实验报告

Steven

2021.12.31

目录

一、实验内容

1. 问题描述与任务要求

(1) 图像分割：读取一幅图像，编写一个分割函数，将其均匀分割成九张子图像(3×3)。

(2) 图像马赛克：读取一幅图像，编写一个马赛克函数，要求输入需要打马赛克的区域（四个坐标参数，模糊程度参数），输出该区域的马赛克效果。

(3) 图像增强：读取一幅图像，编写一个图像增强函数，输入色度、对比度、锐度增强参数，分别输出保存增强结果，并将增强后的四幅图合成一幅图像显示。

(4) 交互式分割：使用鼠标拖动，按照拖出的矩形框作为选定区域进行分割。

(5) 自动马赛克处理：自动定位人脸并进行马赛克处理。

(6) 自适应图像增强：对图像参数进行自动优化，使其表现力更强。

2. 实验配置

(1) 软件：

Pycharm 2021.2.3（基于 Anaconda 3，Python 3.9）

(2) 硬件：

Matebook (Core i7 10th)

(3) 依赖库：

cv2（opencv-python）、PIL（Python Image Library）、numpy（Numerical Python）、tkinter

二、设计文档

1. 整体思路

(1)确定预期目标

功能目标：

完成所有的功能，使整个项目功能完善，实用性强。

交互方式：

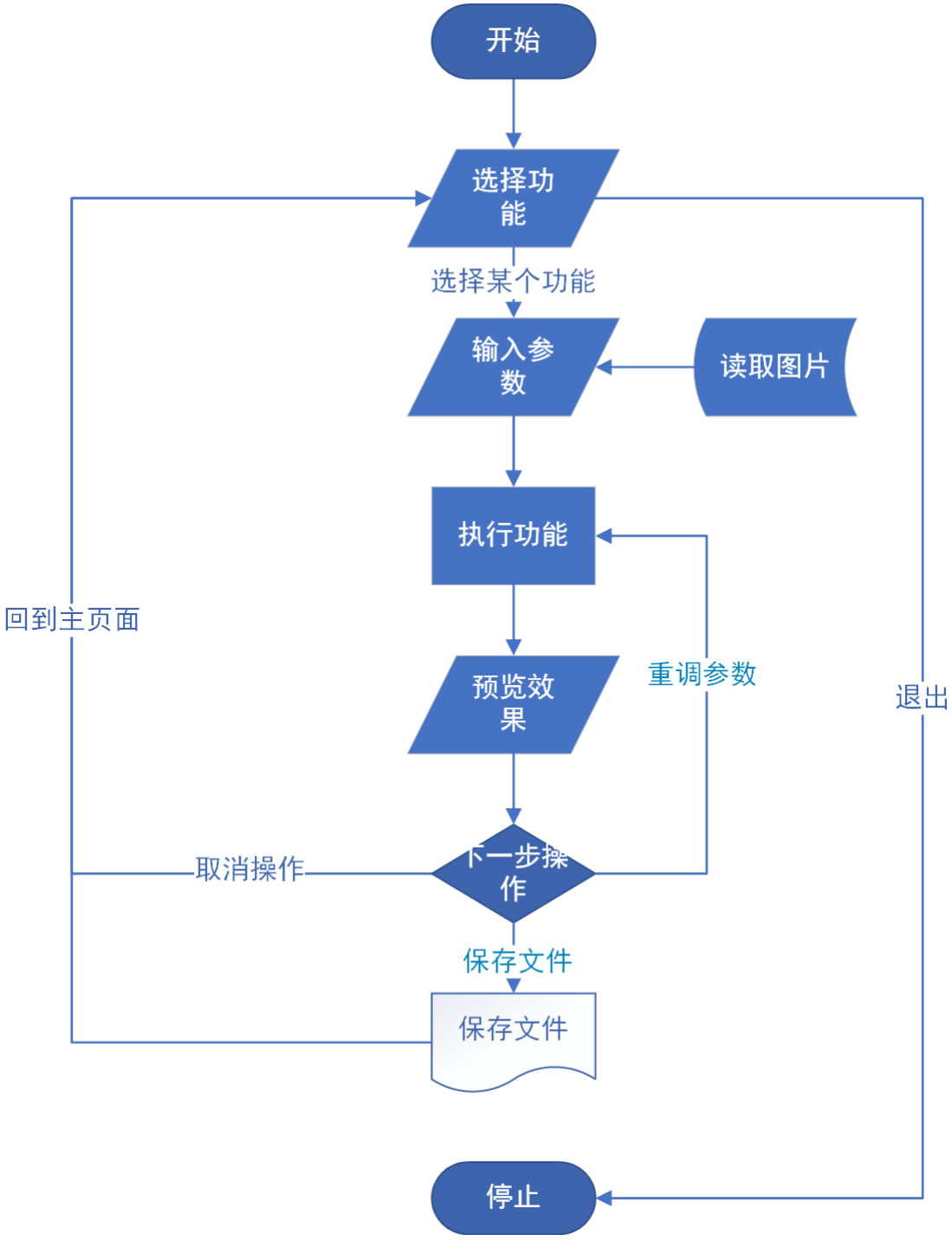
由于本程序需要用户指定的信息较多，且本身处理的对象就是图像，在此情况下使用图形界面交互是十分合理的。所以为了便于预览和操作的，本项目所有交互均应在图形用户界面下完成。

代码结构：

①由于使用构建的窗口进行交互，故为了保证代码的可读性和结构层次合理，整个项目中的代码是前后端分离的，并使用一个额外的对象实现前后端的数据传输。

②由于本项目中功能较多，为了便于管理、生成具体的操作类，本项目的部分代码借鉴了抽象工厂模式来进行实现。

(2)预想操作流程



2. 代码结构及描述

(1) ProcessImage.py

该源文件的所有函数只进行图像的运算、处理，不包含任何的展示功能。

def read(path):	**功能：** 该函数包装了 cv2.imread 函数，使得底层函数不会暴露在操作端 **输入：** (str) path:被读取图片文件的路径 **输出：** (ndarray):经 OpenCV 读取后的图片矩阵
def save(image_CV, path_dir, quality, name):	**功能：** 该函数包装了 cv2.imwrite 函数，支持以多种质量、多张同时的形式保存图片 **输入：** (ndarray/list) image_CV:被保存的图片矩阵或图片矩阵的列表 (str) path_dir:保存目录的绝对路径 (str) quality:设定的保存质量 (str) name:被保存图片/图片组的名称
def capturePart(image_CV, boundary):	**功能：** 截取图的一部分并返回 **输入：** (ndarray) image_CV:经 OpenCV 读取后的图片矩阵 (list) boundary:以列表的形式保存的被截取区域的 x、y 范围 **输出：** (ndarray):原图截取后的在范围内的部分。
def gridToNine(image_CV, blockNum_inRow):	**功能：** 调用 capturePart，截取原图的九宫格图，并作为列表返回 **输入：** (ndarray) image_CV:被截取的图片矩阵 (int) blockNum_inRow:一行内的块数，截取九宫格则该值为 3 **输出：** (list):原图的九宫格图的列表。
def addMosaic(image_CV, boundary, strength):	**功能：** 为图片的指定区域添加指定强度的马赛克 **输入：** (ndarray) image_CV:经 OpenCV 读取后的图片矩阵 (list) boundary:以列表的形式保存的被截取区域的 x、y 范围 (int) strength:马赛克强度，即单个色块的边长（像素数） **输出：** (ndarray):原图添加过马赛克后的图片。
def enhance(image_CV, strength):	**功能：** 将一张图分成四部分，分别根据参数进行亮度、对比度、色度、锐度的调整，并返回参数调整后的拼合图 **输入：** (ndarray) image_CV:经 OpenCV 读取后的图片矩阵

	(list) strength:以列表的形式存放 4 个属性的增强参数 **输出：** (ndarray):原图四个部分分别进行提升后的拼合图像。
def locateFace(image_CV):	**功能：** 定位图片中的人脸位置，并返回其位置（基于像素位置） **输入：** (ndarray) image_CV:经 OpenCV 读取后的图片矩阵 **输出：** (tuple):原图中人脸位置的坐标，分别为(左上顶点的 x、y、宽度，高度)
def autoEnhance(image_CV, lowCut=0.005, highCut=0.005):	**功能：** 自适应优化图像的对比度和色阶，并返回调整后的图片 **输入：** (ndarray) image_CV:经 OpenCV 读取后的图片矩阵 (float) lowCut=0.005:线性映射部分的下界 (float) highCut=0.005:线性映射部分的上界 **输出：** (ndarray):原图经过自适应调整后的图像。
def find(tuple_input, limitation):	**功能：** 查找并返回 tuple_input 中最先达到 limitation 的元组序号，本质上是为了在尽量节省计算机性能的前提下替代 Matlab 中的 find 函数，只被 autoEnhance 函数调用 **输入：** (tuple) tuple_input:被遍历的元组 (int) limitation:大小限制 **输出：** (int):在 tuple_input 中最先达到 limitation 的元素序号
def linearMap(low, high):	**功能：** 自动对比度和色阶算法所需要的线性映射函数，只被 autoEnhance 函数调用 **输入：** (float) low:线性映射的下界 (float) high:线性映射的上界 **输出：** (list):经线性（扩张）映射后的隐射表

(2) Display.py

该源文件的所有函数、类只进行交互所需的展示、获取信息，不进行数据的处理。

函数部分

def ShowMenu(backImage):	**功能：** 创建一个 Tk 根窗口，用于显示主菜单 **输入：** (ndarray) backImage:用于美化主菜单界面的一张装饰图
def CV2Tk(image_CV, width, height):	**功能：** 借助 PIL，将使用 OpenCV 读取的图片转换为可在 tkinter 组件上显示的图片 **输入：**

