

Progetto di Calcolo Numerico: Deblur

Luigi Manieri, Andrea Loretti, Andrea Schinoppi

Febbraio 2021

1 Introduzione

Il problema che ci è stato posto è il seguente:
un'azienda vuole commercializzare un dispositivo di acquisizione immagini, del quale si sa che acquisisce con rumore Gaussiano additivo e sfocatura Gaussiana.

Il modello è quindi: $b = Ax_{true} + \eta$

Dove $A \in R^{mn}$ è la matrice di sfocamento, $\eta \sim \mathcal{N}(0, \sigma^2)$ rumore additivo con varianza σ^2 incognita, $b \in R^{mn}$ è la vettorizzazione dell'immagine corrotta (acquisita) $B \in R^{m \times n}$, $x_{true} \in R^{mn}$ è la vettorizzazione dell'immagine originale $X_{true} \in R^{m \times n}$.

L'obiettivo è quello di ricostruire le immagini originali a partire da quelle generate con sfocatura e rumore additivo.

1.1 Immagini campione

La prima immagine su cui abbiamo lavorato è quella presente nella libreria skimage.data chiamata camera():



Le successive immagini dovevano rispettare alcuni criteri:

- Immagine "fotografica", ovvero con molti dettagli e livelli di grigio sfocati.
- Immagine contenente del testo, che diventerà difficilmente leggibile dopo il processo di sfocamento
- Immagine "geometrica", ovvero con pochi dettagli e contorni netti e ben contrastati

Abbiamo dunque selezionato le tre seguenti, importate tutte dal sito i.imgur.com grazie a skimage.io.imread(), avendo cura di porre il flag "as_gray" = True per importarle in bianco e nero:



2 Analisi dei risultati per l'immagine camera()

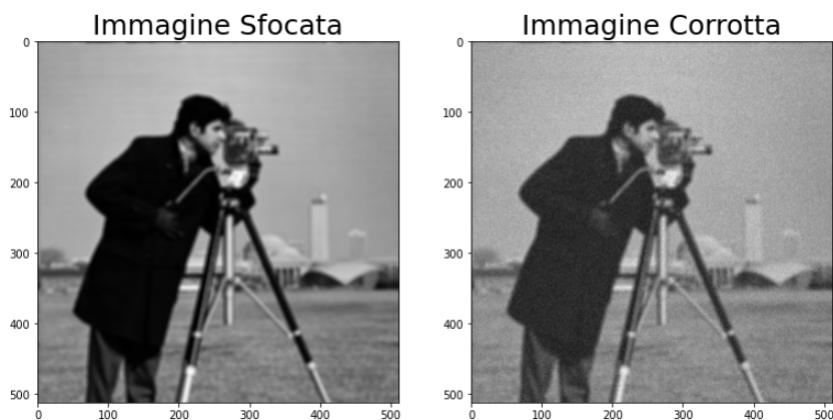
2.1 Aggiunta di sfocatura e rumore additivo

Dopo aver importato la prima immagine, salvandola in una variabile X, abbiamo aggiunto la sfocatura Gaussiana effettuando il prodotto tra X ed A, dove A è la matrice di sfocamento di raggio $d = 7$ e varianza $\sigma = 0.5$.

Il risultato è stato salvato nella variabile X_{blur} .

In seguito è stato generato il rumore η con varianza iniziale $\sigma = 0.1$ ed è stato aggiunto a X_{blur} per ottenere l'immagine iniziale con sfocatura e rumore additivo (chiamata immagine "corrotta" e salvata nella variabile b).

I risultati sono stati i seguenti:



Prima di proseguire, creiamo due variabili molto importanti: max_it e STOP che rappresentano rispettivamente il numero massimo di iterazioni che possono essere eseguite prima di interrompere un ciclo e un valore molto piccolo che indica raggiunto quale ordine di grandezza sarà necessario fermarsi.
Inizialmente max_it è stato posto a 50 e STOP a 10^{-6}

2.2 Ricostruzione dell'immagine originale

Il problema di ricostruire l'immagine originale partendo dalla sua corruzione b , si può riscrivere come un problema di ottimizzazione:

