《数字图像处理》期末项目报告

滕思怡 10190350453
刘雨杰 10194900441

2022.07

目录

1 系统总览和介绍 **1**

2 项目技术栈 **1**

3 图像处理功能原理介绍和实现 **1**

3.1 图像编辑 1
3.1.1 图像翻转 1
3.1.2 图像平移 2
3.1.3 图像旋转 2
3.1.4 图像放缩 2
3.1.5 几何变换 2
3.1.6 图像灰度化 3
3.1.7 图像二值化 3
3.2 图像分析 3
3.2.1 绘制图像直方图 3
3.3 图像边缘检测 4
3.3.1 Roberts 算子 4
3.3.2 Laplacian 算子 5
3.3.3 Sobel 算子 5
3.3.4 LoG 算子 5
3.3.5 Canny 算子 6
3.4 图像平滑 6
3.4.1 时域的平滑 6
3.4.2 频域的平滑 7

3.5 图像锐化 9
3.5.1 空域的锐化 9
3.5.2 频域的锐化 9
3.6 图像形态学 11
3.6.1 膨胀 11
3.6.2 腐蚀 11
3.6.3 开运算 12
3.6.4 闭运算 12
3.7 图像修复 12
3.7.1 添加噪声 12
3.7.2 去除噪声 12
3.8 图像风格迁移 13

4 项目效果展示 19
4.1 图像编辑 19
4.2 图像分析 24
4.3 图像边缘检测 25
4.4 图像平滑与锐化 28
4.5 图像形态学 31
4.6 图像修复（噪声处理） 33
4.7 图像风格迁移 34

1 系统总览和介绍

项目最终完成难度系数：1.5。

在实现课堂基础要求的图像处理功能之外，还通过阅读相关文献、借鉴相关开源代码，实现了任意风格图像和任意内容图像的风格迁移功能，对项目的现实应用有了较为明确的定位。

项目实现功能汇总：

功能分类	具体功能实现
图片上传	可以将图片上传后换存在服务器端
图像编辑	灰度化、二值化、翻转、平移、旋转、放缩
图像分析	绘制灰度直方图、彩色直方图
图像边缘检测	Robert、Sobel、Laplacian、LoG、Canny 算子
图像增强	图像平滑、锐化
图像形态学	腐蚀、膨胀、开运算、闭运算
风格迁移	任意风格和任意内容图像的风格迁移功能

2 项目技术栈

技术栈	
前端	vue.js、npm
python 模块	opencv、numpy、matplotlib、flask、tensorflow 等
环境	linux 系统、python3.8

3 图像处理功能原理介绍和实现

3.1 图像编辑

我们在图像编辑这一部分中，实现了对于图像简单处理的一些功能，具体在下文中描述。

在这一部分的最后，我们将所有图像编辑基本功能归纳为可以使用图像的几何变换功能实现。

3.1.1 图像翻转

图像翻转指的是将图像以某条中轴线为中心进行对换，这里实现了垂直翻转和水平翻转的功能。可以直接使用 OpenCV2 库的 `flip(img, angle)` 函数进行操作，其中：

- `angle=0`，垂直翻转；
- `angle=1`，水平翻转；

- `angle=-1`，对角翻转。

3.1.2 图像平移

图像的平移是在二维平面上的，所以可以抽象为沿着 x 方向 tx 距离，y 方向 ty 距离，可以用移动矩阵来表示：$\begin{bmatrix}1 & 0 & tx\\ 0 & 1 & ty\end{bmatrix}$。我们可以用通过 `numpy.float32()` 构建移动矩阵，并将其赋值给仿射函数 `cv2.warpAffine()` 对图像进行平移操作。

3.1.3 图像旋转

使用 opencv 的 `getRotationMatrix2D((x, y), degree, 1)` 函数实现，其中 x，yr 分别是旋转中心的坐标，$degree$ 是旋转的度数，最后一个参数是缩放因子，设置为 1。

3.1.4 图像放缩

对图像进行放缩有两种方式可以实现，一种是指定尺寸；另一种是给定放缩的比例。

- 指定尺寸：`resize(img, (100, 100))`，直接指定放缩后的尺寸大小；
- 缩放比例：`resize(img, (0,0), fx=2.0, fy=2.0)`，使用放缩比例放缩图片，在项目中使用双线性插值法，`interpolation=cv2.INTER_LINEAR`。

或者也可以使用 `thumbnail()` 函数即可改变图片大小。

3.1.5 几何变换

图像的几何变换 (geometric transformation) 包括平移、旋转、放缩、翻转、前/后向映射、线性内插等等，几何变换不改变像素值，而是改变像素所在的位置。

几何变换不改变像素的值，而只改变像素所在的位置。使用放射矩阵可以对图像进行几何变换。通常使用一个 2×3 的矩阵来实现各种放射变换。其形式如下：

$$A = \begin{bmatrix}a_{00} & a_{01}\\ a_{10} & a_{11}\end{bmatrix}_{2\times 2} B = \begin{bmatrix}b_{00}\\ b_{10}\end{bmatrix}_{2\times 1} M = \begin{bmatrix}A & B\end{bmatrix} = \begin{bmatrix}a_{00} & a_{01} & b_{00}\\ a_{10} & a_{11} & b_{10}\end{bmatrix}_{2\times 3}$$

利用变换矩阵 M，对于任何一个像素 $X = \begin{bmatrix}x\\ y\end{bmatrix}$，有：

$$T = A \cdot \begin{bmatrix}x\\ y\end{bmatrix} + B = M \cdot [x, y, 1]^T$$

特别地，对于旋转操作，有：

$$M = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \end{bmatrix}$$

通过 `cv2.getAffineTransform` 可以根据三个瞄点得到对应的变换矩阵。通过 `cv2.warpAffine` 可以实现各种放射变换。

3.1.6 图像灰度化

灰度化处理就是将一幅彩色图像转化为灰度图像的过程。彩色图像分为 R，G，B 三个分量，分别显示出红绿蓝等各种颜色，灰度化就是使彩色图像的 R，G，B 分量相等的过程。灰度值大的像素点比较亮（像素颜色值最大为 255，为白色），反之比较暗（像素颜色值最小为 0，为黑色）。

在项目中我们直接使用 OpenCV 中的 `cvtColor(img,cv.COLOR_RGB2GRAY)` 语句即可转成灰度图像，随后使用`calcHist()`内置函数进行画灰度图像直方图。

3.1.7 图像二值化

图像二值化就是将图像上的像素点的灰度值设置为 0 或 255 ，也就是将整个图像呈现出明显的黑白效果的过程。在数字图像处理中，图像的二值化可以使图像中数据量大为减少，从而能凸显出目标的轮廓。

在项目中，我们首先将图像转换成灰度图像，再新建一个数组作为映射表。如果色彩深度大于阈值则存 1，小于阈值则存 0。接着再使用 `im.point()` 函数，将映射表传递过去并设置图像模式为二值图像，PIL 库就会逐个遍历像素，将灰度图像的灰度值在数组中映射成 0 或 1，最后得到的图像便是二值图像。

我们尝试了不同阀值所带来的效果，最终将灰度值阈值设置为 110 ，相对而言可以取得较好效果。

3.2 图像分析

在图像分析这一功能的实现中，我们主要实现了图像的直方图，以方便对图像内容进行分析。同时我们实现了直方图均衡化，以增强图像对比度的方法。

3.2.1 绘制图像直方图

为了可以统一化地绘制直方图，我们绘制的步骤是：

1 将输入的图片转换成灰度图

2 统计图片中每一个灰度级出现的次数

3 利用 matplotlib.pyplot 中的 hist 函数绘制图表

直方图均衡化中，我们针对两种不同的图片实现直方图均衡化：

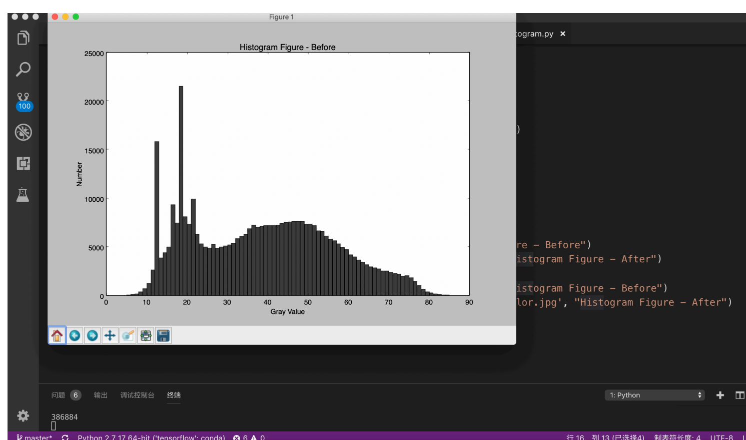


图 1: 绘制直方图效果图

灰度图直方图均衡化。 灰度图的直方图比较方便，我们直接利用 cv2 的 `equalizeHist` 函数，对输入的灰度图进行直方图均衡化后输出即可。

彩色图片直方图均衡化。 彩色图片直方图均衡化，对于简单的 RGB 彩色图像，如果直接分离三种颜色分别直方图均衡化是不会产生效果的。因为直方图的均衡化涉及强度图像的值，而不是颜色分量。因此，对于简单的 RGB 彩色图像，为了不干扰图像的颜色平衡，我们先将图像的颜色空间从 RGB 转换为 YCbCr 颜色空间，该空间将灰度值与颜色分量分离。之后再采用 cv2 的 `equalizeHist` 函数，将图片进行直方图均衡化，之后将图片转变回 RGB 彩色图像，输出图片。