	(ndarray) image_CV:使用 OpenCV 读取的图片矩阵 (int) width:被转换后的图片宽度（基于 tk 组件的设定大小） (int) height:被转换后的图片高度（基于 tk 组件的设定大小） **输出：** (pyimage):用于在 tkinter 组件上显示的图像。
def getImagePath():	**功能：** 包装了 tkinter.filedialog.askopenfilename，以实现被打开文件的选取 **输出：** (str):被打开图片的绝对路径
def setSavePath(title):	**功能：** 包装了 tkinter.filedialog. askdirectory，以实现保存路径的选取 **输出：** (str):文件保存目录的绝对路径

class WindowBase:之后所有功能窗口的父类

def __init__(self, title):	**功能：** 创建一个顶级窗口，设定为 800×450 的大小，并将其居中显示，该窗口用于显示当前功能面板 **输入：** (str) title:当前窗口的标题
def calcResize (width_image, height_image, width_canvas, height_canvas):	**功能：** 计算为适应画布，图像缩略图应调整的实际大小 **输入：** (int) width_image:原图的实际宽度 (int) height_image:原图的实际高度 (int) width_canvas:画布的宽度 (int) height_canvas:画布的高度
def getImagePath():	**功能：** 包装了 tkinter.filedialog.askopenfilename，以实现被打开文件的选取 **输出：** (str):被打开图片的绝对路径
def setSavePath(title):	**功能：** 包装了 tkinter.filedialog. askdirectory，以实现保存路径的选取 **输出：** (str):文件保存目录的绝对路径
def addLabel(self, master, image_CV):	**功能：** 为选定的区域上添加标签组件 **输入：** (Toplevel) master:标签的父级组件 (pyimage) image_CV:标签上显示的初始图片

class GridImage(WindowBase)：裁切九宫格图功能的实现类

def __init__(self):	**功能：** 继承父类__init__函数，并设定了标题名，声明一些全局变量
def	**功能：**

showWindow(self, imgCV_List):	构建九宫格图的操作窗口，包含命名输入框和保存按钮 **输入：** (list) imgCV_List:切好的九宫格图（每张图均为 ndarray）

class DragCapture(WindowBase):拖拽截图功能的实现类

def __init__(self):	**功能：** 继承父类__init__函数，并设定了标题名，声明一些全局变量
def showWindow(self, image_CV):	**功能：** 构建拖拽截图的操作窗口，包含命名输入框、预览按钮、保存按钮、图像预览标签和鼠标监听器 **输入：** (ndarray) image_CV:被操作的的图片
def OnMouse(self, event):	**功能：** 鼠标监听器，获取按下鼠标时在窗口中的 x 和 y 坐标，并存到全局变量中 **输入：** (Event) event:按下鼠标事件
def ReleaseMouse(self, event):	**功能：** 鼠标监听器，获取松开鼠标时在窗口中的 x 和 y 坐标，并存到全局变量中 **输入：** (Event) event:松开鼠标事件
def UpdatePreview(self, image_CV, preview, width_pre, height_pre):	**功能：** 更新窗口中的预览图 **输入：** (ndarray) image_CV:要在窗口中预览的图片 (Label) preview:预览图片的 Label 的句柄 (int) width_pre:显示预览图的 Label 的宽度 (int) width_pre:显示预览图的 Label 的高度
def ResetPreview(self, image_CV, preview, width_pre, height_pre):	**功能：** 重置窗口中的预览图为原图 **输入：** (ndarray) image_CV:要在窗口中预览的图片 (Label) preview:预览图片的 Label 的句柄 (int) width_pre:显示预览图的 Label 的宽度 (int) width_pre:显示预览图的 Label 的高度

class MosaicImage(WindowBase):添加马赛克功能的实现类

def __init__(self):	**功能：** 继承父类__init__函数，并设定了标题名，声明一些全局变量
def showWindow(self, image_CV):	**功能：** 构建添加马赛克的操作窗口，包含四个边界的输入框、强度输入框、命名输入框、预览按钮、保存按钮和图像预览标签 **输入：** (ndarray) image_CV:被操作的的图的图源
def UpdatePreview(self,	**功能：** 更新/重置窗口中的预览图

image_CV, boundary, mosaicStrength, preview):	**输入：** (ndarray) image_CV:要在窗口中预览的图片 (list) boundery:打马赛克的矩形区域的范围 (int) mosaicStrength:马赛克区域的强度 (Label) preview:预览图片的 Label 的句柄

class MosaicImage_Auto(WindowBase):自动添加马赛克的实现类

def __init__(self):	**功能：** 继承父类__init__函数，并设定了标题名，声明一些全局变量
def showWindow(self, image_CV):	**功能：** 构建自动添加马赛克的操作窗口，包含马赛克强度的输入框、命名输入框、预览按钮和保存按钮 **输入：** (ndarray) image_CV: 被操作的的图片
def UpdatePreview(self, image_CV, axis, mosaicStrength, width, height, preview):	**功能：** 更新窗口中的预览图 **输入：** (ndarray) image_CV:要在窗口中预览的图片 (list) axis:人脸区域的位置参数 (int) mosaicStrength:马赛克强度 (int) width:画布的宽度 (int) height:画布的高度 (Label) preview:预览图片的 Label 的句柄
def ResetPreview(self, image_CV, preview, previewWidth, previewHeight):	**功能：** 重置窗口中的预览图为原图 **输入：** (ndarray) image_CV:要在窗口中预览的图片 (Label) preview:预览图片的 Label 的句柄 (int) previewWidth:画布的宽度 (int) previewHeight:画布的高度

class AttributeImage(WindowBase):调整图像参数的实现类

def __init__(self):	**功能：** 继承父类__init__函数，并设定了标题名，声明一些全局变量
def showWindow(self, image_CV):	**功能：** 构建调整参数的操作窗口，包含四种参数的滑块、命名输入框、预览按钮和保存按钮 **输入：** (ndarray) image_CV: 被操作的的图片
def UpdatePreview(self, image_CV, strength, width, height, preview):	**功能：** 更新/重置窗口中的预览图 **输入：** (ndarray) image_CV:要在窗口中预览的图片 (list) strength:各参数的强度 (int) width:画布的宽度

	(int) height:画布的高度 (Label) preview:预览图片的 Label 的句柄

class AttributeImage_Auto(WindowBase):自动调整参数的实现类

def __init__(self):	**功能：** 继承父类__init__函数，并设定了标题名，声明一些全局变量
def showWindow(self, image_CV):	**功能：** 构建自动添加马赛克的操作窗口，包含命名输入框、预览按钮、显示原图按钮和保存按钮 **输入：** (ndarray) image_CV: 被操作的的图片
def UpdatePreview(self, image_CV, width, height, preview):	**功能：** 更新窗口中的预览图 **输入：** (ndarray) image_CV:要在窗口中预览的图片 (int) width:画布的宽度 (int) height:画布的高度 (Label) preview:预览图片的 Label 的句柄
def ResetPreview(self, image_CV, width, height, preview):	**功能：** 重置窗口中的预览图为原图 **输入：** (ndarray) image_CV:要在窗口中预览的图片 (int) width:画布的宽度 (int) height:画布的高度 (Label) preview:预览图片的 Label 的句柄

(3) ControlPanel.py

该源文件的所有函数均用于协调某个功能的数据传输。

def Start():	**功能：** 调用显示主菜单
def CreateNineGrid():	**功能：** 将图像剪切成九宫格，再调用相关展示类显示九宫格图
def Capture():	**功能：** 完成拖拽裁切功能的初始化
def CreateMosaicImg():	**功能：** 完成马赛克图窗口的初始化
def AutoCreateMosaicImg():	**功能：** 完成自动创建马赛克图的窗口初始化
def CreateAttributeImg():	**功能：** 完成修改图片参数窗口的初始化
def AutoAttributeImg():	**功能：** 完成自适应提升图像素质的窗口初始化
def Exit():	**功能：** 退出程序

三、程序源代码

1. 图像处理代码

ProcessImage.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*
"""
本程序主要负责对OpenCV的图像对象进行处理、转换
因为其操作过程不需要全局变量，故采用面向过程的编程思路
注意事项：
(1)由于OpenCV库不支持中文（无论是否UTF-8），所以请保证操作过程中路径名、文件名以及其他与cv2库函数有关的内容不包含中文
(2)由于使用 cv2.CascadeClassifier 时，参数需要为绝对路径，所以请在运行时手动修改函数 locateFace 内的路径
"""
import cv2 as cv
import numpy as np
from PIL import Image, ImageEnhance

__all__ = ['read', 'save', 'capturePart', 'gridToNine', 'addMosaic', 'enhance', 'locateFace', 'autoEnhance', 'find', 'linearMap']


def read(path):
    return cv.imread(path)


def save(image_CV, path_dir, quality, name):
    if isinstance(image_CV, list):
        for i in range(len(image_CV)):
            path_save = path_dir + '/' + (('Untitle_' if name == '' else (name + "_")) + str(i + 1)) + '.jpg'
            if quality == 'Origin':
                cv.imwrite(path_save, image_CV[i], (cv.IMWRITE_JPEG_QUALITY, 100))
            elif quality == 'High':
                cv.imwrite(path_save, image_CV[i], (cv.IMWRITE_JPEG_QUALITY, 72))
            elif quality == 'Medium':
                cv.imwrite(path_save, image_CV[i], (cv.IMWRITE_JPEG_QUALITY, 60))
            elif quality == 'Low':
                cv.imwrite(path_save, image_CV[i], (cv.IMWRITE_JPEG_QUALITY, 45))
    else:
        path_save = path_dir + '/' + ('Untitle' if name == '' else name) + '.jpg'
        if quality == 'Origin':
            cv.imwrite(path_save, image_CV, (cv.IMWRITE_JPEG_QUALITY, 100))
```

```
        elif quality == 'High':
            cv.imwrite(path_save, image_CV, (cv.IMWRITE_JPEG_QUALITY, 72))
        elif quality == 'Medium':
            cv.imwrite(path_save, image_CV, (cv.IMWRITE_JPEG_QUALITY, 60))
        elif quality == 'Low':
            cv.imwrite(path_save, image_CV, (cv.IMWRITE_JPEG_QUALITY, 45))


def capturePart(image_CV, boundary):
    x_left = 0 if boundary[0] < 0 else boundary[0]
    x_right = image_CV.shape[1] - 1 if boundary[1] > image_CV.shape[1] - 1 else
boundary[1]
    y_top = 0 if boundary[2] < 0 else boundary[2]
    y_bottom = image_CV.shape[0] - 1 if boundary[3] > image_CV.shape[0] - 1 else
boundary[3]
    image_new_CV = image_CV[y_top:y_bottom, x_left:x_right]
    return image_new_CV


def gridToNine(image_CV, blockNum_inRow):
    height_origin = image_CV.shape[0]
    width_origin = image_CV.shape[1]
    height_block = height_origin // blockNum_inRow
    width_block = width_origin // blockNum_inRow
    images_grid = []
    for i in range(0, blockNum_inRow):
        for j in range(0, blockNum_inRow):
            boundary = [j * width_block, (j + 1) * width_block, i * height_block, (i +
1) * height_block]
            images_grid.append(capturePart(image_CV, boundary))
    return images_grid


