Denosing

Principle of denoising

Main idea of denosing is using *total variation minimization*[1] method. The *total variation* of a (grayscale) image I is defined as the sum of the gradient norm. In a discrete setting, the total variation becomes

$$J(I) = \sum_{x} |\triangledown I| \cdots\cdots(1)$$

where the sum is taken over all images coordinates $x = [x, y]$.

In the *Chambolle*'s paper[1], the goal is to find a de-noised image U that minimizes

$$min_U |I - U|^2 + 2\lambda J(U) \cdots\cdots\cdot (2)$$

where the norm $|I - U|$ measures the difference between U and the original image I.

In [1] paper, the solution of (2) is described as following:

$$J(w) \;=\; \sum_{i,j} |(\nabla^x w)_{i,j}| + |(\nabla^y w)_{i,j}|$$

where $(\nabla w)_{i,j} = ((\nabla^x w)_{i,j}, (\nabla^y w)_{i,j}) \in X \times X$ is defined by $(\nabla^x w)_{i,j} = w_{i+1,j} - w_{i,j}$ when $i < N$ and 0 if $i = N$, and $(\nabla^y w)_{i,j} = w_{i,j+1} - w_{i,j}$ when $j < M$ and 0 if $j = M$. If both X and $X \times X$ are endowed with the standard Euclidean scalar product, then a discrete divergence is given by $\mathrm{div} = -\nabla^*$, that is

$$(\mathrm{div}\,\xi, w)_X \;=\; -(\xi, \nabla w)_{X\times X} \;\forall w \in X, \xi \in X \times X.$$

(It is easily computed, see [9].)

By standard duality arguments, it is shown in [9] that the solution of (2) is given by $\overline{w} = g + \lambda \mathrm{div}\,\overline{\xi}$ where $\overline{\xi}$ is a solution to

$$\min\{\|g + \lambda \mathrm{div}\,\xi\|^2 \,:\, \xi \in X \times X\,, |\xi^x_{i,j}| \le 1 \text{ and } |\xi^y_{i,j}| \le 1 \forall i,j\}\,. \tag{9}$$

And the computation of *div* is described in [2] paper:

$$(\mathrm{div}\; p)_{ij} = \begin{cases} p^1_{i,j} - p^1_{i-1,j} & \text{if } 1 < i < N, \\ p^1_{i,j} & \text{if } i = 1, \\ -p^1_{i-1,j} & \text{if } i = N, \end{cases} + \begin{cases} p^2_{i,j} - p^2_{i,j-1} & \text{if } 1 < j < N, \\ p^2_{i,j} & \text{if } j = 1, \\ -p^2_{i,j-1} & \text{if } j = N, \end{cases}$$

And then

The adaption of the iterative algorithm of [9] to problem (9) is as follows: we let $\xi^0 = 0$, and for all $n \geq 0$ we let

$$\begin{cases} w^n &= g + \lambda \operatorname{div} \xi^n \\ (\xi_{i,j}^{n+1})^x &= \dfrac{(\xi_{i,j}^n)^x + (\tau/\lambda)(\nabla^x w^n)_{i,j}}{1 + (\tau/\lambda)|(\nabla^x w^n)_{i,j}|}, \\ (\xi_{i,j}^{n+1})^y &= \dfrac{(\xi_{i,j}^n)^y + (\tau/\lambda)(\nabla^y w^n)_{i,j}}{1 + (\tau/\lambda)|(\nabla^y w^n)_{i,j}|}, \end{cases} \tag{10}$$

where $\tau > 0$ is a fixed "time-step". One shows as in [9] that as $n \to \infty$, $w^n \to \overline{w}$, provided $\tau \leq 1/8$ (in fact, experimental convergence is observed as long as $\tau \leq 1/4$). The following variant, which is a simple gradient descent/reprojection method, seems to perform better:

$$\begin{cases} w^n &= g + \lambda \operatorname{div} \xi^n \\ (\xi_{i,j}^{n+1})^x &= \dfrac{(\xi_{i,j}^n)^x + (\tau/\lambda)(\nabla^x w^n)_{i,j}}{\max\{1, |(\xi_{i,j}^n)^x + (\tau/\lambda)(\nabla^x w^n)_{i,j}|\}}, \\ (\xi_{i,j}^{n+1})^y &= \dfrac{(\xi_{i,j}^n)^y + (\tau/\lambda)(\nabla^y w^n)_{i,j}}{\max\{1, |(\xi_{i,j}^n)^y + (\tau/\lambda)(\nabla^y w^n)_{i,j}|\}}. \end{cases} \tag{11}$$

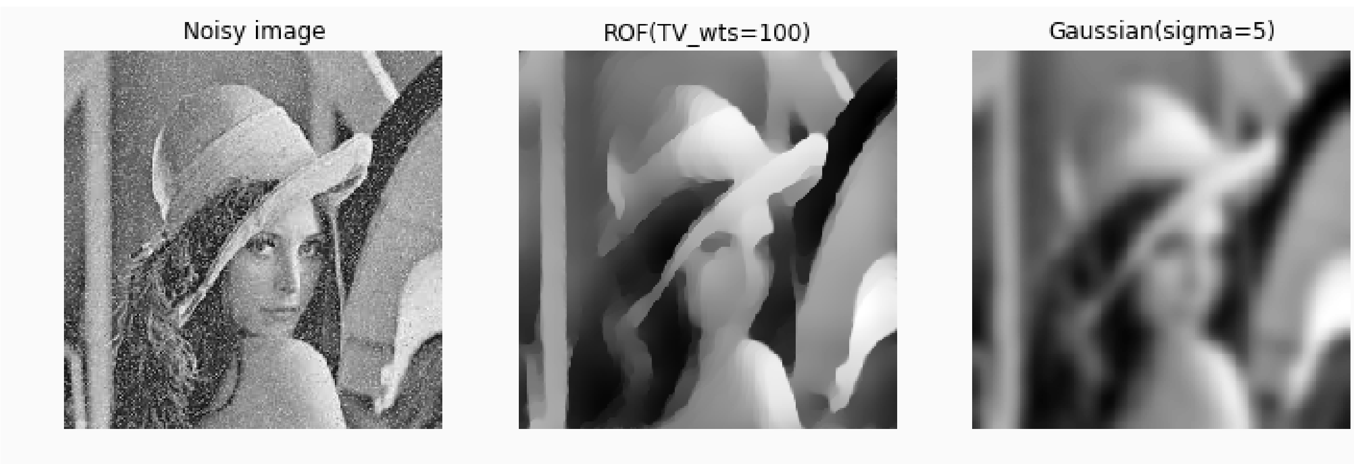
Source Code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import filters

