《计算机视觉》实验报告

姓名：林艺珺 学号：18120189

实验 2

任务 1

在 tree.jpg 中，以图中右侧树为模板，在左侧复制出第 2 棵树
要求尽量做到效果逼真

图 1: tree.jpg

a）核心代码

个人笔记本为 MacOS 平台，选择开发语言为 Python，IDE 为 JetBrains 的 Pycharm。

搜索了网络上的资源，寻找到了 OpenCV 3 的一个功能 Seamless Clone，其作用是将一个图像融合进入另一个背景图像中，并显得自然。函数原型如下：

Python

```
output = cv2.seamlessClone(src, dst, mask, center, flags)
```

C++

```
seamlessClone(Mat src, Mat dst, Mat mask,
                  Point center, Mat output, int flags)
```

src 为所要复制图像，即前景图；dst 为所要复制图像的目标图片，即背景图；mask 为所要复制物体的大致蒙版，大小与前景图一致，全白蒙版同样可用；center 为前景图在背景图的中心位置；flags 为复制的模式选择，有 NORMAL_CLONE 和 MIXED_CLONE 两种，后者会在全白蒙版时根据前景和背景的图案进行选择和合成，而前者会以前景图为主；output 为最终的输出图像。

```
1 import cv2 as cv
2 import numpy as np
```

```
# 读取原始图片
img = cv.imread("tree.jpeg")
# 选定右侧树图像范围
obj = img[188:519, 538:818]
# 为选定右侧树图像范围添加一个全白蒙版
mask = 255 * np.ones(obj.shape, obj.dtype)
# 选定右侧树图像将要放置在原图像位置的中心点
center = (320, 352)
# 无缝将选定图像复制进原始图片
normal_clone = cv.seamlessClone(obj,img,mask,center,cv.NORMAL_CLONE)
# 将结果图片保存并显示
cv.imwrite("normal-clone-example.jpg", normal_clone)
cv.imshow("normalclone", normal_clone)
cv.waitKey(0)
```

b）实验结果截图

图 2: 结果图片

c）实验小结

在实验伊始，我完全用先前人工操作的方式思考了这道实验题：抠图、复制、边缘柔化、灯光调整，然后发现抠图这样的操作，在没有相当实用的 GUI 中是很难实现的，特别是如果要使用代码中严谨的数字来表达抠图过程中的钢笔痕迹的话，对于人物图片中的树来说是及其繁杂的。后来我去官方文档中搜寻有关抠图的函数，grabCut 是一个功能较强的函数，但是在任务图中，树的绿色与海面下部的蓝色，其实不容易被电脑区分，多次试验，即便调高迭代次数，运行速度及其缓慢，但所得的结果都差距不大，并且边缘痕迹明显，无法直接将其挪至画面左边完美融合，呈现效果较差。最终变换思路，从 Photoshop 中类似的“融合方式”的功能中思考，找到了 Seamless Clone 这个函数，顺利完成了实验，在过程中也找到了相关的原理文档进行初步学习。

另外，经过思考，我觉得色彩提取或许也能完成这个实验，效果可能不如 Seamless Clone，但或许可以比默认状态下的 grabCut 好。选取目标物体区域后，选定多个特征颜色并有一定浮动范围，从中提取相应像素点后进行复制并移动。之后有机会再尝试一下这个方法。

任务 2（选做）

请上网搜索混合图像（hybrid image）的原理

比如：https://zhuanlan.zhihu.com/p/104549501

请自己写程序，能生成 cat 和 dog，Marilyn 和 Einstein 的混合图像

a）核心代码

个人笔记本为 MacOS 平台，选择开发语言为 Python，IDE 为 JetBrains 的 Pycharm。

首先根据老师提供的资料，另外自行寻找相关资源进行了 Image Filtering 的理论学习，主要使用的资料是 Cornell 和 CMU 的 Computer Vision 教学讲义/Slides。

