# 代码说明

# WxPython设计

## 设计需求与思路

设计一个GUI，提供用于打开图像文件、将图像转换为灰度、将图像转化为二进制、执行直方图均衡、模糊、应用拉普拉斯锐化和执行图像分割的按钮来实现响应的功能。

首先导入必要的库并定义一个名为“MyFrame”的自定义框架类，该类继承自wx框架类。在MyFrame类的init方法中，创建了一个面板和一个sizer，并将按钮添加到面板中，并将特定标签和函数绑定到其单击事件。

绑定到按钮单击事件的函数（如OnOpen和OnGray）对所选图像执行相应的图像处理操作。例如，OnGray函数使用cv2库的cvtColor函数将图像转换为灰度。

最后，脚本创建MyFrame类的一个实例并将其显示给用户。



## UI设计

在 \_\_init\_\_中 ，添加菜单栏，以及所有按钮，添加scrolled\_window 来放入图像，以便编写缩放功能。

1. **def** \_\_init\_\_(self, \*args, \*\*kw):
2. super().\_\_init\_\_(\*args, \*\*kw)
3. panel = wx.Panel(self)
4. sizer = wx.BoxSizer(wx.VERTICAL)
5. button\_sizer = wx.BoxSizer(wx.HORIZONTAL)
7. panel.SetSizer(sizer)
8. sizer.Add(button\_sizer, 0, wx.HORIZONTAL)

## 按钮事件绑定

使用wxpython 中的wx.Button来创建按钮，下方代码创建了一个名为 "打开图像" 的 wx.Button 对象，并将它添加到 panel 对象中。这个按钮绑定了一个事件处理器函数 OnOpen，这意味着当用户点击按钮时，OnOpen 函数将被调用。其余按钮写法与下方代码相同。



1. self.Laplacian\_button = wx.Button(panel, label="锐化")
2. self.Laplacian\_button.Bind(wx.EVT\_BUTTON, self.OnLaplacian)
3. button\_sizer.Add(self.Laplacian\_button, 0, wx.HORIZONTAL)

## 菜单栏设计‘’

wx.Menu 是 wxPython 中用于创建菜单的类。这里创建了一个菜单对象 file\_menu。

wx.ID\_OPEN 和 wx.ID\_EXIT 是 wxPython 中的预定义常量，分别表示 "打开" 和 "退出" 命令。Append 方法用于向菜单中添加一个菜单项，并使用三个参数：菜单项的标识符、菜单项的标签和菜单项的状态。在这个例子中，两个菜单项都被添加到 file\_menu 中。

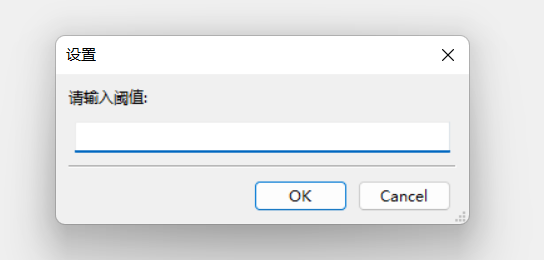
最后，菜单栏对象的 Append 方法用于将 file\_menu 添加到菜单栏中。这意味着在菜单栏中，用户将看到一个名为 "文件" 的菜单，其中包含 "打开" 和 "退出" 两个菜单项。



1. menu\_bar = wx.MenuBar()
2. file\_menu = wx.Menu()
3. open\_item = file\_menu.Append(wx.ID\_OPEN, "打开", "打开图像文件")
4. quit\_item = file\_menu.Append(wx.ID\_EXIT, "退出", "退出程序")
5. menu\_bar.Append(file\_menu, "文件")
6. self.SetMenuBar(menu\_bar)
8. self.Bind(wx.EVT\_MENU, self.OnOpen, open\_item)
9. self.Bind(wx.EVT\_MENU, self.OnQuit, quit\_item)

## 文本框设计

使用TextEntryDialog如下：



1. # 创建一个文本输入对话框
2. dlg = wx.TextEntryDialog(self, '请输入阈值:', '设置')
4. # 显示对话框
5. **if** dlg.ShowModal() == wx.ID\_OK:
6. # 获取文本输入框中的文本
7. text= dlg.GetValue()
8. **try**:
9. self.gamma = int(text)
10. self.OnGamma(event=None)
11. **except**:
12. wx.MessageBox(f"请输入正确的阈值：{text}", "Error", wx.OK | wx.ICON\_ERROR)
13. **return**