def denoise(im, U_init, tv_weight, tolerance=0.01, tau=0.125):

    """
    An implementation of the Rudin-Osher-Fatemi (ROF) denoising model
    using the numerical procedure presented in eq (11) A. Chambolle (2005)

    Input: noisy input image (grayscale), initial guess for U, weight of
    the TV-regularizing term, steplength, tolerance for stop criterion.

    Output: denoised and detextured image, texture residual.
    """
```

```
    im = np.asarray(im)
    m, n = im.shape # size of noisy image

    # initialize
    U = U_init
    Px = im # x-component to the dual field
    Py = im # y-conponent of the dual field
    error = 1

    while (error > tolerance):

        Uold = U

        # grandient of primal variable
        GradUx = np.roll(U, -1, axis=1) - U # x-component of U's gradient
        GradUy = np.roll(U, -1, axis=0) - U # y-component of U's gradient

        # update the dual varible
        PxNew = Px + (tau/tv_weight)*GradUx
        PyNew = Py + (tau/tv_weight)*GradUy
        NormNew = np.maximum(1, np.sqrt(PxNew**2 + PyNew**2))
        Px = PxNew / NormNew # update of x-component (dual)
        Py = PyNew / NormNew # update of y-component (dual)

        # update the primal variable
        RxPx = np.roll(Px, 1, axis=1) # right x-translation of x-component
        RyPy = np.roll(Py, 1, axis=0) # right y-translation of y-component

        DivP = (Px-RxPx) + (Py - RyPy) # divergence of the dual field
        U = im + tv_weight*DivP # update of the primal variable

        # update of error-measure
        error = np.linalg.norm(U - Uold) / np.sqrt(n*m)

    return U, im-U # denoised image and texture residual
```

```
img = cv2.imread("D:/CAO_project/lena-noise.jpg", 0)
G = filters.gaussian_filter(img, 5)
```

```
U, T = denoise(img, img, 100)
plt.figure(figsize=(12, 12))
plt.subplot(1, 3, 1), plt.imshow(img, cmap="gray")
plt.title("Noisy image")
plt.axis("off")
plt.subplot(1, 3, 2), plt.imshow(U, cmap="gray")
plt.title("ROF(TV_wts=100)")
plt.axis("off")
plt.subplot(1, 3, 3), plt.imshow(G, cmap="gray")
plt.title("Gaussian(sigma=5)")
plt.axis("off")
plt.show()
```




```
img = cv2.imread("D:/CAO_project/lena-noise.jpg", 0)
G = filters.gaussian_filter(img, 3)
U, T = denoise(img, img, 25)
plt.figure(figsize=(12, 12))
plt.subplot(1, 3, 1), plt.imshow(img, cmap="gray")
plt.title("Noisy image")
plt.axis("off")
plt.subplot(1, 3, 2), plt.imshow(U, cmap="gray")
plt.title("ROF(TV_wts=25)")
plt.axis("off")
plt.subplot(1, 3, 3), plt.imshow(G, cmap="gray")
plt.title("Gaussian(sigma=3)")
plt.axis("off")
plt.show()
```


```
img = cv2.imread("D:/CAO_project/lena-noise.jpg", 0)
G = filters.gaussian_filter(img, 3)
U, T = denoise(img, img, 20)
plt.figure(figsize=(12, 12))
plt.subplot(1, 3, 1), plt.imshow(img, cmap="gray")
plt.title("Noisy image")
plt.axis("off")
plt.subplot(1, 3, 2), plt.imshow(U, cmap="gray")
plt.title("ROF(TV_wts=20)")
plt.axis("off")
plt.subplot(1, 3, 3), plt.imshow(G, cmap="gray")
plt.title("Gaussian(sigma=3)")
plt.axis("off")
plt.show()
```

Results Analysis

Looking at the above experimental results, it can be seen that the result of the *ROF* algorithm is better than the Gaussian denoising algorithm. *ROF* not only completes the denoising but also retains some important edge information.

And the value of *tv_weight* between 25 and 30, the results will be better. If the selected *tv_weight* is relatively small, such as 20 or 15, then some noise is not filtered out; the *tv_weight* is larger, such as 50, 80, or 100..etc, then some edge information is also smoothed out.

Reference

[1] Antonin Chambolle. Total variation minimization and a class of binary mrf models. In Energy Minimization Methods in Computer Vision and Pattern Recognition, Lecture Notes in Computer Science, pages 136—152. Springer Berlin / Heidelberg, 2005.

[2] Antonon Chambolle. An algorithm for total variation minimization and applications. J. Math. Imaging Vision, 20(1–2):89—97, 2004. Special issue on mathematics and image analysis. 2005.