《生物医学图像处理实验》实验报告

（2021-2022 学年第 2 学期）

实验五：图像编码

课程号：310025010　　**课序号：**01　　**任课教师：**林江莉　　**成绩：**

组长：徐广玄　　**小组成员：徐广玄**

实验报告日期：　　2023 年 5 月 26 日

一、实验内容：

1. 任选以下编码方法中的两种或两种以上实现，并测试所给 4 幅图像的压缩率。

huffman 编码、仙农－费诺码、算术编码、Golomb 编码、行程编码、LZW 编码。

2. 综合多种编码方法，在视觉不失真的情况下尽量获得较大的压缩率，并测试所给 4 幅图像的压缩率。（综合实验）

提示：1.在编码过程中可以应用分块编码、基于比特平面的编码、对预测误差图像编码，RGB 转换为 YUV 空间、采用量化器等思想。2. 保持视觉不失真，可以是有损压缩。

二、编程软件：

Pycharm2021

三、实验步骤及关键代码：

1. 实现 huffman 编码和 LZW 编码，并计算压缩率

```
# -*- coding: utf-8 -*-
import heapq
from collections import defaultdict
from PIL import Image
```

图 1 导入的库函数

```python
# 定义Huffman编码树节点类
class HuffmanNode:
    def __init__(self, pixel=None, freq=0):
        self.pixel = pixel  # 图像像素值
        self.freq = freq  # 频率
        self.left = None
        self.right = None

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

图 2 定义节点

```python
# 生成Huffman编码树
def build_huffman_tree(image):
    pixel_freq = defaultdict(int)
    width, height = image.size

    for y in range(height):
        for x in range(width):
            pixel = image.getpixel((x, y))
            pixel_freq[pixel] += 1

    nodes = [HuffmanNode(pixel, freq) for pixel, freq in pixel_freq.items()]
    heapq.heapify(nodes)

    while len(nodes) > 1:
        left_node = heapq.heappop(nodes)
        right_node = heapq.heappop(nodes)
        parent_node = HuffmanNode(freq=left_node.freq + right_node.freq)
        parent_node.left = left_node
        parent_node.right = right_node
        heapq.heappush(nodes, parent_node)

    return nodes[0]
```

图 3 生成编码树

```python
# 将图像转换为Huffman编码
def compress_image(image, huffman_table):
    width, height = image.size
    compressed_bits = ""

    for y in range(height):
        for x in range(width):
            pixel = image.getpixel((x, y))
            compressed_bits += huffman_table[pixel]

    return compressed_bits
```

图 4 图像转为编码

```python
# 计算压缩率
def calculate_compression_ratio(image, compressed_bits):
    original_size = image.width * image.height * 8
    compressed_size = len(compressed_bits)
    compression_ratio = original_size / compressed_size
    return compression_ratio
```

图 5 计算压缩率

```python
def LZW(image_path):
    # 加载BMP图像
    image_path = image_path
    image = Image.open(image_path)

    # 将图像数据转换为八比特序列
    data = convert_image_to_data(image)
    data = "".join(str(num) for num in data)

    # 初始化字典
    dict_size = 128
    dictionary = {chr(i): i for i in range(dict_size)}

    w = ""
    result = []
    for c in data:
        wc = w + c
        if wc in dictionary:
            w = wc
        else:
            result.append(dictionary[w])
            # Add wc to the dictionary.
            dictionary[wc] = dict_size
            dict_size += 1
            w = c

    # Output the code for w.
    if w:
        result.append(dictionary[w])

    compression_ratio = calculate_compression_ratio_2(image, result)
```

图 6　LZW 主函数

```python
# 将图像数据转换为八比特序列
def convert_image_to_data(image):
    image = image.convert("P")
    width, height = image.size
    data = []

    for y in range(height):
        for x in range(width):
            pixel = image.getpixel((x, y))
            data.append(pixel)

    return data
```

图 7　LZW 转换编码序列

2. 使用 LZW 和比特平面的符合法进行图像压缩，i 代表取多少位比特平面

```python
def LZW_8bits(image_path, i):
    # 加载BMP图像
    image_path = image_path
    image = Image.open(image_path)

    # 将图像数据转换为八比特序列
    data = convert_image_to_data_2(image, i)
    data = "".join(str(num) for num in data)

    # 初始化字典
    dict_size = 128
    dictionary = {chr(i): i for i in range(dict_size)}

    w = ""
    result = []
    for c in data:
        wc = w + c
        if wc in dictionary:
            w = wc
        else:
            result.append(dictionary[w])
            # Add wc to the dictionary.
            dictionary[wc] = dict_size
            dict_size += 1
            w = c

    # Output the code for w.
    if w:
        result.append(dictionary[w])

    compression_ratio = calculate_compression_ratio_2(image, result)
```

图 8 LZW-8bits 算法主体

```python
# 将图像数据转换为八比特序列
def convert_image_to_data_2(image, i):
    image = image.convert("P")
    width, height = image.size
    data = []

    for y in range(height):
        for x in range(width):
            pixel = image.getpixel((x, y))
            pixel = (pixel >> i) << i
            data.append(pixel)
    return data
```

图 9 LZW-8bits 数据转换

四、实验结果：

```
if __name__ == '__main__':
    huffman("CT-1.bmp")
    huffman("lenna_8.bmp")
    huffman("MRI.bmp")
    huffman("US.bmp")
    LZW("CT-1.bmp")
    LZW("lenna_8.bmp")
    LZW("MRI.bmp")
    LZW("US.bmp")
    LZW_8bits("CT-1.bmp", 5)
    LZW_8bits("lenna_8.bmp", 5)
    LZW_8bits("MRI.bmp", 5)
    LZW_8bits("US.bmp", 5)
```

图10 主函数

```
CT-1.bmp Huffman Compression Ratio: 1.60
lenna_8.bmp Huffman Compression Ratio: 1.10
MRI.bmp Huffman Compression Ratio: 1.30
US.bmp Huffman Compression Ratio: 1.03
CT-1.bmp LZW Compression Ratio: 8.42
lenna_8.bmp LZW Compression Ratio: 1.39
MRI.bmp LZW Compression Ratio: 6.28
US.bmp LZW Compression Ratio: 3.51
CT-1.bmp LZW Compression Ratio: 8.42
lenna_8.bmp LZW Compression Ratio: 6.04
MRI.bmp LZW Compression Ratio: 6.86
US.bmp LZW Compression Ratio: 6.17
```

图 11 打印结果

由此可见当取不同的 i（比特平面个数），我们可以获得不同的压缩率，i 越大，压缩率越大。

五、结果分析及实验小结：

编程中问题及其解决：问题：图像变为无间隔字符串出现报错；解决方案：使用 join 函数解决。

编程中的注意事项：主要到 i 为不取的比特平面个数。

六、小组成员分工

由徐广玄独自完成