# 算法设计与编程

## CVImage 转 WxImage

为了使不同情况下的图像可以展示在GUI中编写SetwxImageData函数用于将 numpy 数组转换为 wxPython 图像。

首先，使用 len(numpyarray.shape) 语句测试数组的维度，并将结果保存在 nasize 变量中。如果数组是二维数组，即 nasize 等于 2，则执行从第 5 行到第 11 行的代码块。如果数组是三维数组，即 nasize 等于 3，则代码执行从第 12 行到第 27 行的代码块。

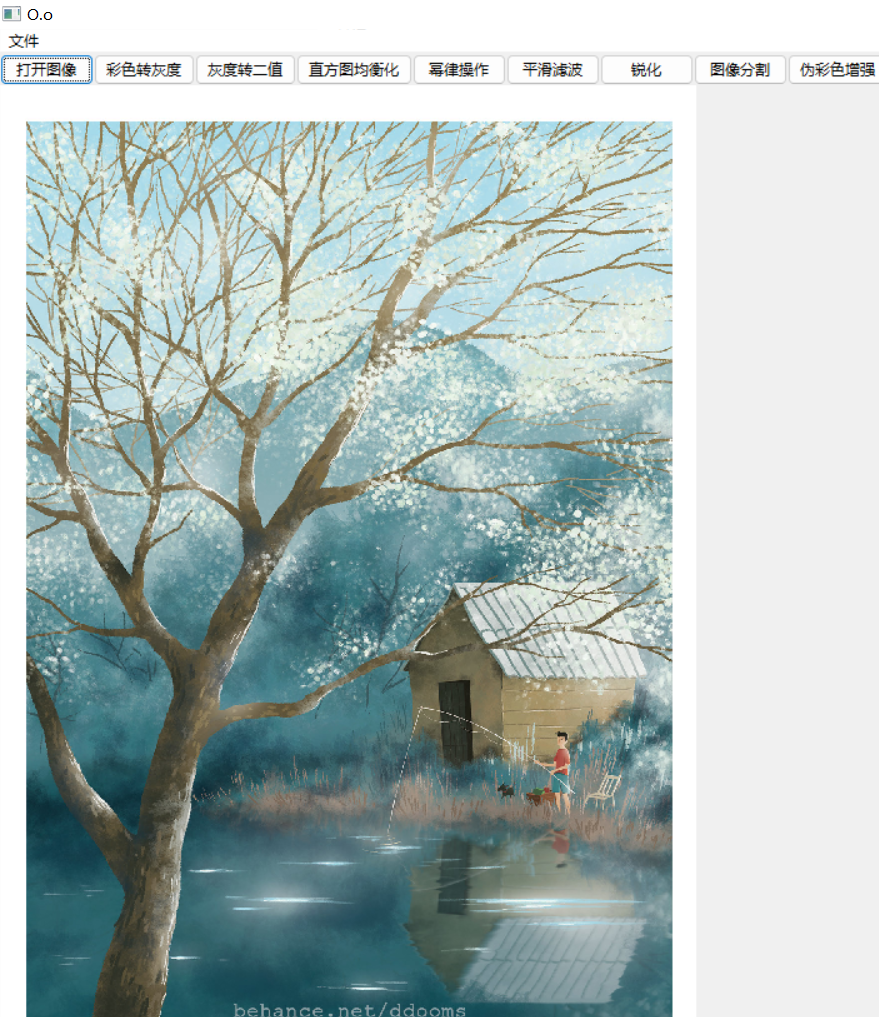
在二维数组的情况下，使用 numpyarray.shape 语句获取数组的高度和宽度，然后创建一个 wx.Image 对象。接下来，使用 np.empty 函数创建一个新数组 Data，并使用三个维度（高度、宽度和 3）。然后将数组 numpyarray 的值复制到 Data 的第一个维度和第二个维度中。最后，使用 wximage.SetData 方法将 Data 数组转换为字节流，并将其设置为 wx.Image 对象的数据。

在三维数组的情况下，使用 numpyarray.shape 语句获取数组的高度、宽度和维度，然后创建一个 wx.Image 对象。接下来，根据数组的维度执行不同的操作。如果维度为 3，即 dim 等于 3，则使用 wximage.SetData 方法将 numpyarray 数组转换

如果维度为 4，即 dim 等于 4，则使用 np.empty 函数创建一个新数组 Data，并使用三个维度（高度、宽度和 3）。然后将数组 numpyarray 的前三个维度（即第一个、第二个和第三个）复制到 Data 数组中。最后，使用 wximage.SetData 方法将 Data 数组转换为字节流，并将其设置为 wx.Image 对象的数据。

