Progetto - Metodi Numerici

12 gennaio 2022

1 Introduzione

Il programma è stato scritto in Matlab ed è composto dallo script principale *main.m* e da due funzioni che hanno il compito di eseguire l'inpainting nei due casi richiesti dal progetto. Nelle due sezioni che seguono verranno descritte brevemente le due funzioni *interp_inpainting(im,mask)* e *PDE_inpainting(im,mask)* che sono state implementate per eseguire rispettivamente l'image inpainting usando come metodi numerici l'interpolazione e la diffusione. Mentre il file principale *main.m* contiene l'inizializzazione delle immagini $I(x, y)$ e $mask(x, y)$ attraverso la funzione *imRead()* e la chiamata alle funzioni sopra citate i cui output corrispondono alle immagini restaurate con i due metodi richiesti dal progetto.

2 Interpolazione

Il primo metodo utilizzato per effettuare l'inpainting dell'immagine, che verrà trattato in questa sezione, è l'interpolazione. In particolare si considera una immagine in scala di grigi alla quale è associata una matrice di dimensione $498x495$ da restaurare, i cui valori sono espressi in binario con una codifica a 8 bit senza segno. Il valore di ogni pixel può quindi variare tra 0 e 255. Per poter lavorare su tali immagini le si converte in double, in modo da avere i pixel con un valore che varia tra 0 e 1. Per eseguire questa conversione si è utilizzata la funzione Matlab *im2double* che esegue il cast in double e allo stesso tempo la normalizzazione.

Nel caso delle immagini si può modellare numericamente l'inpainting con il metodo dell'interpolazione considerando i valori dei pixel buoni come dati

Figura 1: Interpolazione immagine scala di grigi

scattered. Quindi si ha un insieme di pixel che corrispondono alla zona buona dell'immagine di cui si considerano le coordinate x, y e i corrispettivi valori, questi vengono poi utilizzati per poter trovare una funzione interpolante. Tale funzione interpolante permette di definire i valori dei pixel appartenenti alla regione danneggiata e quindi restaurare l'immagine.

In Matlab si usa la funzione *scatteredInterpolant(P,V,method)* per definire l'interpolante nel caso in cui si lavora con dati scattered. Tale funzione ha in ingresso come parametri le coordinate dei punti corrispondenti ai pixel buoni dell'immagine e i corrispettivi valori espressi in formato double. Inoltre la funzione utilizzata permette di scegliere tra due diversi tipi di interpolazione: l'interpolazione lineare e l'interpolazione Nearest Neighbour. Il codice Matlab implementato è illustrato nella Lista 1

```
1
2  function [Irestored_linear, Irestored_nearest] =
       interp_inpainting(I,mask)
3
4  [i, j] = find(mask == 1);
5  coord = cat(2,i,j);
6
7  [i2, j2] = find(mask ~= 1);
8  coordScat = cat(2, i2, j2);
9
10 ind = sub2ind(size(I), i, j);
11 indScat = sub2ind(size(I), i2, j2);
12
```

```
v = I(ind);

F_linear = scatteredInterpolant(coord, v, 'linear');
F_nearest = scatteredInterpolant(coord, v, 'nearest');

reconstructedImage_linear = I;
reconstructedImage_nearest = I;

newV = F_linear(coordScat);
reconstructedImage_linear(indScat) = newV;

newV = F_nearest(coordScat);
reconstructedImage_nearest(indScat) = newV;

Irestored_linear = reconstructedImage_linear;
Irestored_nearest = reconstructedImage_nearest;

end
```

Listing 1: Codice Matlab funzione interp_inpainting(im mask)

Il codice nella Lista 1 è quello che si è implementato nella funzione *interp_inpainting(im,mask)* del progetto e da come output due immagini restaurate , una secondo interpolazione lineare e l'altra secondo interpolazione Nearest Neighbour così come è possibile vedere nella Figura 1.