根据理论学习，首先使用 OpenCV 提供的函数进行实验，此部分相关内容见“实验小结”。

之后根据 numpy 库提供的计算及 OpenCV 库提供的部分转换、输入输出函数，使用 Cornell Project 1 开源在 Github 上的 skeleton code 的框架进行程序设计。（源代码中保留了 Cornell 框架中提供的函数英文注释）

二维卷积计算

根据二维卷积公式，其中 F 是原始图像，H 是 Kernel 核（大小 (2k+1)*(2k+1)），G 为输出图像。

$$G[i,j] = \Sigma_{u=-k}^{k}\Sigma_{v=-k}^{k}H[u,v]F[i-u,j-v]$$

```
def convolve_2d(img, kernel):
# 输入：img：单通道图像；kernel：高斯核。
# 输出：卷积后图像。
    k_height = kernel.shape[0]
    k_width = kernel.shape[1]
    conv_height = img.shape[0] - k_height + 1
    conv_width = img.shape[1] - k_width + 1
    # 创建全零conv
    conv = np.zeros((conv_height, conv_width))
    for i in range(0, conv_height):
        for j in range(0, conv_width):
            conv[i][j] = wise_element_sum(img[i:i + k_height, j:j +
                k_width], kernel)
    return conv

def wise_element_sum(img, kernel):
# 卷积计算辅助函数
    res = (img * kernel).sum()
    if (res < 0):
        res = 0
    elif res > 255:
        res = 255
    return res
```

二维高斯核计算 根据二维高斯公式，进行程序设计。

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

```python
def gaussian_blur_kernel_2d(sigma, height, width):
# 输入: sigma: 高斯模糊半径; width, height: 高斯核的大小。
# 输出: 高斯核。
    # 创建全零高斯核
    kernel = np.zeros([height, width])
    # 中心点为原点
    center_h = height/2
    center_w = width/2
    pi = 3.1415926
    # sigma为零时，使用OpenCv库中的算法根据kernel大小计算
    if sigma == 0:
        sigma = ((height-1) * 0.5 - 1) * 0.3 + 0.8
    for i in range(0, height):
        for j in range(0, width):
            x = i - center_h
            y = j - center_w
            kernel[i, j] = (np.exp(- (x**2 + y**2) / (2 * (sigma**2))))
                / (2 * pi * (sigma**2))
    return kernel
```

高低通滤波后图像

将 RGB 图片分为三通道，根据 sigma 和 kernel size 计算高斯核，再单通道进行卷积计算，最后将三通道合成并返回。

```python
def low_pass(img, sigma, size):
# 输入: img: 原始图像; sigma: 高斯模糊半径; size: 高斯核的大小。
# 输出: 低通图。
    low_b = img[:, :, 0]    # Blue通道
    low_g = img[:, :, 1]    # Green通道
    low_r = img[:, :, 2]    # Red通道
    kernel = gaussian_blur_kernel_2d(sigma, size, size)
    # 分通道卷积计算
    low_b = convolve_2d(low_b, kernel)
    low_g = convolve_2d(low_g, kernel)
    low_r = convolve_2d(low_r, kernel)
    # 三通道合成
    lowimg = np.dstack([low_b, low_g, low_r])
    return lowimg

```

```python
def high_pass(img, sigma, size):
# 输入：img：原始图像；sigma：高斯模糊半径；size：高斯核的大小。
# 输出：高通图。
    lowimg = low_pass(img, sigma, size)
    # 将原始图片缩放至卷积后图片相同大小
    dim = (lowimg.shape[1], lowimg.shape[0])
    resized = cv.resize(img, dim)
    highimg = resized - low_pass(img, sigma, size)
    return highimg
```