3.3 图像边缘检测

图像边缘是图像最基本的特征，所谓边缘是指图像局部特性的不连续性。灰度或结构等信息的突变处称之为边缘。例如，灰度级的突变、颜色的突变、纹理结构的突变等。边缘是一个区域的结束，也是另一个区域的开始，利用该特征可以分割图像。

在图像边缘检测的模块，我们实现了多种不同的差分算子进行了边缘检测的实现，具体原理在下文中详细描述。

3.3.1 Roberts 算子

Roberts 算子又称为交叉微分算法，它是基于交叉差分的梯度算法，通过局部差分计算检测边缘线条。

Roberts 算子的模版为

$$d_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, d_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

，从模版就可以看出当图像边缘接近于正 45 度或负 45 度时，Roberts 算子处理效果更理想。

在项目实现中，我们可以用 `numpy` 定义模板，再调用 `cv2.filter2D()` 函数实现边缘提取。

3.3.2 Laplacian 算子

这部分比较方便处理，只需要将图片转变成灰度图片，然后直接利用 cv2 的 Laplacian 进行计算。

Laplacian 算子的模版为

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

在项目实现中，同样可以用 `numpy` 定义模板，再调用 `cv2.filter2D()` 函数实现边缘提取，也可以直接用 `cv2.Laplacian()` 。

3.3.3 Sobel 算子

Sobel 算子是一种用于边缘检测的离散微分算子，它结合了高斯平滑和微分求导。该算子用于计算图像明暗程度近似值，根据图像边缘旁边明暗程度把该区域内超过某个数的特定点记为边缘。

Roberts 算子的模版为

$$d_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, d_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

,

与 Laplacian 不同的是，Sobel 还涉及到差分的方向问题。我们分别对图片计算了 x 方向的梯度和 y 方向的梯度，之后对这两部分的计算进行了合并得到了最后的边缘检测结果。

在项目实现中，同样可以用 `numpy` 定义模板，再调用 `cv2.filter2D()` 函数实现边缘提取，也可以直接用 `cv2.sobel()` 。

3.3.4 LoG 算子

LOG 边缘检测算子是 David Courtnay Marr 和 Ellen Hildreth 在 1980 年共同提出的，也称为 Marr & Hildreth 算子，它根据图像的信噪比来求检测边缘的最优滤波器。该算法首先对图像做高斯滤波，然后再求其拉普拉斯二阶导数，根据二阶导数的过零点来检测图像的边界，即通过检测滤波结果的零交叉来获得图像或物体的边缘。

LOG 算子实际上是把 Gauss 滤波和 Laplacian 滤波结合了起来，先平滑掉噪声，再进行边缘检测。

常见的 LoG 算子是 5×5 的模板：

$$\begin{bmatrix} -2 & -4 & -4 & -4 & -2 \\ -4 & 0 & 8 & 0 & -4 \\ -4 & 8 & 24 & 8 & -4 \\ -4 & 0 & 8 & 0 & -4 \\ -2 & -4 & -4 & -4 & -2 \end{bmatrix}.$$

所以在实验代码实现中，我们首先使用 `cv2.GaussianBlur()` 再用 `cv2.Laplacian()` 即可。

3.3.5 Canny 算子

首先，由于 Canny 只能处理灰度图，所以将读取的图像转成灰度图。Canny 算子涉及到两个阀值，通过查阅相关的资料了解到，较大的阈值用于检测图像中明显的边缘，但一般情况下检测的效果不会那么完美，边缘检测出来会是断断续续的。所以这时候我们需要用较小的第一个阈值用于将这些间断的边缘连接起来。

在项目实现中，我们可以用 `cv2.canny(img，minVal，maxVal)` 来用 canny 算子进行边缘检测，第一个参数是输入图像，第二个和第三个参数是 $minVal$ 和 $maxVal$，即边缘检测的两个灰度值的阈值。

3.4 图像平滑

每一幅图像都包含某种程度的噪声，噪声可以理解为由一种或者多种原因造成的灰度值的随机变化，如由光子通量的随机性造成的噪声等等。而图像平滑技术或者是图像滤波技术就是用来处理图像上的噪声。平滑可以抑制高频成分，但也会让图像变得模糊。

3.4.1 时域的平滑

局部平均 局部平均也称为均值滤波。原理是通过对一个 $n \times n$ 的像素点构成的矩阵中的信息取平均值，用这个平均值来替代矩阵中心位置的像素信息。可以直接使用 cv2 中自带的 blur 函数来实现这一功能。

例如 2D 卷积就是一种局部平均，OpenCV 中用 `cv2.filter2D()`实现卷积操作，比如我们的核是下面这样（3×3 区域像素的和除以 9）：

$$M = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

缺陷：均值滤波本身存在着固有的缺陷，即它不能很好地保护图像细节，在图像去噪的同时也破坏了图像的细节部分，从而使图像变得模糊，不能很好地去除噪声点，特别是椒盐噪声。

中值滤波 与均值滤波的原理类似，只不过是把像素点的中值而不是均值替代矩阵中心位置的像素信息。相比均值滤波，对椒盐噪声的平滑效果更好，且模糊程度有所减轻。

加权平均 与均值滤波的原理类似，只不过是有一个与矩阵相同大小的模版，每次通过模版给领域像素加权后进行平均，并且以这个加权平均值代替矩阵中心的像素值。换句话说，均值滤波是加权系数全为 1 的加权平均。

多帧平均 多帧平均的方法可用于图像的去噪。实验的原理十分简单，就是通过对同一个景象多张图片的拍摄，将所有图片中每一个像素点信息均用平均值法处理，在最终输出的结果中，用多张图片的平均值来替代每一个像素点的信息即可完成去噪。

3.4.2 频域的平滑

在一幅图像中，其低频成分对应者图像变化缓慢的部分，对应着图像大致的相貌和轮廓。而其高频成分则对应着图像变化剧烈的部分，对应着图像的细节（图像的噪声也属于高频成分）。低频滤波器，顾名思义，就是过滤掉或者大幅度衰减图像的高频成分，让图像的低频成分通过。低频滤波器可以平滑图像，去除图像的噪声。而与此相反的高频滤波器，则是过滤低频成分，通过高频成分，可以达到锐化图像的目的，这将在下一节中详细介绍。

理想低通滤波器的滤波非常尖锐，而高斯低通滤波器的滤波则非常平滑。Butterworth 低通滤波器则介于两者之间，当 Butterworth 低通滤波器的阶数较高时，接近于理想低通滤波器，阶数较低时，则接近于高斯低通滤波器。

在这三种低通滤波器的表达式中，我们都用 D_0 来表示其截止频率。$D(u,v)$ 表示距离频率矩形中心的距离。在我们的项目中，我们实现了上述提到的三种低通滤波器的图像平滑功能。