Nel caso di una immagine RGB si è eseguita l'interpolazione sui singoli canali rosso, verde e blu. Questo viene eseguito nello script *main.m* chiamando la funzione *interp_inpainting(im,mask)* per ognuno dei tre canali in modo da avere come output per ogni canale la corrispondente immagine restaurata. Per poter poi avere l'immagine RGB si è sovrapposto le tre immagini ottenute in output, modellate numericamente come matrici $498x495x1$, in modo da ottenere una matrice tridimensionale di dimensione $498x495x3$. Tale procedimento è illustrato nella Lista 2.

```

    im_R = interp_inpainting(im(:,:,1), mask);
    im_G = interp_inpainting(im(:,:,2), mask);
    im_B = interp_inpainting(im(:,:,3), mask);
    inpainted_image = cat(3, im_R, im_G, im_B);
```

Listing 2: codice matlab interpolazione immagine RGB

Il risultato di tale interpolazione è evidenziato in Figura 2

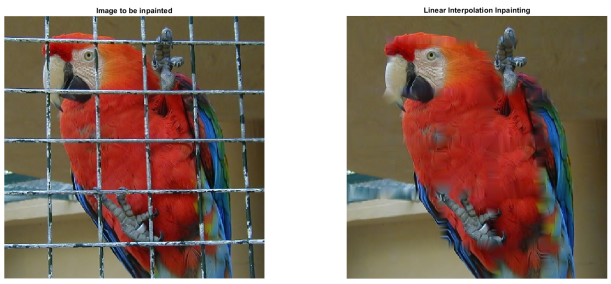


Figura 2: Interpolazione immagine RGB

3 Image Inpainting: PDE Heat Equation

Il secondo metodo numerico che si è implementato per effettuare l'inpainting è quello per diffusione attraverso l'uso della seguente PDE:

$$u_t = \lambda \Delta u + \chi_{\Omega \setminus D}(f - u) \qquad in \Omega$$

dove

$$\chi_{\Omega \setminus D}(x) = \begin{cases} 1, & \text{if } x \in \Omega \setminus D \\ 0, & \text{otherwise} \end{cases}$$

```
function [Irestored, peaksnr] = PDE_inpainting(I, mask)
[imX, imY] = size(I);
dx=1;
dy=1;
dt = 0.4;
t_max = 600;
ts = 1:dt:t_max;
ts_n = size(ts,2);

lambda = 0.5;
r = lambda*(dt/dx^2);

U0 = zeros(size(I));

L_X=(diag(-2*r*ones(imX,1)) + diag(r*ones(imX-1,1),1) +
diag(r*ones(imX-1,1),-1));
```

```
16
17      L_Y=(diag(-2*r*ones(imY,1)) + diag(r*ones(imY-1,1),1) +
        diag(r*ones(imY-1,1),-1));
18      ...
```

Listing 3: codice matlab

$$L\_X\_L\_Y = \begin{bmatrix} -2r & r & 0 & 0 & \cdots & 0 \\ r & -2r & r & 0 & \cdots & 0 \\ 0 & r & -2r & r & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & r & -2r \end{bmatrix}$$

Per la soluzione della equazione differenziale alle derivate parziali si è definita come condizione iniziale al tempo $t = 0$ una immagine in scala di grigi della stessa dimensione dell'immagine I sulla quale si effettua l'inpainting, dove però per ogni pixel si ha un valore pari a 0, quindi si ha una immagine nera. Si definiscono poi le derivate seconde del laplaciano attraverso differenze finite centrali e si ottiene le due matrici a banda L_XeL_Y.

Poichè la PDE è risolta in modo esplicito bisogna prestare particolare importanza alla stabilità, in questo caso infatti si ha un metodo condizionatamente stabile. Al fine di avere un metodo stabile si deve scegliere accuratamente la *CFL condition* che in questo caso è stata indicata dal parametro r, funzione dei passi spaziali dx e dy, quello temporale dt e il parametro di diffusività λ. Dove si ha $dx = dy = 1$ in quanto si è considerato come discretizzazione spaziale la distranza tra un pixel ed un altro. Si è visto sperimentalmente che in questo caso r deve essere minore o uguale a $\frac{1}{5}$ per avere stabilità:

$$r = \lambda \frac{dt}{dx} = \lambda \frac{dt}{dy} \leq \frac{1}{5}$$

In questo caso si ha che la regione dell'immagine da dover restaurare si trova anche in prossimità dei bordi laterali. Per questo motivo quando si esegue l'inpainting senza condizioni al contorno, si ottiene come output un'immagine che presenta lungo i suoi bordi delle regioni in cui l'algoritmo non è stato in grado di ricostruire in modo corretto le porzioni di immagine danneggiata così come è possibile vedere in Figura 3.

