Alma Mater Studiorum · Università di Bologna

**SCUOLA DI SCIENZE**
**Corso di Laurea in Informatica**

# Rimozione di artefatti da movimento in un'immagine con tecniche di deep learning

**Relatore:**
**Chiar.ma Prof.ssa**
**Elena Loli Piccolomini**

**Presentata da:**
**Eduard Marchidan**

**Sessione II**
**2020-21**

# Introduzione

La sfocatura delle immagini è un problema ricorrente nel campo fotografico e medico, dovuto alla mancanza di messa a fuoco, movimento dello strumento di cattura dell'immagine o del soggetto catturato.
Ciò causa vari problemi, poichè nel caso di una fotografia di un soggetto in movimento potrebbe non essere più possibile riscattare la foto, o in campo medico se durante una tomografia il soggetto si muove potrebbe essere necessario rieseguire la scansione sottoponendo il soggetto ad ulteriori radiazioni.
Il problema è di difficile risoluzione perchè è necessario eseguire una operazione di deconvoluzione. Generalmente nella deconvoluzione abbiamo solo l'immagine degradata in input è ciò lo rende un problema inverso malposto poichè è difficile, da un'immagine sfocata, trovare l'immagine non degradata.
Nel tempo sono stati sviluppati numerosi metodi matematici che si avvalgono di algoritmi di ottimizzazione, analisi statistiche, metodi geometrici e algebrici.
Più recentemente con la diffusione delle reti neurali e poi delle reti convoluzionali la ricerca nel campo si è spostata su di esse per la loro capacità di inferire i dati mancanti di un dataset incompleto grazie all'allenamento su dataset completi di dati simili.
In particolare in questo lavoro verrà analizzata una tecnica di Deep Prior composta da 2 reti generative non aventi bisogno di allenamento pregresso.
Lo scopo di questa tesi è duplice:

1. Modificare la tecnica di Deep Prior, permettendogli di funzionare su immagini anche di grandi dimensioni, diminuendo il tempo e la memoria richiesta per l'esecuzione.

2. Modificare il rumore in input della rete generativa, provando varie distribuzioni di probabilità e aggiungendo dati strutturali dell'immagine direttamente nel rumore per determinare se vi è un effetto sui risultati.

Nel primo capitolo verranno esposte le basi matematiche dei problemi di convoluzione e deconvoluzione mostrando anche esempi e applicazioni per poi passare ad analizzare i due metodi di ottimizzazione più usati.
Nel secondo capitolo vengono mostrati il metodo Deep Prior e la tecnica SelfDeblur,

basata su di esso, presa in oggetto.
Nel terzo capitolo sono proposte le modifiche alla tecnica di Deep Prior presa in oggetto con esempi del loro funzionamento.
Nell'ultimo capitolo sono esposti i risultati con dati numerici e figure per valutare visivamente le soluzioni.

# Indice

# Capitolo 1

# Convoluzione e Deconvoluzione

## 1.1 Convoluzione Lineare

La convoluzione è un'operazione che consiste nell'effettuare una combinazione lineare dei pixel vicini utilizzando un set predefinito di pesi chiamato *kernel*. Data un'immagine $x$ e un kernel $k$, la loro convoluzione sarà:

$$y = k \otimes x \tag{1.1}$$

$y$ è l'immagine risultante dalla filtrazione di $x$ tramite il kernel $k$, la convoluzione avviene eseguendo l'operazione:

$$y(m,n) = k(m,n) * x(m,n) = \sum_{p=0}^{m-1} \sum_{q=0}^{n-1} k(p,q)x(m-p, n-q) \tag{1.2}$$

In altre parole ciascun pixel di $y$ è dato dalla moltiplicazione dei pixel adiacenti a quello selezionato di $x$ per i corrispondenti pixel in $k$. Come si può notare in figura 1.1, a causa della convoluzione, $y$ avrà dimensione $|x| - \frac{|k|}{2}$, questo è detto *effetto wraparound* poichè avviene ai bordi dell'immagine ed è facilmente risolvibile aggiungendo all'immagine originale uno *zero padding* cioè un bordo i cui pixel sono 0 in maniera da non contribuire niente alla convoluzione ma permettendole di essere eseguita anche sui pixel ai bordi. La grandezza di questo padding è $\frac{|k|}{2}$. La convoluzione lineare presenta due proprietà importanti:

- Principio di sovrapposizione
- Shift invariance

**Principio di superposizione**

$$k \otimes (f_0 + f_1) = k \otimes f_0 + k \otimes f_1 \tag{1.3}$$

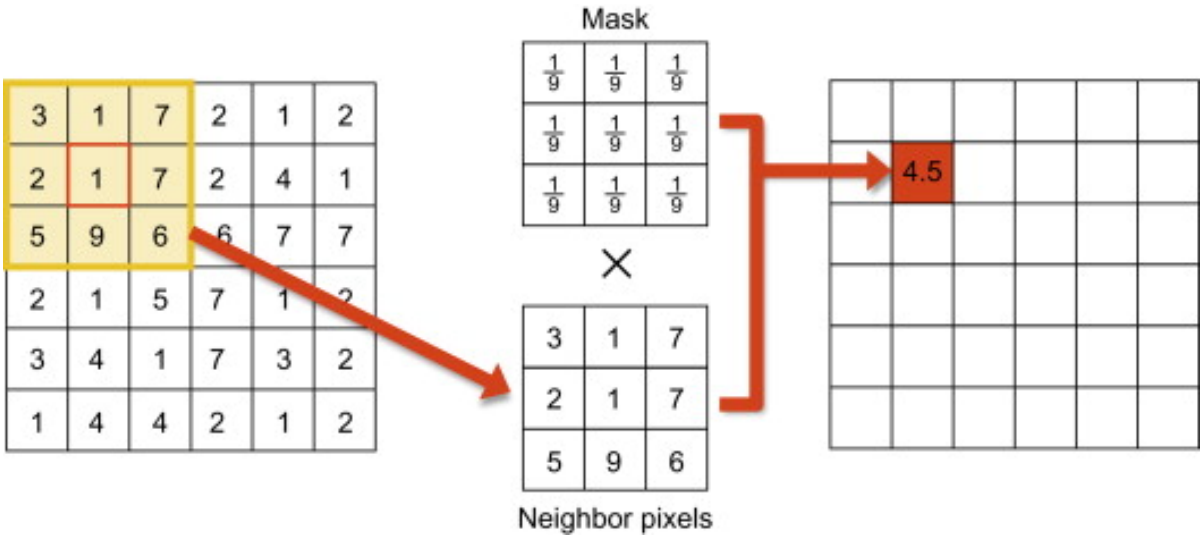



Figura 1.1: Esempio di convoluzione [5]

**Invarianza spaziale**

$$y(m,n) = x(m+p, n+q) \Leftrightarrow (k \otimes y)(m,n) = (k \otimes x)(m+p, n+q) \tag{1.4}$$

Quest'ultima proprietà ci dice che la convoluzione si comporta allo stesso modo dovunque.

## 1.2 Applicazioni della convoluzione

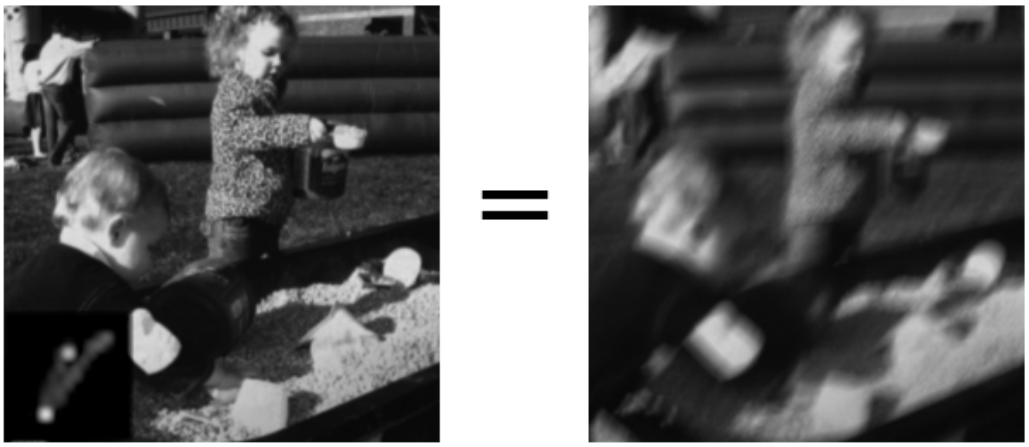
### 1.2.1 Sfocatura

La sfocatura è ottenuta tramite una convoluzione lineare, convolvendo un'immagine non corrotta con un kernel di sfocatura (*blur kernel*). La sfocatura è anche detta *smoothing* in alcuni casi, poichè "leviga" l'immagine. Esistono diversi tipi di sfocatura, tra i più comuni abbiamo:

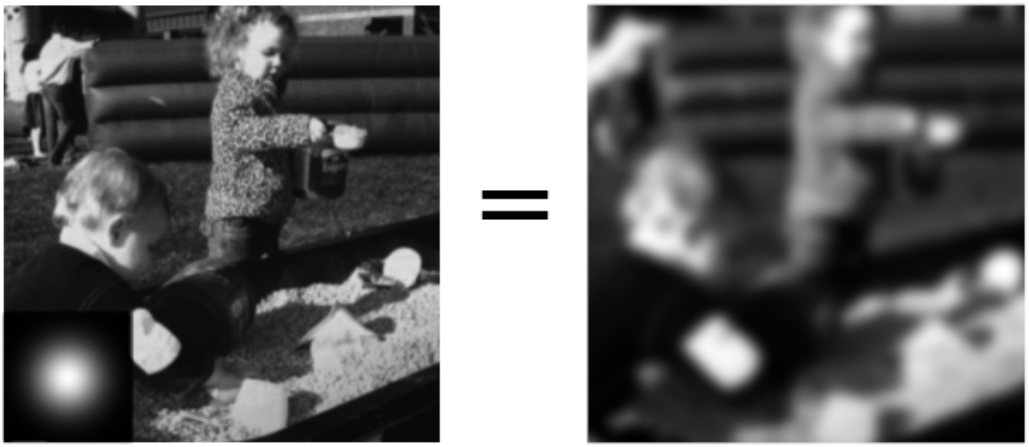
- Sfocatura gaussiana
- Sfocatura da movimento