主函数

```python
high = cv.imread("图片1.jpg")
low = cv.imread("图片2.jpg")
# 为plt显示进行的转换
high = high/255
low = low/255
# size为21是经过调试选择
lowpass = low_pass(low, 0, 21)
highpass = high_pass(high, 0, 21)

plt.figure("Hybrid Image")
# plt颜色通道是RGB，OpenCV是BGR，直接输出会导致颜色出错，因此进行转换
# numpy的数据格式为Float64，而OpenCV此处需要Float32.因此进行转换
plt.subplot(1,3,1)
plt.imshow(cv.cvtColor(lowpass.astype(np.float32), cv.COLOR_BGR2RGB))
plt.xticks([])
plt.yticks([])
plt.subplot(1,3,2)
# 为得到和Cornell实验网站上一致的高通图，输出时增加了亮度
plt.imshow(cv.cvtColor((highpass+0.5).astype(np.float32), cv.
    COLOR_BGR2RGB))
plt.xticks([])
plt.yticks([])
# Hybrid图像合成，其中数字可以更改
hybrid = lowpass * 1 + highpass * 1
plt.subplot(1,3,3)
plt.imshow(cv.cvtColor(hybrid.astype(np.float32), cv.COLOR_BGR2RGB))
plt.xticks([])
plt.yticks([])
plt.show()

cv.waitKey(0)
```

b）实验结果截图

图 3: Cat & Dog 代码运行结果截图

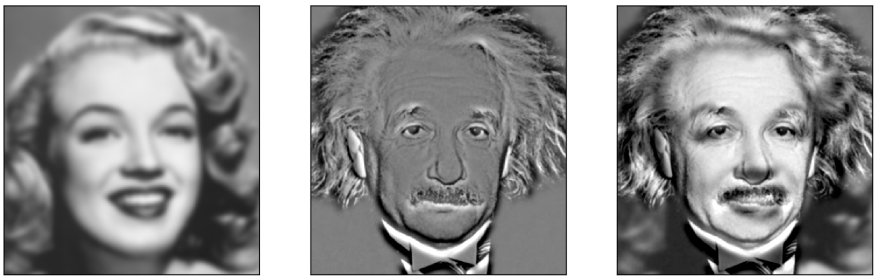

图 4: Einstein & Marilyn 代码运行结果截图

c）实验小结

本次实验伊始，经过理论学习后，直接去 OpenCV 的库中寻找相应函数并进行实验，但是在实验过程中发现结果输出与预想有差异。

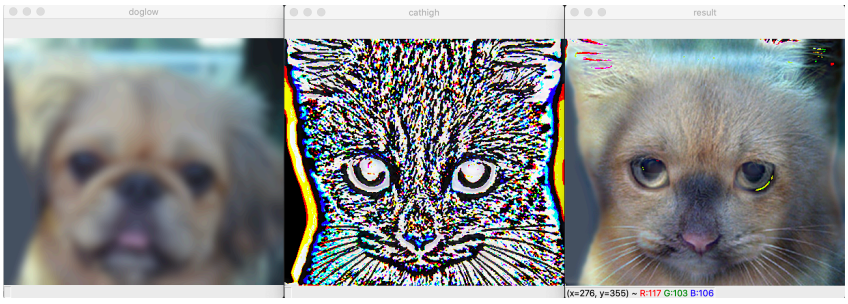

图 5: 使用 Filter2D+getGaussianKernel 或 GaussianBlur 进行实验结果

值得一提的是，其中的高斯函数与二维高斯函数公式的计算方式不同。其并未乘上系数 $\frac{1}{2\pi\sigma^2}$，而是进行求和得出，不知此点是否导致最终输出的不理想。虽然 OpenCv 是一个非常强大的库，拥有很多打包好的可以直接使用的函数，但是在某些情况下，底层的计算逻辑无法调整，会得出与期望不一致的结果，并且找不到原因。

另外，在写输出的时候，发现不同的函数库对于数据格式的要求变化很大，一个数据不符合要求就可能导致报错无法运行，因此在查找函数的同时，要时刻注意每个字段对应的数据类型，以减少后续 debug 的工作量。

最后，在寻找相关资源的时候找到一本国外大学都在使用的经典教材《Computer Vision: Algorithms and Applications》，有空的时候可以自己研读一下，以提升在计算机视觉方面的知识储备。