1 去噪

常见的去噪方法有均值滤波、高斯滤波、中值滤波等。这次尝试了均值滤波与高斯滤波。

* 1. 代码

（这部分只是简单调用了cv2里的函数）

def Gaussian(gray):

''' 高斯滤波降噪

:param gray: 灰度化后的图像

:return: 降噪后的图像

'''

dst = cv.GaussianBlur(gray, (7,7), sigmaX=2.3)

return dst

def Median(gray):

''' 中值滤波降噪

:param gray: 灰度化后的图像

:return: 降噪后的图像

'''

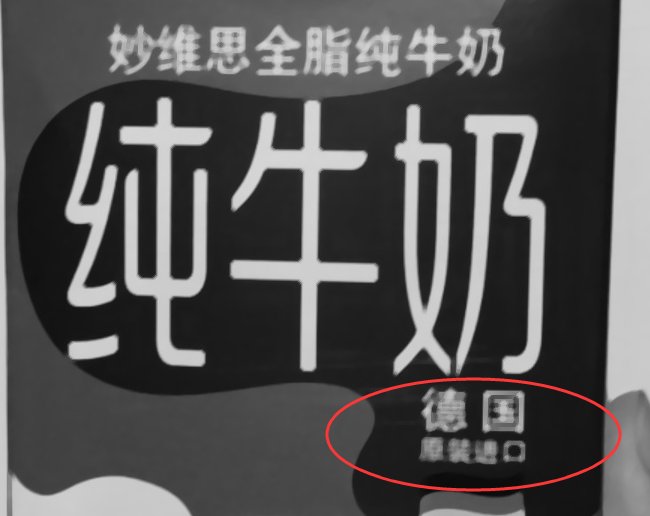
dst = cv.medianBlur(gray, 7)

return dst

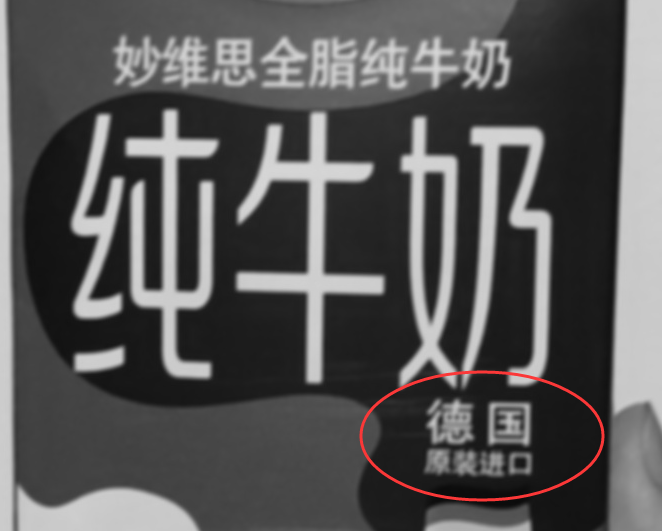
* 1. 结果比较

但是在尝试的过程中发现，内核大小相同的时候，高斯滤波对于这张图片一些重要细节的保留会比中值滤波更优秀一些。如下图所示，高斯滤波处理后，文字的显示依然较清晰。

中值滤波：



高斯滤波：



同时，关于高斯函数的参数，我也尝试了不同的值。其中，关于内核和sigma的比值，根据调查，大概3：1的时候效果更好。至于内核大小，这里举两个例子。（原图见文件）

内核3，sigma1：



内核7，sigma2.3：



可以观察到，对于墙上凹凸的区域，将参数设置为内核7，sigma2.3的时候，能在一定程度上平滑不必要的边缘，从而更好地处理图片。

2 边缘提取

2.1 Sobel

Sobel算子：[-1 0 1

-2 0 2

-1 0 1]

利用Sobel算子与原图像卷积，可以检测出垂直方向的边缘。由于Sobel算子比较敏感，易受影响，所以之前需要通过高斯平滑来降噪。

2.2 边缘梯度并量化方向

计算公式：edge = pow((pow(fx\_float, 2) + pow(fy\_float, 2)), 0.5)

tan = np.arctan(fy / fx)

量化方向：将不同角度量化到特定方向

angle = tan / np.pi \* 180

angle[angle < -22.5] = 180 + angle[angle < -22.5]

\_angle = np.zeros\_like(angle, dtype = np.uint8)

\_angle[np.where(angle <= 22.5)] = 0

\_angle[np.where((angle > 22.5) & (angle <= 67.5))] = 45

\_angle[np.where((angle > 67.5) & (angle <= 112.5))] = 90

\_angle[np.where((angle > 112.5) & (angle <= 157.5))] = 135

2.3 非最大化抑制

对梯度幅值进行非最大化抑制，可以细化边缘，使得结果更加准确、清晰。

对于量化后的方向，不同方向的dx和dy：

if angle[y, x] == 0:

dx1, dy1, dx2, dy2 = -1, 0, 1, 0

elif angle[y, x] == 45:

dx1, dy1, dx2, dy2 = -1, 1, 1, -1

elif angle[y, x] == 90:

dx1, dy1, dx2, dy2 = 0, -1, 0, 1

elif angle[y, x] == 135:

dx1, dy1, dx2, dy2 = -1, -1, 1, 1

同时，也要考虑x，y是边缘的情况：

if x == 0:

dx1 = max(dx1, 0)

dx2 = max(dx2, 0)

if y == 0:

dy1 = max(dy1, 0)

dy2 = max(dy2, 0)

if x == W - 1:

dx1 = min(dx1, 0)

dx2 = min(dx2, 0)

if y == H - 1:

dy1 = min(dy1, 0)

dy2 = min(dy2, 0)

遍历每个像素点，寻找梯度方向上其他点的值，如果不是最大则抑制：

if edge[y, x] < max(edge[y + dy1, x + dx1], edge[y + dy2, x + dx2]):

\_edge[y, x] = 0

2.4 双阈值筛选

双阈值筛选是为了去掉孤立的低阈值边缘。

大于高阈值设为255， 小于低阈值设为0：

if \_edge[y, x] < low:

\_edge[y, x] = 0

elif \_edge[y, x] > high:

\_edge[y, x] = 255

如果在高低阈值之间，则要判断像素是否在主边缘上；周围一圈如果有大于高阈值的，则将该点设为255，反之设为0：

if np.max(\_edge[y-1:y+2, x-1:x+2] \* t) >= high:

\_edge[y, x] = 255

else:

\_edge[y, x] = 0

（最终结果见文件）

3 拟合——霍夫变换

霍夫变换可以用来辨别物件的特征，比如线条。利用cv.HoughLines就可以实现霍夫变换。不同的阈值会带来不同的效果，如：

阈值150：



阈值200：



阈值240：



很明显可以看出，阈值低的时候，会将图片中的所有汉字的边缘也拟合出来，但如果我们需要的只是牛奶盒边缘的话，则可以适当调高阈值。

4 总结

至此，通过高斯滤波、Canny、霍夫变换，我们实现了去噪、边缘检测、拟合的功能。

（具体结果图见文件）