课堂实验四：彩色图像处理

一、实验目的

使用python 软件对图像进行彩色处理。使学生通过实验熟悉使用 python软件进行图像彩色处理的有关方法，并体会到图像彩色处理技术以及 对图像处理的效果。

二、实验内容及步骤

(1) 彩色图像的分析
调入并显示彩色图像flower1.tif；
拆分这幅图像，并分别显示其R，G，B分量；
根据各个分量图像的情况讨论该彩色图像的亮度、色调等性质。
(2) 彩色图像的直方图均衡
接内容(1)；
显示这幅图像的R，G，B分量的直方图，分别进行直方图均衡处理，并显示均衡后的直方图和直方图均衡处理后的各分量；
将处理完毕的各个分量合成彩色图像并显示其结果；
观察处理前后图像的彩色、亮度、色调等性质的变化。
(3) 假彩色处理
调入并显示红色可见光的灰度图像vl_red.jpg、绿色可见光的灰度图像vl_green.jpg和蓝色可见光的灰度图像vl_blue.jpg；以及近红外灰度图像infer_near.jpg和中红外灰度图像infer_mid.jpg；
以图像vl_red.jpg为R；图像vl_green.jpg为G；图像vl_blue.jpg为B，
将这三幅图像组合成可见光RGB彩色图像；
分别以近红外图像infer_near.jpg和中红外图像infer_mid替换R分量，
形成假彩色图像；
观察处理的结果，注意不同波长红外线图像组成图像的不同结果

(4) 伪彩色处理1：灰度切片处理
调入并显示灰度图像head.jpg；
利用python提供的函数对图像在8~256级的范围内进行切片处理，并使用hot模式和cool模式进行彩色化；
观察处理的结果。
(5) 彩色变换(选做)
调入并显示灰度图像Lenna.jpg；
使用不同相位的正弦函数作为变换函数，将灰度图像变换为RGB图像。
其中红色分量R的变换函数为-sin( )，绿色分量G的变换函数为-cos( )；
蓝色分量B的变换函数为sin( )；
显示变换曲线及变换合成的彩色图像并观察彩色变换图像的色调与原始图像灰度之间的关系；
将RGB的变换公式至少互换一次(例如R与G互换)，显示变换曲线、变换结果并观察处理的结果。
(6)打印全部结果并进行讨论。
利用python软件实现彩色图像处理的程序：

三、实验代码及结果

```
import cv2
import numpy as np
```

```python
import matplotlib.pyplot as plt


# ========================================================================
# 实验1：彩色图像分析
# ========================================================================
def experiment1():
    # 读取并显示彩色图像
    img = cv2.imread('daitu.jpg')
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # 拆分RGB通道
    r, g, b = cv2.split(img_rgb)
    # 显示结果并保存拆分后的通道图像
    plt.figure(figsize=(12, 6))
    plt.subplot(221), plt.imshow(img_rgb), plt.title('origin')
    plt.subplot(222), plt.imshow(r, cmap='gray'), plt.title('red')
    plt.subplot(223), plt.imshow(g, cmap='gray'), plt.title('green')
    plt.subplot(224), plt.imshow(b, cmap='gray'), plt.title('blue')
    plt.tight_layout()
    plt.show()

    # 保存拆分后的通道图像
    cv2.imwrite('daitu_red.jpg', r)
    cv2.imwrite('daitu_green.jpg', g)
    cv2.imwrite('daitu_blue.jpg', b)


# ========================================================================
# 实验2：直方图均衡
# ========================================================================
def experiment2():
    img = cv2.imread('daitu.jpg')
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    r, g, b = cv2.split(img_rgb)

    # 原始直方图
    plt.figure(figsize=(12, 8))
    colors = ('r', 'g', 'b')
    for i, channel in enumerate([r, g, b]):
        hist = cv2.calcHist([channel], [0], None, [256], [0, 256])
        plt.subplot(3, 2, 2 * i + 1)
        plt.plot(hist, color=colors[i])
        plt.title(f'origin {["Red", "Green", "Blue"][i]} Histogram')

    # 直方图均衡
    r_eq = cv2.equalizeHist(r)
    g_eq = cv2.equalizeHist(g)
    b_eq = cv2.equalizeHist(b)

    # 均衡后直方图
    for i, channel in enumerate([r_eq, g_eq, b_eq]):
        hist = cv2.calcHist([channel], [0], None, [256], [0, 256])
        plt.subplot(3, 2, 2 * i + 2)
        plt.plot(hist, color=colors[i])
        plt.title(f'processed {["Red", "Green", "Blue"][i]} Histogram')
```