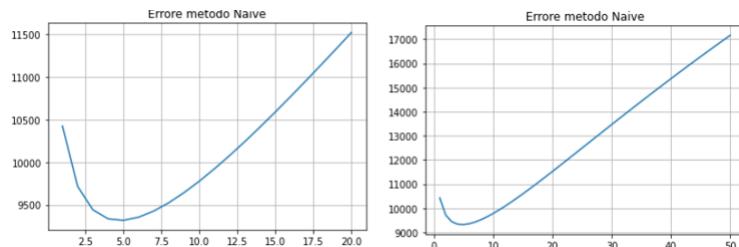
$$x^* = \arg \min_x \frac{1}{2} \|Ax - b\|_2^2$$

2.2.1 Metodo del gradiente (naive)

Per cominciare è stato utilizzato l'algoritmo di discesa del gradiente con scelta del passo tramite backtracking con i parametri $\alpha = 1.1$, $\rho = \frac{1}{2}$ e $c_1 = \frac{1}{4}$.

In seguito è stato ridotto il numero massimo di iterazioni da 50 a 20 per poter visualizzare il grafico in una zona più ristretta.

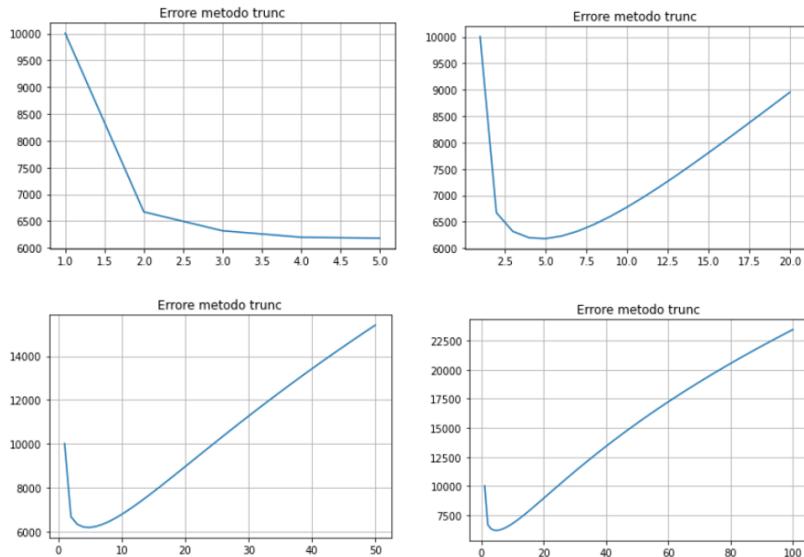
I grafici dell'errore sono risultati essere i seguenti:



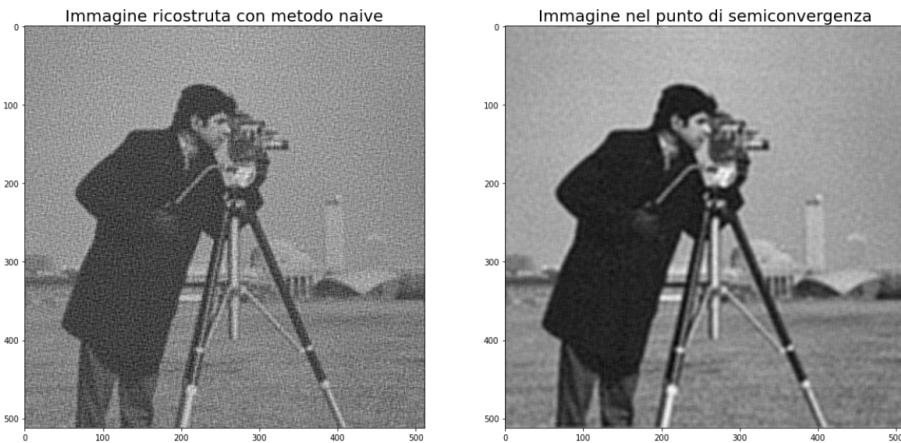
2.2.2 Metodo del gradiente con punto di semiconvergenza

Successivamente abbiamo modificato il metodo naive affinché restituisse l'errore tra l'immagine ricostruita al passo k e l'immagine originale e abbiamo cercato per quale k si avesse semiconvergenza.

