

به نام خدا
دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

سیستم‌های چندرسانه‌ای

گزارش کار تمرین ۴
پروژه فشرده‌سازی JPEG

استاد درس: دکتر خرسندی

محمد رضا شهرستانی

۹۷۲۸۰۵۴

نیم‌سال دوم ۱۴۰۰-۱۴۰۱

توضیح مختصر بخش‌های پروژه

این پروژه، پیاده‌سازی مراحل فشرده‌سازی تصویر JPEG است.

- در مرحله‌ی اول با کتابخانه‌ی Pillow اقدام به خواندن فایل تصویر کرده و سپس RGB را به YCbCr تبدیل می‌کنیم. سپس با استفاده از 4:2:0 chroma subsampling عمل نمونه‌برداری را انجام می‌دهیم. (شکل ۱)

```
# part 1
img = Image.open('photo1.png')
ycbcr = img.convert('YCbCr')
chroma_subsampling = sampling(np.array(ycbcr, dtype=np.uint8))
```

شکل ۱

- در مرحله‌ی دوم باید تبدیل کسینوس گسسته را پیاده‌سازی کرد. ابتدا داده‌ها را به بلوک‌های 8×8 تبدیل کرده و سپس برای هر بلوک تبدیل دو بعدی کسینوس گسسته را اعمال کرده و ضرایب AC و DC را خروجی می‌گیریم. (شکل ۲)

```
# part 2 & 3
dc, ac = convert_dc_ac(chroma_subsampling)
```

شکل ۲

- در مرحله‌ی سوم با استفاده از ضرایب مرحله قبل و جداول کوانتیزاسیون داده شده، ماتریس‌های کوانتیزاسیون را ساخته و داده‌ها را کوانتیزه می‌کنیم. (شکل ۳)

```

luminance_table = np.array([[16, 11, 10, 16, 24, 40, 51, 61],
                             [12, 12, 14, 19, 26, 58, 60, 55],
                             [14, 13, 16, 24, 40, 57, 69, 56],
                             [14, 17, 22, 29, 51, 87, 80, 62],
                             [18, 22, 37, 56, 68, 109, 103, 77],
                             [24, 35, 55, 64, 81, 104, 113, 92],
                             [49, 64, 78, 87, 103, 121, 120, 101],
                             [72, 92, 95, 98, 112, 100, 103, 99]])

chrominance_table = np.array([[17, 18, 24, 47, 99, 99, 99, 99],
                              [18, 21, 26, 66, 99, 99, 99, 99],
                              [24, 26, 56, 99, 99, 99, 99, 99],
                              [47, 66, 99, 99, 99, 99, 99, 99],
                              [99, 99, 99, 99, 99, 99, 99, 99],
                              [99, 99, 99, 99, 99, 99, 99, 99],
                              [99, 99, 99, 99, 99, 99, 99, 99],
                              [99, 99, 99, 99, 99, 99, 99, 99]])

```

شکل ۳

- در مرحله‌ی چهارم در فایل `Huffman.py` الگوریتم کدگذاری هافمن را پیاده‌سازی می‌کنیم. در این فایل درخت هافمن و نودهای آن شبیه‌سازی شده است. (شکل ۴)

```

class HuffmanTree:
    def __init__(self, arr):
        self.value_to_bitstring_table = dict()
        q = PriorityQueue()

        for val, freq in cal_freq(arr).items():
            node = _Node().init_leaf(value=val, freq=freq)
            q.put(node)

        while q.qsize() >= 2:
            left = q.get()
            right = q.get()
            node = _Node().init_node(left_child=left, right_child=right)
            q.put(node)

        self.root = q.get()
        self.create_huffman_table()

```

شکل ۴

- در مرحله‌ی پنجم پیمایش زیگزاگ را اضافه کرده و تابع RLC برای ضرایب AC و تابع DPCM برای ضرایب DC را در فایل Hoffman.py پیاده‌سازی می‌کنیم. (شکل ۵)

```
def DPCM(list):
    list[1:] = list[1:] - list[0]
    return list

def RLC(list):
    last_nonzero = -1
    for i, s in enumerate(list):
        if s != 0:
            last_nonzero = i
    symbols = []
    run_length = 0
    for i, s in enumerate(list):
        if i > last_nonzero:
            symbols.append((0, 0))
            break
        elif s == 0 and run_length < 15:
            run_length += 1
        else:
            symbols.append((run_length, s))
            run_length = 0
    return symbols
```

شکل ۵

- در مرحله‌ی آخر کدگذاری مرحله‌ی ۴ را بر روی ضرایب مرحله‌ی ۳ اعمال می‌کنیم. (شکل ۶)

```
# part 6
dc_y, dc_c, ac_y, ac_c = huffman.huffman_coding(dc, ac)
```

شکل ۶

پاسخ پرسش‌ها

۱- اشکال استفاده از رنگ مشکی آن است که اطلاعاتی از نحوه‌ی پخش پیکسل‌ها در حواشی تصویر به ما داده نمی‌شود.

روش پیشنهادی دیگر آن است که به جای گذاشتن رنگ مشکی در پر کردن بلاک‌های ناقص، رنگ پیکسل‌های گوشه‌ی تصاویر را بگذاریم و آن بلاک‌ها را پر کنیم.

۲- اگر اندازه‌ی بلاک‌ها بیش از 8×8 باشد محاسبات ریاضی پیچیده می‌شود و و زمان لازم برای فشرده‌سازی نیز بالا می‌رود و اگر از آن مقدار کم‌تر باشد، میزان داده برای فشرده کردن یک بلاک کم‌تر و در نتیجه کیفیت فشرده‌سازی پایین می‌آید.^۱

۳- خروجی الگوریتم و نرخ فشرده‌سازی حاصل از تقسیم تعداد بایت خروجی بر تعداد اعداد با فرض یک بایت بودن هر عدد، در شکل ۷ قابل مشاهده است.



```
output is b'00000110001101101110011010111100110111'
rate of compression is 3.1578947368421053
```

شکل ۷

۴- حجم فایل قبل و بعد از فشرده‌سازی در شکل ۸ و ۹ مشاهده می‌شود. طبق گفته‌ی تدریس‌یار به علت فشرده بودن فایل png حجم فایل فشرده بالاتر رفته است ولی از فایل اصلی jpeg که ۷ مگابایت حجم داشته کم حجم‌تر است.

```
Size of photo1.png is 2069007 bytes
Size of compressed_jpeg.bin is 2580835 bytes
```

شکل ۸

| Name | Date modified | Type | Size |
|---------------------------------------------------------------------------------------------------------|----------------------|----------|----------|
|  compressed_jpeg.bin | ۱۴۰۱/۰۴/۰۸ ۰۴:۱۵ ب.ظ | BIN File | 2,521 KB |
|  photo1.png | ۱۴۰۱/۰۳/۱۶ ۰۱:۲۸ ق.ظ | PNG File | 2,021 KB |

شکل ۹

¹ <https://stackoverflow.com/questions/10780425/why-jpeg-compression-processes-image-by-8x8-blocks>