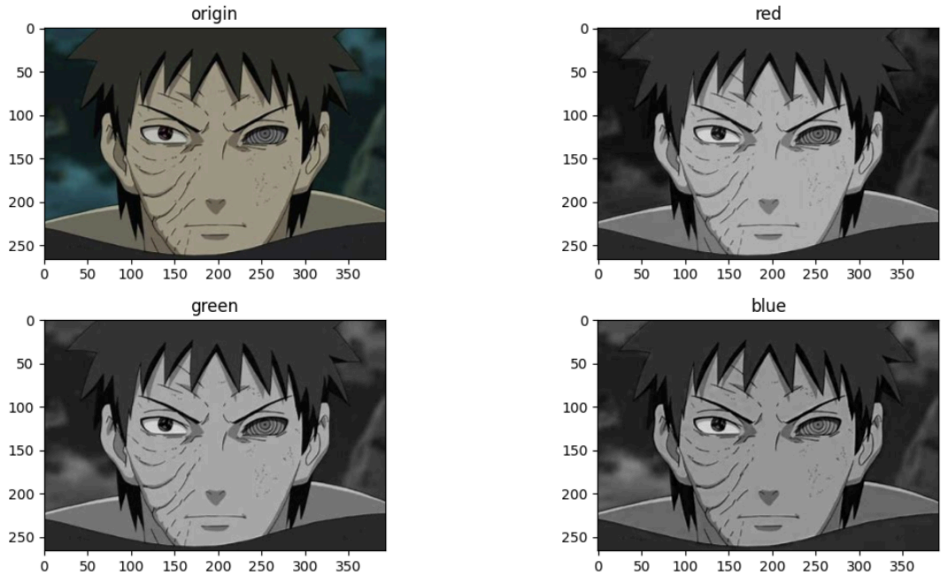
```
    plt.tight_layout()
    plt.show()

    # 合成并显示结果
    img_eq = cv2.merge([r_eq, g_eq, b_eq])
    plt.figure(figsize=(12, 4))
    plt.subplot(121), plt.imshow(img_rgb), plt.title('orign')
    plt.subplot(122), plt.imshow(img_eq), plt.title('processed')
    plt.tight_layout()
    plt.show()


# ========================================================================
# 实验3：假彩色处理
# ========================================================================
def experiment3():
    # 读取所有图像
    def load_gray(path):
        return cv2.imread(path, cv2.IMREAD_GRAYSCALE)

    vl_red = load_gray('daitu_red.jpg')
    vl_green = load_gray('daitu_green.jpg')
    vl_blue = load_gray('daitu_blue.jpg')
    infer_near = load_gray('infer_near.jpg')
    infer_mid = load_gray('infer_mid.jpg')

    # 可见光合成
    visible = cv2.merge([vl_red, vl_green, vl_blue])

    # 近红外替换
    false_near = cv2.merge([infer_near, vl_green, vl_blue])

    # 中红外替换
    false_mid = cv2.merge([infer_mid, vl_green, vl_blue])

    # 显示结果
    plt.figure(figsize=(15, 5))
    plt.subplot(131), plt.imshow(visible), plt.title('origin RGB')
    plt.subplot(132), plt.imshow(false_near), plt.title('near image')
    plt.subplot(133), plt.imshow(false_mid), plt.title('mid image')
    plt.tight_layout()
    plt.show()


# def generate_infrared_effects(img_rgb):
#     """用现有RGB图像生成模拟红外效果"""
#     # 转换为HSV颜色空间
#     hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)
#
#     # 生成近红外模拟（增强植被效果）
#     near_infrared = cv2.addWeighted(img_rgb[:, :, 1], 0.7, img_rgb[:, :, 2],
0.3, 0)  # 混合G和B通道
#
#     # 生成中红外模拟（增强热效应）
```

```
#     mid_infrared = cv2.convertScaleAbs(hsv[:, :, 2], alpha=1.5, beta=50)  # 强化明度通道
#
#     return near_infrared, mid_infrared


def generate_and_save_infrared(img_path='daitu.jpg'):
    # 读取原始图像
    img = cv2.imread(img_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # 生成模拟红外图像
    def generate_effects(rgb):
        # 近红外模拟（增强绿色通道）
        near_ir = cv2.addWeighted(rgb[:, :, 1], 0.8,  # 绿色通道权重
                                  rgb[:, :, 2], 0.2,  # 蓝色通道权重
                                  0)
        # 中红外模拟（增强红色通道边缘）
        mid_ir = cv2.Laplacian(rgb[:, :, 0], cv2.CV_16S)  # 红色通道边缘
        mid_ir = cv2.convertScaleAbs(mid_ir)
        return near_ir, mid_ir

    # 生成并保存红外图像
    infer_near, infer_mid = generate_effects(img_rgb)

    # 确保为8位格式
    infer_near = cv2.normalize(infer_near, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    infer_mid = cv2.normalize(infer_mid, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)

    # 保存文件
    cv2.imwrite('infer_near.jpg', infer_near)
    cv2.imwrite('infer_mid.jpg', infer_mid)
    print("红外图像已保存为 infer_near.jpg 和 infer_mid.jpg")



# ======================================================================
# 实验4：灰度切片
# ======================================================================
def experiment4():
    # 步骤1：读取BGR三通道彩色图像
    img_color = cv2.imread('daitu.jpg', cv2.IMREAD_COLOR)

    # 步骤2：转换为灰度图
    img = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
    # 创建掩模
    mask = np.zeros_like(img)
    mask[img >= 8] = 1  # 8-256级范围

    # Hot模式
    hot_img = cv2.applyColorMap(img, cv2.COLORMAP_HOT)
    hot_img = cv2.bitwise_and(hot_img, hot_img, mask=mask)

    # Cool模式
```

