Total variation denoising

1. Introducere

"Total variation denoising" (TVD) este o abordare pentru reducerea zgomotului, astfel încât să se rezerve marginile ascuţite în semnalul de bază . Spre deosebire de un filtru convenţional de trecere scăzută, deznodământul TVD este definit în termeni de o problemă de optimizare.

Filtrul de zgomot este obţinut prin reducerea la minimum a unei anumite funcţii de cost. Orice algoritm care rezolva problema de optimizare poate fi utilizat pentru a implementa deznodământul TV. Cu toate acestea, nu este de natură privată, deoarece funcţia de cost TVD nu este diferenţiată.

Pentru variaţia totală, care presupune că sunt datele conţin zgomot. Astfel, semnalul y (n) poate fi scris sub forma:

$$y(n) = x(n) + w(n), \qquad n = 0, \ldots, N-1$$

Unde x (n) reprezinta vectorul de pixeli din imagine, iar w (n) este zgomotul adaugat în imagine . Denoising TV estimează semnalul x (n) prin rezolvarea problemei de optimizare:

$$\arg\min_x \left\{ F(x) = \frac{1}{2} \sum_{n=0}^{N-1} |y(n) - x(n)|^2 + \lambda \sum_{n=1}^{N-1} |x(n) - x(n-1)| \right\}$$

2. Notite :

1. Semnalul punctului este reprezentat de vector
   x= [x(0), ..., x(N−1)]'
2. Norma l_1 a unui vector este definită ca fiind

   $\| v \| = \sum_n | v(n) |$

$$\|\mathbf{v}\|_2 = \left[\sum |v(n)|^2\right]^{\frac{1}{2}}$$

3. Norma l_2 a unui vector este definită ca fiind
4. Matricea D este definita:

$$\mathbf{D} = \begin{bmatrix} -1 & 1 & & & & \\ & -1 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & -1 & 1 & \\ & & & & -1 & 1 \end{bmatrix}$$

5. Variaţia totală a semnalului punctului N (x) este dată de

$$\mathrm{TV}(\mathbf{x}) := \|\mathbf{D}\mathbf{x}\|_1 = \sum_{n=1}^{N-1} |x(n) - x(n-1)|$$

6. În final funcţia poate fi scrisă sub forma:

$$\arg\min_{\mathbf{x}\in\mathbb{R}^N} \left\{ F(\mathbf{x}) = \frac{1}{2}\|\mathbf{y} - \mathbf{x}\|_2^2 + \lambda \|\mathbf{D}\mathbf{x}\|_1 \right\}$$

3. Minimizare si majorare

Pentru rezolvarea acestei probleme se cauta o problema de optimizare care este mai uşor de rezolvat decat problema iniţială. F(x) este considerată ca fiind funcţia de cost a problemei.

Pentru început, putem rescrie funcţia pentru a folosi functii patratice (care sunt mai uşor de minimizat)

$$\frac{1}{2|t_k|}t^2 + \frac{1}{2}|t_k| \geqslant |t| \qquad \forall t \in \mathbb{R}$$

4. Implementare Total Variation Denoising

```
% se citeste imaginea
I = imread('lena512.bmp');
% se aplica zgomot de tip salt and pepper
SaltPepperI = imnoise(I,'salt & pepper',0.01);
imwrite(SaltPepperI,'flower_salt_pepper.jpg');

% dimensiunile imaginii
[rows, columns, numberOfColorChannels] = size(SaltPepperI);

% se convertesc pixelii la double
y = double(SaltPepperI);

lam = 70;
% numarul de iteratii
Nit = 50;
% se converteste matricea de pixeli intr-un vector
y = y(:);
% functia de cost
cost = zeros(1, Nit);
N = length(y);

% matrice unitate
I = speye(N);
D = I(2:N, :) - I(1:N-1, :);
DDT = D * D';

x = y;

% initializare
Dx = D*x;
Dy = D*y;

for k = 1:Nit
        F = sparse(1:N-1, 1:N-1, abs(Dx)/lam) + DDT;
        x = y - D'*(F\Dy);
        Dx = D*x;
        % calcularea functiei de cost
        cost(k) = 0.5*sum(abs(x-y).^2) + lam*sum(abs(Dx));
end
% convertirea rezultatului in matrice
X = reshape(x, rows, columns);
```

```
% convertirea valorilor obtinute in pixeli
RES = uint8(255 * mat2gray(X));
figure,imshow(SaltPepperI)
hold
figure,imshow(RES)
```

5. Implementare cu gradient

```
I = imread('lena512.bmp');
SaltPepperI = imnoise(I,'salt & pepper',0.01);
imwrite(SaltPepperI,'flower_salt_pepper.jpg');

GaussianI = imnoise(I,'gaussian',0.01);
PoisonI = imnoise(I,'poisson');
SpeckleI = imnoise(I,'speckle');

[rows, columns, numberOfColorChannels] = size(SaltPepperI);

y = double(SaltPepperI);

% numarul de iteratii
N = 50;

nim = y;
u = y;
[height, width]=size(SaltPepperI);
for k=1:N
        [ux, uy]=imgradientxy(y, 'IntermediateDifference');
        lambda = 0.1;
        E=sum(sqrt(ux(:).^2 + uy(:).^2)) + lambda*sum(abs(u(:) - nim(:)));
        Res = gradient(E);
end
```

6. Implementare folosind Median Filter

```
p = 4;
pad=uint8(zeros(size(i)+2*(p-1)));

for x=1:size(i,1)
        for y=1:size(i,2)
        pad(x+p-1,y+p-1)=i(x,y);
```

```
        end
end

for i= 1:size(pad,1)-(p-1)
        for j=1:size(pad,2)-(p-1)
        kernel=uint8(ones((p)^2,1));
        t=1;
        for x=1:p
        for y=1:p
        kernel(t)=pad(i+x-1,j+y-1);
        t=t+1;
        end
        end
        filt=sort(kernel);
        out(i,j)=filt(ceil(((p)^2)/2));
        end
end
```

7. Rezultat folosind Total Variation denoising