### 1.2.2 Aumentare la nitidezza delle immagini

Dall'inglese "Sharpen", questa operazione serve a dare maggiori dettagli alle texture (soprattuto agli spigoli degli oggetti che diventano "affilati"). La sfocatura riduce le alte frequenze, per aumentare la nitidezza di un'immagine utilizziamo questa proprietà



(a) Sfocatura da movimento



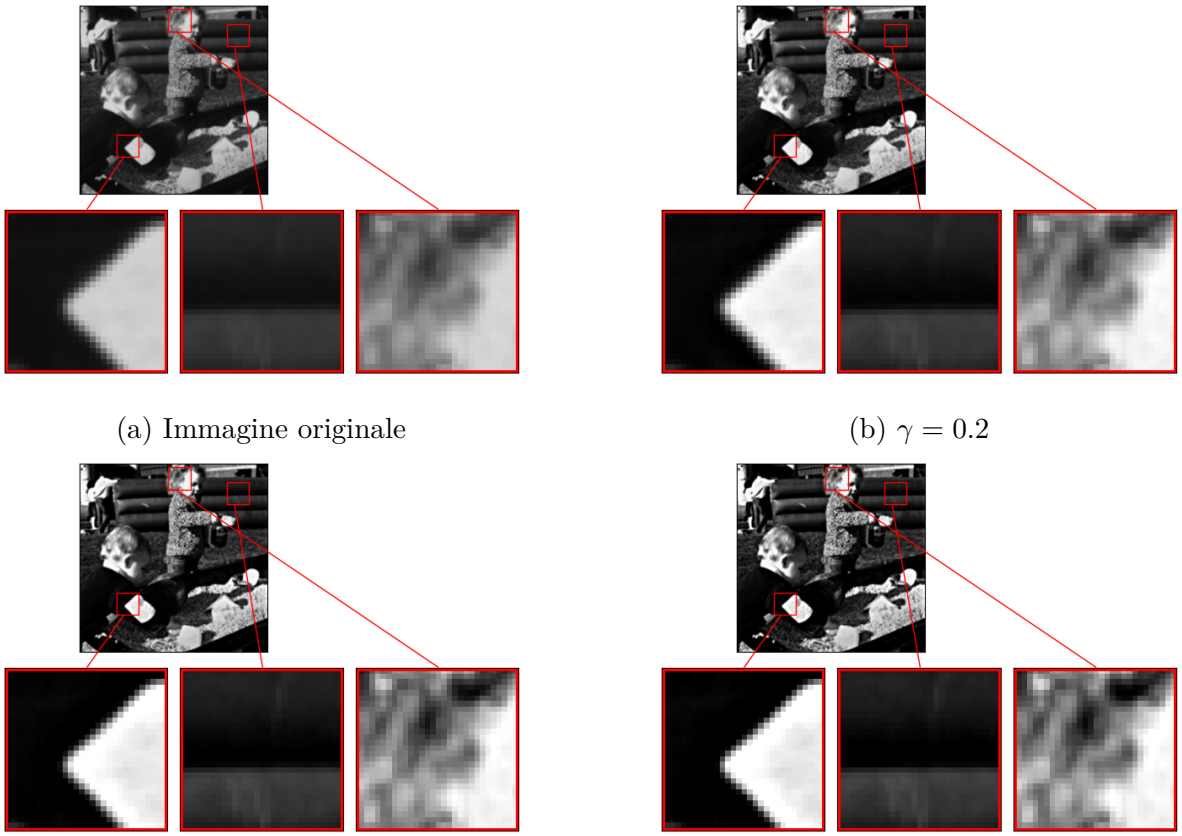
(b) Sfocatura gaussiana

Figura 1.2: Esempi di sfocatura, l'immagine viene convoluta per il kernel in basso a sinistra

a nostro vantaggio, aggiungendo una parte dell'immagine originale al risultato della sfocatura:

$$y_{sharp} = x + \gamma(x - (k_{blur} \otimes x)) \tag{1.5}$$

Dove $\gamma$ è un coefficiente per determinare quanto aumentare la nitidezza dell'immagine.



(a) Immagine originale

(b) $\gamma = 0.2$

(c) $\gamma = 0.5$

(d) $\gamma = 0.7$

Figura 1.3: Sharpening al variare di $\gamma$ utilizzando un kernel gaussiano

## 1.3 Canny Edge Detector

E' un operatore di rilevamento dei bordi (edge detection) che usa un algoritmo a più stadi per riconoscere molti bordi di tipi diversi. E' stato sviluppato da John F. Canny nel 1986 [2].
I passi dell'algoritmo sono:

- Rimuovere il rumore "levigando" (smoothing) l'immagine con un filtro gaussiano
- Trovare i gradienti d'intensità dell'immagine
- Effettuare una soppressione dei pixel non massimi (non-max suppression) per ottenere dei bordi
- Sottoporre i bordi trovati ad una soglia di isteresi

**Operatore Sobel**
L'operatore Sobel permette di trovare i gradienti d'intensità.

$$G = \sqrt{G_x^2 + G_y^2} \qquad \Theta = \arctan(\frac{G_y}{G_x}) \tag{1.6}$$

$G$ è la magnitudo, $\Theta$ è la direzione del gradiente.

**Soppressione dei pixel non massimi**

E' una tecnica per assottigliare i bordi, controlliamo se ciascun pixel è il massimo locale nel suo intorno nella direzione del gradiente. Se è un massimo locale viene matenuto altrimenti viene rimosso.
Per ogni pixel, i pixel nel suo intorno sono posizioni nelle direzioni:
orizzontale, verticale e diagonale. Compariamo ora ogni pixel con i suoi due vicini nella direzione del gradiente così solo i pixel massimi saranno mantenuti.

**Soglia di Isteresi**
La soppressione dei pixel non massimi fornisce una migliore rappresentazione dei bordi di un'immagine ma lascia alcuni bordi più intensi degli altri. I più intensi tendono ad essere bordi mentre i meno intensi potrebbero essere spigoli mal interpretati a partire dal rumore. Per capire quali sono i veri bordi utilizziamo due soglie:

- Una *alta* per cui le curve che la superano sono i veri spigoli.
- Una *bassa* per cui siamo sicuri che le curve non sono spigoli

## 1.4 Deconvoluzione

La deconvoluzione è l'operazione inversa della convoluzione, cioè a partire dal risultato di una convoluzione, vogliamo trovare o l'immagine originale sapendo il kernel o il kernel conoscendo l'immagine originale.

$$x = y \oplus k$$
$$k = y \oplus x$$

dove $\oplus$ è l'operatore di deconvoluzione.

## 1.5 Blind Deconvolution

La Blind Deconvolution è una deconvoluzione in cui vogliamo trovare sia l'immagine originale che il kernel. Ciò è estremamente difficile poichè il problema è "mal posto" cioè

ci sono infinite possibili soluzioni $(k, x)$ al problema. Un problema per essere ben posto deve rispettare tre condizioni:

- La soluzione esiste
- La soluzione è unica
- La soluzione dipende continuamente dai dati del problema

Oltre al fatto che il problema è mal posto si deve tenere di conto che lavorando con immagini digitali esse conterranno rumore casuale addittivo $\eta$. Quindi il problema di convoluzione è:

$$y = k \otimes x + \eta \tag{1.7}$$

Può essere anche generalizzato vettorizzando le immagini $x$, $y$ e usando una matrice di Toeplitz (una matrice a diagonali costanti) come:

$$y = H(k)x + \eta \tag{1.8}$$

Esistono diversi metodi per risolvere questo problema e sono divisi in due categorie:

- Metodi di ottimizzazione
- Metodi di Deep Learning

I metodi di ottimizzazione sono i metodi tradizionali e possono essere divisi a loro volta in due categorie:

- Basati su VB(Variational Bayes)
- Basati su MAP (Maximum A Posteriori)

### 1.5.1 Maximum A Posteriori

Questo metodo cerca di trovare $(x, k)$ massimizzando $p(x, k|y)$:

$$p(x, k|y) = \frac{p(y|x, k)p(x)p(y)}{p(y)} \tag{1.9}$$

$p(y|x, k)$ è una funzione di verosimiglianza gaussiana con media $k * x$ e covarianza $\lambda I$, $p(x)$ e $p(y)$ sono regolarizzatori detti *"prior"*. Possiamo trasformare 1.9 in un problema di ottimizzazione:

$$\min_{x,k} -2 \log p(x, k|y) \equiv \min_{x,k} \frac{1}{\lambda} \|k \otimes x - y\|_2^2 + g_x(x) + g_k(k) \tag{1.10}$$

Dove $g_x(x)$(normalmente della forma $g_x(x) = \sum_i g_x(x_i)$) e $g_k(k)$ sono regolarizzatori sull'immagine latente e sul kernel rispettivamente.

Il problema 1.10 può essere riformulato come un'ottimizzazione con minimi quadrati regolarizzati[3]:

$$(\hat{x}, \hat{k}) = arg\min_{x,k} \|k \otimes x - y\|^2 + \lambda\phi(x) + \tau\varphi(k) \tag{1.11}$$

dove $\phi$ e $\varphi$ sono prior. Non è triviale risolvere il problema nell'equazione 1.11 e normalmente viene risolto a passi alternati:

$$\hat{x}^{(i+1)} = arg\min_{x}(\|x \otimes \hat{k}^{(i)} - y\|^2 + \lambda\phi(x)) \tag{1.12}$$

e

$$\hat{k}^{(i+1)} = arg\min_{k}(\|\hat{x}^{(i+1)} \otimes k - y\|^2 + \tau\varphi(k)) \tag{1.13}$$

Nella maggior parte dei metodi di blind deconvolution la grandezza del kernel $(m, n)$ è un iperparametro impostato manualmente. Sarebbe ottimo se conoscessimo la grandezza del kernel a priori ma nella maggior parte dei casi non è così. Un'idea è impostarla alla grandezza del kernel di convoluzione della ground truth ma ciò porterebbe l'applicazione a comportarsi bene nel caso test ma non assicurando che si comporti bene in tutti gli altri casi.
Potremmo altrimenti impostare una grandezza del kernel abbastanza elevata da comprendere la maggior parte dei kernel di convoluzione possibili del nostro ambito d'applicazione ma come viene fatto notare in [18] questo può portare alla presenza di una quantità significativa di rumore nella stima del kernel.