理想低通滤波 理想低通滤波器在以原点为圆心、D_0 为半径的园内，通过所有的频率，而在圆外截断所有的频率。函数如下：

$$H(u,v) = \begin{cases} 1, D(u,v) \leq D_0 \\ 0, D(u,v) > D_0 \end{cases}$$

特点：

图像边缘模糊；

随着滤波器截止频率的增加，图像边缘模糊程度减少；

出现振铃效应，边缘渐变部分出现灰度值强弱震荡和加边效应。

巴特沃斯低通滤波 函数表达式如下，其中 n 称为 Butterworth 低通滤波器的阶数：

$$H(u,v) = \frac{1}{1+(D(u,v)/D_0)^{2n}}$$

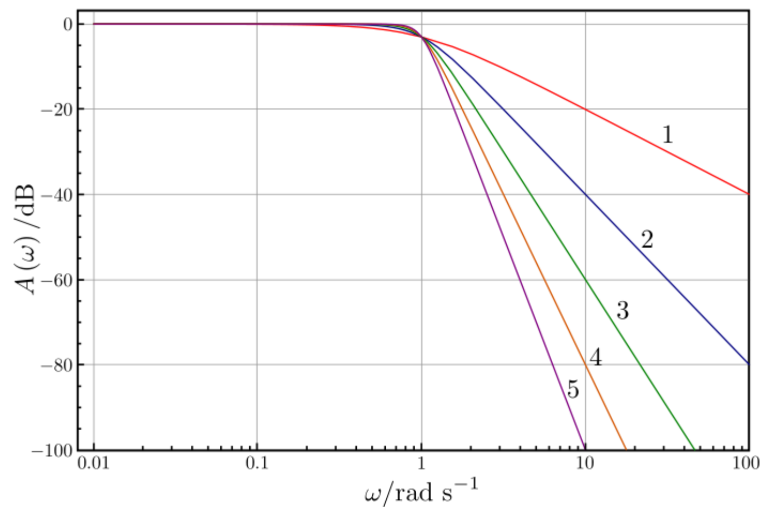


图 2: 巴特沃斯低通滤波器的频率响应曲线

特点:

在高频与低频之间连续衰减，不像理想低通滤波器那样陡峭和具有明显的不连续性；对噪声的平滑效果不如理想低通滤波器；

相较于理想低通滤波器，图像边缘模糊程度降低；

无明显振铃效应；

截止频率越大，图像越清晰；滤波器阶数越大，平滑效果越好，但运算量也越大。

高斯低通滤波 函数表达式如下:

$$H(u,v) = e^{\frac{-D^2(u,v)}{2D_0^2}}$$

其中 D_0 为截止频率。

高斯滤波器的过度特性非常平坦，因此不会产生振铃现象。

指数低通滤波 函数如下:

$$H(u,v) = e^{-[\frac{D(u,v)}{D_0}]^n}$$

特点:

在抑制噪声的同时，图像中边缘的模糊程度比巴特沃斯低通滤波器大；

无明显振铃效应（边缘抖动现象）；

截止频率越大，图像越清晰；滤波器阶数越大，平滑效果越好，但运算量也越大。

梯形低通滤波 函数如下：

$$H(u,v)=\begin{cases}1, D(u,v)<D_0\\ \frac{D(u,v)-D_1}{D_0-D_1}, D_0\le D(u,v)\le D_1\\ 0, D(u,v)>D_1\end{cases}$$

其中 D_0, D_1 均为规定值。

特点：

处理效果介于理想低通滤波器和指数低通滤波器之间，处理后的图像出现一定模糊和振铃效应；

$D_1>D_0>0$，当 D_0 不变，D_1 增大，图像越清晰；当 D_1 不变，D_0 增大，振铃效应越明显。

3.5 图像锐化

图像锐化 (image sharpening) 与图像平滑是作用相反的操作。它是通过加强图像的轮廓，增强图像的边缘及灰度跳变的部分，使图像变得清晰。

3.5.1 空域的锐化

微分法 在本项目中分别实现了采用 Robert,Laplacian,Sobel 以及 Prewitt 算子的微分法图像锐化，在前文已经对这些算子的实现做了一定的介绍，此处就不再赘述。

反锐化掩模法 将原图像通过反锐化掩模进行模糊预处理 (相当于采用低通滤波) 后与原图逐点做差值运算，然后乘上一个修正因子再与原图求和，以达到提高图像中高频成分、增强图像轮廓的目的.

反锐化掩模技术最早是应用于摄影技术中，以增强图像的边缘和细节。光学上的操作方法是将聚焦的正片和散焦的负片在底片上进行叠加，结果是增强了正片高频成份，从而增强了轮廓。散焦的负片相当于 “模糊” 模板（掩模），它与锐化的作用正好相反，因此，该方法被称为反锐化掩模法。

3.5.2 频域的锐化

在本项目中，与图像的平滑一样，也实现了前三种高通滤波的图像锐化。

理想高通滤波 与上述理想低通滤波刚好相反，以截止频率为 D0 为半径的圆内所有频率分量完全衰减，圆外所有频率分量无损通过。理想高通滤波器处理的图像中出现振铃效应（边缘有抖动现象）。

函数如下：

$$H(u,v)=\begin{cases}0, D(u,v)\le D_0\\ 1, D(u,v)>D_0\end{cases}$$

特点：

图像边缘、线性特征、细节增强；

随着滤波器截止频率增加，图像边缘、细节减少；

出现振铃效应。

巴特沃斯高通滤波 函数如下：

$$H(u,v)=\frac{1}{1+(D_0/D(u,v))^{2n}}$$

特点：

相较于理想高通滤波器，图像锐化效果更好，图像边缘、线性特征、细节突出显示；

随着截止频率的增加，图像中细节减少；滤波器阶数越大，锐化效果越好，但运算量也越大。

高斯高通滤波器 函数如下：

$$H(u,v)=1-e^{\frac{-D^2(u,v)}{2D_0^2}}$$

特点：

相较于巴特沃斯高通滤波器，衰减更慢，对噪声抑制作用较好，无明显振铃效应;

与巴特沃斯高通滤波器的处理效果相当，图像中的边缘、线性特征、细节突出显示；

随着截止频率的增加，图像中细节减少；滤波器阶数越大，锐化效果越好，但运算量也越大。

指数高通滤波器 函数如下：

$$H(u,v)=e^{-[\frac{D_0}{D(u,v)}]^n}$$

特点：

指数高通滤波器比巴特沃斯高通滤波器处理效果差些，处理后的图像振铃效应（边缘抖动现象）不明显。

随着截止频率的增加，图像中细节减少；滤波器阶数越大，锐化效果越好，但运算量也越大。

梯形高通滤波器 函数如下：

$$H(u,v)=\begin{cases}0, D(u,v)<D_1\\ \frac{D(u,v)-D_1}{D_0-D_1}, D_1\le D(u,v)\le D_0\\ 1, D(u,v)>D_0\end{cases}$$

特点：

梯形滤波器会产生一定的振铃效应（轻微抖动现象），但因计算简单经常被使用。

处理效果介于理想高通滤波器和指数高通滤波器之间，处理后的图像出现一定模糊和振铃效应。$D_0>D_1>0$，当 D_0 不变，D_1 增大，振铃效应越明显；当 D_1 不变，D_0 增大，图像细节减少。

3.6 图像形态学

形态学，即数学形态学 (Mathematical Morphology)，在图像处理中的主要应用是提取图像中对于表达和描绘区域形状有意义的图像分量，使后续的识别工作能够抓住目标对象最为本质的形状特征，如像素边界和连通区域等。二值图像的基本形态学运算包括腐蚀、膨胀、开操作和闭操作。

膨胀和腐蚀是形态学图像处理的两个基本运算，膨胀可以使图像扩大（如字体图像加粗）；腐蚀可以使图像缩小（如消除图像中不重要的细节部分）。

在我们的项目中，分别实现了膨胀、腐蚀、开运算与闭运算四种图像形态学方法，对输入的图像能够作出相应的处理。

3.6.1 膨胀

使用结构形态元 B 处理图像 A，膨胀的计算公式为：

$$A\oplus B=\{z|((\hat{B})_z\cap A\neq\emptyset)\}$$

预先定义好结构元，可以使用 `cv2.dilate()` 实现该运算。

3.6.2 腐蚀

使用结构形态元 B 处理图像 A，腐蚀的计算公式为：

$$A\ominus B=\{z|(B)_z\subseteq A\}$$

可以使用 `cv2.erode()` 实现该功能。

3.6.3 开运算

开运算是通过先对图像腐蚀再膨胀实现，能够排除小团块物体 (假设物体较背景明亮)，开运算的结果删除了不能包含结构元素的对象区域，平滑了对象的轮廓，断开了狭窄的连接，去掉了细小的突出部分。可以使用 `cv2.open()` 实现该运算。