# 为图片添加马赛克
def addMosaic(image_CV, boundary, strength):
    mosaicPart_height = boundary[3] - boundary[2]  # 马赛克区域的高度
    mosaicPart_width = boundary[1] - boundary[0]  # 马赛克区域的宽度
    mosaicImage = image_CV.copy()
    if strength != 0 and strength != 1:  # 当strength==0 或==1 时，将视为不进行马赛克处理
        for i in range(0, mosaicPart_width // strength + 1):
            for j in range(0, mosaicPart_height // strength + 1):
                leftEdge = boundary[0] + i * strength
                rightEdge = min(boundary[0] + (i + 1) * strength, image_CV.shape[1] - 1)
                topEdge = boundary[2] + j * strength
                bottomEdge = min(boundary[2] + (j + 1) * strength, image_CV.shape[0] -
1)
```

```
                part_image = image_CV[topEdge:bottomEdge, leftEdge:rightEdge]  # 获取被打码部分的子图，用于计算颜色均值
                avg_color = np.average(np.average(part_image, axis=0), axis=0)  # 先计算每行的均值，再计算总的均值
                mosaicImage[topEdge:bottomEdge, leftEdge: rightEdge] = avg_color  # 填色
    return mosaicImage


# 返回调整参数后的文件
def enhance(image_CV, strength):
    height_block = image_CV.shape[0] // 2
    width_block = image_CV.shape[1] // 2
    # 截取并将其转化为可调整的对象
    image_brightPart = Image.fromarray(cv.cvtColor(image_CV[0:height_block - 1, 0:width_block - 1], cv.COLOR_BGR2RGBA))
    image_colorPart = Image.fromarray(
        cv.cvtColor(image_CV[0:height_block - 1, width_block:2 * width_block - 1], cv.COLOR_BGR2RGBA))
    image_contrastPart = Image.fromarray(
        cv.cvtColor(image_CV[height_block:2 * height_block - 1, 0:width_block - 1], cv.COLOR_BGR2RGBA))
    image_sharpPart = Image.fromarray(
        cv.cvtColor(image_CV[height_block:2 * height_block - 1, width_block:2 * width_block - 1], cv.COLOR_BGR2RGBA))
    # 调整并转化为OpenCV 图片对象
    image_brightened = ImageEnhance.Brightness(image_brightPart).enhance(strength[0] + 1)  # 将 Image 对象转换为可调整对象
    image_brightened_CV = cv.cvtColor(np.asarray(image_brightened), cv.COLOR_RGB2BGR)  # 将其转换为OpenCV 对象
    image_colored = ImageEnhance.Color(image_colorPart).enhance(strength[1] + 1)  # 调整亮度后的可调整对象
    image_colored_CV = cv.cvtColor(np.asarray(image_colored), cv.COLOR_RGB2BGR)  # 将其转换为OpenCV 对象
    image_contrasted = ImageEnhance.Contrast(image_contrastPart).enhance(strength[2] + 1)  # 调整亮度后的可调整对象
    image_contrasted_CV = cv.cvtColor(np.asarray(image_contrasted), cv.COLOR_RGB2BGR)  # 将其转换为OpenCV 对象
    image_sharped = ImageEnhance.Sharpness(image_sharpPart).enhance(strength[3] + 1)  # 调整亮度后的可调整对象
    image_sharped_CV = cv.cvtColor(np.asarray(image_sharped), cv.COLOR_RGB2BGR)  # 将其转换为OpenCV 对象
    # 将图片按照对应位置赋值
    image_enhanced = np.zeros((image_CV.shape[0], image_CV.shape[1], 3), np.uint8)
    image_enhanced[0:height_block - 1, 0:width_block - 1] = image_brightened_CV
    image_enhanced[0:height_block - 1, width_block - 1:2 * width_block - 2] = image_colored_CV
```

```
    image_enhanced[height_block - 1:2 * height_block - 2, 0:width_block - 1] = image_contrasted_CV
    image_enhanced[height_block - 1:2 * height_block - 2, width_block - 1:2 * width_block - 2] = image_sharped_CV
    return image_enhanced


def locateFace(image_CV):
    face_cascade = cv.CascadeClassifier(

"C:\\Users\\Steven\\PycharmProjects\\imgProcess\\resource\\haarcascade_frontalface_default.xml")
    grayImg = cv.cvtColor(image_CV, cv.COLOR_BGR2GRAY)
    location_face = face_cascade.detectMultiScale(grayImg, scaleFactor=1.2, minNeighbors=5, minSize=(5, 5))
    return location_face  # 返回的参数是一个元组，分别为左上顶点的x、y、宽度，高度


def autoEnhance(image_CV, lowCut=0.005, highCut=0.005):
    image_enhanced = image_CV.copy()
    height = image_CV.shape[0]
    width = image_CV.shape[1]
    pixelAmount = width * height
    histB = np.histogram(image_CV[:, :, 0], 256, [0, 256])[0]
    histG = np.histogram(image_CV[:, :, 1], 256, [0, 256])[0]
    histR = np.histogram(image_CV[:, :, 2], 256, [0, 256])[0]
    cumB = histB.cumsum()
    cumG = histG.cumsum()
    cumR = histR.cumsum()
    minB = find(cumB, pixelAmount * lowCut)
    minG = find(cumG, pixelAmount * lowCut)
    minR = find(cumR, pixelAmount * lowCut)

    maxB = find(cumB, pixelAmount * (1 - highCut))
    maxG = find(cumG, pixelAmount * (1 - highCut))
    maxR = find(cumR, pixelAmount * (1 - highCut))

    RedMap = linearMap(minR, maxR)
    GreenMap = linearMap(minG, maxG)
    BlueMap = linearMap(minB, maxB)
    for i in range(0, height):
        for j in range(0, width):
            image_enhanced[i, j, 0] = BlueMap[image_CV[i, j, 0]]
            image_enhanced[i, j, 1] = GreenMap[image_CV[i, j, 1]]
            image_enhanced[i, j, 2] = RedMap[image_CV[i, j, 2]]
    return image_enhanced
```

```
def find(tuple_input, limitation):
    此函数被 autoEnhance 函数调用，用于查找元组中最先满足达到 limitation 的序号
    （由于 np.where() 不能完全替代 Matlab 中的 find() 函数，而用列表生成式会有很大的性能损失，所以设计这么个函数）
    for i in range(len(tuple_input)):
        if tuple_input[i] >= limitation:
            return i