### 1.5.2 Variational Bayes

Il metodo Variational Bayes può essere visto come un'estensione dell'algoritmo EM (*expectation-maximization*) che da una stima MAP del valore più probabile di ogni parametro genera l'intera distribuzione a posteriori dei parametri. Per vedere meglio questo metodo dobbiamo semplificare 1.10 come un problema di minimizzazione/massimizzazione ad un solo parametro [9]:

$$\max_k p(k|y) \equiv \min_k -2\log p(y|k)p(k) \tag{1.14}$$

dove $p(y|k) = \int p(x, y|k)dx$. Una volta ottenuto $x$, $k$ può essere trovato con tecniche tradizionali di deconvoluzione. Il problema con 1.14 è che $p(y|k)$ richiede una marginalizzazione su $x$ che è un integrale intrattabile su prior realistici. Così usiamo VB per approssimare questa marginalizzazione trovando un upper bound su $-\log p(y|k)$. Per essere applicata VB dobbiamo esprimere $p(x)$ come una GSM(Gaussian scale mixture):

$$p(x) = \exp\left[-\frac{1}{2}g_x(x)\right] = \prod_i \exp\left[-\frac{1}{2}g_x(x_i)\right] = \prod_i \int \mathcal{N}(x_i; 0, \gamma_i)p(\gamma_i)d\gamma_i \tag{1.15}$$

dove ogni $\mathcal{N}(x_i; 0, \gamma_i)$ rappresenta una gaussiana con media 0, varianza $\gamma_i$ e distribuzione a priori $p(\gamma_i)$. Dato questo $p(x)$ possiamo trovare l'upper bound:

$$-\log p(y|k) \leq -\int\int q(x,\gamma) \log \frac{p(x,\gamma,y|k)}{q(x,\gamma)} dx d\gamma = F[q(x,\gamma),k] \tag{1.16}$$

dove $F[q(x,\gamma),k]$ è chiamata energia libera, $q(x,\gamma)$ è una distribuzione arbitraria su $x$ e $\gamma = [\gamma_1, \gamma_2, ...]^T$ è il vettore delle varianze di 1.15. L'uguaglianza si ottiene quando $q(x,\gamma) = p(x,\gamma|y,k)$ ma $p(x,\gamma|y,k)$ non ha una soluzione in forma chiusa quindi è intrattabile. La VB serve proprio a restringere la forma di $q(x,\gamma)$ per poterlo approssimare. Per questo motivo fattorizziamo [15] $q(x,\gamma) = q(x)q(\gamma)$ permettendo così di trovare una soluzione approssimata. Tuttavia per risolvere $p(x)$ ad ogni iterazione dobbiamo calcolare la covarianza di $x$ dipendente da $\gamma$, cosa che richiederebbe $O(m^3)$ iterazioni dove $m$ è la grandezza della matrice. Per risolvere anche questo problema fattorizziamo di nuovo $q(x,\gamma) = \prod_i q(x_i)q(\gamma_i)$, sostituendo a 1.16 otteniamo:

$$-\log p(y|k) \leq -\int\int \prod_i q(x_i)q(\gamma_i) \log \frac{p(x,\gamma,y|k)}{\prod_i q(x_i)q(\gamma_i)} dx d\gamma \tag{1.17}$$

Possiamo quindi minimizzare l'upper bound:

$$\min_{q(x,\gamma),k} F\left[q(x,\gamma),k\right] \quad \textbf{con} \quad q(x,\gamma) = \prod_i q(x_i)q(\gamma_i) \tag{1.18}$$

**Input:** Immagine sfocata y, prior $p(x)$
**Output:** Blur Kernel k e immagine pulita x
1: while criterio di fermata non e' soddisfatto, do
2: aggiorna $q(\gamma) = \prod_i q(\gamma_i)$
3: aggiorna $q(x) = \prod_i q(x_i)$
4: aggiorna k
5: stima l'immagine latente utilizzando k

## 1.6 Applicazioni della Blind Deconvolution

**Denoising**
Lo scopo del *denoising* (rimozione del rumore) è ottenere un'immagine pulita a partire da una degradata. A volte il modello della degradazione può seguire un modello $y = x + \epsilon$ dove $\epsilon$ è campionato da una certa distribuzione. Più spesso, non conosciamo la distribuzione e dobbiamo trovarla.

(a) Immagine degradata

(b) Immagine pulita

Figura 1.4: Denoising di immagini con Deep Prior

**Super-resolution**
Lo scopo della super-risoluzione è di trovare un'immagine a bassa risoluzione $x \in \mathbb{R}^{3\times W\times H}$ e un fattore di scaling $t$ che genera un'immagine ad alta risoluzione $y \in \mathbb{R}^{3\times tW\times tH}$.
La funzione da minimizzare è:

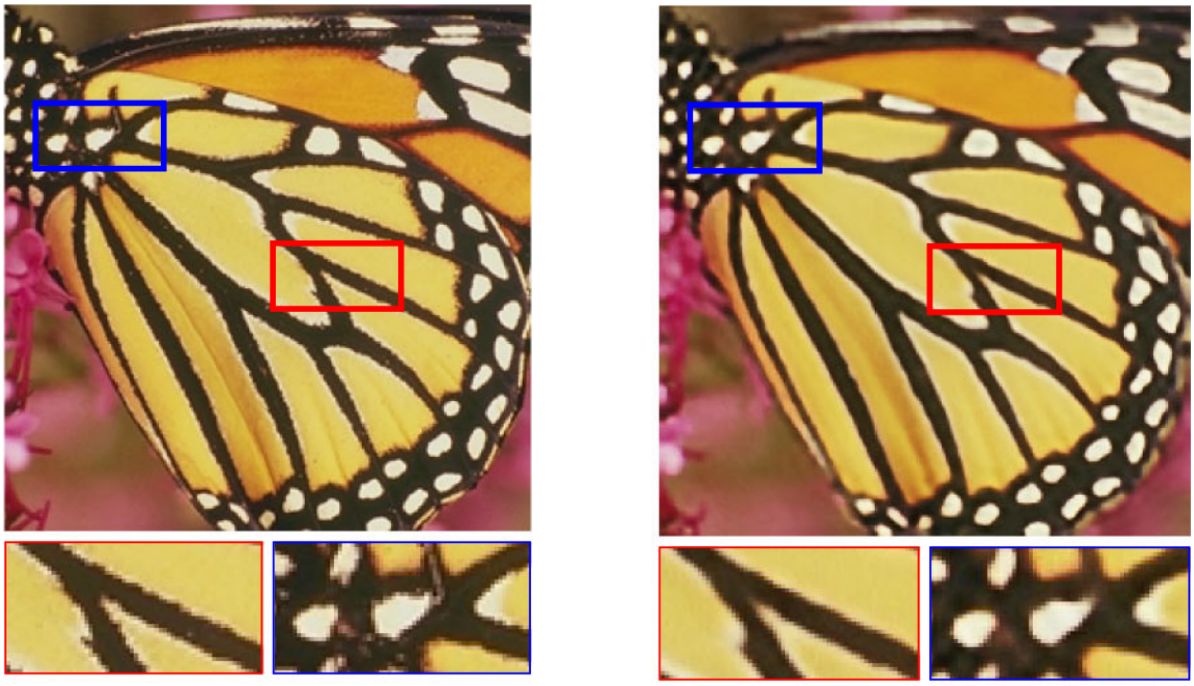
$$E(y;x) = \|d(y) - x\|^2$$

dove $d(\cdot)$ è un operatore di downsampling (sottocampionamento). Questo è evidentemente un problema mal posto, dato che esistono infinite immagini $y$ che si riducono a $x$ (l'operatore $d(\cdot)$ è iniettivo).

**Inpainting**
Nel problema di "inpainting" viene data un'immagine $x$ con dei pixel mancanti in corrispondenza di una maschera binaria $m \in \{0,1\}^{H\times W}$. Lo scopo è di riscotruire i dati mancanti. La funzione da minimizzare è :

$$E(y;x) = \|(y - x) \odot m\|^2$$

dove $\odot$ è il prodotto di Hadamard.



(a) Immagine a bassa risoluzione

(b) Immagine ad alta risoluzione

Figura 1.5: Super-risoluzione di immagini con Deep Prior

## 1.7 Neural Blind Deconvolution

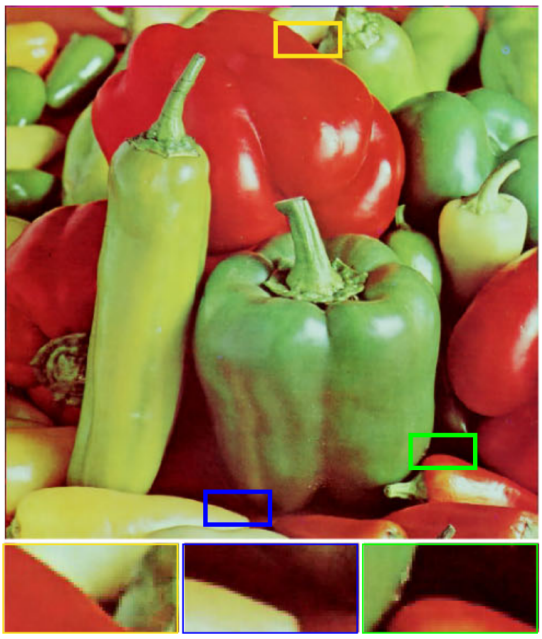
La neural blind deconvolution consiste nell'utilizzo di reti neurali per trovare $x$ e $k$. Un metodo utilizzato spesso nella letteratura [14] [16] consiste in:

1. Estrazione di feature
2. Stima del *blur kernel*
3. Ottieni l'immagine originale

L'implementazione di questi passi cambia ma l'idea è la stessa.

**Estrazione di feature**
Questo passo è molto importante poichè consiste nel trovare dei dettagli (*feature*) che riescano a dare indicazioni importanti sull'immagine anche se sfocata. Ciò è possibile poichè la convoluzione è shift invariant e quindi l'operatore è applicato su ogni pixel dell'immagine. Molti cercano gli spigoli (*sharp edges*) poichè sono in grado di mostrare con la maggior chiarezza l'operazione di convoluzione.


(a) Immagine con maschera binaria di pixel mancanti



(b) Immagine ridipinta

Figura 1.6: Inpating di immagini con Deep Prior

**Stima del *blur kernel***

Il kernel di sfocatura è calcolato a partire dal risultato della estrazione delle feature che può essere separato in $\tilde{x}$ e $\tilde{y}$.

$$\tilde{k} = arg\min_{k} \|k * \tilde{x} - \tilde{y}\|_2^2 + \gamma\|\tilde{k}\|_2^2 \tag{1.19}$$

La soluzione della minimizzazione nell'equazione 1.19 può essere risolta in spazio di fourier:

$$k = F^{-1}\frac{\overline{F\tilde{x}} \odot F\tilde{y}}{|F\tilde{x}|^2 + \gamma} \tag{1.20}$$

dove $F$ è la trasformata di fourier, $\odot$ è il prodotto di Hadamard in cui si moltiplica ciascun elemento di una matrice per il suo corrispettivo nell'altra matrice , $\overline{v}$ è l'operazione di coniugazione complessa. La divisione e il valore assoluto sono tutte operazione sugli elementi.