3.6.4 闭运算

闭运算在数学上是先膨胀再腐蚀的结果，能够排除小型黑洞 (黑色区域)，能够平滑对象的轮廓，但是与开运算不同的是闭运算一般会将狭窄的缺口连接起来形成细长的弯口，并填充比结构元素小的洞。可以使用 `cv2.close()` 实现该运算。

3.7 图像修复

3.7.1 添加噪声

本项目实现了对选定图片添加高斯噪声和椒盐噪声的功能，相关参数可以在前端界面进行设定。

高斯噪声 高斯噪声（Gaussian noise）是一种具有正态分布（也称作高斯分布）概率密度函数的噪声。换句话说，高斯噪声的值遵循高斯分布或者它在各个频率分量上的能量具有高斯分布。它被极其普遍地应用为用以产生加成性高斯白噪声（AWGN）的迭代白噪声。

对于每个输入像素，我们可以通过与符合高斯分布的随机数相加, 得到输出像素；获得一个符合高斯分布的随机数有好几种方法，比如最基本的一个方法是使用标准的正态累积分布函数的反函数，除此之外还有 Box-Muller 变换、ziggurat 算法等更加高效的方法。python 的 random 库提供了产生高斯随机数的方法：`random.gauss(mu, sigma)`

椒盐噪声 椒盐噪声的添加较为简单，在获取了阈值之后，只需通过遍历图像中的每个像素点，与设定阈值比较从而添加椒盐噪声即可。

3.7.2 去除噪声

滤波类算法 BM3D 是一种传统的算法，通过设计滤波器对图像进行处理。特点是速度比较快，很多卷积滤波可以借助 FFT 变换来加速。

BM3D 算法总体分为两大步，每一大步又分为三小步：相似块分组、协同滤波和聚合。从 2007 年提出以来，它在近十年内都被认为是很好的算法，甚至目前一些论文也经常引用到这个模型作为对比。

该算法的主要原理是通过相似判定找到与参考块相近的二维图像块，并将相似块按照组合成三维群组，对三维群组进行协同滤波处理，再将处理结果聚合到原图像块的位置。

稀疏表示类算法 从稀疏分解与稀疏表示的角度看，含噪的图像信号包括两部分，干净图像信号与噪声。由于干净的图像信号是又一定的结构的，故干净图像信号被认为是图片中的稀疏成分，通过设定稀疏表示阈值可以保留下来。而噪声是随机的，不相关，因此没有结构特性，所以 = 可以从原本的图像信号中提取干净有结构的图像信号，去除随机噪声信号。

聚类低秩 近几年，低秩矩阵恢复（LRMR）广泛用于图像处理用途图像恢复，比如去噪、去模糊等。一幅清晰的自然图像其数据矩阵往往是低秩或者近似低秩的，但存在随机幅值任意大但是分布稀疏的误差破坏了原有数据的低秩性。低秩矩阵恢复是将退化图像看做一组低维数据加上噪声形成的，因此退化前的数据就可以通过低秩矩阵来逼近。

外部先验 如果从有噪音的图片本身无法找到规律，也可以借助其他类似但又没有噪音的图片，来总结图片具有的固有属性。这一类方法利用的外部图片来创造先验条件，然后用于约束需要预测的图片。这种想法和 Deep learning 中有监督/半监督等学习方法类似。

3.8 图像风格迁移

这是我们项目的一大特点，主要参考了以下论文：A Neural Algorithm of Artistic Style[1], Perceptual Losses for Real-Time Style Transfer and Super-Resolution[2], Meta Networks for Neural Style Transfer[3]，以及 github 开源代码：StyleTransferTrilogy[4] 和 DeepLearningExamples[5] 。

实现的是任意风格任意内容的极速风格迁移（实际产出速度根据选择训练参数以及机器性能的不同有所差异）。

VGG19 VGG19 是 Google DeepMind 发表在 ICLR 2015 上的论文《VERY DEEP CONVOLUTIONAL NETWORK SFOR LARGE-SCALE IMAGE RECOGNITION》中提出的一种 DCNN 结构。

本项目使用到的 VGG19 模型就是在 imagenet 数据集上预训练的模型。

一般认为，深度卷积神经网络的训练是对数据集特征的一步步抽取的过程，从简单的特征到复杂的特征。训练好的模型学习到的是对图像特征的抽取方法，所以在 imagenet 数据集上训练好的模型理论上来说，也可以直接用于抽取其他图像的特征，这也是迁移学习的基础。这样的效果往往没有在新数据上重新训练的效果好，但能够节省大量的训练时间，在特定情况下非常有用。

首先，我们下载预训练好的 VGG19 模型。

---

[1] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge: A Neural Algorithm of Artistic Style. CoRR (2015).

[2] Justin Johnson, Alexandre Alahi, Li Fei-Fei: Perceptual Losses for Real-Time Style Transfer and Super-Resolution. ECCV (2) 2016: 694-711.

[3] Falong Shen, Shuicheng Yan, Gang Zeng: Meta Networks for Neural Style Transfer. CoRR (2017).

[4] https://github.com/CortexFoundation/StyleTransferTrilogy

[5] https://github.com/AaronJny/DeepLearningExamples/tree/master/tf2-neural-style-transfer

模型编写 首先，从预训练的 VGG19 模型中，获取卷积层部分的参数，用于构建我们自己的模型。VGG19 中的全连接层舍弃掉，这一部分对提取图像特征基本无用。此处提取出来的 VGG 参数全部是作为常量使用的，不会再被训练，在反向传播的过程中也不会改变。

此外，将输入层设置为变量；这是因为我们需要实现任意风格任意内容的风格迁移，使得最开始输入一张噪音图片，然后不断地根据内容 loss 和风格 loss 对其进行调整，直到一定次数后，该图片兼具了风格图片的风格以及内容图片的内容。当训练结束时，输入层的参数就是我们生成的图片。

模型训练 主要分为以下 7 个步骤

1. 使用 VGG 中的一些层的输出来表示图片的内容特征和风格特征。

2. 将内容图片输入网络，计算内容图片在网络指定层上的输出值。

3. 计算内容损失。这样定义内容损失：内容图片在指定层上提取出的特征矩阵，与噪声图片在对应层上的特征矩阵的差值的 L2 范数。即求两两之间的像素差值的平方。

对应每一层的内容损失函数：

$$L_i = \frac{1}{2 * M * N} \sum_{ij} (X_{ij} - P_{ij})^2$$

其中，X 是噪声图片的特征矩阵，P 是内容图片的特征矩阵。M 是 P 的长 * 宽，N 是信道数。

最终的内容损失为，每一层的内容损失加权和，再对层数取平均。

4. 将风格图片输入网络，计算风格图片在网络指定层上的输出值。

5. 计算风格损失。使用风格图像在指定层上的特征矩阵的 GRAM 矩阵来衡量其风格，风格损失可以定义为风格图像和噪音图像特征矩阵的格莱姆矩阵的差值的 L2 范数。

对于每一层的风格损失函数：

$$L_i = \frac{1}{4 * M^2 * N^2} \sum_{ij} (G_{ij} - A_{ij})^2$$

其中 M 是特征矩阵的长 * 宽，N 是特征矩阵的信道数。G 为噪音图像特征的 Gram 矩阵，A 为风格图片特征的 GRAM 矩阵。

最终的风格损失为，每一层的风格损失加权和，再对层数取平均。

6. 最终用于训练的损失函数为内容损失和风格损失的加权和。

$$L_{total} = \alpha L_{content} + \beta L_{style}$$

7. 当训练开始时，我们根据内容图片和噪声，生成一张噪声图片。并将噪声图片喂给网络，计算 loss，再根据 loss 调整噪声图片。将调整后的图片喂给网络，重新计算 loss，再调整，再计算……直到达到指定迭代次数，此时，噪声图片已兼具内容图片的内容和风格图片的风格，进行保存即可。

注：由于 *TensorFlow 2.0* 默认使用动态图模式，所以在单次迭代过程中，利用 *AutoGraph* 技术将迭代过程构造成静态图，加速计算，约可以节约一半的训练时间。