def linearMap(low, high):
    """
    该函数被 autoLevel 函数调用，用于计算自动优化过程中的隐射表。
    其本质就是一个带有极化效果的线性映射函数
    :param low: 隐射表下限
    :param high: 隐射表上限
    :return: 某通道的隐射表，根据此表可以计算原颜色在优化后的新图中的色阶值
    """
    mapping = [0] * 256
    for i in range(256):
        if i < low:
            mapping[i] = 0
        elif i > high:
            mapping[i] = 255
        else:
            mapping[i] = np.uint8((i - low) / (high - low) * 255)
    return mapping
```

2. 效果展示代码

Display.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*
from tkinter import *
from tkinter import filedialog
import cv2 as cv
from PIL import Image, ImageTk  # 该库的函数将 OpenCV 的图转换为 TK 中图的格式
import ControlPanel
import ProcessImage as Img

__all__ = ['ShowMenu', 'CV2Tk', 'getImagePath', 'setSavePath', 'WindowBase',
'DragCapture', 'MosaicImage', 'GridImage',
          'AttributeImage', 'MosaicImage_Auto', 'AttributeImage_Auto']
```

```
# 展示主菜单
def ShowMenu(backImage):
    window = Tk()  # 此为根窗口
    window.configure()
    window.title("图像处理小程序")
    # 获取屏幕尺寸以计算布局参数，使窗口居屏幕中央
    width_screen = window.winfo_screenwidth()
    height_screen = window.winfo_screenheight()
    alignstr = '%dx%d+%d+%d' % (800, 450, (width_screen - 800) / 2, (height_screen -
450) / 2)
    window.geometry(alignstr)
    window.resizable(width=False, height=False)  # 设置窗口的长宽均不可变

    # 往窗口上添加按钮
    area_function = Frame(window, width=200)
    Label(window, text="行到水穷处，坐看云起时。", font="楷体",
fg='black').pack(anchor='ne', padx=5, pady=5)
    Button(area_function, text='创建九宫格图', width=18, command=lambda:
ControlPanel.CreateNineGrid()).pack(side='top',

pady=10)
    Button(area_function, text='拖拽以进行截图', width=18, command=lambda:
ControlPanel.Capture()).pack(side='top', pady=10)
    Button(area_function, text='指定区域马赛克', width=18, command=lambda:
ControlPanel.CreateMosaicImg()).pack(side='top',

pady=10)
    Button(area_function, text='面部马赛克', width=18, command=lambda:
ControlPanel.AutoCreateMosaicImg()).pack(side='top',

pady=10)
    Button(area_function, text='修改图像参数', width=18, command=lambda:
ControlPanel.CreateAttributeImg()).pack(side='top',

pady=10)
    Button(area_function, text='自适应优化图像', width=18, command=lambda:
ControlPanel.AutoAttributeImg()).pack(side='top',

pady=10)
    Button(area_function, text='退出程序', width=18, command=lambda:
ControlPanel.Exit()).pack(side='top', pady=10)
    area_function.pack(padx=60, side='left')

    Label(window, width=450, height=450, image=CV2Tk(backImage, 450,
450)).pack(anchor='ne')
```

```
    window.mainloop()  # 进入消息循环


def CV2Tk(image_CV, width, height):
    global image_convert
    image_convert = cv.resize(image_CV, (width, height), interpolation=cv.INTER_AREA)
    image_convert = cv.cvtColor(image_convert, cv.COLOR_BGR2RGBA)  # 转换颜色从 BGR 到
RGBA
    image_convert = Image.fromarray(image_convert)  # 将图像转换成 Image 对象
    image_convert = ImageTk.PhotoImage(image=image_convert)  # 将 image 对象转换为
imageTK 对象
    return image_convert


# 获取读取的文件路径
def getImagePath():
    path_image = filedialog.askopenfilename(title="请选择文件", filetypes=[('图像文件',
['*.jpg', '*.jpeg'])])
    return path_image


# 获取保存文件的目录路径
def setSavePath(title):
    filePath = filedialog.askdirectory(title=title)
    return filePath


class WindowBase:
    def __init__(self, title):
        self.window = Toplevel()
        self.window.title(title)
        # 获取屏幕尺寸以计算布局参数，使窗口居屏幕中央
        screenwidth = self.window.winfo_screenwidth()
        screenheight = self.window.winfo_screenheight()
        alignstr = '%dx%d+%d+%d' % (800, 450, (screenwidth - 800) / 2, (screenheight -
450) / 2)
        self.window.geometry(alignstr)
        self.window.resizable(width=False, height=False)  # 设置窗口的长宽均不可变

    # 为适应画布，需要计算图片应调整至的大小
    @staticmethod
    def calcResize(width_image, height_image, width_canvas, height_canvas):
        if float(width_image / height_image) > float(width_canvas / height_canvas):
            width_resize = width_canvas
            height_resize = int((width_canvas * height_image / width_image))
        else:
```

```
            width_resize = int((height_canvas * width_image / height_image))
            height_resize = height_canvas
        return [width_resize, height_resize]

    # 获取读取的文件路径
    @staticmethod
    def getImagePath():
        path_image = filedialog.askopenfilename(title="请选择图片", filetypes=[('图像文件', ['*.jpg', '*.jpeg'])])
        return path_image

    # 获取保存文件的目录路径
    @staticmethod
    def setSavePath(title):
        filePath = filedialog.askdirectory(title=title)
        return filePath

    def addLabel(self, master, image_CV):
        height_image = image_CV.shape[0]  # 原图的高度
        width_image = image_CV.shape[1]  # 原图的宽度
        [width_resize, height_resize] = self.calcResize(width_image, height_image, 400, 400)
        image_TK = CV2Tk(image_CV, width_resize, height_resize)
        previewImg = Label(master, width=400, height=400, image=image_TK)
        return previewImg


# 拖拽截取图片
class DragCapture(WindowBase):
    def __init__(self):
        super().__init__(title="拖拽以截取图片")
        self.startX = 0
        self.startX = 0
        self.startY = 0
        self.endX = 0
        self.endY = 0
        self.image_temp = None

    def showWindow(self, image_CV):
        # 暂存图，为了能够即时预览与保存
        self.image_temp = image_CV.copy()
        # 添加选项栏
        area_option = Frame(self.window)
        # 添加命名栏
        Label(area_option, text="请为文件命名(不含中文)").pack(side='top', pady=5)
        name = Entry(area_option, width=18)
```

```
        name.pack(side='top', pady=5)
        # 添加预览和重设按钮
        Button(area_option, text="预览更改", width=18,
               command=lambda: self.UpdatePreview(self.image_temp, previewImg, 400,
400)).pack(side='top', pady=5)
        Button(area_option, text="重置为原图", width=18,
               command=lambda: self.ResetPreview(image_CV, preview=previewImg,
width_pre=400,
                                                 height_pre=400)).pack(side='top', pady=5)
        # 添加保存菜单
        area_save = Frame(area_option)
        Button(area_save, text="保存文件-原画质", width=18,
               command=lambda: Img.save(image_CV=self.image_temp,
path_dir=self.setSavePath("保存至"), quality="Origin",
                                        name=name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-高画质", width=18,
               command=lambda: Img.save(image_CV=self.image_temp,
path_dir=self.setSavePath("保存至"), quality="High",
                                        name=name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-中画质", width=18,
               command=lambda: Img.save(image_CV=self.image_temp,
path_dir=self.setSavePath("保存至"), quality="Medium",
                                        name=name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-低画质", width=18,
               command=lambda: Img.save(image_CV=self.image_temp,
path_dir=self.setSavePath("保存至"), quality="Low",
                                        name=name.get())).pack(side='top', pady=5)
        area_save.pack(side='top')
        area_option.pack(side='left', padx=60)

        # 添加动态预览框
        previewImg = self.addLabel(master=self.window, image_CV=image_CV)
        previewImg.bind("<Button-1>", self.OnMouse)
        previewImg.bind("<ButtonRelease-1>", self.ReleaseMouse)
        previewImg.pack(side='right', padx=30)
        self.window.mainloop()

    def OnMouse(self, event):
        self.startX = event.x
        self.startY = event.y
        print(self.startX, self.startY)

    def ReleaseMouse(self, event):
        self.endX = event.x
        self.endY = event.y
        print(self.endX, self.endY)
```

```
    def UpdatePreview(self, image_CV, preview, width_pre, height_pre):
        # 确定原本的缩放方式
        if float(image_CV.shape[1] / image_CV.shape[0]) > 1:
            rate = int(image_CV.shape[1] / 400)
        else:
            rate = int(image_CV.shape[0] / 400)
        boundary = [min(self.startX, self.endX), max(self.startX, self.endX),
min(self.startY, self.endY),
                    max(self.startY, self.endY)]

        self.image_temp = Img.capturePart(image_CV, [i * rate for i in
boundary]).copy()
        [width, height] = self.calcResize(self.image_temp.shape[1],
self.image_temp.shape[0], width_pre, height_pre)
        image_TK = CV2Tk(self.image_temp, width, height)
        preview.configure(image=image_TK)

    def ResetPreview(self, image_CV, preview, width_pre, height_pre):
        self.image_temp = image_CV.copy()
        [width, height] = self.calcResize(self.image_temp.shape[1],
self.image_temp.shape[0], width_pre, height_pre)
        imgTK = CV2Tk(image_CV, width, height)
        preview.configure(image=imgTK)


# 创建马赛克图
class MosaicImage(WindowBase):
    # 功能-创建马赛克图-更新预览窗口的预览图
    def __init__(self):
        super().__init__(title="图片添加马赛克")
        self.image_temp = None

    def showWindow(self, image_CV):
        self.image_temp = image_CV
        # 向Frame 中添加组件
        area_option = Frame(self.window)
        row_x_l = Frame(area_option)
        row_x_r = Frame(area_option)
        row_y_f = Frame(area_option)
        row_y_b = Frame(area_option)
        row_strength = Frame(area_option)
        row_preview = Frame(area_option)
        Label(row_x_l, text="X 左边界:", width=8).pack(side='left')
        val_x_l = Entry(row_x_l, width=10)
        val_x_l.pack(side='left')
```

```
        Label(row_x_r, text="X 右边界:", width=8).pack(side='left')
        val_x_r = Entry(row_x_r, width=10)
        val_x_r.pack(side='left')
        Label(row_y_f, text="Y 上边界:", width=8).pack(side='left')
        val_y_f = Entry(row_y_f, width=10)
        val_y_f.pack(side='left')
        Label(row_y_b, text="Y 下边界:", width=8).pack(side='left')
        val_y_b = Entry(row_y_b, width=10)
        val_y_b.pack(side='left')
        Label(row_strength, text="马赛克强度(像素数):", width=15).pack(side='left')
        val_strength = Entry(row_strength, width=3)
        val_strength.pack(side='left')
        row_x_l.pack(side='top', pady=5)
        row_x_r.pack(side='top', pady=5)
        row_y_f.pack(side='top', pady=5)
        row_y_b.pack(side='top', pady=5)
        row_strength.pack(side='top', pady=5)
        Label(area_option, text="请为文件命名(不含中文)").pack(side='top', pady=5)
        row_name = Entry(area_option, width=18)
        row_name.pack(side='top', pady=5)

        Button(row_preview, text="预览更改", width=8,
               command=lambda: self.UpdatePreview(image_CV, [int(val_x_l.get()),
int(val_x_r.get()), int(val_y_f.get()),
                                                              int(val_y_b.get())],
int(val_strength.get()),
                                                   preview)).pack(side='left', pady=5)
        Button(row_preview, text="重置", width=8,
               command=lambda: self.UpdatePreview(image_CV, None, 0,
preview)).pack(side='left', pady=5)
        row_preview.pack(side='top', pady=5)
        area_save = Frame(area_option)
        Button(area_save, text="保存文件-原画质", width=18,
               command=lambda: Img.save(image_CV=self.image_temp,
path_dir=self.setSavePath("保存至"), quality="Origin",
                                        name=row_name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-高画质", width=18,
               command=lambda: Img.save(image_CV=self.image_temp,
path_dir=self.setSavePath("保存至"), quality="High",
                                        name=row_name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-中画质", width=18,
               command=lambda: Img.save(image_CV=self.image_temp,
path_dir=self.setSavePath("保存至"), quality="Medium",
                                        name=row_name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-低画质", width=18,
               command=lambda: Img.save(image_CV=self.image_temp,
```

```
path_dir=self.setSavePath("保存至"), quality="Low",
                                    name=row_name.get())).pack(side='top', pady=5)
        area_save.pack(side='top')
        area_option.pack(side='left', padx=60)
        # 设置画布参数
        preview = self.addLabel(self.window, image_CV)
        preview.pack(side='right', padx=30)
        self.window.mainloop()

    def UpdatePreview(self, image_CV, boundary, mosaicStrength, preview):
        if boundary is not None:  # boundary is None 说明是更新原图而非展示预览图