在所有图像的情况下，都会返回 wx.Image 对象。

1. **def** SetwxImageData(self,numpyarray):
3. nasize=len(numpyarray.shape)#测试数组的位数
4. **if** nasize==2:
5. Height,Width=numpyarray.shape
6. wximage=wx.Image(Width,Height)
7. Data=np.empty([Height,Width,3],dtype='byte')
8. Data[:,:,0]=numpyarray
9. Data[:,:,1]=numpyarray
10. Data[:,:,2]=numpyarray
11. wximage.SetData(Data.tobytes())
12. **elif** nasize==3:
13. Height,Width,dim=numpyarray.shape
14. wximage=wx.Image(Width,Height)
15. **if** dim==3:#保存的是(R,G,B)
17. wximage.SetData(numpyarray)
18. **elif** dim==4:#某些格式,较少见
19. Data=np.empty([Height,Width,3],dtype='byte')
20. Data[:,:,:]=numpyarray[:,:,0:3]
21. wximage.SetData(Data.tobytes())
22. **elif** dim==1:#其实仍然是灰度图像
23. Data=np.empty([Height,Width,3],dtype='byte')
24. Data[:,:,0]=numpyarray[:,:,0]
25. Data[:,:,1]=numpyarray[:,:,0]
26. Data[:,:,2]=numpyarray[:,:,0]
27. wximage.SetData(Data.tobytes())
28. **return** wximage



## 彩色图转灰度图

常用的灰度转换算法有以下几种：

平均法：将每个像素的红色、绿色和蓝色通道的值相加，再除以 3，得到的结果就是灰度值。

加权平均法：将每个像素的红色、绿色和蓝色通道的值分别乘以不同的权重，再相加，得到的结果就是灰度值。常用的权重是 0.299、0.587、0.114。

最大最小法：将每个像素的红色、绿色和蓝色通道的值的最大值和最小值的平均值作为灰度值。

这里使用了 OpenCV 库中的 cv2.cvtColor 函数将彩色图像转换为灰度图像。

1. self.image\_temp = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)



## 灰度图转二值图

常见的二值化算法有以下几种：

固定阈值法：使用固定的阈值，对灰度图像进行二值化。

自适应阈值法：根据图像的灰度分布情况，自动计算出合适的阈值，对灰度图像进行二值化。

全局阈值法：将整幅图像的所有像素分成两部分，使用不同的阈值进行二值化。

局部阈值法：将图像分成若干个区域，对每个区域分别使用不

这里使用了 OpenCV 库中的 cv2.cvtColor 函数进行二值化。

1. self.image\_temp = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)
2. # 设置阈值，将图片转换为二进制图
3. \_, self.image\_temp = cv2.threshold(self.image\_temp, 127, 255, cv2.THRESH\_BINARY)

## 

## 直方图均衡化

直方图均衡化原理：

计算图像的直方图。

计算累计分布函数（CDF）。CDF 是直方图的积分，表示图像灰度值小于等于当前灰度值的像素数占总像素数的比例。

根据 CDF 计算新的灰度值。新的灰度值是将 CDF 的值乘以图像的总像素数并取整后的结果。

将新的灰度值赋给原图像的每个像素。

1. self.image\_temp = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)
2. # 进行直方图均衡化
3. self.image\_temp = cv2.equalizeHist(self.image\_temp)



## 幂律操作

幂律操作算法：

将图像的灰度值设为 x。

设置幂指数 y。

计算新的灰度值 z = x^y。

将新的灰度值 z 赋给原图像的每个像素。

1. self.image\_temp = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)
3. self.image\_temp = np.uint8(cv2.pow(self.image\_temp / 255.0, self.gamma) \* 255)



## 平滑滤波

平滑滤波是指使用滤波器对图像进行平滑处理，以去除噪声或模糊不清的细节。

平滑滤波的原理是：使用滤波器对图像的每个像素周围的像素进行加权平均。常用的滤波器有均值滤波器、高斯滤波器、中值滤波器等。这里使用blur

1. self.image\_temp = cv2.blur(self.image, (5, 5))



## 图像锐化

图像锐化是指对图像进行滤波，使得图像边缘和细节更加清晰，对比度更高。

图像锐化的原理是：使用滤波器对图像的每个像素与周围像素的差值进行加权平均。常用的滤波器有锐化滤波器、拉普拉斯滤波器等。

1. self.image\_temp = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)
2. kernLaplace = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])  # Laplacian kernel
3. self.image\_temp = cv2.filter2D(self.image\_temp, -1, kernLaplace, borderType=cv2.BORDER\_REFLECT)