Per farlo abbiamo cercato il punto in cui l'errore del metodo naive era minore. Questa volta abbiamo creato i grafici dell'errore utilizzando quattro max_it differenti: il valore del punto di semiconvergenza, 20, 50 e 100:



Infine abbiamo stampato le immagini:



Non è difficile accorgersi della ripida salita dell'errore superato il punto di semiconvergenza causata dal rumore additivo, infatti è chiaro anche guardando le foto che nel punto in cui l'errore è minimo, l'immagine risulta più nitida e meno disturbata.

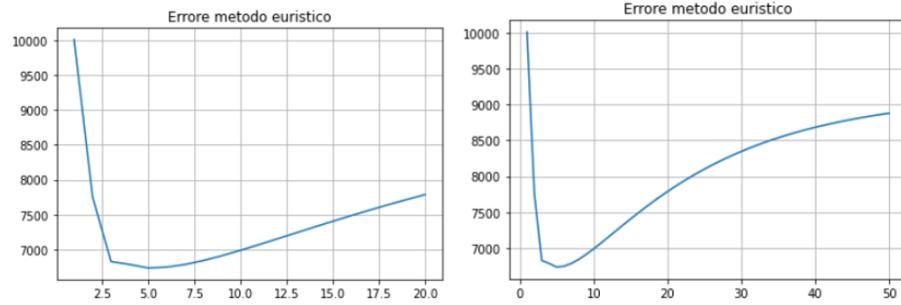
2.2.3 Problema regolarizzato

Per risolvere la semiconvergenza, si introduce il problema regolarizzato

$$x^* = \arg \min_x \frac{1}{2} \|Ax - b\|_2^2 + \frac{\lambda}{2} \|x\|_2^2$$

con $\lambda > 0$ parametro di regolarizzazione.

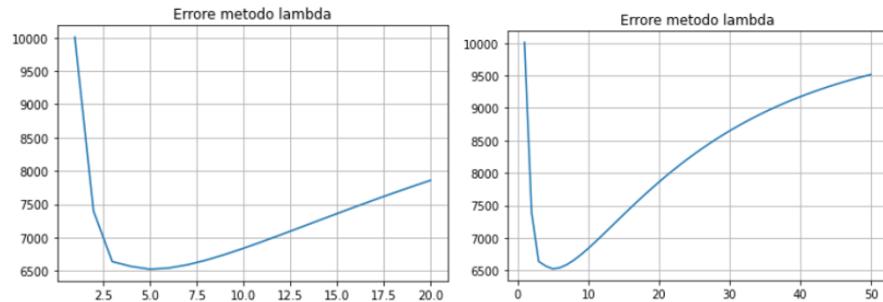
Per prima cosa abbiamo utilizzato un lambda euristico scelto da noi pari a $\lambda = 0.04$ e il grafico dell'errore è stato il seguente:



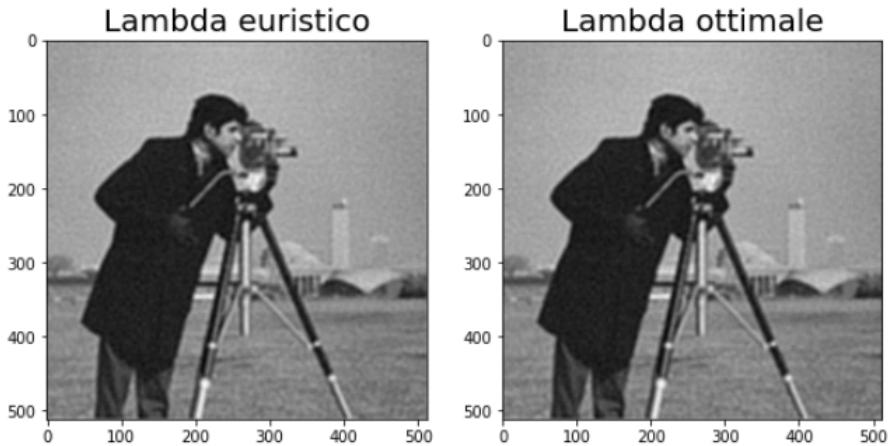
In seguito abbiamo stimato un lambda ottimale pari a circa $\lambda = 0.036$ grazie al principio di discrepanza che prevede di scegliere il più grande λ tale che

$$\|Ax_\lambda - b\|_2^2 \leq \|\eta\|_2^2$$

Anche in questo caso abbiamo il grafico dell'errore, sempre dopo 20 e poi 50 iterazioni:

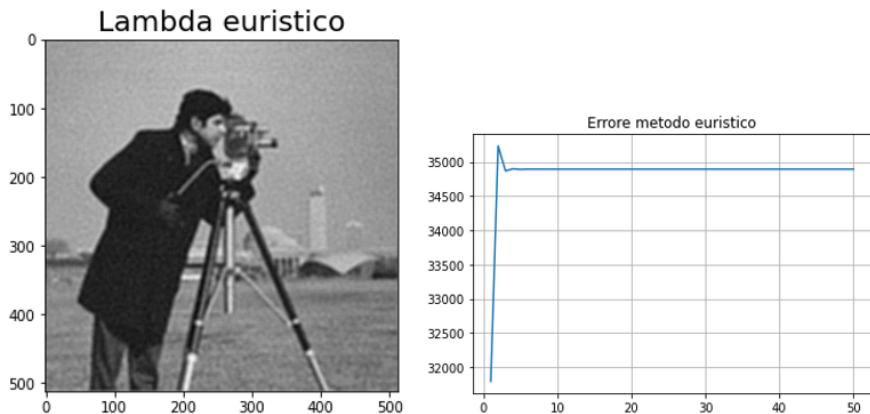


Infine abbiamo confrontato le immagini ricostruite con i due diversi lambda:



2.2.4 Risultati con λ molto grande

Abbiamo deciso di verificare cosa sarebbe successo se avessimo posto il nostro lambda euristico $\lambda_{eur} = 1$.

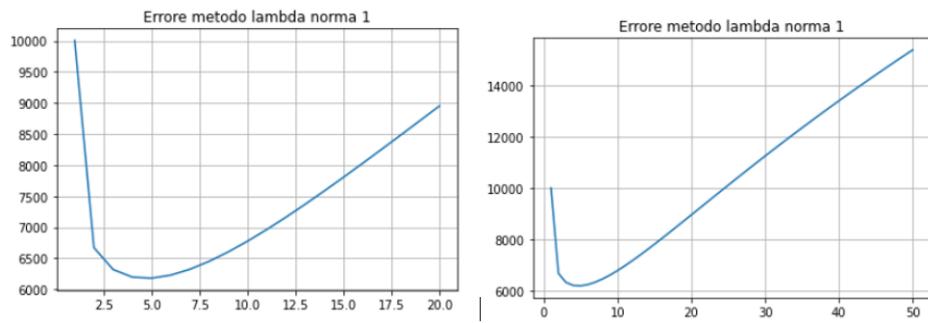


Il risultato per quanto riguarda l'immagine ricostruita è stato abbastanza sorprendente in quanto pensavamo che la ricostruzione sarebbe stata molto peggiore. Per quanto riguarda invece l'errore ci aspettavamo sarebbe stato molto alto, e così è stato.

2.2.5 Problema regolarizzato con norma 1

Questa volta come regolarizzatore è stata utilizzata la norma 1, ovvero:
 $x^* = \arg \min_x \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1$ e funzione obiettivo con gradiente pari ad $A^T(Ax - b) + \lambda \operatorname{sign}(x)$.

In questo caso abbiamo seguito gli stessi passaggi del metodo precedente ma con il nuovo regolarizzatore e ottenuto i seguenti errori:



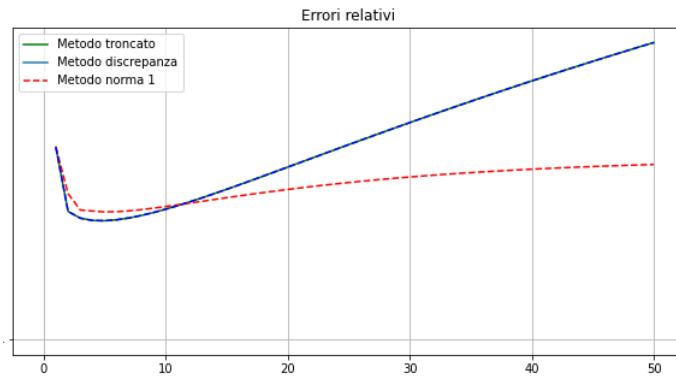
L'immagine ottenuta in questo caso è stata:



2.3 Confronto degli errori

Dopo aver svolto tutti i passaggi abbiamo ottenuto i seguenti errori relativi a cui abbiamo affiancato i corrispondenti errori percentuali, approssimati:

- Errore relativo metodo troncato: 0.0903961323476625 → 9%
- Errore relativo metodo discrepanza: 0.0954306472839483 → 9.5%
- Errore relativo metodo norma 1: 0.09039584155265878 → 9%
- Errore relativo immagine corrotta: 0.20823578237296936 → 20%

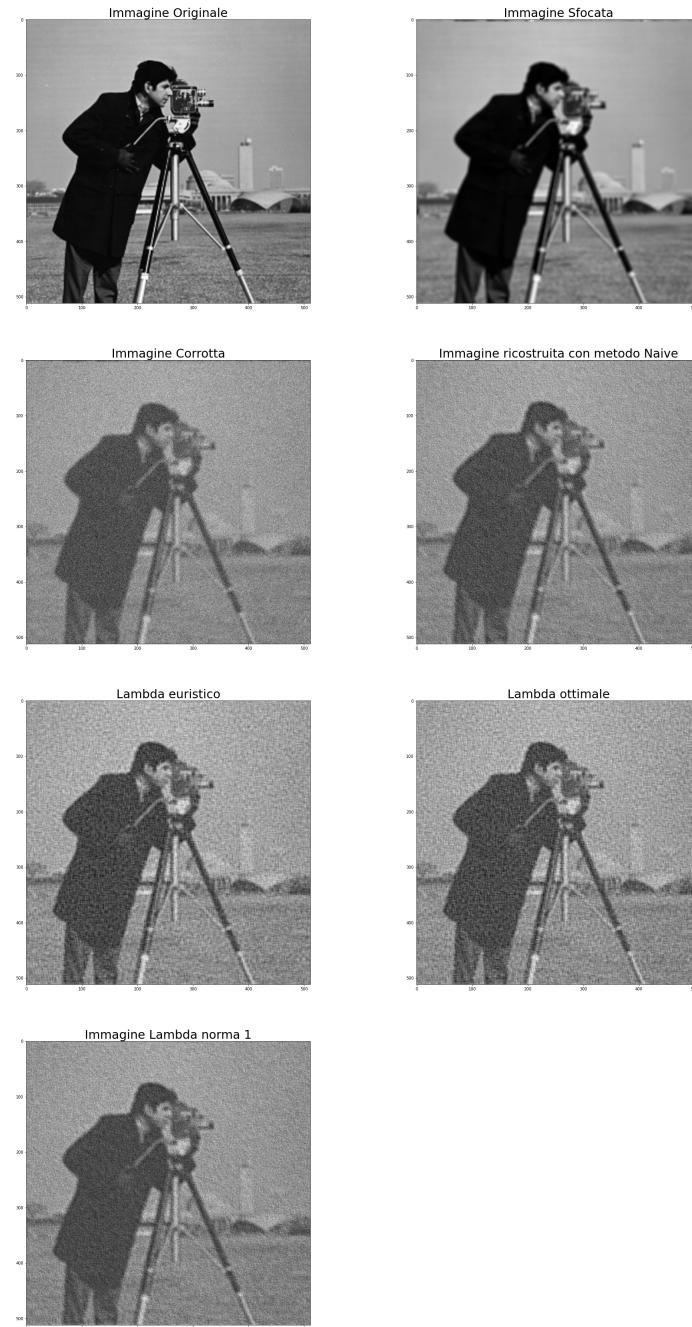


Abbiamo inoltre calcolato il valore del PSNR per i vari metodi, qui espressi in dB:

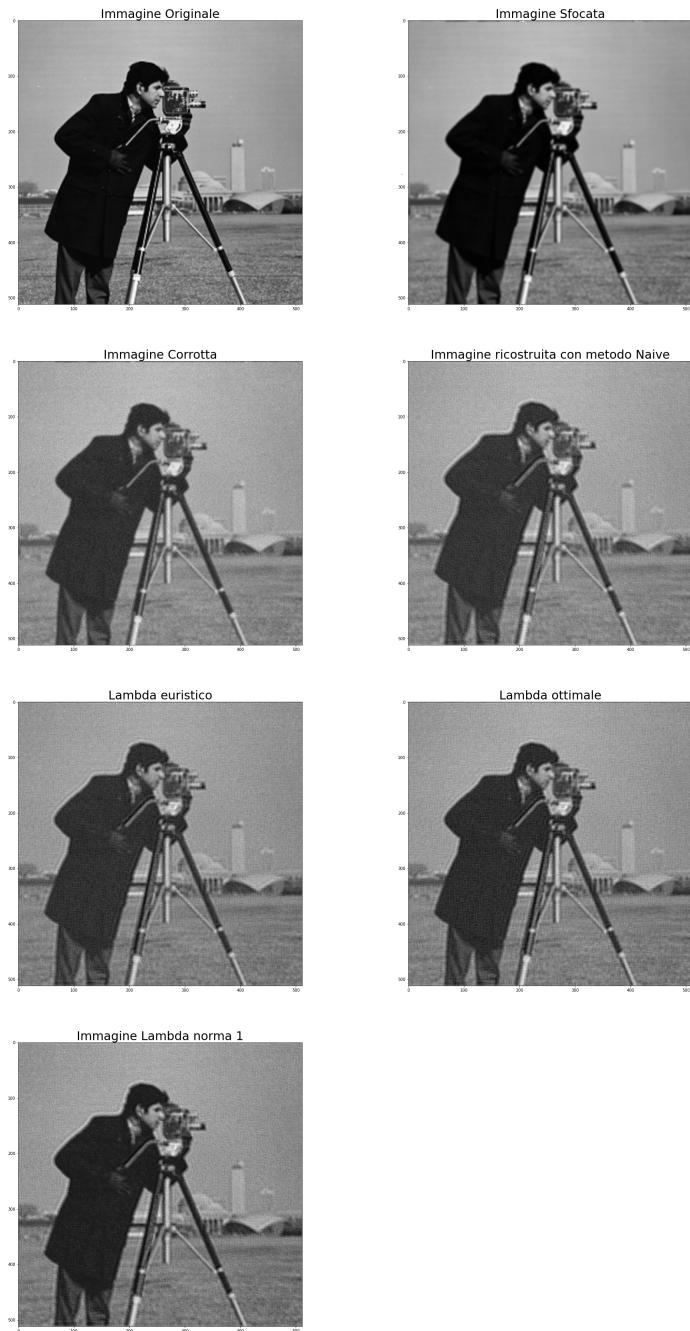
- PSNR metodo naive con troncamento x_{trunc} : 80.40908444937372
- PSNR metodo con lambda calcolato tramite discrepanza x_{disc} : 80.10096823791793
- PSNR metodo norma 1 come regolarizzatore x_{lambda_orm1} : 80.40905237211521
- PSNR immagine corrotta: 80.70018738963859