**Immagine originale**

A questo punto ci sono vari metodi per trovare l'immagine originale, l'idea più semplice è ottenere l'immagine iniziale nello spazio di fourier a partire dall'equazione 1.1:

$$x = F^{-1}(Fx) = F^{-1}\left(\frac{Fy}{Fk}\right) \tag{1.21}$$

Dove $F$ è la trasformata di fourier e $F^{-1}$ è l'antitrasformata di fourier che serve a tornare indietro dallo spazio di fourier. Questo metodo è piuttosto diretto e si basa sul presupposto che il kernel di sfocatura sia molto vicino a quello originale ma ciò non è detto, quindi normalmente si procede con un approccio euristico come in [14] dove dopo aver trovato l'immagine latente (non corrotta) rieseguiamo tutto il procedimento per migliorare la stima del kernel.
Un altra possibilità invece che trovare direttamente l'immagine latente nello spazio di fourier consiste nel procedere con un'ottimizzazione alternata come nel metodo MAP per trovare un'immagine latente e kernel sempre più vicini alla *ground truth*.

**L'approccio di SelfDeblur**
In questo caso invece di cercare $x$ e $k$ utilizziamo le reti generative $G_x$ e $G_k$. Possiamo inoltre rimuovere i regolarizzatori poichè le reti sono abbastanza potenti da generare dei prior adatti. Quindi il problema di neural blind deconvolution può essere formulato come:

$$\min_{G_x,G,k} \|G_k(z_k) \otimes G_x(z_x) - y\|^2 \tag{1.22}$$

Dove $z_x$ e $z_k$ sono presi dalla distribuzione uniforme. Notiamo come $z_k$ è un vettore 1D e $G_k(z_k)$ è trasformato in una matrice 2D del *blur kernel*.

## 1.8 Variazione Totale (TV)

La variazione totale o Total Variation (TV) è un regolarizzatore che protegge dalle discontinuità in attività di image processing. Per un kernel $H$ un' immagine soddisfa la relazione $x \approx Hy$ dove l'approssimazione è presente per tenere conto del rumore. L'immagine stimata $y$ può essere descritta come il minimizzatore di:

$$J_{TV} = \frac{1}{2}\|Hx - y\|^2 + \lambda TV(y) \tag{1.23}$$

Dove la norma TV è definita come:

$$TV(y) = \int_\Omega \sqrt{|\nabla y|^2} dwdh \tag{1.24}$$

Dove $\nabla y$ è il laplaciano di $y$. L'equazione 1.24 può essere approssimata a [13]:

$$TV_{fro}(y) = \sum_{i,j} \sqrt{|y_{i+1,j} - y_{i,j+1}|^2 + |y_{i,j+1} - y_{i,j}|^2 + \alpha^2} \tag{1.25}$$

Dove $\alpha$ è un termine molto piccolo che permette di differenziare $TV(y) = 0$.
l'equazione 1.25 è la formula implementata come termine di regolarizzazione per Selfdeblur$_{tv}$ più avanti nella tesi.

# Capitolo 2

# Deep Image Prior e SelfDeblur

## 2.1 Deep Image Prior

Deep Image Prior [8] è una tecnica che permette di ottenere statistiche di basso livello da un'immagine utilizzando reti neurali non allenate. Questa tecnica mostra come la struttura stessa della rete neurale è in grado di catturare informazioni importanti riguardo all'immagine, in particolare è in grado di notare l'utilizzo di sequenze di convoluzioni guardando la relazione di pixel vicini gli uni agli altri.

**Formulazione del problema come minimizzazione dell'energia**
Senza allenamento la rete non è in grado di eseguire operazioni troppo complicate come classificare oggetti ma permette di modellizzare distribuzioni condizionali di immagini $p(x|x_0)$ che si hanno generalmente durante la ristorazione di immagini dove $x$ è una versione corrotta di $x_0$. Ma nel caso di DIP invece che formulare il problema in termini di distribuzione di probabilità, lo formuliamo come minimizzazione dell'energia:

$$x^* = arg\min_x E(x; x_0) + R(x) \tag{2.1}$$

Dove $E(x; x_0)$ è un termine dipendente dall'applicazione, $x_0$ è l'immagine a bassa risoluzione, con rumore o occlusa e $R(x)$ è un regolarizzatore non strettamente correllato all'applicazione ma che cattura la generica regolarità delle immagini naturali.

**Regolarizzatori**
Un regolarizzatore o "image prior" generalmente è ottenuto a partire da un dataset utilizzando reti CNN. Il metodo DIP utilizza un regolarizzatore catturato dalla parametrizzazione della rete neurale.

$$\theta^* = arg\min_\theta E(f_\theta(z); x_0),\ x^* = f_{\theta^*}(z) \tag{2.2}$$

Il minimo (locale) $\theta^*$ è ottenuto da un ottimizzazione come la discesa del gradiente partendo da un'inizializzazione casuale di $\theta$. Poichè nessun aspetto della rete $f_\theta$ è appreso dai dati. Partendo dai pesi casuali $\theta_0$, li aggiorniamo iterativamente per minimizzare

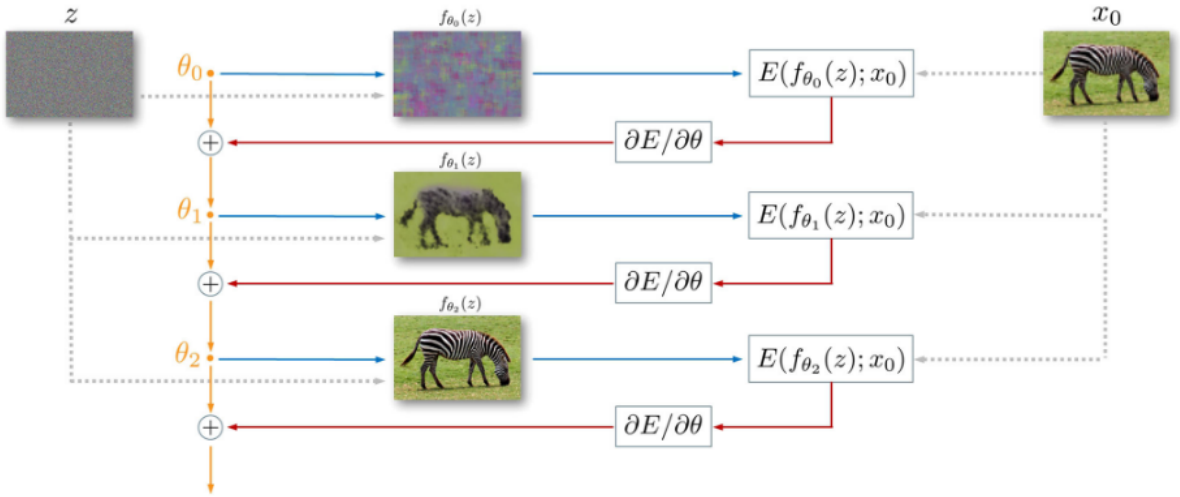



Figura 2.1: Restauro delle immagini utilizzando DIP

l'energia in 2.2. Ad ogni iterazione $t$ i pesi di $\theta$ sono mappati ad un'immagine $x = f_\theta(z)$ dove $z$ è un tensore fissato e la mappatura $f$ è una rete neurale con parametri $\theta$. Il gradiente dell'energia relativamente ai parametri $\theta$ viene poi utilizzato per aggiornare i parametri.

## 2.2 SelfDeblur

La rete SelfDeblur [12] è composta da due reti generative [4], $G_x$ che genera l'immagine non corrotta e $G_k$ che genera la matrice di convoluzione (kernel).

**Rete $G_x$**
Adottiamo una rete DIP cioè un Autoencoder asimmetrico con skip connections. Come si può vedere nella figura 2.2 i primi 5 strati dell'encoder sono connessi con gli ultimi 5 strati dell'encoder. In fondo aggiungiamo uno strato convoluzionale per ottenere l'output. Per mantenere il valore di $x$ nel range $[0, 1]$ utilizziamo una funzione sigmoide nonlineare nello strato di output.

**Rete $G_k$**
Il *blur kernel* $k$ generalmente contiene molte meno informazioni dell'immagine $x$ e quindi può essere generato da una rete più semplice. Quindi utilizziamo una rete totalmente

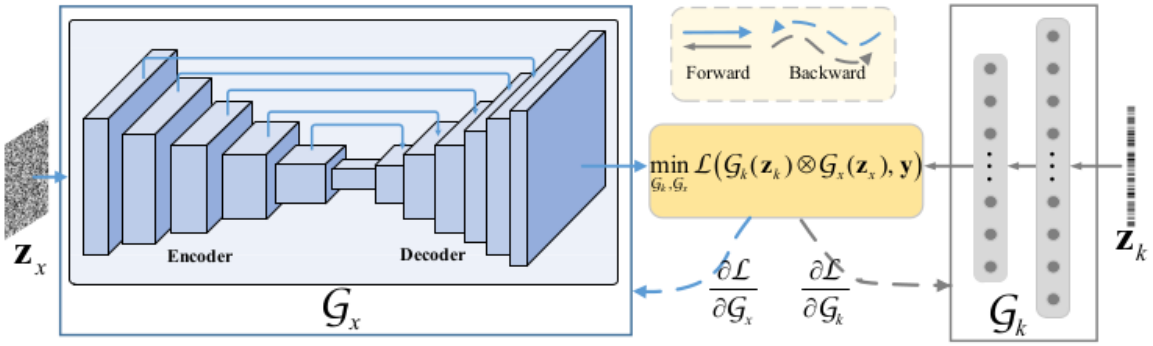



Figura 2.2: Architettura della rete SelfDeblur

connessa FCN(Fully-Connected Network). Come mostra la figura 2.2 $G_k$ prende in input un rumore 1D $z_k$ con dimensione 200 come input, presenta un hidden layer di 1000 nodi e uno strato di output di $K^2$ nodi. Per garantire che il kernel non sia negativo e che $\sum_j (G_k(z_k))_j = 1 \; \forall j$ utilizziamo una funzione SoftMax nonlineare all'output di $G_k$. Alla fine l'output 1D di dimensioni $K^2$ viene trasformato in un *blur kernel* $K \times K$.