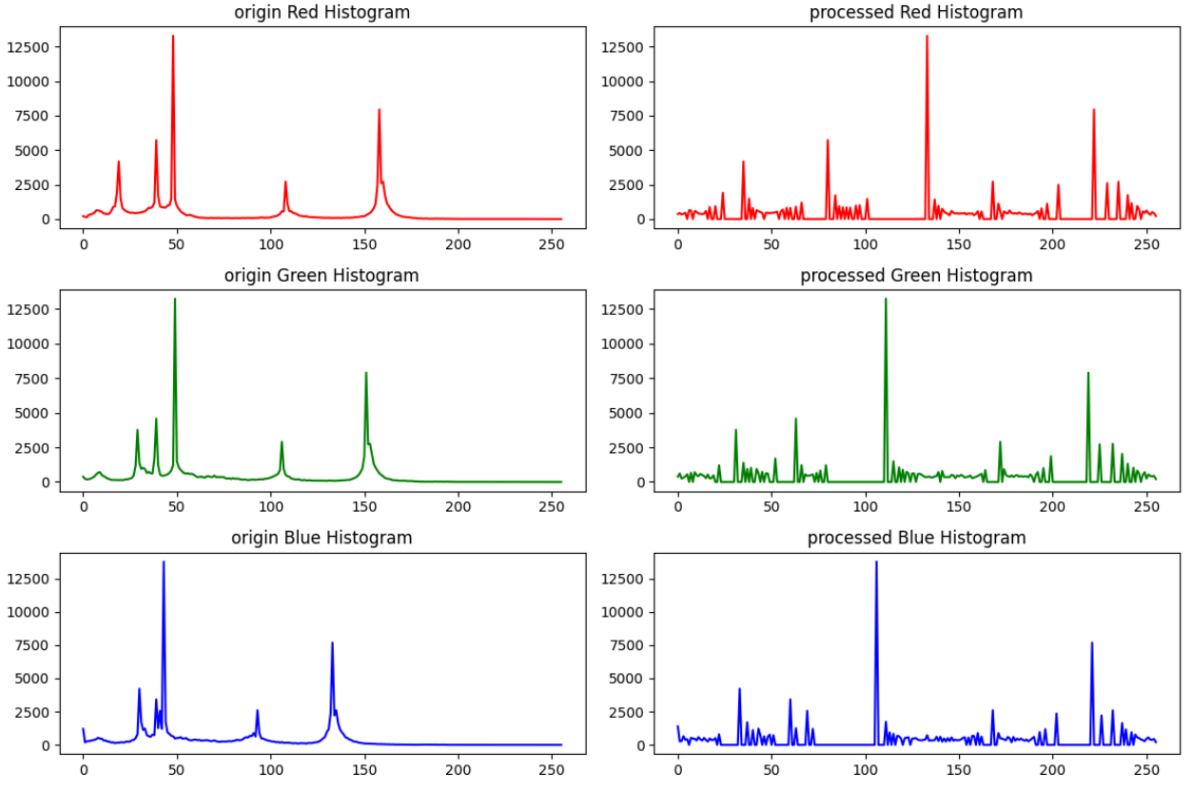
```
    cool_img = cv2.applyColorMap(img, cv2.COLORMAP_COOL)
    cool_img = cv2.bitwise_and(cool_img, cool_img, mask=mask)

    # 显示结果
    plt.figure(figsize=(12, 6))
    plt.subplot(131), plt.imshow(img, cmap='gray'), plt.title('original')
    plt.subplot(132), plt.imshow(cv2.cvtColor(hot_img, cv2.COLOR_BGR2RGB)),
plt.title('Hot')
    plt.subplot(133), plt.imshow(cv2.cvtColor(cool_img, cv2.COLOR_BGR2RGB)),
plt.title('Cool')
    plt.tight_layout()
    plt.show()


# ========================================================================
# 实验5：彩色变换（选做）
# ========================================================================
def experiment5():
    img = cv2.imread('daitu.jpg', cv2.IMREAD_GRAYSCALE)
    img_norm = img.astype(np.float32) / 255.0

    # 生成变换通道
    theta = 2 * np.pi
    R = -np.sin(img_norm * theta)
    G = -np.cos(img_norm * theta)
    B = np.sin(img_norm * theta)

    # 归一化到[0,1]
    R = (R + 1) / 2
    G = (G + 1) / 2
    B = (B + 1) / 2

    # 合成图像
    rgb_sin = cv2.merge([R, G, B])

    # 交换RG通道
    rgb_swap = cv2.merge([G, R, B])

    # 显示结果
    plt.figure(figsize=(12, 6))
    plt.subplot(131), plt.imshow(img, cmap='gray'), plt.title('origin')
    plt.subplot(132), plt.imshow(rgb_sin), plt.title('sin')
    plt.subplot(133), plt.imshow(rgb_swap), plt.title('exchange')
    plt.tight_layout()
    plt.show()


# ========================================================================
# 主程序
# ========================================================================
if __name__ == "__main__":
    experiment1()
    experiment2()
    generate_and_save_infrared()
    experiment3()
    experiment4()
```

```
    experiment5()  # 选做实验
```