            self.image_temp = Img.addMosaic(image_CV=image_CV, boundary=boundary,
strength=mosaicStrength)
        else:
            self.image_temp = image_CV.copy()
        width_resize, height_resize = self.calcResize(image_CV.shape[1],
image_CV.shape[0], 400, 400)
        image_TK = CV2Tk(self.image_temp, width_resize, height_resize)
        preview.configure(image=image_TK)


# 创建九宫格图
class GridImage(WindowBase):
    def __init__(self):
        super().__init__(title="生成九宫格图")
        self.image_CV_List = []
        self.tkList = []

    def showWindow(self, imgCV_List):
        for i in imgCV_List:
            self.image_CV_List.append(i.copy())

        # 添加Frame 组件
        area_option = Frame(self.window)
        Label(area_option, text="请为文件命名(不含中文)").pack(side='top', pady=5)
        area_name = Entry(area_option, width=18)
        area_save = Frame(area_option)
        Button(area_save, text="保存文件-原画质", width=18,
               command=lambda: Img.save(image_CV=self.image_CV_List,
path_dir=self.setSavePath("保存至"), quality="Origin",
                                    name=area_name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-高画质", width=18,
               command=lambda: Img.save(image_CV=self.image_CV_List,
path_dir=self.setSavePath("保存至"), quality="High",
                                    name=area_name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-中画质", width=18,
```

```
            command=lambda: Img.save(image_CV=self.image_CV_List,
path_dir=self.setSavePath("保存至"), quality="Medium",
                                     name=area_name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-低画质", width=18,
               command=lambda: Img.save(image_CV=self.image_CV_List,
path_dir=self.setSavePath("保存至"), quality="Low",
                                     name=area_name.get())).pack(side='top', pady=5)
        area_name.pack(side='top', pady=5)
        area_save.pack(side='top', pady=5)
        area_option.pack(side='left', padx=60)

        # 计算缩放宽度和高度
        [reWidth, reHeight] = self.calcResize(imgCV_List[0].shape[1],
imgCV_List[0].shape[0], 400 // 3, 400 // 3)
        self.tkList = [CV2Tk(i, reWidth, reHeight) for i in self.image_CV_List]
        # 计算绘图所需的的锚点以及其他变量
        edge = 5  # 设定边框为5像素（在绘图区中）
        topEdge = int((400 - (3 * reHeight) - 4 * edge) / 2)  # 上边界宽度

        # 创建画布
        preview = Canvas(self.window, width=400, height=400)
        preview.pack(side='right', padx=30)
        # 绘图
        for row in range(0, 3):
            for col in range(0, 3):
                preview.create_image((col + 1) * edge + col * reWidth, topEdge + (row +
1) * edge + row * reHeight,
                                     image=self.tkList[col + 3 * row], anchor='nw')
        self.window.mainloop()


# 调整图像参数
class AttributeImage(WindowBase):
    def __init__(self):
        super().__init__(title="图像参数调整")
        self.tempImg = None

    def showWindow(self, image_CV):
        self.tempImg = image_CV
        # 添加参数标签栏
        area_attribute = Frame(self.window)
        area_label = Frame(area_attribute)
        Label(area_label, text='亮度', width=4).pack(side='left')
        Label(area_label, text='色度', width=5).pack(side='left')
        Label(area_label, text='对比度', width=5).pack(side='left')
        Label(area_label, text='锐度', width=4).pack(side='left')
```

```
        area_label.pack(side='top')
        # 添加参数滑块区域
        area_scale = Frame(area_attribute)
        val_brightness = Scale(area_scale, from_=2, to=-1, resolution=0.1, width=4,
length=150)
        val_brightness.pack(side='left')
        val_color = Scale(area_scale, from_=2, to=-1, resolution=0.1, width=5,
length=150)
        val_color.pack(side='left')
        val_contrast = Scale(area_scale, from_=2, to=-1, resolution=0.1, width=4,
length=150)
        val_contrast.pack(side='left')
        val_sharpness = Scale(area_scale, from_=2, to=-1, resolution=0.1, width=5,
length=150)
        val_sharpness.pack(side='left')
        area_scale.pack(side='top')
        # 添加功能区域
        Label(area_attribute, text="请为文件命名(不含中文)").pack(side='top', pady=5)
        area_name = Entry(area_attribute, width=18)
        area_name.pack(side='top', pady=5)
        Button(area_attribute, text="效果预览", width=18,
               command=lambda: self.UpdatePreview(image_CV,
[float(val_brightness.get()), float(val_color.get()),
                                                  float(val_contrast.get()),
float(val_sharpness.get())],
                                             resizeWidth, resizeHeight,
preview)).pack(side='top')
        area_save = Frame(area_attribute)
        Button(area_save, text="保存文件-原画质", width=18,
               command=lambda: Img.save(image_CV=self.tempImg,
path_dir=self.setSavePath("保存至"), quality="Origin",
                                        name=area_name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-高画质", width=18,
               command=lambda: Img.save(image_CV=self.tempImg,
path_dir=self.setSavePath("保存至"), quality="High",
                                        name=area_name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-中画质", width=18,
               command=lambda: Img.save(image_CV=self.tempImg,
path_dir=self.setSavePath("保存至"), quality="Medium",
                                        name=area_name.get())).pack(side='top', pady=5)
        Button(area_save, text="保存文件-低画质", width=18,
               command=lambda: Img.save(image_CV=self.tempImg,
path_dir=self.setSavePath("保存至"), quality="Low",
                                        name=area_name.get())).pack(side='top', pady=5)
        area_save.pack(side='top', pady=5)
        area_attribute.pack(side='left', padx=60)
```

```
        # 添加画布
        [resizeWidth, resizeHeight] = self.calcResize(image_CV.shape[1],
image_CV.shape[0], 400, 400)
        preview = self.addLabel(self.window, image_CV=image_CV)
        preview.pack(side='right', padx=30)
        self.window.mainloop()

    # 功能-调整图像参数-更新预览窗口的预览图
    def UpdatePreview(self, image_CV, strength, width, height, preview):
        newPic = Img.enhance(image_CV, strength)
        self.tempImg = newPic.copy()
        image_TK = CV2Tk(self.tempImg, width, height)
        preview.configure(image=image_TK)


# 自动创建马赛克图
class MosaicImage_Auto(WindowBase):
    def __init__(self, axis):
        super().__init__("定位人脸并添加马赛克")
        self.tempMosaicImg = None
        self.facesLocation = axis

    def showWindow(self, image_CV):
        self.tempMosaicImg = image_CV.copy()
        # 向Frame中添加组件
        attribute = Frame(self.window)
        Label(attribute, text="马赛克强度(色块边长):").pack(side='top')
        MosaicStrengthVal = Entry(attribute)
        MosaicStrengthVal.pack(side='top')
        Label(attribute, text="请为文件命名(不含中文):").pack(side='top')
        NameVal = Entry(attribute)
        NameVal.pack(side='top')
        # 添加按钮组件
        Button(attribute, text="显示预览效果", width=18,
               command=lambda: self.UpdatePreview(image_CV=image_CV,
axis=self.facesLocation,

mosaicStrength=int(MosaicStrengthVal.get()), width=width_resize,
                                                  height=height_resize,
preview=previewImg)).pack(side='top',

pady=5)
        Button(attribute, text="显示原图", width=18,
               command=lambda: self.ResetPreview(image_CV, previewImg, 400,
400)).pack(side='top', pady=5)
        Button(attribute, text="保存文件-原画质", width=18,
```

```
            command=lambda: Img.save(image_CV=self.tempMosaicImg,
path_dir=self.setSavePath("另存为"), quality="Origin",
                                     name=NameVal.get())).pack(side='top', pady=5)
        Button(attribute, text="保存文件-高画质", width=18,
            command=lambda: Img.save(image_CV=self.tempMosaicImg,
path_dir=self.setSavePath("另存为"), quality="High",
                                     name=NameVal.get())).pack(side='top', pady=5)
        Button(attribute, text="保存文件-中画质", width=18,
            command=lambda: Img.save(image_CV=self.tempMosaicImg,
path_dir=self.setSavePath("另存为"), quality="Medium",
                                     name=NameVal.get())).pack(side='top', pady=5)
        Button(attribute, text="保存文件-低画质", width=18,
            command=lambda: Img.save(image_CV=self.tempMosaicImg,
path_dir=self.setSavePath("另存为"), quality="Low",
                                     name=NameVal.get())).pack(side='top', pady=5)
        attribute.pack(side=LEFT, padx=60)
        # 设置画布参数
        [width_resize, height_resize] = self.calcResize(image_CV.shape[1],
image_CV.shape[0], 400, 400)
        image_TK = CV2Tk(image_CV, width_resize, height_resize)
        previewImg = Label(self.window, width=400, height=400, image=image_TK)
        previewImg.pack(side='right', padx=30)
        self.window.mainloop()

    def UpdatePreview(self, image_CV, axis, mosaicStrength, width, height, preview):
        mosaicPic = Img.addMosaic(image_CV=image_CV, boundary=[(axis[0])[0],
(axis[0])[0] + (axis[0])[2], (axis[0])[1],
                                                               (axis[0])[1] +
(axis[0])[3]], strength=mosaicStrength)
        self.tempMosaicImg = mosaicPic.copy()
        imgWithTK = CV2Tk(mosaicPic, width, height)
        preview.configure(image=imgWithTK)

    def ResetPreview(self, image_CV, preview, previewWidth, previewHeight):
        self.tempMosaicImg = image_CV
        [width, height] = self.calcResize(image_CV.shape[1], image_CV.shape[0],
previewWidth, previewHeight)
        imgTK = CV2Tk(image_CV, width, height)
        preview.configure(image=imgTK)


# 自动优化参数
class AttributeImage_Auto(WindowBase):
    def __init__(self):
        super().__init__(title="自动优化参数")
        self.tempImg = None
```

```
    def showWindow(self, image_CV):
        self.tempImg = image_CV

        # 添加参数标签栏
        area_function = Frame(self.window)
        Label(area_function, text="请为图片命名:").pack(side='top', pady=5)
        NameVal = Entry(area_function)
        NameVal.pack(side='top', pady=5)
        Button(area_function, text="效果预览", width=18,
               command=lambda: self.UpdatePreview(image_CV, 400, 400,
preview)).pack(side='top', pady=5)
        Button(area_function, text="显示原图", width=18,
               command=lambda: self.ResetPreview(image_CV, 400, 400,
preview)).pack(side='top', pady=5)
        Button(area_function, text="保存文件-原画质", width=18,
               command=lambda: Img.save(self.tempImg, self.setSavePath("另存为"),
quality="Origin",
                                        name=NameVal.get())).pack(side='top', pady=5)
        Button(area_function, text="保存文件-高画质", width=18,
               command=lambda: Img.save(self.tempImg, self.setSavePath("另存为"),
quality="High",
                                        name=NameVal.get())).pack(side='top', pady=5)
        Button(area_function, text="保存文件-中画质", width=18,
               command=lambda: Img.save(self.tempImg, self.setSavePath("另存为"),
quality="Medium",
                                        name=NameVal.get())).pack(side='top', pady=5)
        Button(area_function, text="保存文件-低画质", width=18,
               command=lambda: Img.save(self.tempImg, self.setSavePath("另存为"),
quality="Low",
                                        name=NameVal.get())).pack(side='top', pady=5)
        area_function.pack(side='left', padx=60)

        # 添加画布
        [resizeWidth, resizeHeight] = self.calcResize(image_CV.shape[1],
image_CV.shape[0], 400, 400)
        image_TK = CV2Tk(image_CV, resizeWidth, resizeHeight)
        preview = Label(self.window, width=400, height=400, image=image_TK)
        preview.pack(side='right', padx=30)

        self.window.mainloop()

    # 功能-调整图像参数-更新预览窗口的预览图
    def UpdatePreview(self, image_CV, width, height, preview):
        newPic = Img.autoEnhance(image_CV)
        self.tempImg = newPic
```

```
        [resizeWidth, resizeHeight] = self.calcResize(image_CV.shape[1],
image_CV.shape[0], width, height)
        imgTK = CV2Tk(self.tempImg, resizeWidth, resizeHeight)
        preview.configure(image=imgTK)

    def ResetPreview(self, image_CV, width, height, preview):
        self.tempImg = image_CV
        [resizeWidth, resizeHeight] = self.calcResize(image_CV.shape[1],
image_CV.shape[0], width, height)
        imgTK = CV2Tk(self.tempImg, resizeWidth, resizeHeight)
        preview.configure(image=imgTK)
```