Figura 3: Inpainting senza condizioni Neumann

Per poter effettuare una corretta ricostruzione delle regioni in cui è presente il problema, sono state utilizzate le condizioni al contorno di *Neumann*.

$$\begin{aligned} u'_x(0,t) &= 0 \qquad & u'_y(0,t) &= 0 \\ u'_x(imX,t) &= 0 \qquad & u'_y(imY,t) &= 0 \end{aligned}$$

Di seguito si risolve numericamente la prima condizione su u'_x utilizzando le differenze finite centrali, lo stesso può essere fatto per u'_y.

$$\frac{\partial u_x(0,t)}{\partial x} = \frac{u_1^j - u_{-1}^j}{2\Delta x} = 0$$

$$u_{-1}^j = u_1^j$$

$$\frac{\partial u_x(imX,t)}{\partial x} = \frac{u_{imX+1}^j - u_{imX-1}^j}{2\Delta x} = 0$$

$$u_{imX+1}^j = u_{imX-1}^j$$

dove u_{-1} e u_{imX+1} sono i *Ghost point* e si è dimostrato che sono uguali rispettivamente a u_1 e u_{imX-1}. Tale uguaglianza permette di svolgere i successivi calcoli senza avere dipendenza da tali *Ghost point* :

$$\frac{u_1^{j+1} - u_1^j}{\Delta t} = \frac{(u_{-1}^j - 2u_0^j + u_1^j)}{\Delta x^2} = \frac{(-2u_0^j + 2u_1^j)}{\Delta x^2}$$

$$\frac{u_{imX}^{j+1} - u_{imX}^j}{\Delta t} = \frac{(u_{imX-1}^j - 2u_{imX}^j + u_{imX+1}^j)}{\Delta x^2} = \frac{(-2u_{imX}^j + 2u_{imX-1}^j)}{\Delta x^2}$$

si ha quindi:

$$\begin{aligned}
u_0^{j+1} - u_0^j &= -2ru_0^j + 2ru_1^j && i = 0 \\
u_i^{j+1} - u_i^j &= ru_{i-1}^j - 2ru_i^j + ru_{i+1}^j && i = 1, ..., imX - 1 \\
u_{imX}^{j+1} - u_{imX}^j &= -2ru_{imX}^j + 2ru_{imX-1}^j && i = imX
\end{aligned}$$

Tenendo conto delle condizioni di Neumann le due matrici L_X e L_Y possono essere riscritte nel seguente modo:

$$L\_X, L\_Y = \begin{bmatrix} -2r & 2r & 0 & 0 & \cdots & 0 \\ r & -2r & r & 0 & \cdots & 0 \\ 0 & r & -2r & r & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 2r & -2r \end{bmatrix}$$

Le nuove matrici L_X ed L_Y, che ora implementano le condizioni al contorno di *Neumann*, sono state implementate in Matlab attraverso la modifica delle vecchie matrici così come illustrato nella Lista 4.

```
1    ...
2    L_X(1,2)=2*r;
3    L_X(imX,imX-1)=2*r;
```

```
    L_Y(1,2)=2*r;
    L_Y(imY,imY-1)=2*r;
    ...
```

Listing 4: Codice Matlab: Condizioni di Neumann

Dopo aver definito le due nuove matrice L_X e L_Y con le nuove condizioni al contorno si prosegue in Matlab con la risoluzione della equazione differenziale alle derivate parziali con il metodo esplicito, così come illustrato nella Lista 5.