**Neural Blind Deconvolution non vincolata usando la norma TV**

Usando il modello sopracitato otteniamo un modello non vincolato il cui risultato degrada velocemente all'aumentare del rumore dell'immagine originale. Per migliorare i risultati combiniamo $G_x$ con la norma TV per catturare gli image prior. Possiamo così riscrivere il modello come:

$$\min_{G_k, G_x} \|G_k(z_k) \otimes G_x(z_x) - y\|^2 + \lambda TV(G_x(z_x)) \tag{2.3}$$

Dove $\lambda$ denota il parametro regolarizzatore controllato dal livello di rumore $\sigma$.

### 2.2.1 Ottimizzazione

Il processo di ottimizzazione dell' equazione 2.3 può essere spiegato come un apprendimento supervisionato *zero-shot* [1] dove le reti generative $G_k$ e $G_y$ sono allenate solo con un'immagine test (l'immagine sfocata y) e nessuna *ground truth*. Il metodo proposto presenta due algoritmi di ottimizzazione differenti.

**Alternating Optimization**
Con questo metodo i parametri di $G_k$ e $G_x$ sono aggiornati in maniera alternata. Prima i parametri di $G_k$ sono aggiornati con l'algoritmo ADAM [6] fissando $G_x$, poi fissando $G_k$ e aggiornando $G_x$ analogamente.

**Input:** Immagine sfocata y
**Output:** Blur Kernel k e immagine pulita x
Preleva $z_x$ e $z_k$ da una distribuzione uniforme con seme 0
$k = G_k^0(z_k)$
for t = 1 to T do:
    x = $G_x^{t-1}(z_x)$
    calcola il gradiente rispetto a $G_k$
    aggiorna $G_k^t$ usando l'algoritmo ADAM
    $k = G_K^t(z_k)$
calcola il gradiente rispetto a $G_x$
    aggiorna $G_x^t$ usando l'algoritmo ADAM
$x = G_x^T(z_x),\ k = G_x^t(z_k)$

**Joint Optimization**
Nei tradizionali framework basati su MAP viene preferita la Alternating Optimization poichè è in grado di gestire i vincoli di non-negatività, uguaglianza e permette di trattare il risultato dell'ottimizzazione in maniera da evitare soluzioni triviali. In questo caso però l'equazione 2.3 non è vincolata e le due reti generative sono abbastanza complesse da evitare soluzioni triviali, quindi la Joint Optimization è preferita.

**Input:** Immagine sfocata y
**Output:** Blur Kernel k e immagine pulita x
1: Preleva $z_x$ e $z_k$ da una distribuzione uniforme con seme 0
2: for t = 1 to T do:
3: k = $G_k^{t-1}(z_k)$
4: x = $G_x^{t-1}(z_x)$
5: calcola i gradienti rispetto a $G_k$ e $G_x$
6: aggiorna $G_k^t$ e $G_x^t$ usando l'algoritmo ADAM
7: $x = G_x^T(z_x),\ k = G_x^t(z_k)$

### 2.2.2 Funzione obiettivo

La funzione obiettivo serve per valutare quanto vicino è il risultato della rete alla *ground-truth*. Questa funzione viene generalmente minimizzata in maniera da avere un risultato il più simile possibile alla *ground-truth*. Nel caso di SelfDeblur è richiesta la minimizzazione di $G_x$ e $G_k$ per 2.3 di modo che la differenza tra la loro convoluzione e l'immagine sfocata sia il minore possibile, ma cosa significa in pratica?
La sottrazione tra immagini non è una metrica utile per determinare la somiglianza tra di esse, [12] come molti altri, utilizzano SSIM [4.3] o MSE [4.1] per determinare la loro somiglianza.

In particolare è stata utilizzata la metrica MSE per le prime 1000 iterazioni (di 5000) in maniera da inizializzare i pesi della rete perchè MSE sia minimo, per poi utilizzare la SSIM per il resto delle iterazioni massimizzandola il più possibile. L'utilizzo di due funzioni obiettivo non è casuale infatti usando solo la SSIM come obiettivo può portare alla generazione di artefatti visivi come quelli in figura 2.3c.

(a) Immagine originale (b) Immagine con MSE (c) Immagine senza MSE

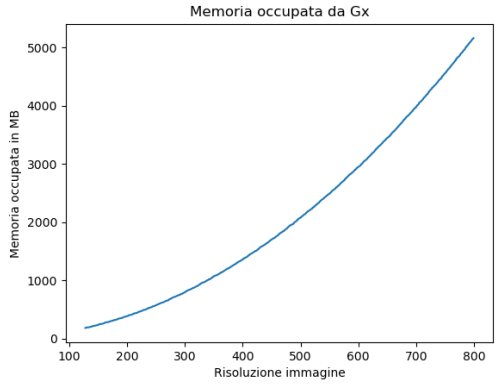
Figura 2.3: Immagine originale e risultati di SelfDeblur

# Capitolo 3

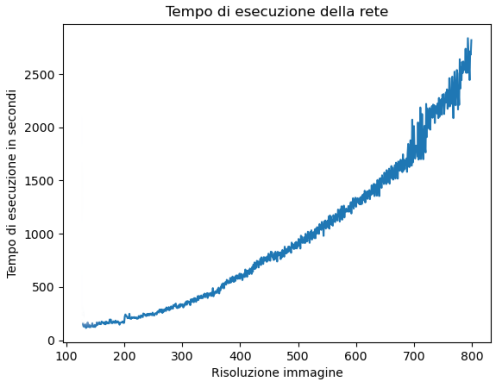
# Miglioramenti al metodo SelfDeblur

## 3.1 Implementazione basata su patches

SelfDeblur utilizza due reti generative $G_k$ e $G_x$, la prima non utilizza molto spazio in memoria essendo una FCN con soli 1000 nodi nella hidden layer. $G_x$ invece essendo un asymmetric autoencoder con skip connections richiede molta memoria: la quale cresce esponenzialmente con il crescere della risoluzione dell'immagine in input come da figura 3.1a. Il tempo di esecuzione risente dello stesso problema 3.1b.

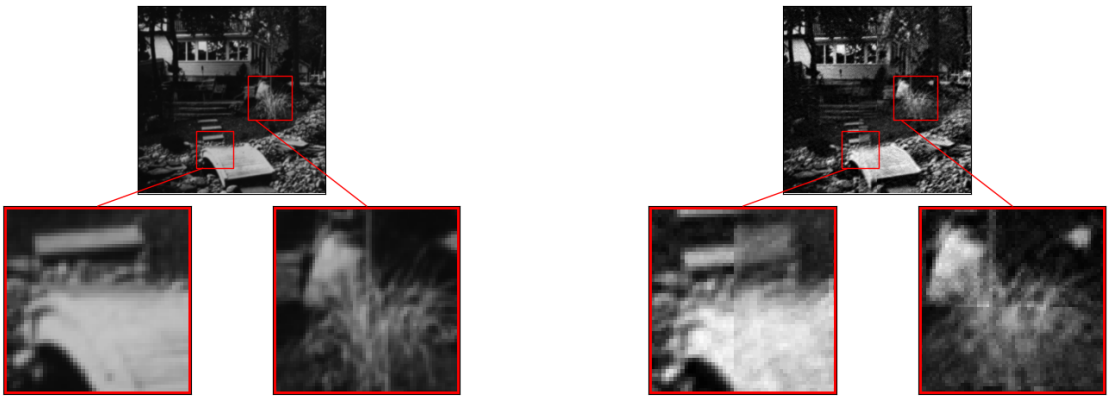



(a) Memoria occupata dalla rete $G_x$ al crescere della risoluzione dell'input




(b) Tempo di esecuzione della rete al crescere della risoluzione dell'input

Partendo da questi risultati, abbiamo pensato di eseguire la rete su patch dell'immagine originale con un overlap pari alla grandezza del kernel in input. Il problema è che la rete genererà un kernel diverso per ogni patch, generalmente spostati ma a volte anche kernel deformati 3.3.



(a) Immagine originale (b) Immagine ricostruita con patch

Figura 3.2: Ricostruzione immagine con patch

Un kernel spostato causerà una translazione dell'immagine originale a causa della convoluzione portando a degli artefatti nella zona di overlap e lungo i bordi ben visibili in figura 3.2b.

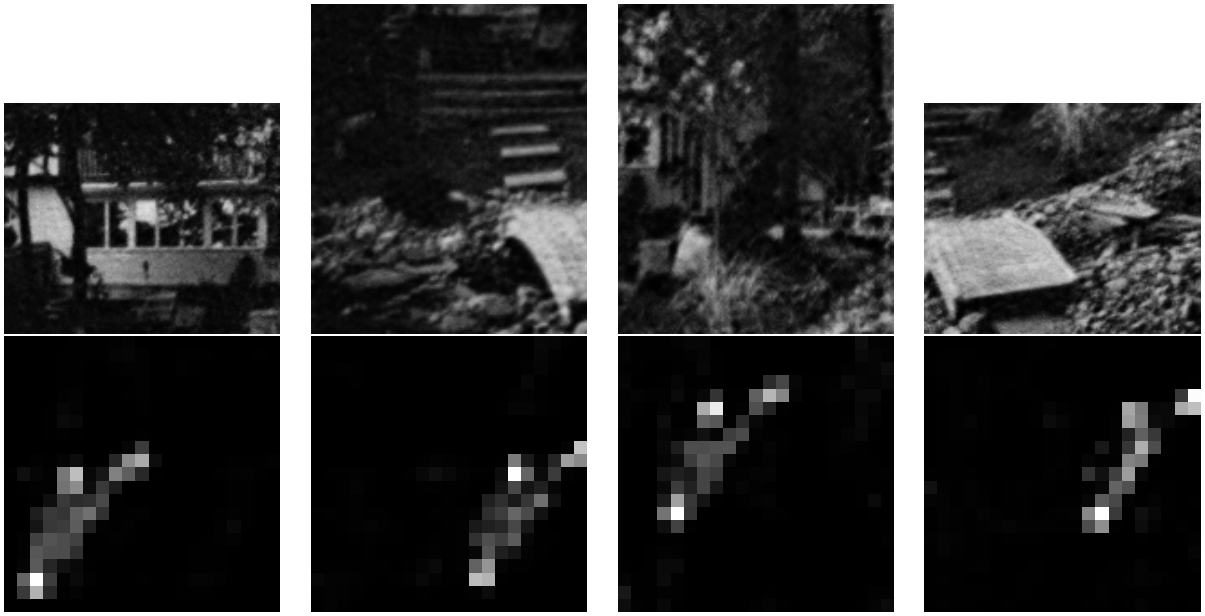
Visti i risultati, abbiamo provato immaginare possibili soluzioni.