核心代码展示如下：

```python
import tensorflow as tf
import settings
import models
import numpy as np
import scipy.misc


def loss(sess, model):
    """
    定义模型的损失函数
    :param sess: tf session
    :param model: 神经网络模型
    :return: 内容损失和风格损失的加权和损失
    """
    # 先计算内容损失函数
    # 获取定义内容损失的vgg层名称列表及权重
    content_layers = settings.CONTENT_LOSS_LAYERS
    # 将内容图片作为输入，方便后面提取内容图片在各层中的特征矩阵
    sess.run(tf.assign(model.net['input'], model.content))
    # 内容损失累加量
    content_loss = 0.0
    # 逐个取出衡量内容损失的vgg层名称及对应权重
    for layer_name, weight in content_layers:
        # 提取内容图片在layer_name层中的特征矩阵
        p = sess.run(model.net[layer_name])
        # 提取噪音图片在layer_name层中的特征矩阵
        x = model.net[layer_name]
        # 长x宽
        M = p.shape[1] * p.shape[2]
        # 信道数
        N = p.shape[3]
        # 根据公式计算损失，并进行累加
        content_loss += (1.0 / (2 * M * N)) * tf.reduce_sum(tf.pow(p - x, 2)) * weight
    # 将损失对层数取平均
    content_loss /= len(content_layers)
```

```python
    # 再计算风格损失函数
    style_layers = settings.STYLE_LOSS_LAYERS
    # 将风格图片作为输入，方便后面提取风格图片在各层中的特征矩阵
    sess.run(tf.assign(model.net['input'], model.style))
    # 风格损失累加量
    style_loss = 0.0
    # 逐个取出衡量风格损失的vgg层名称及对应权重
    for layer_name, weight in style_layers:
        # 提取风格图片在layer_name层中的特征矩阵
        a = sess.run(model.net[layer_name])
        # 提取噪音图片在layer_name层中的特征矩阵
        x = model.net[layer_name]
        # 长x宽
        M = a.shape[1] * a.shape[2]
        # 信道数
        N = a.shape[3]
        # 求风格图片特征的gram矩阵
        A = gram(a, M, N)
        # 求噪音图片特征的gram矩阵
        G = gram(x, M, N)
        # 根据公式计算损失，并进行累加
        style_loss += (1.0 / (4 * M * M * N * N)) * tf.reduce_sum(tf.pow(G - A, 2)) *
            weight
    # 将损失对层数取平均
    style_loss /= len(style_layers)
    # 将内容损失和风格损失加权求和，构成总损失函数
    loss = settings.ALPHA * content_loss + settings.BETA * style_loss

    return loss


def gram(x, size, deep):
    """
    创建给定矩阵的格莱姆矩阵，用来衡量风格
    :param x:给定矩阵
    :param size:矩阵的行数与列数的乘积
    :param deep:矩阵信道数
    :return:格莱姆矩阵
    """
    # 改变shape为（size,deep)
```

```
    x = tf.reshape(x, (size, deep))
    # 求xTx
    g = tf.matmul(tf.transpose(x), x)
    return g


def train():
    # 创建一个模型
    model = models.Model(settings.CONTENT_IMAGE, settings.STYLE_IMAGE)
    # 创建session
    with tf.Session() as sess:
        # 全局初始化
        sess.run(tf.global_variables_initializer())
        # 定义损失函数
        cost = loss(sess, model)
        # 创建优化器
        optimizer = tf.train.AdamOptimizer(1.0).minimize(cost)
        # 再初始化一次（主要针对于第一次初始化后又定义的运算，不然可能会报错）
        sess.run(tf.global_variables_initializer())
        # 使用噪声图片进行训练
        sess.run(tf.assign(model.net['input'], model.random_img))
        # 迭代指定次数
        for step in range(settings.TRAIN_STEPS):
            # 进行一次反向传播
            sess.run(optimizer)
            # 每隔一定次数，输出一下进度，并保存当前训练结果
            if step % 50 == 0:
                print 'step {} is down.'.format(step)
                # 取出input的内容，这是生成的图片
                img = sess.run(model.net['input'])
                # 训练过程是减去均值的，这里要加上
                img += settings.IMAGE_MEAN_VALUE
                # 这里是一个batch_size=1的batch，所以img[0]才是图片内容
                img = img[0]
                # 将像素值限定在0-255，并转为整型
                img = np.clip(img, 0, 255).astype(np.uint8)
                # 保存图片
                scipy.misc.imsave('{}-{}.jpg'.format(settings.OUTPUT_IMAGE,step), img)
        # 保存最终训练结果
        img = sess.run(model.net['input'])
```

```
        img += settings.IMAGE_MEAN_VALUE
        img = img[0]
        img = np.clip(img, 0, 255).astype(np.uint8)
        scipy.misc.imsave('{}.jpg'.format(settings.OUTPUT_IMAGE), img)


if __name__ == '__main__':
    train()
```

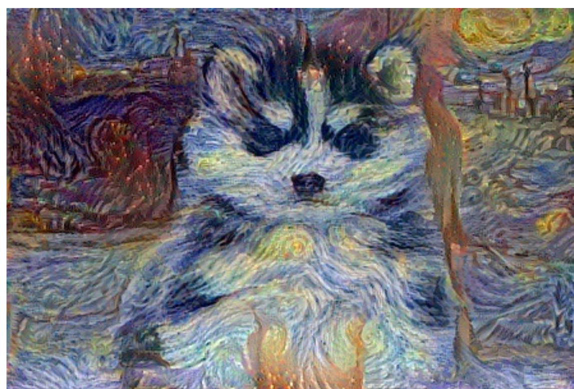
参数设置 需要注意的是，我们的项目提供了三个可供自由设定的训练参数，用户可以根据自身需要（如训练效果、训练机器性能等）进行参数设定，分别为：

- `epoch`：一个超参数，定义了学习算法在整个训练数据集中的工作次数。默认为 1（耗时最少，但效果一般）;
- `per epoch`：每个 epoch 训练次数。默认为 100;
- `learning rate`：学习率，默认为 0.03。

如果想要降低 loss ，可以在增大训练 epoch 数的同时，选择较小的学习率，或者使用学习率衰减；但这需要花费更多的时间。

此处演示了同一张内容图在同一风格图片（星空）下不同训练参数的最终风格迁移效果对比。

(a) $epoch = 1$



(b) $epoch = 20$

图 3: 训练次数效果对比 (per_epoch=100,learning_rate=0.03)

4 项目效果展示

我们在项目的首页最上方采用滚动的条幅展示数字图像中的经典用例图以美观首页，并在首页的下方展示了项目的详细资料信息。

图 4: 项目首页

4.1 图像编辑

在这一部分中，我们都以下面这张图作为原图进行效果展示，省略处理的步骤部分截图。

图 5: 图像处理前的**原图**

图 6: 图像灰度化展示

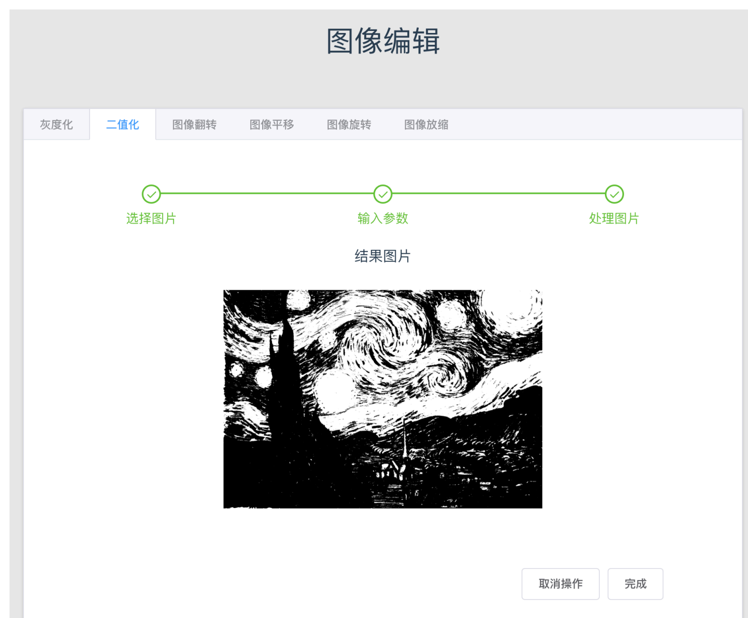


图 7: 图像二值化展示

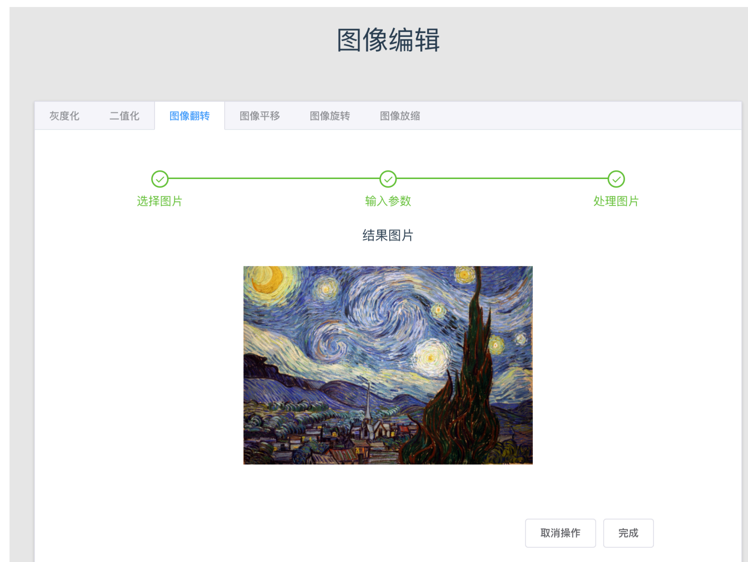


图 8: 图像翻转效果图展示–水平翻转

图 9: 图像翻转效果图展示–垂直翻转

图 10: 图像翻转效果图展示–对角翻转

图 11: 图像平移效果图展示–移动参数 (10, 10)