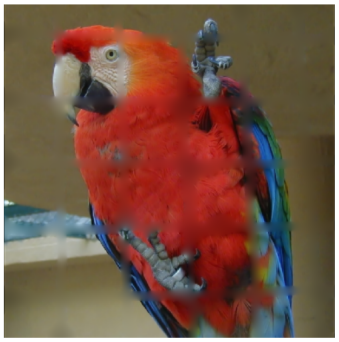
```
    ...
    chi = zeros(size(mask));
    chi_ind = find(mask == 1);
    chi(chi_ind) = 1;
    U_old = U0;
    U_new = size(U_old);
    for k = 1:ts_n

        U_new = U_old+L_X*U_old+U_old*L_Y+dt*(chi.*(im-U_old)
   );
        U_old = U_new;
    end
    Irestored = U_new;

    end
```

Listing 5: Codice Matlab: Inpainting

Come è possibile vedere dal codice nella Lista 5 si risolve la PDE con un metodo iterativo con passo temporale $k = 1, 2, .., ts\_n$. Si usa un metodo esplicito, quindi si calcola il nuovo U_{new} a partire dal valore passato U_{old}, ricordando che come condizione iniziale si è definito $U_0 = zeros(size(I))$. Inoltre si ha la presenza del *Fidelity-Term* $chi.*(im-U_{old})$ che ha il compito, nella zona esterna a quella da restaurare $\Omega \setminus D$, di mantenere l'immagine sulla quale si sta facendo inpainting il più simile possibile all'immagine iniziale. Questo è stato implementato in Matlab con la matrice *chi* che ha la stessa dimensione della immagine di partenza e ha valori pari ad 1 solo al di fuori della zona danneggiata D e uguali a 0 nella zona danneggiata.

Alla fine dell'iterazione si ottiene l'immagine restaurata così come si può vedere in Figura 4. Inoltre si nota che non si ha più il problema lungo i bordi, che come visto nella Figura 3, si verificava nel caso in cui non si consideravano le condizioni al contorno sui bordi dell'immagine.

(a) Grayscale PDE inpainting



(b) RGB PDE inpainting

Figura 4: Inpainting per diffusione

3.1 Peak Signal-To-Noise ratio

Per poter determinare la bontà dell'immagine ottenuta attraverso le tecniche di inpainting si utilizza il Peak Signal to Noise Ratio(PSNR). L'unità di misura del PSNR è il *Decibel* e maggiore è tale valore, migliore sarà la qualità dell'immagine ricostruita . Per poter calcolare il PSNR si deve calcolare prima il Mean Square Error (MSE), ovvero l'errore quadratico medio tra l'immaggine buona e quella che è stata ottenuta dall'inpainting e con questo si calcola il PSNR ovvero l'errore di picco. Minore è il valore del MSE minore sarà l'errore.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} ||I(i,j) - I_{restored}(i,j)||_2^2$$

$$PSNR = 10 \log(\frac{MAX_i^2}{MSE})$$

In questo caso si sta calcolando il PSNR di immagini dove i valori dei pixel sono espressi in binario a 8 bit, quindi $MAX_i = 255$. Nel caso i valori dei pixel fossero espressi in double, allora $MAX_i = 1$.

Si è implementato il calcolo del PSNR in Matlab usando la funzione $psnr(A, ref)$ disponibile nella libreria Image Processing Toolbox. Il PSNR che si è ottenuto con questo algoritmo è di 34.47 dB.

```
peaksnr = psnr(Irestored,Igood);
```

Listing 6: Codice Matlab calcolo PSNR

3.2 Conclusione

Il metodo utilizzato in questo caso per eseguire il restauro dell'immagini, come già visto, è quello di utilizzare l'equazione simile a quella del calore in due dimensioni in cui è presente un ulteriore termine che in letteratura è definito *Fidelity-Term* per i motivi sopra descritti. Tale approccio però è robusto solamente nel caso in cui si hanno immagini uniformi poiché si ha una diffusione isotropa sull'intera immagine e quindi tale equazione non tiene conto degli edge che sono le regioni dell'immagine in cui è più difficile ottenere un risultato dell'inpainting ottimale. Un metodo alternativo che permette di avere una minore diffusività nelle zone in cui sono presenti gli edge è l'utilizzo di PDE non lineari, un esempio è la famosa equazione di Perona-Malik che permette di ridurre la diffusione in quelle regioni dell'immagine che hanno un'alta probabilità di essere degli edge.