3. 控制代码

ControlPanel.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*
import Display as DISP
import ProcessImage as IMG  # imgProcess 的简称


# 加载菜单页
def Start():
    """
    该函数用于加载菜单页，以开始整个进程
    :return: None
    """
    backImage = IMG.read("resource/Decorate.jpeg")
    DISP.ShowMenu(backImage)


# 创建九宫格图
def CreateNineGrid():
    """
    该函数用于调用 TK 的文件选择，并调用
    :return: OpenCV 格式的 9 张图的列表
    """
    filePath = DISP.getImagePath()
    GridCreator = DISP.GridImage()
    ninePics = IMG.gridToNine(IMG.read(filePath), 3)
    GridCreator.showWindow(ninePics)


def Capture():
    filePath = DISP.getImagePath()
    Captureor = DISP.DragCapture()
```

```
    Captureor.showWindow(IMG.read(filePath))


# 为图片打马赛克
def CreateMosaicImg():
    filePath = DISP.getImagePath()
    MosaicCreator = DISP.MosaicImage()
    MosaicCreator.showWindow(IMG.read(filePath))


def AutoCreateMosaicImg():
    filePath = DISP.getImagePath()
    axis = IMG.locateFace(IMG.read(filePath))
    MosaicCreator = DISP.MosaicImage_Auto(axis=axis)
    MosaicCreator.showWindow(IMG.read(filePath))


# 为图像调整参数
def CreateAttributeImg():
    filePath = DISP.getImagePath()
    AttributeSetor = DISP.AttributeImage()
    AttributeSetor.showWindow(IMG.read(filePath))


def AutoAttributeImg():
    filePath = DISP.getImagePath()
    AttributeSetor = DISP.AttributeImage_Auto()
    AttributeSetor.showWindow(IMG.read(filePath))