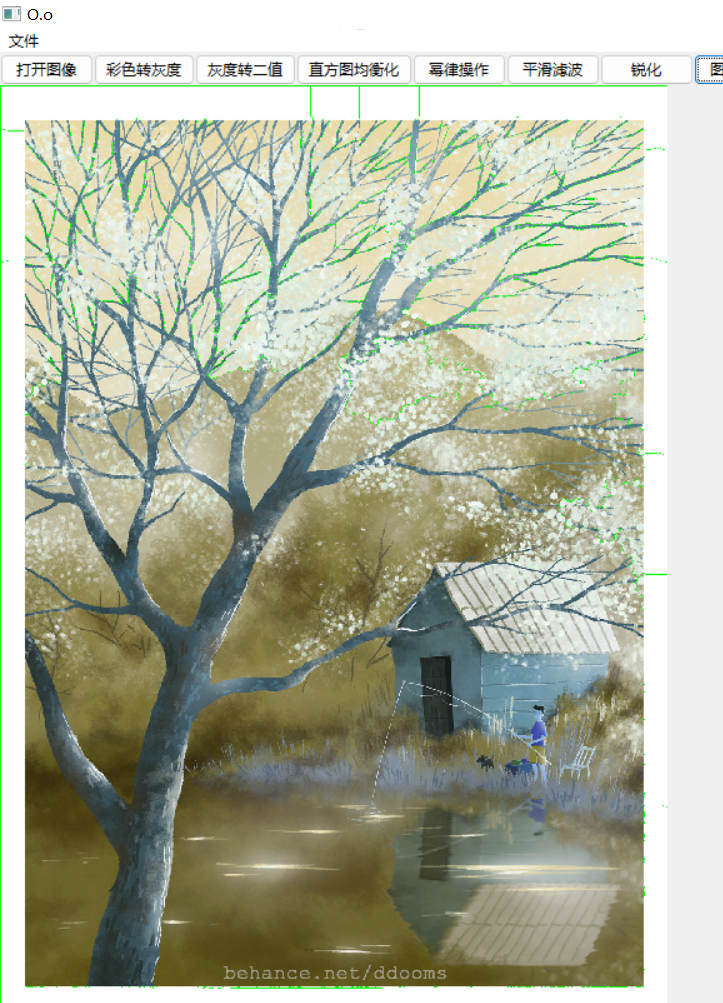


## 图像分割

图像分割是指将图像划分成若干个连通域，使得每个连通域内的像素具有相似的特征。图像分割常用于图像处理中的物体分类、轮廓提取、物体跟踪等任务。

图像分割的原理是：使用不同的算法对图像进行处理，将图像划分成若干个连通域。常用的图像分割算法有阈值分割、联通域分割、分水岭分割等。

1. blurred = cv2.pyrMeanShiftFiltering(self.image\_temp, 10, 100)
3. gray = cv2.cvtColor(blurred, cv2.COLOR\_BGR2GRAY)
4. # 二值化
5. ret, binary = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)
7. kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))
9. nb = cv2.morphologyEx(binary, cv2.MORPH\_OPEN, kernel)
11. sure\_bg = cv2.dilate(nb, kernel)
13. dist = cv2.distanceTransform(sure\_bg, cv2.DIST\_L2, 3)
14. dist\_output = cv2.normalize(dist, 0, 2.0, cv2.NORM\_MINMAX)
15. ret, surface = cv2.threshold(dist, dist.max() \* 0.6, 255, cv2.THRESH\_BINARY)
16. surface\_fg = np.uint8(surface)
17. unkown = cv2.subtract(sure\_bg, surface\_fg)
18. ret, markers = cv2.connectedComponents(surface\_fg)
20. markers = markers + 1
21. markers[unkown == 255] = 0
22. markers = cv2.watershed(self.image\_temp, markers=markers)



## 伪色彩增强

伪色彩增强是指使用计算机算法对图像进行处理，使得图像的对比度和色彩更加鲜艳。

伪色彩增强的原理是：将图像的每个像素按照一定的规则进行转换，使得图像的灰度分布更加均匀。常用的伪色彩增强算法有直方图均衡化、幂律操作等。

伪色彩增强的一个常用应用是将灰度图像转换为伪彩色图像，使得图像看起来更加生动。伪彩色图像是通过将灰度图像的灰度值映射到色彩空间来实现的。

1. self.image\_temp = cv2.cvtColor(self.image, cv2.COLOR\_BGR2GRAY)
3. # Process and display
4. self.image\_temp = cv2.applyColorMap(self.image\_temp, cv2.COLORMAP\_JET)
5. self.image\_temp = cv2.cvtColor(self.image\_temp, cv2.COLOR\_BGR2RGB)