图 12: 图像旋转效果图展示–旋转角度 45 度

图 13: 图像放缩效果图展示–放缩比例 $(0.5, 0.2)$

4.2 图像分析

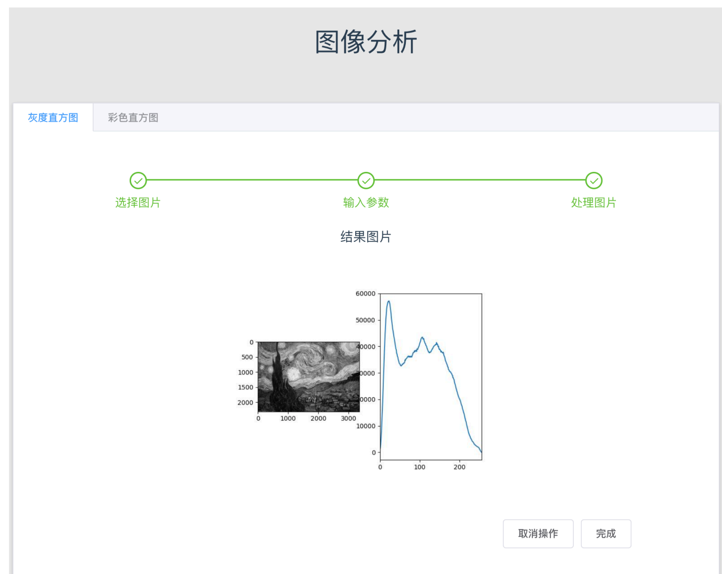


图 14: 灰度直方图效果展示

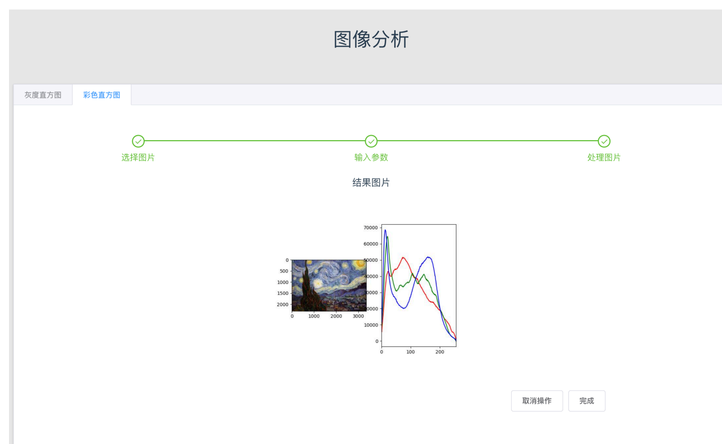


图 15: 彩色直方图效果展示

4.3 图像边缘检测

在这一部分中，我们都以下面这张图作为原图进行效果展示，省略处理的步骤部分截图。

图 16: 图像边缘检测使用的**原图**

图 17: 边缘检测–Roberts 算子

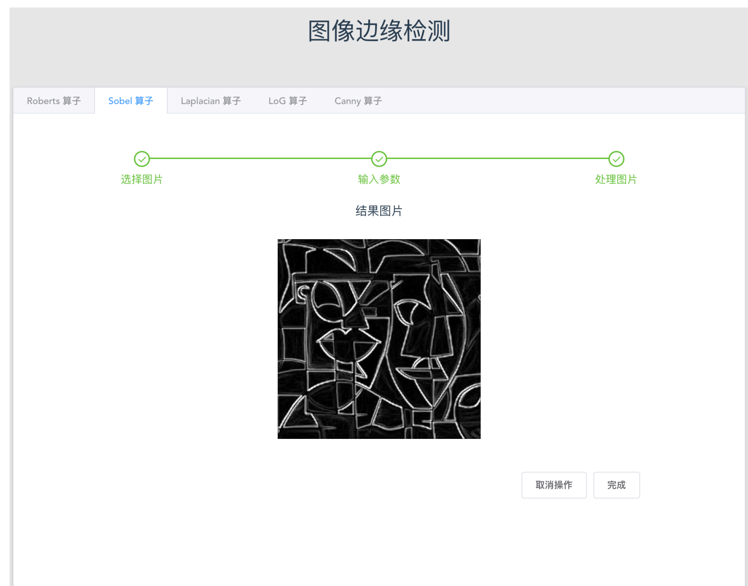


图 18: 边缘检测–Sobel 算子

图 19: 边缘检测–Laplacian 算子

图 20: 边缘检测–LoG 算子

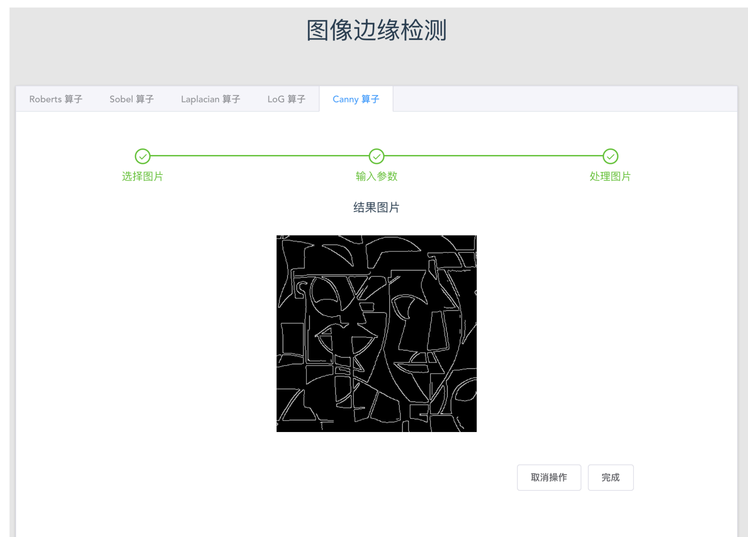