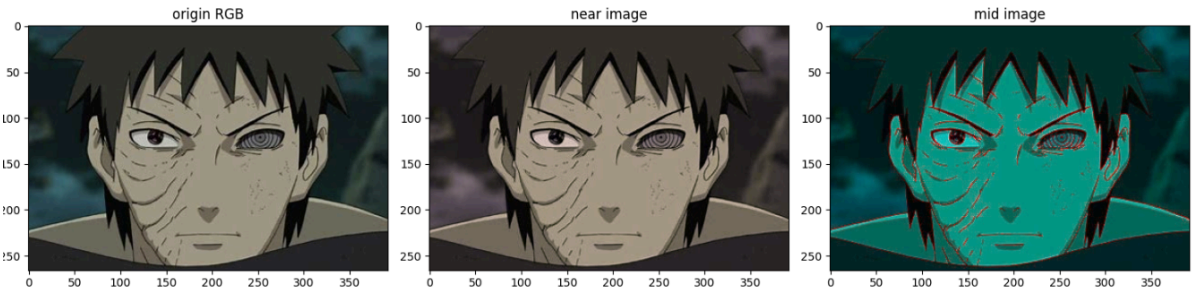
三原色



直方图

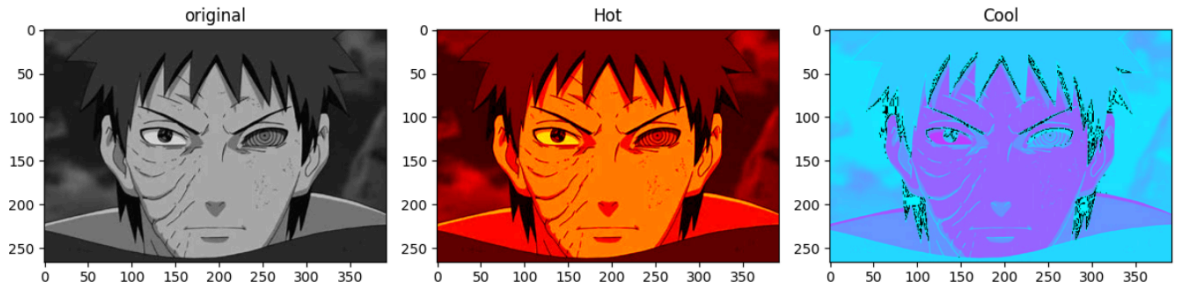


合成图像



---

hot-cold模式



四、思考题

问题1：为什么彩色直方图均衡后的图像除了对比度增强外，还有色调变化?

原因分析:

1. **分通道独立均衡**:
    - 直方图均衡是对每个颜色通道（R、G、B）单独进行的操作，而各通道的原始分布不同。例如:
        - 若原始图像的红色通道集中在暗区，均衡后红色分量会被拉伸到更宽的动态范围，可能导致红色分量过亮。
        - 若绿色通道原本分布均匀，均衡后变化较小。
    - 这种通道间的**独立调整**会改变颜色分量的相对比例，导致整体色调偏移。
2. **颜色相关性破坏**:
    - RGB颜色空间中，颜色由三通道的**联合分布**决定。单通道均衡会破坏通道之间的自然相关性。
    - 例如：原本的肤色可能由特定比例的R/G/B组成，均衡后比例失衡可能导致肤色偏红或偏青。
3. **非线性增强效果**:
    - 直方图均衡是非线性变换，可能在某些区域过度增强特定颜色分量。

- 若某个通道（如蓝色）在暗区有大量像素，均衡后可能产生不自然的亮蓝色区域。

实验验证：

- 在实验2中，可观察到均衡后的图像可能偏向某一种色调（如偏红），这是因为红色通道的均衡效果更显著。

问题2：假彩色处理是否可以有多种方案？其他方案的可能结果？

其他可行方案：

1. **通道替换组合多样化**：

   方案1

   ：用近红外替换绿色通道（G）

   - 结果：植被（高近红外反射）会显示为亮绿色，水体（低反射）显示为暗色。

   方案2

   ：用中红外替换蓝色通道（B）

   - 结果：高温区域（如火灾）会显示为亮蓝色，低温区域为暗蓝色。

   方案3