def Exit():
    print("您已正常退出，感谢使用")
    exit(0)
```

4. main

```
import ControlPanel

ControlPanel.Start()
```

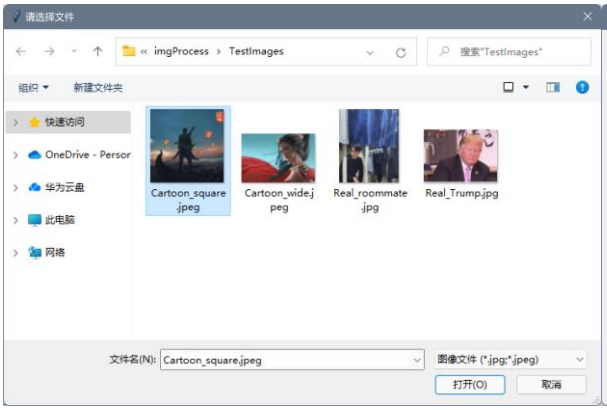
四、运行结果

1. 过程截图

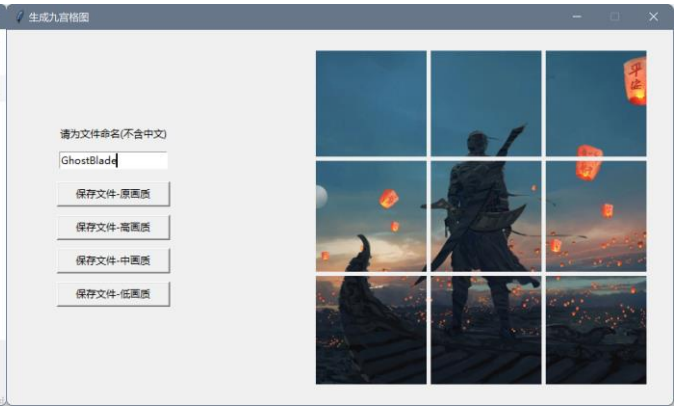
(1) 主页面展示

（图 1.1 主菜单展示）

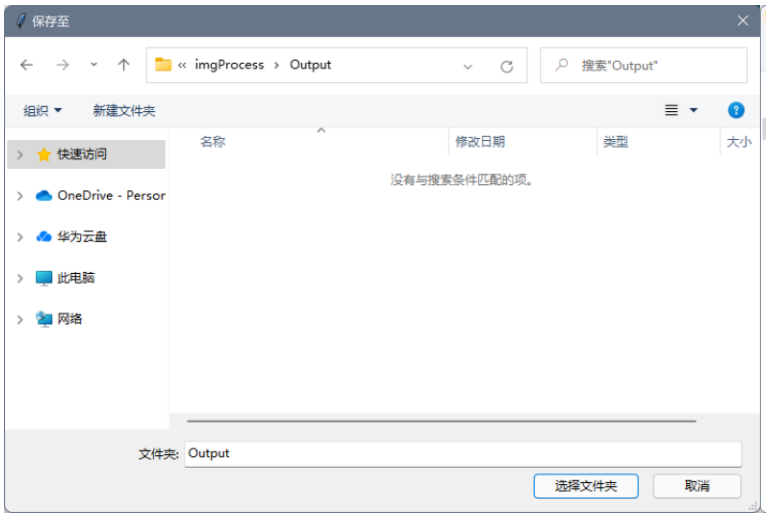
(2) 创建九宫格图展示



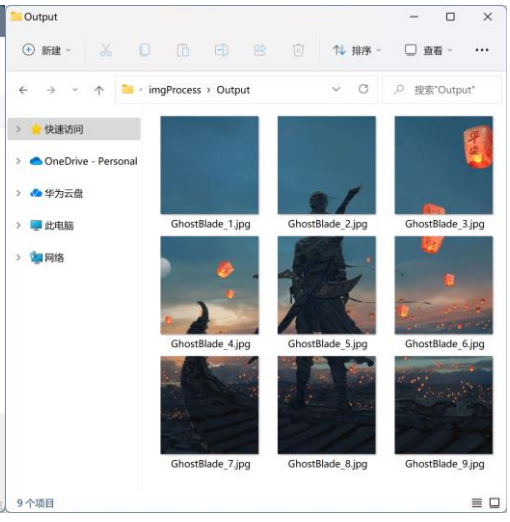
（图 2.1 选择文件）



（图 2.2 九宫格图展示）

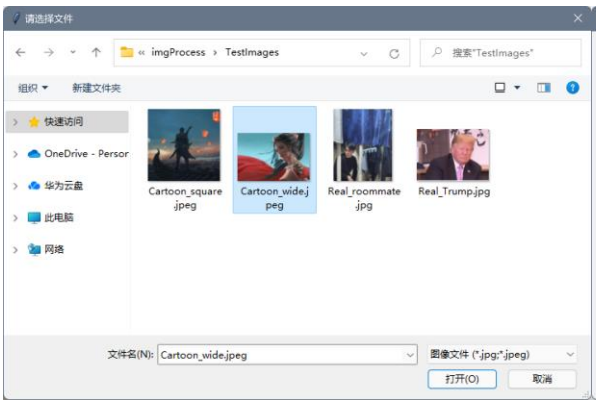


（图 2.3 选择保存路径）



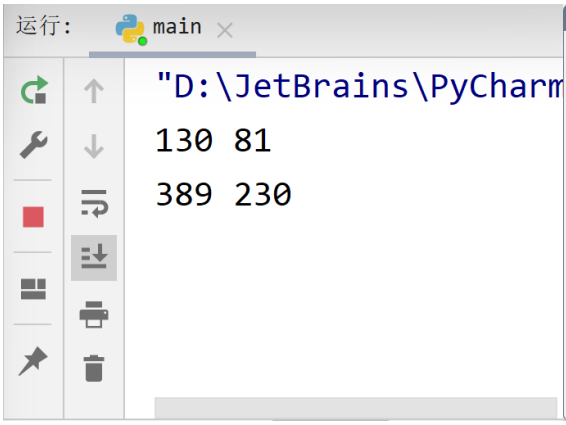
（图 2.4 保存完成）

(3) 拖拽截图展示

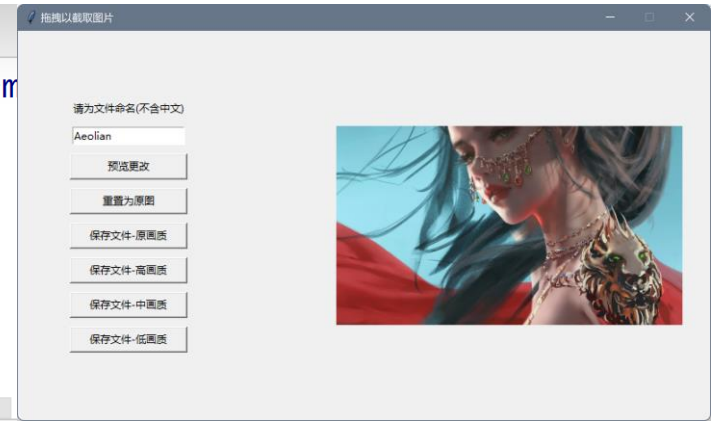


（图3.1 选择文件）

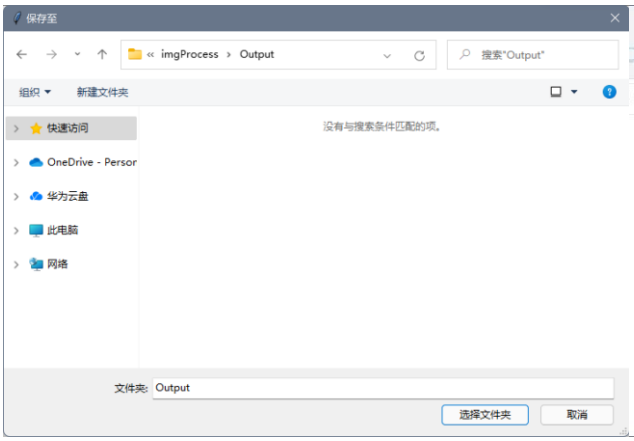
（图3.2 拖拽截图界面展示）



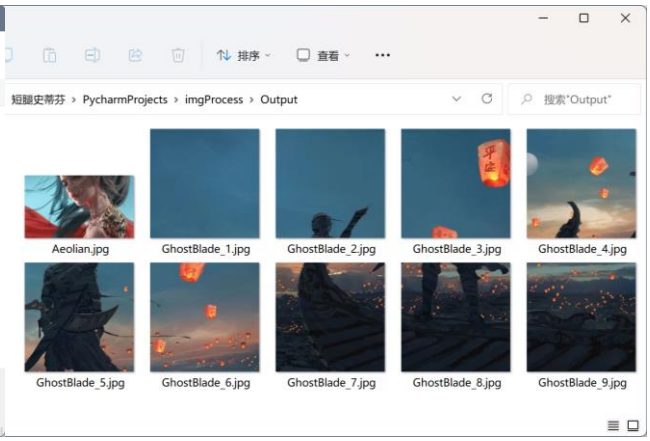
（图3.3 拖拽过程中，控制台会显示坐标）



（图3.4 拖拽完成后，效果预览）

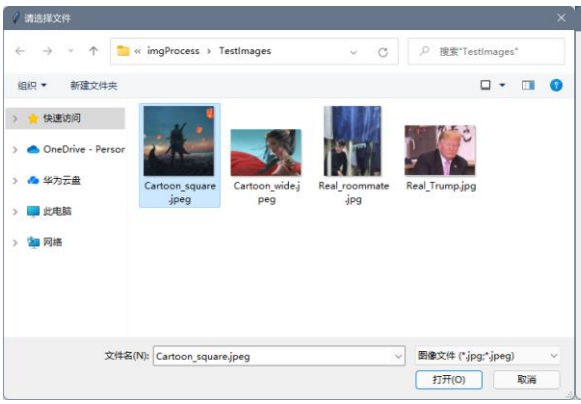


（图3.4 选择保存路径）

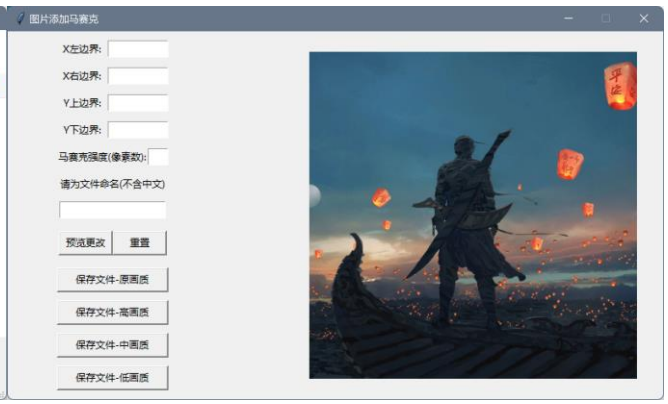


（图3.5 保存完成）

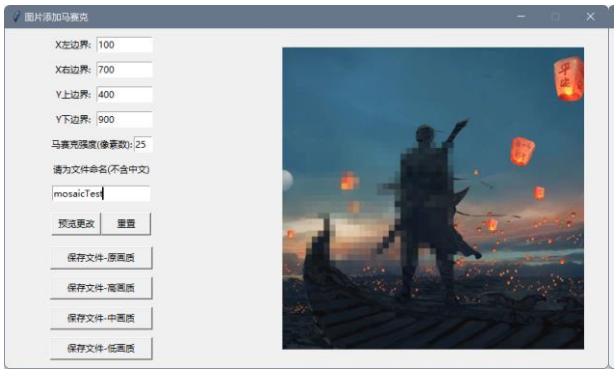
(4) 马赛克展示



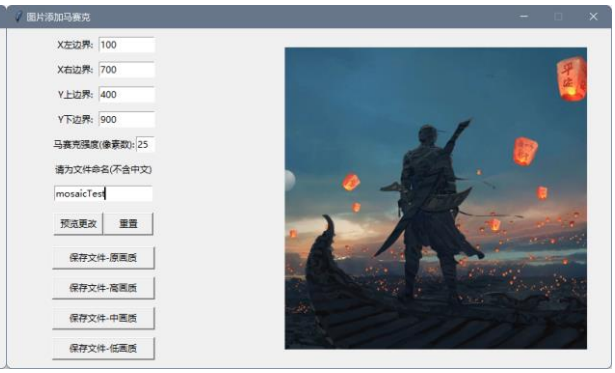
（图 4.1 选择文件）



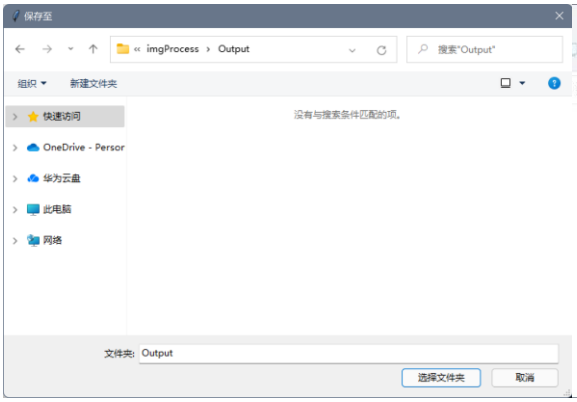
（图 4.2 马赛克界面展示）



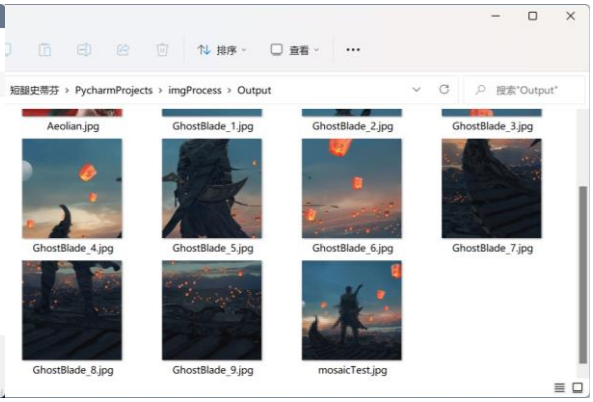
（图 4.3 马赛克效果展示）



（图 4.4 原图效果展示）



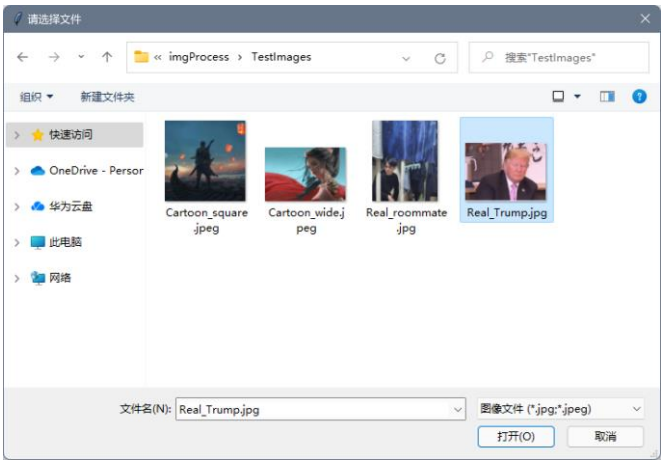
（图 4.5 选择保存路径）



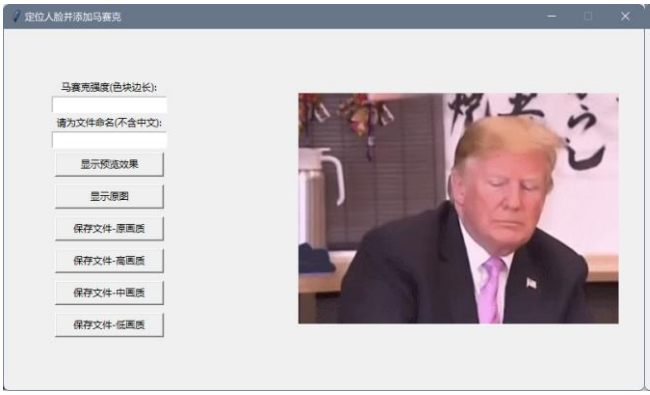
（图 4.6 保存完成）

（图 4.7 附：马赛克处理后的效果图）

(5) 定位面部并添加马赛克

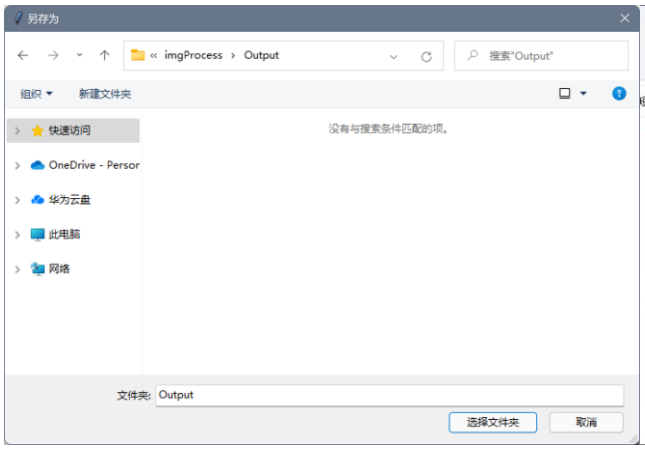


（图5.1 选择文件）

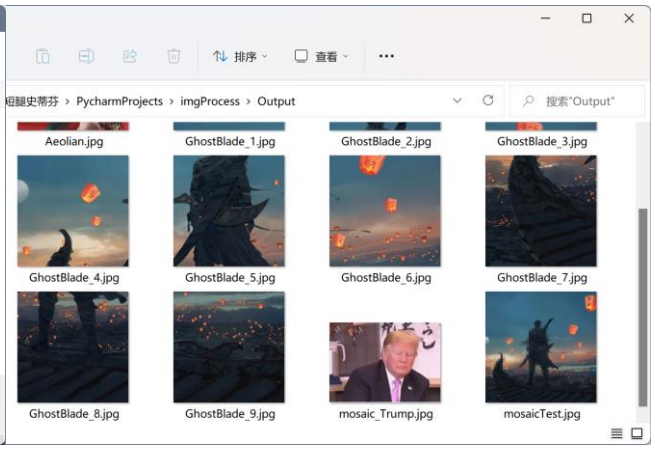


（图5.2 自动定位马赛克界面展示）

（图5.3 马赛克效果展示）

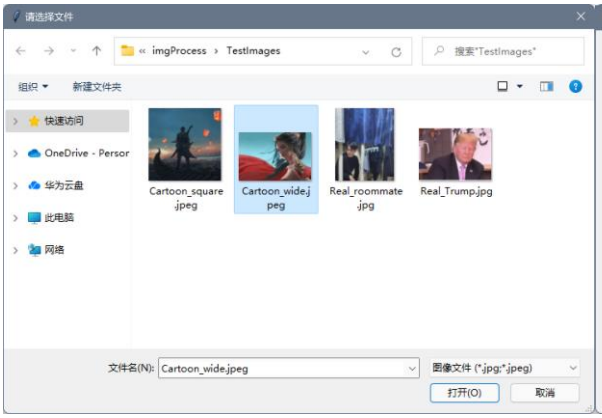


（图5.4 选择保存路径）



（图5.5 保存完成）

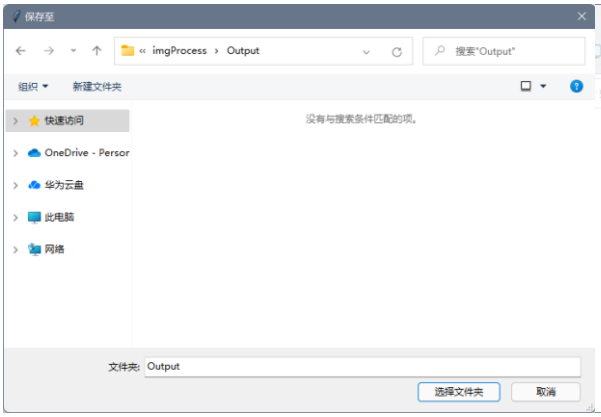
(6) 修改图像参数



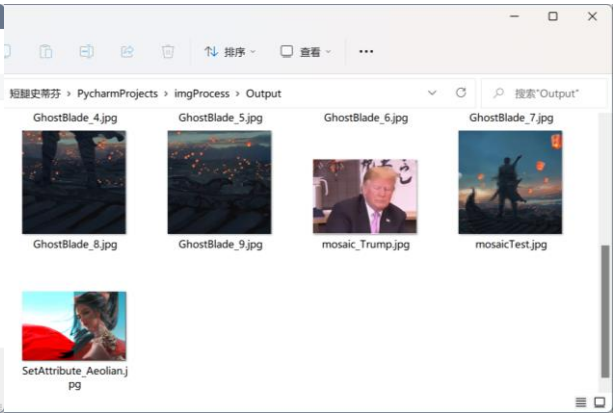
（图 6.1 选择文件）

（图 6.2 设置参数界面展示）

（图 6.3 调整参数后效果展示）

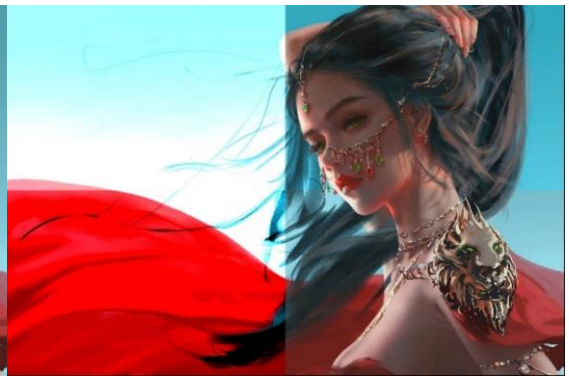


（图 6.4 选择保存路径）



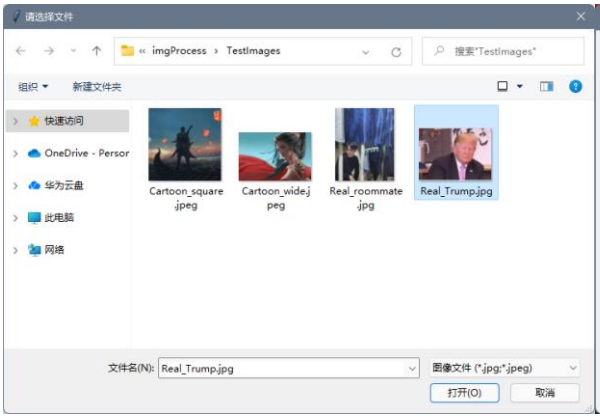
（图 6.5 保存完成）

（图 6.6 附 1：原图）

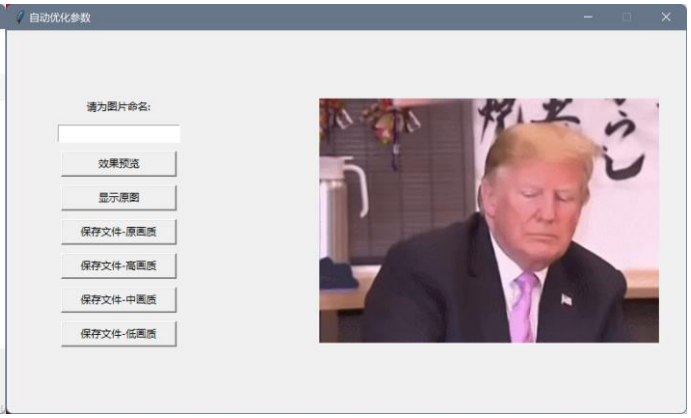


（图 6.7 附 2：处理后的图像）

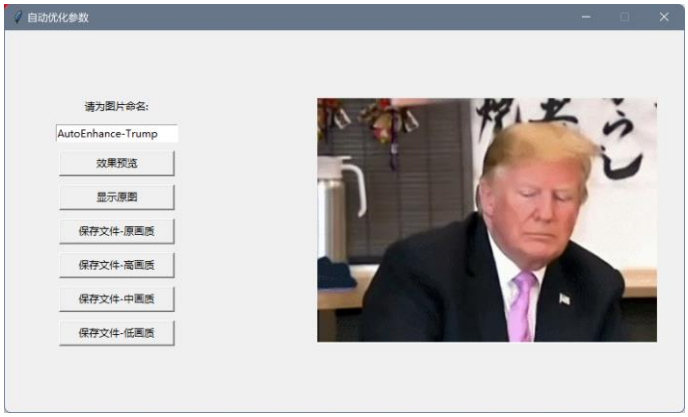
(7) 自适应提升参数



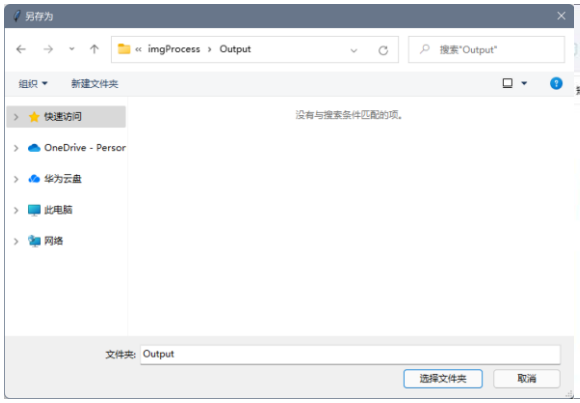
（图 7.1 选择文件）



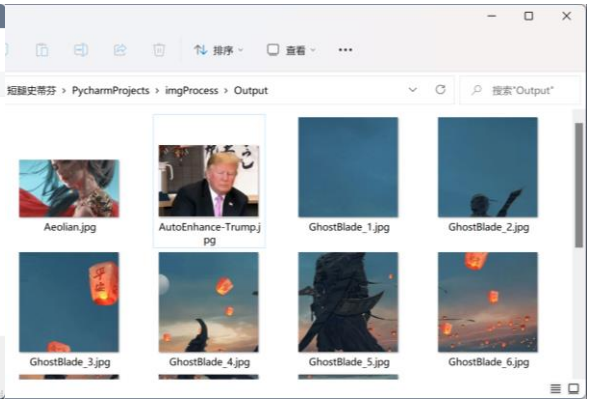