图 21: 边缘检测–Canny 算子

4.4 图像平滑与锐化

在这一部分中，我们所使用的原图如下所示：

(a) 图像平滑　　(b) 图像锐化

图 22: 图像平滑与锐化–原图

图 23: 图像平滑–邻域平均法

图 24: 图像平滑–中值滤波

图 25: 图像平滑–理想低通滤波

图 26: 图像平滑–巴特沃斯低通滤波

图 27: 图像平滑–高斯低通滤波

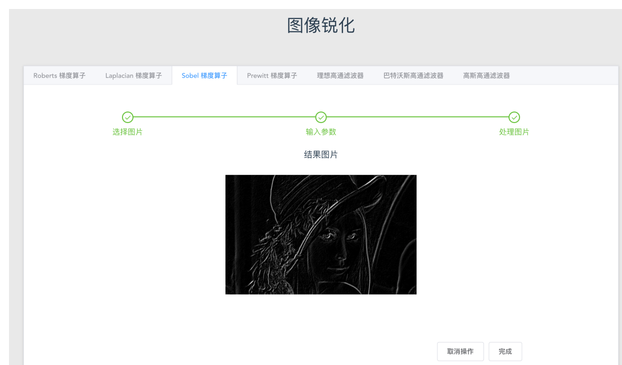


图 28: 图像锐化–Sobel

图 29: 图像锐化–巴特沃斯高通滤波

4.5 图像形态学

在这一部分中，我们使用的图片原图如下所示：

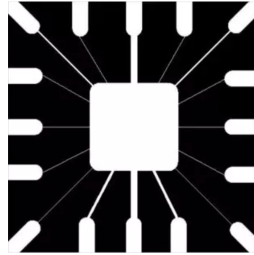


图 30: 图像形态学–原图

四种图像形态学的效果展示：

图 31: 图像形态学–腐蚀

图 32: 图像形态学–膨胀