**Utilizzo del primo kernel per non-blind deconvolution**
Troviamo il kernel sulla prima patch e poi eseguiamo una non-blind deconvolution (con la versione non-blind del metodo SelfDeblur) sulle altre patch utilizzando il kernel trovato. Il risultato sarà privo di artefatti da translazione poichè è sempre stato utilizzato lo stesso kernel.

Rimagono problemi con questo approccio:

- Il risultato è peggiore dell'esecuzione su immagine completa, poichè la rete minimizza kernel per una patch specifica e non è detto che sia il kernel ottimale per l'immagine completa. Ciò può essere mitigato centrando e ridimensionando il kernel come in figura 3.4 migliorando allo stesso tempo il risultato 4.1.

- Il primo kernel potrebbe risultare deforme rispetto agli altri, risulta quindi una buona idea controllare anche il secondo kernel è vedere se (una volta centrati entrambi per evitare problemi con la translazione) sono abbastanza simili con una metrica come MSE che si comporta bene per immagini "*sparse*". Generalizzando ad $n$ patch per essere sicuri che il kernel sia un buon rappresentante, il kernel dovrebbe essere simile ad almeno $\frac{n}{2}$ patch.
  Nel caso peggiore tutti i kernel potrebbero essere abbastanza diversi l'uno dall'altro.
  A questo punto possiamo:



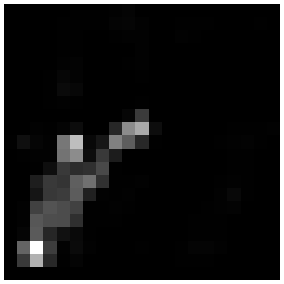
(a) Patch 0 (b) Patch 1 (c) Patch 2 (d) Patch 3

Figura 3.3: Patch e kernel associati

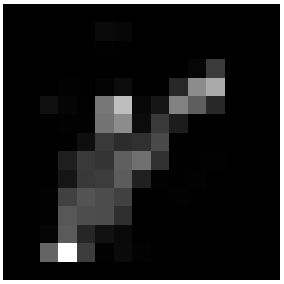
- Prendere il kernel con una maggior somiglianza a tutti gli altri
- Centrare tutti i kernel e fare una media, il kernel risultante dovrebbe rappresentare al meglio tutti gli altri poichè pixel in cui più kernel si sovrappongono hanno una maggior intensità e pixel in cui c'è poca sovrapposizione avranno una minore intensità.

## 3.2 Input con Canny Edge Detection

La rete genera l'immagine latente e il kernel a partire da un rumore generato casualmente in ingresso, nel nostro caso campionato da una distribuzione gaussiana. Al meglio delle nostre conoscenze non vi è molta ricerca in letteratura sull'importanza del rumore in input, [11] hanno studiato come il variare della dimensione dell'input influisce sul risultato finale, [17] hanno studiato come determinate distribuzioni di rumore possono portare a un risultato meno rumoroso.
Secondo l'ipotesi delle varietà, i dati reali si trovano su varietà a basse dimensioni incorporate in spazi ad alta dimensione [7]. Quindi i dati dovrebbero avere una rappresentazione a bassa dimensionalità efficiente. Avendo i dati ad alta dimensionalità possiamo eseguire una riduzione di dimensionalità (Es. Principal Component Analisys), in questo caso però non abbiamo la distribuzione a dimensioni elevate, per questo utilizziamo una rete neurale in grado di imparare la funzione in grado di mappare il rumore in input all'



(a) Primo kernel stimato



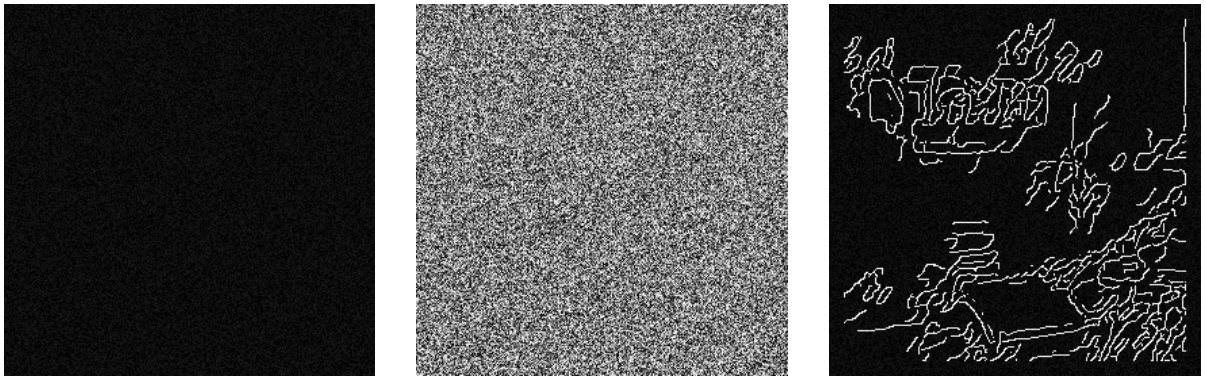
(b) Kernel centrato e ridimensionato

Figura 3.4: Kernel originale e kernel centrato e ridimensionato

immagine latente.
Teoricamente la rete dovrebbe essere in grado di mappare qualunque input all'output corretto ma in realtà non è così:

- Ci sono multipli minimi locali e non è detto (anzi è molto improbabile) che la rete trovi il minimo globale azzerando la funzione loss.
- Non è detto che la funzione loss possa essere minimizzata poichè la rete, per quanto complessa, è composta da un numero finito di parametri ed una determinata struttura impedendole di mappare "qualsiasi" input a "qualsiasi" output.
- Nel nostro caso la nostra funzione loss è $1 - \text{SSIM}$ cioè ci basiamo su una metrica, come tale non è in grado di racchiudere tutte le informazioni dell'immagine con cui è effettuato il confronto, portando ad un risultato imperfetto
- Anche se la metrica presa in considerazione fosse l'uguaglianza all'immagine confrontata potremmo comunque non avere l'immagine latente poichè l'immagine di confronto è l'immagine sfocata in input e ci sono infinite soluzioni di $(x, k)$ $t.c.$ $y = x \otimes k$ che generano l'immagine sfocata di partenza.

In questo lavoro abbiamo prima provato a cambiare le distribuzioni che generano il rumore in input, ciò ha portato a risultati differenti come da risultati 4.2 e 4.3. Testando due rumori diversi su tutto il dataset, quello gaussiano e la variazione totale del rumore gaussiano, abbiamo notato che sulle singole immagini davano risultati generalmente opposti. Nelle immagini in cui il rumore gaussiano dava buoni risultati, la variazione totale dava pessimi risultati (rispetto alla mediana) e viceversa. Infatti nel complesso il miglioramento è minimo 4.4 e non significativo.
Ipotizziamo che il motivo per cui una certa distribuzione si comporti in modo migliore con una determinata immagine sia perchè il rumore abbia una distribuzione più adatta



(a) Rumore Gaussiano

(b) Rumore Gaussiano regolarizzato con TV

(c) Rumore Gaussiano con Canny Edge Detection

Figura 3.5: Comparazione tra rumori in input, si consiglia di ingrandire per vedere il rumore gaussiano

ad una determinata funzione "generabile" dalla rete che porta ad un minimo locale migliore degli altri.
Seguendo questa linea di pensiero abbiamo provato ad incorporare nel rumore dei dati strutturali che l'immagine sfocata ha in comune con l'immagine latente: gli spigoli (edges). Abbiamo usato l'algoritmo Canny Edge Detection [2] con vari *sigma* per valutare i risultati 3.5. Abbiamo scelto questo algoritmo poichè è veloce, l'implementazione è facilmente reperibile e rimane uno standard nella edge detection nonostante vi siano metodi moderni più evoluti e complessi.

# Capitolo 4

# Risultati Numerici

## 4.1 Metriche

Le metriche sono metodi che determinano la qualità del risultato di una determinata operazione. In questo caso dobbiamo prendere in considerazioni metriche in grado di determinare la somiglianza tra l'immagine restaurata e l'immagine originale (se disponibile) a partire dall'immagine corrotta. Le metriche possono essere:

- Soggettive: Si basano sulla percezione di chi analizza il risultato. Sono difficilmente quantizzabili e variano da individuo ad individuo ma in casi in cui non è presente un termine comparativo come quando non abbiamo l'immagine originale, il metodo più semplice per determinare la qualità della restaurazione è la percezione soggettiva poichè è difficile per un algoritmo determinare la "sfocatezza" di un'immagine senza un qualcosa con cui compararla. Un esempio di metrica soggettiva è il confronto di due immagini una a fianco dell'altra, una persona è in grado di dare la propria opinione sulla somiglianza tra le due.
- Oggettive: Utilizzano metodi matematici o algoritmici per determinare la qualità del prodotto dando un risultato quantizzato e universalmente ripetibile.

**Mean Squared Error (MSE)**
La metrica MSE come dice il nome è la media della differenza tra gli elementi di due matrici (in questo caso due immagini) elevata al quadrato. Questa metrica non è invariante in spazio cioè controlla precisamente la differenza pixel per pixel, quindi la metrica produce risultati peggiori se il risultato è soggetto ad un operatore spaziale (transformazioni lineari come translazione, rotazione, convoluzione).

$$\text{MSE} = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}(Y_{ij} - \hat{Y}_{ij})^2 \tag{4.1}$$

Dove $n$ e $m$ sono rispettivamente la larghezza e altezza dell'immagine, $Y$ è l'immagine originale non corrotta, $\hat{Y}$ è l'immagine restaurata e l'operatore $\{\cdot\}_{ij}$ indica l'elemento $ij$ della matrice. Più basso è MSE migliore è il risultato.

**Peak Signal to Noise Ratio (PSNR)**
La metrica PSNR è il rapporto tra il massimo valore del segnale e la potenza del rumore. Il rumore può essere descritto dal MSE.

$$\text{PSNR} = 10 \times \log_{10}\left(\frac{\text{MAX}(Y)^2}{\text{MSE}}\right) = 20 \times \log_{10}\left(\frac{\text{MAX}(Y)}{\sqrt{\text{MSE}}}\right) \tag{4.2}$$

Maggiore è il PSNR minore è il rumore indicando la possibilità di un risultato migliore. Anche questa metrica non è invariante in spazio e inoltre non indica strettamente una miglior immagine restaurata ma indica un basso rumore.

