码长度为:

$$L = \sum_{i=1}^n p(x_i) \cdot l(x_i)$$

其中 $l(x_i)$ 是像素 x_i 的编码长度。

离散余弦变换 (DCT)

离散余弦变换 (DCT) 是一种用于将图像从空间域转换到频域的技术。它能够将图像大部分的能量集中在较少的低频分量上, 从而在压缩时能够去除不重要的高频分量。

二维的DCT变换定义为:

$$F(u, v) = \frac{1}{4} \alpha(u) \alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x, y) \cos \left[\frac{(2x+1)u\pi}{2N} \right] \cos \left[\frac{(2y+1)v\pi}{2M} \right]$$

其中:

- $f(x, y)$ 是空间域中的图像像素值;
- $F(u, v)$ 是变换后的频域系数;
- $\alpha(u) = \begin{cases} \frac{1}{\sqrt{N}}, & u = 0 \\ \sqrt{\frac{2}{N}}, & u > 0 \end{cases}$ 。

逆DCT变换将频域信号恢复为空间域信号, 公式为:

$$f(x, y) = \frac{1}{4} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} \alpha(u) \alpha(v) F(u, v) \cos \left[\frac{(2x+1)u\pi}{2N} \right] \cos \left[\frac{(2y+1)v\pi}{2M} \right]$$

通过DCT变换后的系数中, 低频成分 (靠近左上角的值) 包含了图像的大部分信息, 而高频成分则可以通过量化降低或置零, 来达到压缩的目的。

实验过程

哈夫曼编码

利用哈夫曼树进行哈夫曼编码, 并解码, 对比前后图像。

```
In [49]: import cv2
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
# 读取图像
image = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
```

```

ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('off')

# 实例化霍夫曼编码类
huffman = HuffmanCoding()
# 压缩图像
compressed_data = huffman.compress(image)
print("Compressed data length:", len(compressed_data))
with open('huffman.txt', 'w') as f:
    f.write(compressed_data)
# 解压缩图像
decompressed_data = huffman.decompress(compressed_data, image.shape)
ax[1].imshow(decompressed_data, cmap='gray')
ax[1].set_title('Decompressed Image')
ax[1].axis('off')

plt.show()

```

Compressed data length: 1941978

Original Image



Decompressed Image



编解码后图像和原图像完全一致，哈夫曼编码是无损压缩方式。

DCT压缩

```

In [51]: import cv2
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
# 读取图像
image = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE).astype('float')
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('off')

img_dct=cv2.dct(image)
img_dct_log=np.log(abs(img_dct))
img_recor=cv2.idct(img_dct)
recor_tmp=img_dct[0:50,0:50]
recor_tmp2=np.zeros(image.shape)
recor_tmp2[0:50,0:50]=recor_tmp

```

```
img_recor1=cv2.idct(recor_tmp2)
ax[1].imshow(img_recor1, cmap='gray')
ax[1].set_title('Decompressed Image')
ax[1].axis('off')

plt.show()
```

Original Image



Decompressed Image



可以看到DCT压缩为有损压缩。