（图 7.2 自动调整参数界面展示）



（图 7.3 调整后效果展示）



（图 7.4 选择保存路径）



（图 7.5 保存完成）

（图 7.6 附 1：原图）

（图 7.5 附 2：增强后的图，优化了偏色问题）

2. 分析与结论

(1) tkinter 的功能分析

tkinter 是一个使用组件进行窗口设计的 Python 工具库，这和大多数编程语言自带的图形库（如 Java 的 awt、swing）类似，乃至提供的组件也大致相同。由于我之前曾在 Java 中尝试使用过 awt 写点东西，所以在学习使用 tkinter 的过程中进展较快。

不过有一点挺令人意外的，tkinter 虽说只占用单个线程，但却能实现一些动态的更新，如更新 label 组件的图片只需要 configure 函数即可。相比之下，在 Java 上实现这一功能要复杂许多（嗯，大概是的）。

(2) 使用面向对象编程的意义

在 Display.py 中，我定义了一个窗口父类，WindowBase。它最大的意义不仅在于能够使其所有子类继承顶级窗口，而是为每个顶级窗口分配了“全局范围”。

在 tkinter 中，若希望使用图片，则需要提前将其声明为全局变量——对于有 6 个功能，且图片需求各不同的该程序来说，这意味着全局变量的声明会很复杂。而在面向对象的编程思想下，每个对象的实例变量就是全局变量，所以在需要修改图片时，只需要在各个子类的__init__中额外声明实例变量即可，这使得代码的逻辑结构更加明了。

(3) 交互

总体来说，交互部分达到了预期效果，将需要指定选项和路径的部分都使用 tk 类的函数包装了起来，使得仅通过鼠标就能完成所有功能的交互。

并且所有窗口都遵循同样的设计语言，所有的二级菜单都是左侧是操作栏右侧是预览框，且所有窗口都位于同一位置，这在美观性和易用性上很有意义，同时也使得界面在布局时更加有矩可循。

(4) 提升鲁棒性的措施

①图形界面取代命令行输入。

以往的交互，每一步的可选项都是让用户输入，但总有可能输入内容不在可选项中。以往的措施无非是使用 default、hasNextInt、try-catch 等方式，但终究只是补救措施。使用图形界面则直接避免了这些问题的发生。

②马赛克边界问题

马赛克实现原理是在方形范围内计算颜色均值，然后为方形的所有像素赋值。但实际中有可能出现需要计算的矩形不完全位于图片中的情况。这种情况其实很好解决，只需要在划定马赛克块时，实际右边界取理论右边界和图像右边界的较小值即可，下边界同理。

③拖动截图问题

拖动截图时，无法提前预知起始点和终止点的位置，所以不能保证计算结果的正负。

解决方法更简单了，加绝对值….

(5) 性能优化措施

主要体现在自适应优化部分，有一段目的是找到累加表中最先达到 limitation 的列表序号，但由于 numpy.where 不能完全替代 Matlab 的 find 函数，若是使用列表生成式的话，只能将整个列表生成完毕才能停止，对于一个 1920×1080 的图片来说这个额外计算量是巨大的，所以自定了一个 find 函数用于满足该功能。

(6) 结论

本程序比较成功，在完全实现了所有要求的基础上保持了较好的鲁棒性。

五、心得体会

实验过程中所遇到的问题，以及解决方式，实验过程中的感悟等

1. 问题

tkinter是新学习的东西，大多数问题就来自于tkinter的应用上，不过好在都已经解决了。

(1) 根窗口与顶级窗口

一开始完全没想到窗口是区分根窗口和顶级窗口的，也因此一开始出现了线程阻塞的问题，即关掉第一个根窗口才能显示第二个根窗口。

(2) 图片的全局声明

对于tkinter组件，图像须声明全局才能使用，这导致了许多图像参数未能传达的问题。

(3) 图像拼接时存在的问题

之前还曾遇到一个问题，就是使用cv2分别处理图像后，再将其拼接起来时会有一道黑边。已经确定像素点对齐了，但是这条黑边仍然存在。只能在拼接时使右下角的图片多向左上方移一个像素，以消除黑边。

2. 感悟

Python说是一门简单的编程语言，但其实也不尽然。它能以较快的方式和非常简洁的方式构建框架、实现功能，但也要因此承受诸多的问题，如可定制化程度不高、难以构架大体量程序。但尽管如此，它仍是一门特性鲜明，受广泛认可的编程语言，尤其是在人工智能方面。今后的学习情况不得而知，但当我需要快速构建、实现功能的时候，Python一定是首选工具。