**Structural Similarity Index Measure (SSIM)**
La metrica SSIM è un metodo per determinare la qualità percepita di un'immagine ed è usata per misurare la somiglianza tra due immagini. La differenza tra questa metrica e le due precendenti è che il PSNR e il MSE misurano l'errore assoluto mentre il SSIM considera le informazioni strutturali dell'immagine. L'informazione strutturale è l'idea che pixel vicini hanno una forte interdipendenza tra loro.

$$\text{SSIM}(x, y) = [l(x, y)]^\alpha \cdot [c(x, y)]^\beta \cdot [s(x, y)]^\gamma \tag{4.3}$$

dove:

- $l(x, y) = \frac{\mu_x \mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1}$
- $c(x, y) = \frac{\sigma_x \sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2}$
- $s(x, y) = \frac{\sigma_{xy} + c_3}{\sigma_x \sigma_y + c_3}$

$\mu_x$, $\mu_y$, $\sigma_x^2$, $\sigma_y^2$ sono rispettivamente la media e la varianza di $x$ e $y$ e $\sigma_{xy}$ è la covarianza. $c_1$, $c_2$ e $c_3$ sono variabili per stabilizzare le divisioni con piccolo denominatore. $\alpha$, $\beta$, $\gamma$ sono pesi generalmente impostati uguali ad 1.

**Percentuale di immagini affette da artefatti visivi**
Questa è una metrica soggettiva (valutata da una persona) che si è resa necessaria perchè le metriche puramente numeriche non erano in grado di catturare con precisione la presenza di artefatti visivi e. Questi artefatti variano tra distorsioni significative dell'immagine e sfocatura non rimossa o rimossa solo parzialmente. La metrica è data da:

$$P_a = \frac{n_{immagini\ con\ artefatti}}{n_{immagini\ totali}} * 100 \tag{4.4}$$

(a) Immagine ancora sfocata

(b) Immagine con artefatti

Esempi di immagini con artefatti e sfocatura non rimossa sono 4.1a e 4.1b. Essendo una metrica soggettiva non pretende di essere universale, ma dà una buona idea di quanto le metriche in alcuni casi non riescano a valutare con precisione un'immagine.

## 4.2 Dataset e metodologia

Il dataset utilizzato per i risultati è quello di Levin [9]. E' un dataset di 4 immagini in scala di grigi con risoluzione di $255 \times 255$ sfocate utilizzando 8 kernel di movimento realistici diversi per un totale di 32 immagini.
Le metriche prese in considerazione sono quelle esposte nella sezione precedente.
Oltre le metriche verrano mostrati alcuni dei risultati a confronto per poterli valutare visivamente.
I seguenti termini saranno utilizzati nello studio per semplicità:

- L'immagine originale verrà denominata GT (Ground Truth)
- L'implementazione basata su patch in cui è stato preso il primo kernel per effettuare la non-blind deconvolution delle altre patch verrà chiamato $\text{Patch}_1$
- L'implementazione basata su patch centrando e ridimensionando il primo kernel trovato sarà detta $\text{Patch}_{1c}$
- SelfDeblur è la tecnica originale dell'articolo [12] con rumore gaussiano come input
- Il metodo che utilizza un input regolarizzato con TV (Total Variation) sarà chiamato $\text{SelfDeblur}_{gtv}$

- Il metodo con funzione loss regolarizzata con TV, SelfDeblur$_{tv}$
- Il metodo con Canny Edge Detection, SelfDeblur$_{c}$
- Il metodo con Canny Edge Detection e regolarizzazione della funzione loss con TV, SelfDeblur$_{ctv}$

Le metriche MSE e PSNR sono implementate come da formula, la SSIM è implementata da una libreria, tutti i risultati sono stati valutati con queste implementazioni delle metriche.

## 4.3 Implementazione basata su patch

### 4.3.1 Complessità computazionale

Dal punto di vista della complessità computazionale in tempo, l'algoritmo originale è esponenziale (assumiamo base 2 per semplicità di esposizione) nella dimensione dell'input 3.1b, assumendo un immagine con larghezza uguale a lunghezza per semplicità:

$$O(2^{s^2})$$

Dove $s$ è la lunghezza di una delle dimensioni. Dividendo l'immagine in $n$ patch, il numero di pixel di ogni patch sarà $\frac{s^2}{n}$ e la complessità di calcolare tutte le patch sarà:

$$O(n \cdot 2^{\frac{s^2}{n}}) \tag{4.5}$$

Vogliamo trovare $n$ che minimizzi la complessità, quindi cerchiamo la derivata:

$$\frac{\partial(n \cdot 2^{\frac{s^2}{n}})}{\partial n} = \frac{2^{\frac{s^2}{n}}(n - s^2\log(2))}{n} \tag{4.6}$$

Notiamo che la derivata diventa negativa per $n < s^2\log(2)$ e positiva per $n > s^2\log(2)$ quindi:

$$n = s^2\log(2) \tag{4.7}$$

è punto di minimo. Se sostituiamo 4.7 in 4.5 vediamo che la complessità è:

$$O(s^2\log(2) \cdot 2^{\frac{s^2}{s^2\log(2)}}) = O(s^2\log(2) \cdot 2^{\frac{1}{\log(2)}}) = O(s^2) \tag{4.8}$$

$\log(2) \cdot 2^{\frac{1}{\log(2)}}$ è una costante quindi la complessità da esponenziale è diventata quadratica sulla dimensione in input e lineare sul numero di pixel in input!
Per quanto riguarda la complessità spaziale il miglioramento è ancora più visibile poichè ad ogni iterazione la memoria viene liberata, seguendo un procedimento analogo la complessità spaziale è:

$$O(2^{\frac{1}{\log(2)}}) = O(1)$$

Da esponenziale la complessità in spazio diventa costante!

**Esempio**
Prendiamo un' immagine $2048 \times 2048$ quindi $s = 2048$ la complessità sarà minima per:

$$n = s^2 \log(2) = 2907269.992011301 \approx 2907270 \text{ patch}$$

Ovviamente ciò non è possibile, poichè se la rete lavorasse su patch di dimensione $1.2 \times 1.2$ non riuscirebbe a trovare la sfocatura, in pratica non ci conviene lavorare con patch sotto $64 \times 64$. Ma queste considerazioni ci aiutano comunque a capire il miglioramento passando ad un'implementazione a patch.
Un esempio più realistico è prendere un'immagine $1024 \times 1024$ dividerla in 32 patch da $128 \times 128$.
Il tempo necessario ad eseguire la rete è circa $21727.77s \approx 362,13$ min $\approx 6$ ore e la memoria necessaria è 26612MB $\approx 26.6GB$ il che la rende praticamente inutilizzabile per immagini di grandi dimensioni.
Con le patch il tempo di esecuzione diventa $1920s \approx 32$ min e la memoria necessaria rimane costante a circa 200MB.

### 4.3.2 Risultati metriche

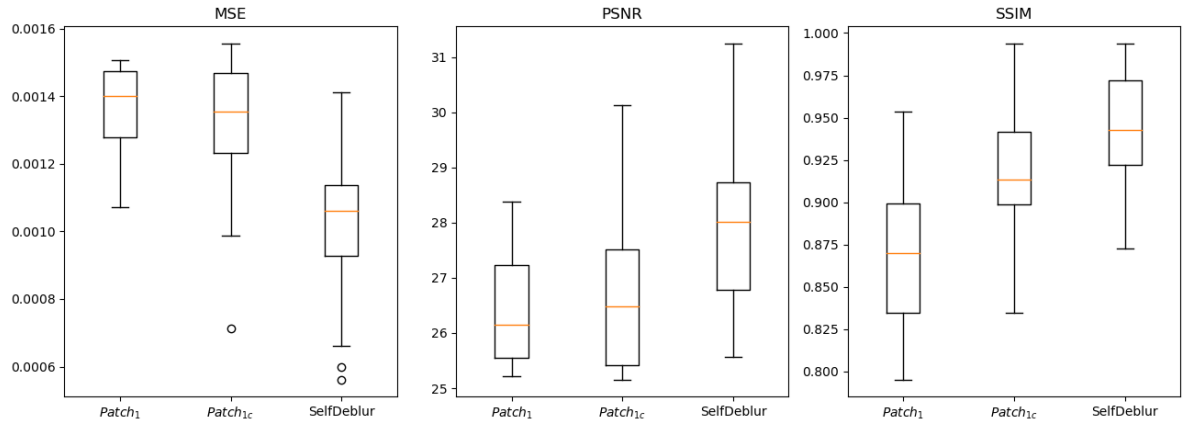



Figura 4.2: Confronto tra i metodi con implementazione delle patches e metodo originale visualizzati come boxplot

C'è un lieve calo di tutte le metriche quando vengono usate le patch ma per immagini di grandi dimensioni a meno di avere un enorme potenza di calcolo l'esecuzione della rete originale può anche essere impossibile poichè la memoria richiesta è troppo grande.
Interessante notare come in questo caso $P_a$ sia molto importante come metrica poichè anche se le immagini si comportano bene con le altre, quando vengono esaminate visivamente presentano artefatti, il disturbo causato da questi artefatti varia tra la mancanza

| Metodo | MSE | PSNR | SSIM | $P_a$ |
|---|---|---|---|---|
| SelfDeblur | 0.001014 | 27.906835 | 0.942813 | 3.125% |
| $\text{Patch}_1$ | 0.001372 | 26.444221 | 0.871094 | 53.125% |
| $\text{Patch}_{1c}$ | 0.001323 | 26.637991 | 0.918116 | 78.125% |

Tabella 4.1: Confronto tra i valori medi dei risultati generati dai metodi implementati con patches e metodo originale

di dettagli e artefatti che effettivamente rendono l'immagine distorta.
Il fatto che $Patch_{1c}$ si comporti meglio nelle altre metriche ma presenti un maggior numero di artefatti è inputabile al fatto che la rete ha minimizzato su un determinato kernel ma noi gliene forniamo uno che è stato translato e di dimensione minore portando sicuramente a migliori metriche poichè non ci dovrebbe essere errore di translazione ma la rete non riesce a generare una immagine realistica con questo kernel modificato ad hoc.
In figura 4.3 il metodo $\text{Patch}_{1c}$ introduce artefatti che distorcono troppo l'immagine per essere considerato un successo, nella figura 4.4 accade il contrario, è $\text{Patch}_1$ a generare artefatti. In ogni caso l'immagine presenta sensibilmente più rumore rispetto al metodo originale.