3 Variazione dei parametri

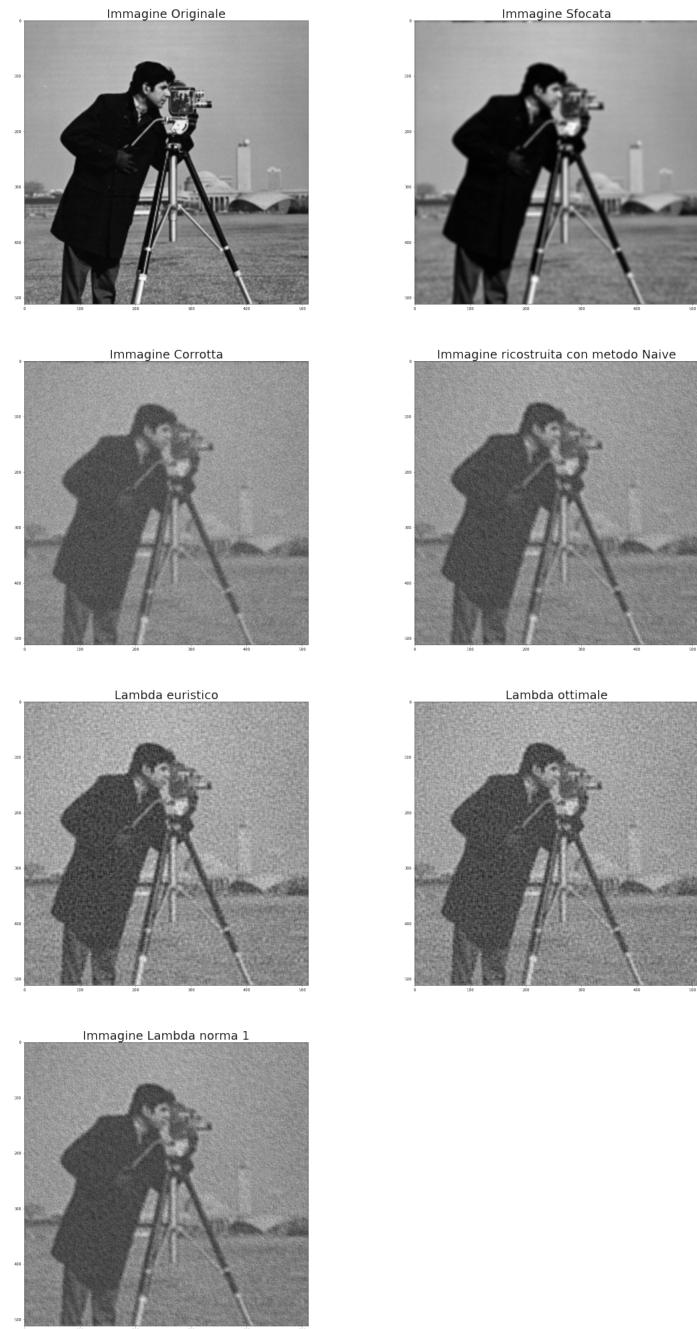
3.1 $\sigma = 0.3$, d=7



3.2 $d = 5, \sigma = 0.1$



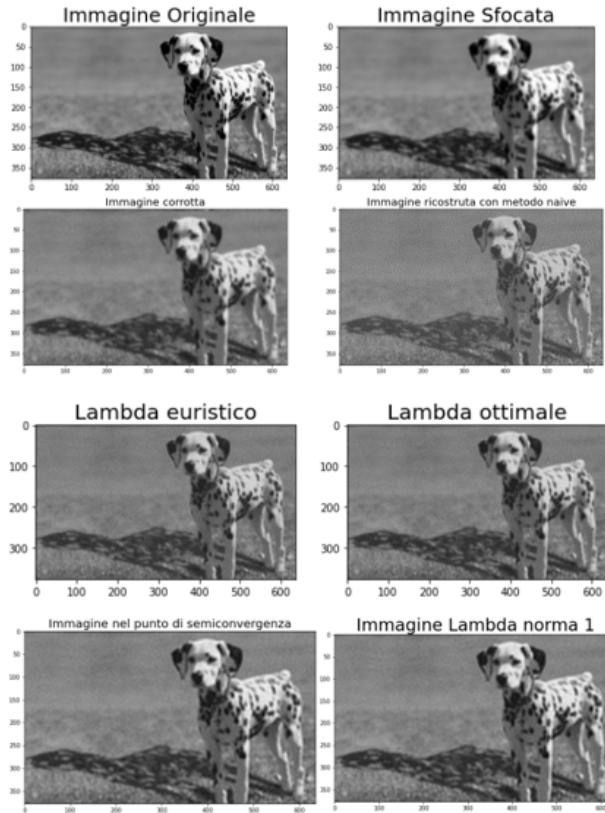
3.3 $\sigma = 0.3$, $d = 10$



4 Utilizzo delle altre immagini

4.1 Immagine fotografica

In questo caso l'aggiunta di sfocatura e rumore additivo non inficia particolarmente la visione dell'immagine: il soggetto resta piuttosto distinguibile, così come l'ombra che proietta e lo sfondo.