图 33: 图像形态学–开运算

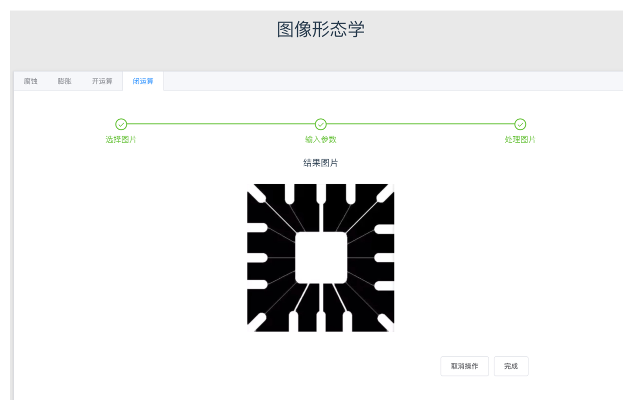


图 34: 图像形态学–闭运算

4.6 图像修复（噪声处理）

图 35: 图像修复–原图

图 36: 图像修复–高斯噪声

图 37: 图像修复–椒盐噪声

4.7 图像风格迁移

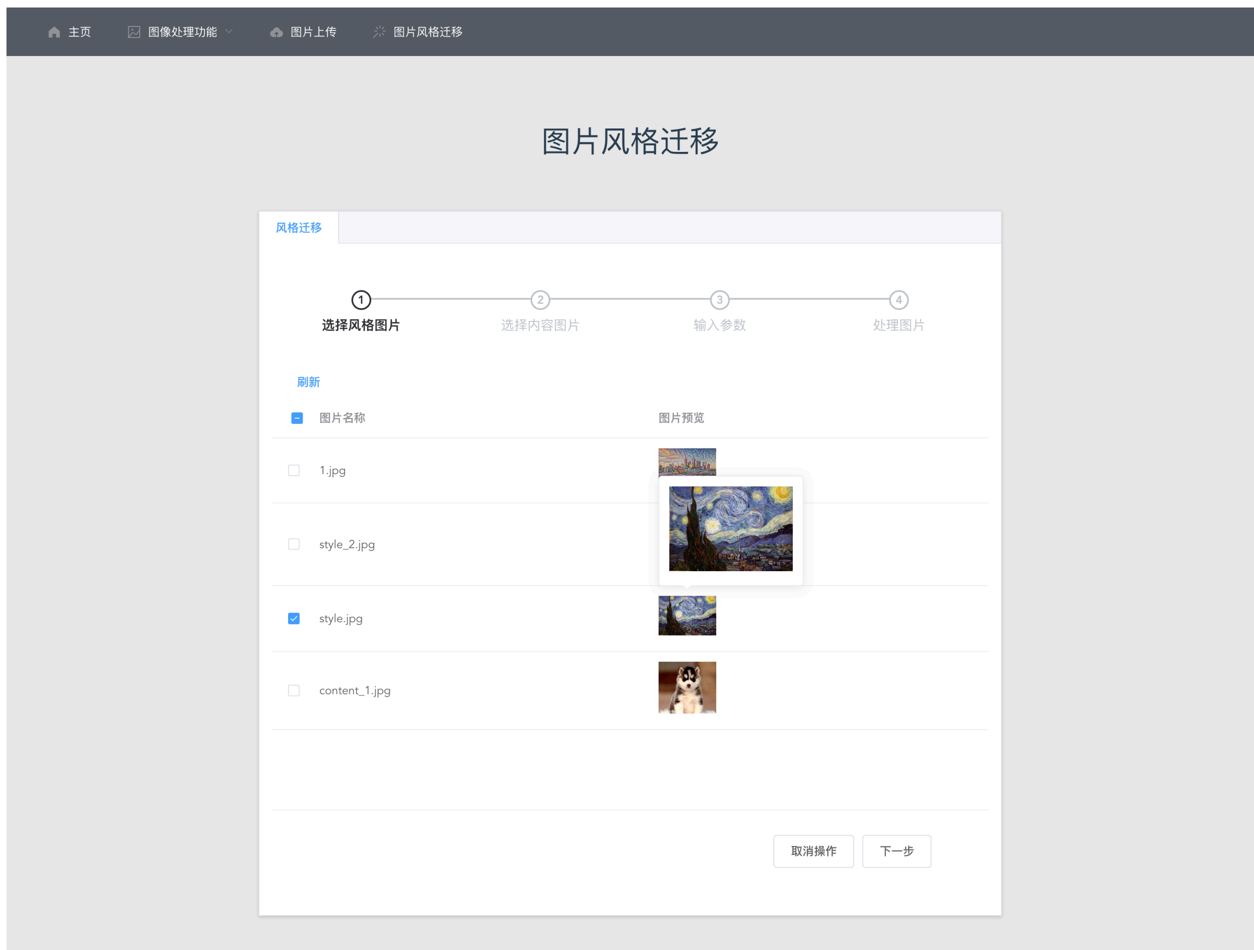


图 38: 图像风格迁移——选择风格照片

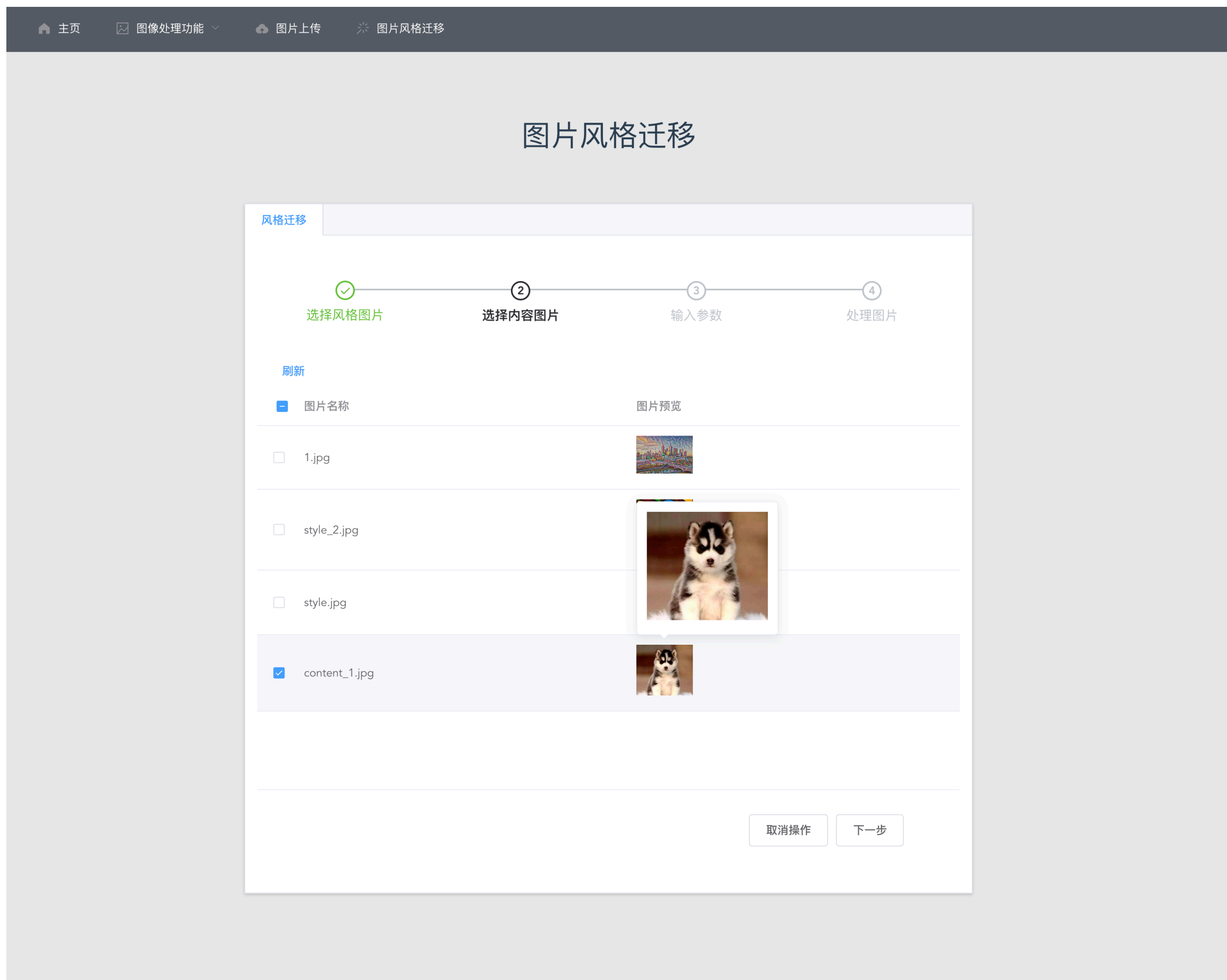


图 39: 图像风格迁移——选择内容照片

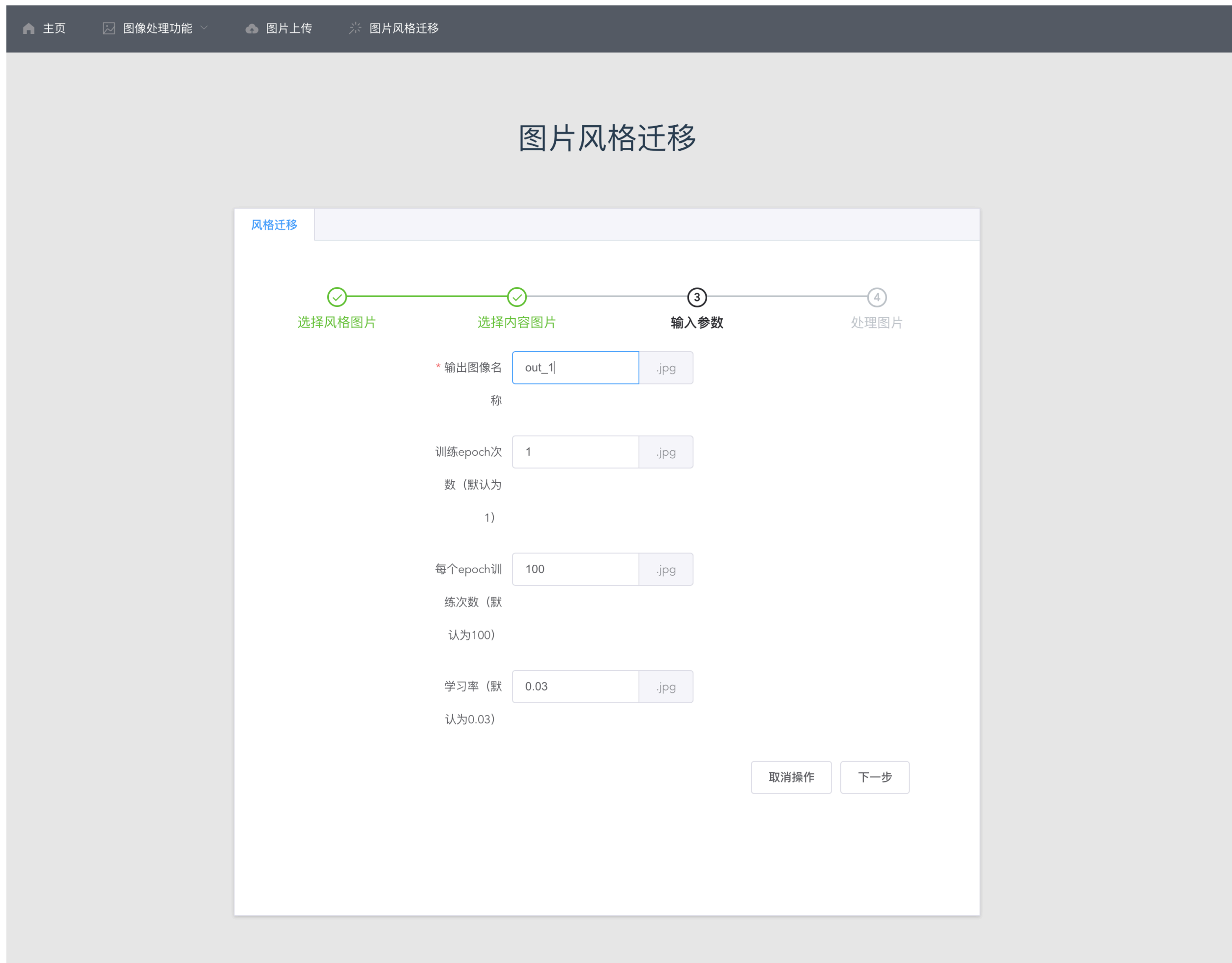


图 40: 图像风格迁移——输入参数

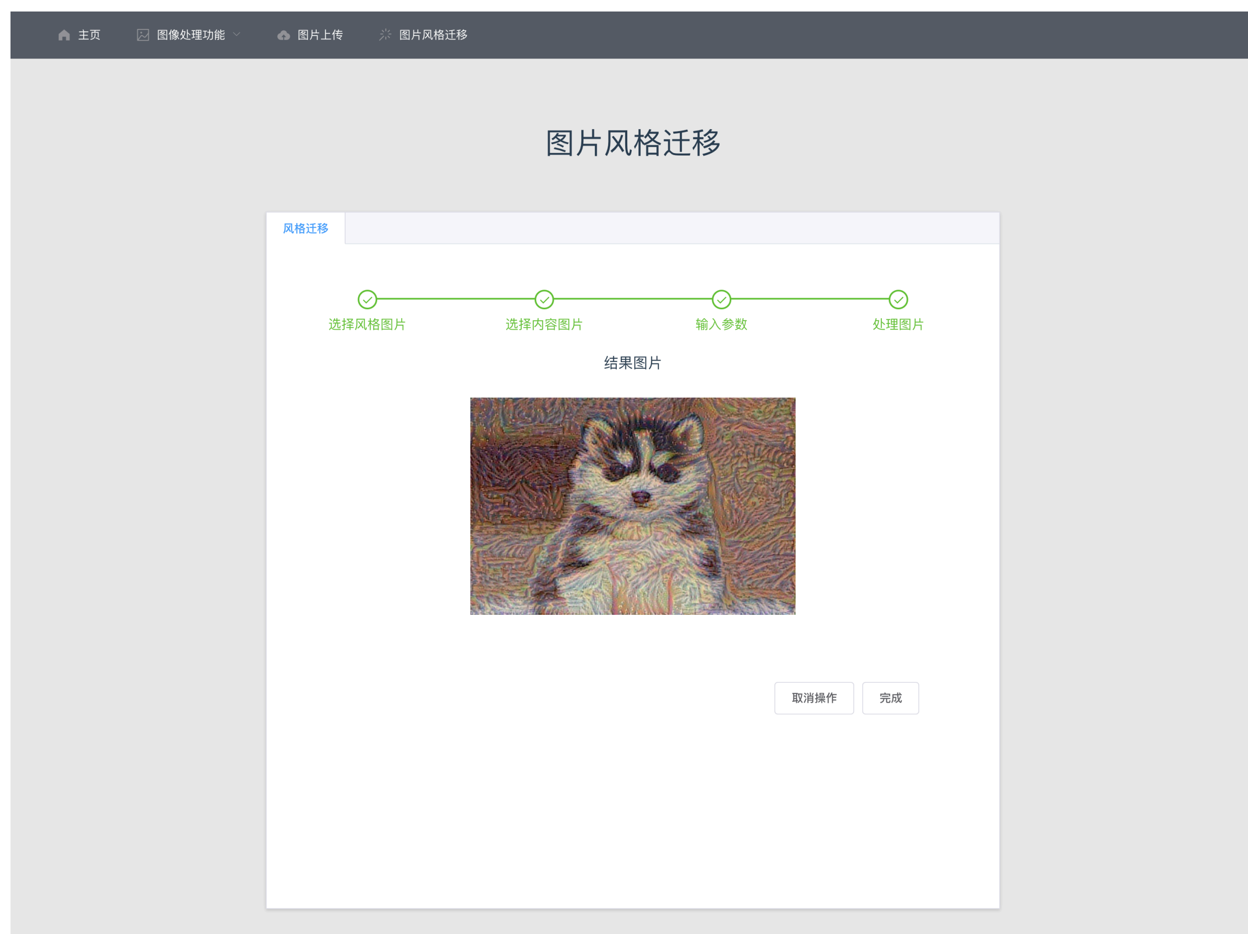


图 41: 图像风格迁移——生成图片（最终结果,epoch=1）