   ：多波段融合（如近红外+中红外同时替换两个通道）

   - 结果：复杂场景中不同地物通过颜色组合区分（如植被=红色，水体=蓝色）。

2. **波段运算增强**：

   方案4

   ：近红外与可见光波段叠加（如

   ```
   R = 0.5*红外 + 0.5*R
   ```

   - 结果：红外信息与可见光颜色混合，突出特定目标（如伪装植被）。

3. **伪彩色映射扩展**：

   方案5

   ：将红外波段映射到HSV颜色空间的色调通道

   - 结果：不同红外强度对应不同颜色（如低→蓝色，高→红色）。

可能结果示例：

- **健康植被检测**：近红外替换红色通道时，健康植被显示为亮红色；若替换绿色通道，则显示为亮绿色。
- **火灾监测**：中红外替换红色通道时，火点显示为亮红色；替换蓝色通道则显示为亮蓝色。

问题3：灰度切片处理中，图像head.jpg使用多少级切片合适？

选择依据：

1. **图像特性**：
   - 若图像灰度分布范围窄（如医学图像集中在某区间），需**精细分级**（如16~32级）。
   - 若灰度分布广且对比度低（如自然场景），**粗粒度分级**（如8级）更有效。

2. **应用需求**：
    - **诊断需求**（如医学影像）：需多级切片（16~256级）以区分细微组织差异。
    - **可视化需求**：8~16级可平衡视觉效果与计算复杂度。
3. **实验结果验证**：
    - 在实验4中，使用8级切片时：
        - **优点**：颜色对比明显，便于快速区分主要结构。
        - **缺点**：可能丢失细节（如软组织的渐变层次）。
4. **直方图均衡的色调变化**源于通道独立操作对颜色平衡的破坏。
5. **假彩色方案多样性**可通过通道替换、波段融合和颜色空间映射实现。
6. **灰度切片分级**需权衡图像特性与应用需求，8~16级适用于多数场景。