In questo caso gli errori che abbiamo ottenuto sono i seguenti:

Errore relativo metodo troncato: $0.1027779313511707 \rightarrow 10.2\%$

Errore relativo metodo discrepanza: $0.10891168783577426 \rightarrow 10.8\%$

Errore relativo metodo norma 1: $0.10275317087076742 \rightarrow 10.2\%$

Errore relativo immagine corrotta: $0.22634142360074902 \rightarrow 22.6\%$

Per quanto riguarda invece i valori del PSNR, sempre espressi in dB:

- PSNR metodo naive con troncamento x_{trunc} : 81.14100086854138
- PSNR metodo con lambda calcolato tramite discrepanza x_{disc} : 80.81534016749174
- PSNR metodo norma 1 come regolarizzatore x_{lambda_norm1} : 81.13324080585483
- PSNR immagine corrotta: 81.15959761048735

4.2 Immagine con testo

Contrariamente a quanto avvenuto in precedenza, in questo caso l'aggiunta di sfocatura e rumore additivo ha compromesso la possibilità di leggere il testo scritto con carattere più piccolo presente nell'immagine (ad esempio gli ingredienti)



Gli errori ottenuti sono i seguenti:

Errore relativo metodo troncato: $0.11742309085437944 \rightarrow 11.7\%$

Errore relativo metodo discrepanza: $0.12268418040813897 \rightarrow 12.2\%$

Errore relativo metodo norma 1: $0.11743099022286684 \rightarrow 11.7\%$

Errore relativo immagine corrotta: $0.23714523481196195 \rightarrow 23.7\%$

I valori del PSNR (dB):

PSNR metodo naïve con troncamento: 79.29910739332993

PSNR metodo lambda calcolato tramite discrepanza: 78.91139911488125

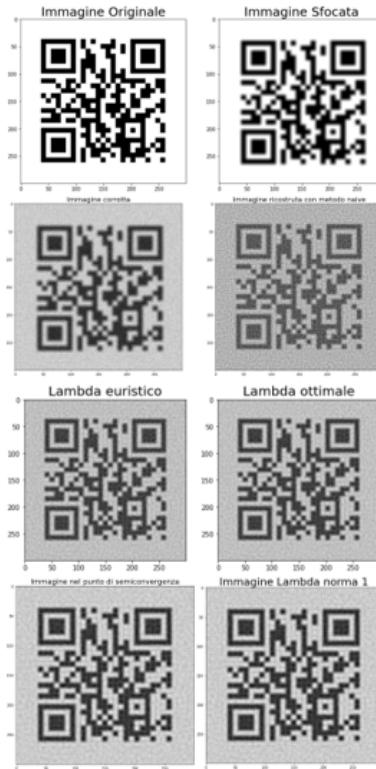
PSNR metodo norma 1 come regolarizzatore: 79.29146142837601

PSNR metodo immagine corrotta: 79.89459347371215

4.3 QR code

Nel caso del QR code non abbiamo ottenuto effetti particolarmente significativi a livello visivo, infatti per l'occhio umano il codice QR resta visibile ma non comprensibile.

Per un lettore di codici QR invece la situazione cambia: il tempo impiegato a decifrare il codice dopo le varie ricostruzioni era sempre molto maggiore rispetto al tempo impiegato per decifrare l'originale.



In quest'ultimo caso gli errori relativi ottenuti sono stati i seguenti:
Errore relativo metodo troncato: $0.18411461003662002 \rightarrow 18.4\%$
Errore relativo metodo discrepanza: $0.1878934809975741 \rightarrow 18.7\%$
Errore relativo metodo norma 1: $0.18293304903183702 \rightarrow 18.2\%$
Errore relativo immagine corrotta: $0.3934450759387857 \rightarrow 39.3\%$

Per quanto riguarda il PSNR (dB):

PSNR metodo naive con troncamento: 74.33049480158107

PSNR metodo lambda calcolato tramite discrepanza: 73.787621377772

PSNR metodo norma 1 come regolarizzatore: 74.39951483578898

PSNR metodo immagine corrotta: 73.91891508061575

4.3.1 Comparazione errori QR

Nel caso del QR code abbiamo notato una maggior differenza nel grafico che confronta gli errori rispetto alle altre immagini:

