

ℓ_1 Image Inpainting

The ℓ_1 –regularized image reconstruction problem is an alternative method to solve the image inpainting problem:

$$\min_{x \in \mathbb{R}^{mn}} \|\Psi x\|_1 \quad \text{s.t.} \quad \|Ax - b\|_\infty \leq \delta, \quad (3)$$

where $\Psi_{mn \times mn}$ transfers the image x to the frequency domain; $A_{s \times mn}$ and $b \in \mathbb{R}^s$ are as in the Total Variation Minimization Problem formulation, and $\delta > 0$ is the error threshold between the undamaged pixels b and the reconstructed version Ax .

LP Formulation

We first reformulate (3) as a linear program by noting that

$$\begin{aligned} & \min_{x \in \mathbb{R}^{mn}} \|\Psi x\|_1 \quad \text{s.t.} \quad \|Ax - b\|_\infty \leq \delta \\ &= \min_{x \in \mathbb{R}^{mn}} \mathbf{1}^\top |\Psi x| \quad \text{s.t.} \quad |Ax - b| \leq \delta \mathbf{1} \\ &= \min_{x, t \in \mathbb{R}^{mn}} \mathbf{1}^\top t \quad \text{s.t.} \quad t \geq \pm \Psi x, \quad \pm (Ax - b) \leq \delta \mathbf{1}. \end{aligned} \quad (4)$$

Here, $\mathbf{1}$ denotes the all-one vectors of appropriate sizes and $|v|$ is interpreted element-wise on vector v .

Dual Problem

Next, we derive the associated dual of (4). Rewrite (4)

$$\begin{aligned} & \min_{x, t \in \mathbb{R}^{mn}} \mathbf{1}^\top t \quad \text{s.t.} \quad t \geq \pm \Psi x, \quad \pm (Ax - b) \leq \delta \mathbf{1} \\ &= \min_{x, t \in \mathbb{R}^{mn}} \mathbf{1}^\top t \quad \text{s.t.} \quad \pm \Psi x + t \geq 0, \quad \pm Ax \geq -\delta \mathbf{1} \pm b \\ &= \min_{x, t \in \mathbb{R}^{mn}} \begin{bmatrix} 0^\top & \mathbf{1}^\top \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} \quad \text{s.t.} \quad \begin{bmatrix} \Psi & I_{mn} \\ -\Psi & I_{mn} \\ A & 0_{s \times mn} \\ -A & 0_{s \times mn} \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ -\delta \mathbf{1} + b \\ -\delta \mathbf{1} - b \end{bmatrix}. \end{aligned}$$

where I_r denotes the $r \times r$ identity matrix and $0_{p \times q}$ is the $p \times q$ all-zero matrix.

The dual is then given by

$$\begin{aligned}
& \max_{\substack{u, v \in \mathbb{R}^{mn} \\ y, z \in \mathbb{R}^s}} [0^\top \mid 0^\top \mid -\delta \mathbf{1}^\top + b^\top \mid -\delta \mathbf{1}^\top - b^\top] [u \quad v \quad y \quad z]^\top \\
& \text{s.t.} \quad \begin{bmatrix} \Psi^\top & -\Psi^\top & A^\top & -A^\top \\ I_{mn} & I_{mn} & 0_{mn \times s} & 0_{mn \times s} \end{bmatrix} [u \quad v \quad y \quad z]^\top = \begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}, \quad u, v, y, z \geq 0 \\
& = \max_{\substack{u, v \in \mathbb{R}^{mn} \\ y, z \in \mathbb{R}^s}} b^\top (y - z) - \delta \mathbf{1}^\top (y + z) \\
& \text{s.t.} \quad \Psi^\top (u - v) + A^\top (y - z) = 0, \quad u + v = \mathbf{1}, \quad u, v, y, z \geq 0 \\
& = \max_{\substack{t \in \mathbb{R}^{mn} \\ r, w \in \mathbb{R}^s}} b^\top w - \delta \mathbf{1}^\top (w + r) \\
& \text{s.t.} \quad \Psi^\top t + A^\top w = 0, \quad r \geq 0, \quad -1 \leq t \leq 1 \\
& = \max_{\substack{t \in \mathbb{R}^{mn} \\ w \in \mathbb{R}^s}} (b^\top - \delta \mathbf{1}^\top) w \\
& \text{s.t.} \quad \Psi^\top t + A^\top w = 0, \quad -1 \leq t \leq 1.
\end{aligned}$$

Implementation

We implemented the ℓ_1 inpainting algorithm with *linprog* function of MATLAB R2020a. The interior-point method was chosen for better performance.

```

% parameters, preprocesses of image and mask
U      = imread('.\test_images\512_512_lena.png');
if size(U, 3) == 3
    U    = rgb2gray(U);
end
U      = double(U) / 255;
[m, n] = size(U);

Ind     = imread('.\test_masks\512_512_random50.png');
Ind     = logical(ceil(Ind / 255));
s       = sum(Ind, 'all');
delta   = 0.06;

bsz = 8;
Psi = get_Psi(m, n, bsz);

% block-stack U and Ind in Psi style
u    = blk_stack(U, bsz);
ind  = blk_stack(Ind, bsz);

% form A and b
i     = 1:s;
j     = zeros(1, s);
count = 1;
for col = 1:m*n
    if ind(col) == 1
        j(count) = col;
        count = count + 1;
    end
end
end

```

```

A = sparse(i, j, ones(1, s), s, m*n);
b = A * u;

% linprog
% min(c'x) s.t. Mx <= d
del = delta*ones(s, 1);
I = speye(m*n);
ze = sparse(s, m*n);
c = [zeros(m*n, 1); ones(m*n, 1)];
M = [-Psi -I; Psi -I; -A ze; A ze];
d = [zeros(2*m*n, 1); del-b; del+b];
options = optimoptions('linprog', 'Algorithm', 'interior-point', ...
    'ConstraintTolerance', 1e-3, 'Display', 'iter');
x = linprog(c, M, d, [], [], [], [], options);

% scale and transform the stacked image back into matrix
x = uint8( x(1:m*n)*255 );
X = blk_unstack(x, bsz);

% evaluate PSNR
psnr = PSNR((U*255), double(X));
imshow(X);

```

Results

We tested the model on sample grey-scale images of 256×256 and 512×512 pixels. The results were obtained with a 4.0 GHz Quad-Core Intel Core i7-4790K processor and 32 GB 2133 MHz memory. The quality of the reconstructed images was assessed via the PSNR value:

$$\text{PSNR} := 10 \cdot \log_{10} \frac{mn}{\|x - u^*\|^2},$$

where x is the reconstructed image and $u^* = \text{vec}(U^*)$ is the ground truth.

Overall Performance

The ℓ_1 block model reconstructed 256×256 images contaminated by 50% random noise with $\text{PSNR} \approx 20$. The average runtime is around 40 sec with a maximum of 20 iterations. Curiously, we noticed that the algorithm took significantly longer at each iteration, and more iterations to converge, if the block size bsz had been set at 16 (the value recommended by the lecturer for low-res images). For instance, image (a) took nearly 8 min to reconstruct at $\text{bsz} = 16$ despite a slightly higher PSNR, and images (b) and (c) even failed to converge within a 200-iteration limit. The reason to this is most likely that matrix Ψ becomes much denser with larger bsz .


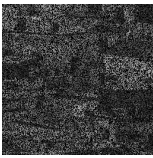



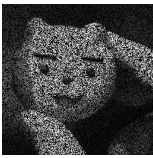


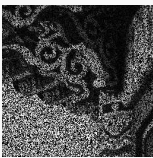

	Ground Truth (256×256)	Ground Truth + 50% Noise	Reconstructed Images	PSNR	Runtime (Iterations)
(a)				21.6	38.4 sec (17)
(bsz = 16)				23.9	7.9 min
(b)				22.3	44.2 sec (20)
(c)				20.4	44.5 sec (19)

Table 1. Reconstruction results of 256×256 random noise contaminated images, $\delta = 6 \times 10^{-2}$, bsz = 8, constraint tolerance = 10^{-3} , optimality tolerance = 10^{-6}

In the task of reconstructing 512×512 image polluted by 50% random noise, the algorithm achieves roughly the same PSNR compared to low-res images within similar number of iterations, while the runtime becomes roughly four times as long. We also observed that as the noise intensity increased, the image quality deteriorates drastically although the algorithm seems to converge faster.








Noise Percentage	Ground Truth	30%	50%	70%
Ground Truth + Noise				
Reconstructed Images	-			
PSNR	-	25.5	22.6	17.0
Runtime (Iterations)	-	4.4 min (26)	3.1 min (21)	2.8 min (21)

Table 2. Reconstruction results of 512×512 images contaminated by 30%, 50%, and 70% random noises, $\delta = 6 \times 10^{-2}$, $\text{bsz} = 8$, constraint tolerance $= 10^{-3}$, optimality tolerance $= 10^{-6}$

In reconstructing images with non-random damages, the algorithm performs roughly the same as in the 50% random noise case with mesh, handwriting, and mild scratches. However, when inpainting the hard-scratched image, the algorithm performs poorly with $\text{PSNR} = 13.9$, a score even lower than with the 70% random noise, which covers more area of the image than the scratches. From this, we conclude that the algorithm is sensitive to the distribution of the damage. In particular, the algorithm performs better if the damage distributes more evenly, i.e., there is no large "chunks" of damage area on the image (as in hard scratches).






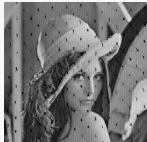
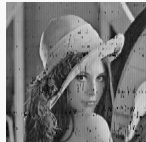






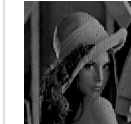

Damage Types	Ground Truth	Mesh	Handwriting	Mild Scratches	Hard Scratches
Damaged Images					
Reconstructed Images	-				
PSNR	-	21.6	21.5	25.3	13.9
Runtime (Iterations)	-	4.2 min (26)	4.2 min (21)	4.4 min (24)	3.7 min (22)

Table 3. Reconstruction results of 512×512 images with non-random damage, $\delta = 6 \times 10^{-2}$, $\text{bsz} = 8$, constraint tolerance $= 10^{-3}$, optimality tolerance $= 10^{-6}$

Effect of adjusting Optimality Tolerance

We investigate the effect of adjusting optimality tolerance on the reconstruction quality by inpainting the 50% random noise contaminated image.

Optimality Tolerance	Ground Truth	TV (Interior Point)	0.006	0.06	0.3	0.6
Reconstructed Images						
PSNR	-	34.5	26.3	22.6	11.3	6.5
Runtime (Iterations)	-	19.0 sec (11)	4.6 min (30)	3.1 min (21)	3.3 min (22)	3.2 min (21)

Effect of adjusting δ

We investigate the effect of altering the error threshold δ on the reconstruction quality by inpainting the 50% random noise contaminated image. We observe that the image quality slightly decreases as the threshold grows from 0.006 to 0.06, but is still within the acceptable range. Nevertheless, when δ reaches 0.3, the reconstructed image becomes visibly darker and coarser, and at $\delta = 0.6$, the image is almost unidentifiable.

Comparing with the Total Variation model in part 1 of the project, the ℓ_1 block model is inferior in both reconstruction quality and speed. The excessive runtime of the ℓ_1 model is probably due to







δ	Ground Truth	TV (Interior Point)	0.006	0.06	0.3	0.6
Reconstructed Images						
PSNR	-	34.5	26.3	22.6	11.3	6.5
Runtime (Iterations)	-	19.0 sec (11)	4.6 min (30)	3.1 min (21)	3.3 min (22)	3.2 min (21)

Table 5. Reconstruction results of 512×512 images contaminated by 50% random noise, $\delta = 0.006, 0.06, 0.3, 0.6$, $\text{bsz} = 8$, constraint tolerance $= 10^{-3}$, optimality tolerance $= 10^{-6}$