## 4.4 Modifica del rumore in input

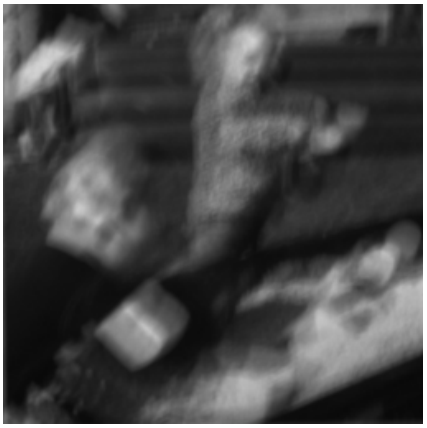
| Metodo | MSE | PSNR | SSIM |
|---|---|---|---|
| SelfDeblur | 0.001189 | 27.600815 | 0.874733 |
| $\text{SelfDeblur}_{gtv}$ | **0.000520** | **31.190073** | **0.994476** |

Tabella 4.2: Metriche su immagine in figura 4.5

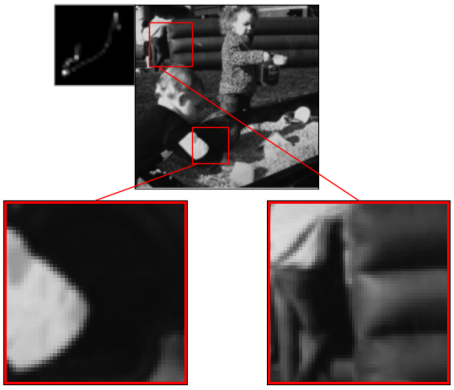
| Metodo | MSE | PSNR | SSIM |
|---|---|---|---|
| SelfDeblur | **0.000716** | **28.786700** | **0.981956** |
| $\text{SelfDeblur}_{gtv}$ | 0.000974 | 27.453961 | 0.918181 |

Tabella 4.3: Metriche su immagine in figura 4.6

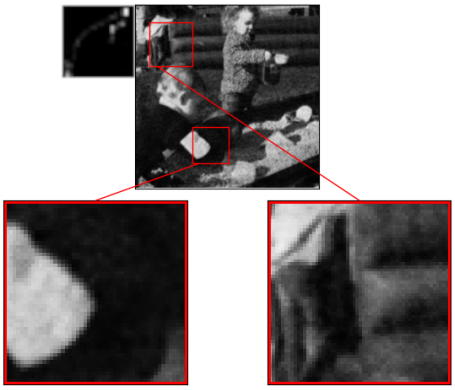
Le tabelle 4.2 e 4.3 mostrano il comportamento della rete con distribuzioni diverse, generalmente i valori delle metriche con distribuzioni diverse sono nello stesso intorno ma in diverse istanze si comportano in maniera notevolmente differente. Possiamo vedere nelle figure 4.5 e 4.6 che $\text{SelfDeblur}_{gtv}$ tende a generare immagini più *liscie* (smooth) ma



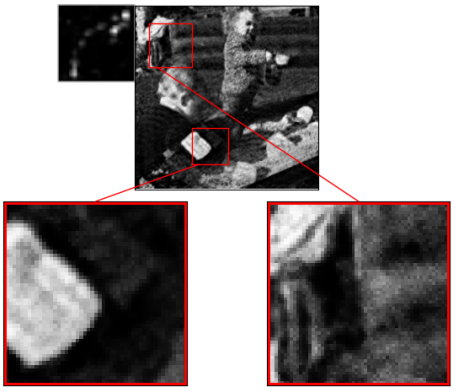
(a) Input



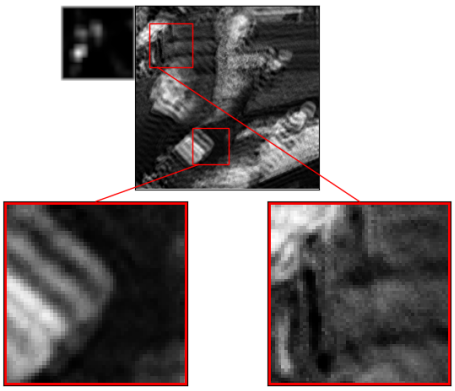
(b) GT



(c) Selfdeblur



(d) $\text{Patch}_1$



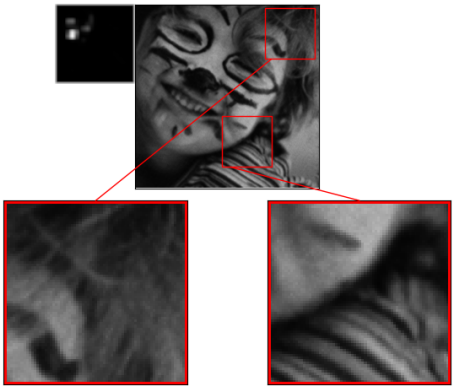
(e) $\text{Patch}_{1c}$

Figura 4.3: Confronto tra i metodi con patches, metodo originale e GT

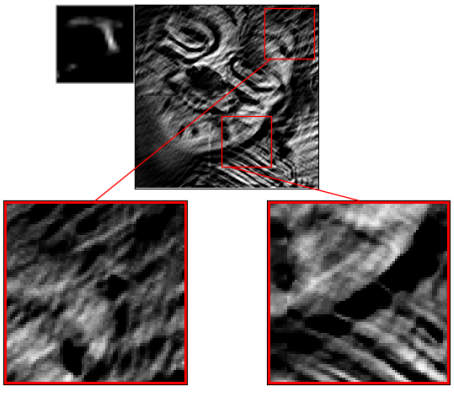
entrambi i metodi producono ottimi risultati in ogni caso.

(a) Input

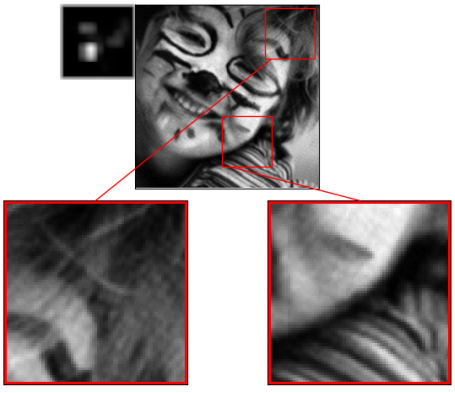
(b) GT



(c) Selfdeblur



(d) $\text{Patch}_1$



(e) $\text{Patch}_{1c}$

Figura 4.4: Confronto tra i metodi con patches, metodo originale e GT

Nel complesso i risultati sono simili, vedi tabella 4.4.

(a) Input (b) GT

(c) SelfDeblur (d) SelfDeblur$_{gtv}$

Figura 4.5: Confronto tra i risultati con diverse distribuzioni di probabilità in input

| Metodo | MSE | PSNR | SSIM |
|---|---|---|---|
| SelfDeblur | 0.001014 | 27.906835 | 0.942813 |
| SelfDeblur$_{gtv}$ | **0.000958** | **28.212702** | **0.949149** |

Tabella 4.4: Metriche su calcolate sull'intero dataset

## 4.5 Input con Canny Edge Detection

### 4.5.1 Risultati sul dataset

Il metodo SelfDeblur al tempo della scrittura è uno dei metodi allo stato dell'arte per la rimozione di sfocatura da un'immagine come dimostrano i risultati visibili in [12], come

| Metodo | MSE | PSNR | SSIM | $P_a$ |
|---|---|---|---|---|
| SelfDeblur | 0.001014 | 27.906835 | 0.942813 | **3.125%** |
| SelfDeblur$_{tv}$ | 0.001018 | 27.884284 | 0.940731 | 6.250% |
| SelfDeblur$_{c}$ | 0.000933 | **28.235203** | **0.962755** | 31.125% |
| SelfDeblur$_{ctv}$ | **0.000925** | 28.344359 | 0.960425 | 28.125% |

Tabella 4.5: Media delle metriche testate su tutto il dataset

descritto nell'articolo l'aggiunta della Total Variation come da eq. 2.3 con $\lambda = 1$ non porta ad un generale miglioramento dei risultati, anzi per $\lambda < 0.7 \wedge \lambda > 1.2$ i risultati presentano anche leggeri peggioramenti.

Il metodo utilizzante la Canny Edge Detection presenta risultati ragguardevoli, tutte le metriche sono migliorate rispetto alla tecnica originale, dal punto di vista visivo 4.9, possiamo notare guardando la guancia del soggetto che i metodi con l'edge detection hanno meno rumore, inoltre il kernel è centrato portando a meno errore da translazione causato dalla convoluzione. Il kernel più verosimile viene però generato dalla rete con TV e ciò è rispecchiato dalle metriche 4.6, non è stata fatta alcuna modifica alla generazione del kernel in SelfDeblur$_{c}$ e SelfDeblur$_{ctv}$ quindi la centratura che vediamo probabilmente è causata dal fatto che la struttura è già presente nell'input noise e se la rete generasse un kernel spostato la struttura si dovrebbe spostare portando ad un'energia necessaria maggiore.
Dal punto di vista della $P_a$ si può notare come i metodi con edge detection causino un maggior numero di artefatti, in particolare gli artefatti sono solo causati da sfocatura non rimossa o rimossa non del tutto 4.8.

| Metodo | MSE | PSNR | SSIM |
|---|---|---|---|
| SelfDeblur | 0.001231 | 26.163145 | 0.916163 |
| SelfDeblur$_{tv}$ | 0.001018 | 27.884284 | 0.940731 |
| SelfDeblur$_{c}$ | **0.000882** | **27.609505** | **0.975885** |
| SelfDeblur$_{ctv}$ | 0.000942 | 27.322994 | 0.971176 |

Tabella 4.6: Metriche su immagine in figura 4.9

| Metodo | MSE | PSNR | SSIM |
|---|---|---|---|
| SelfDeblur | 0.001034 | 28.599968 | 0.941098 |
| SelfDeblur$_{tv}$ | 0.001043 | 28.563012 | 0.946146 |
| SelfDeblur$_{c}$ | 0.000850 | 29.451491 | 0.973097 |
| SelfDeblur$_{ctv}$ | **0.000703** | **30.271881** | **0.988554** |

Tabella 4.7: Metriche su immagine in figura 4.10

(a) Input

(b) GT

(c) SelfDeblur

(d) SelfDeblur$_{gtv}$

Figura 4.6: Confronto tra i risultati con diverse distribuzioni di probabilità in input

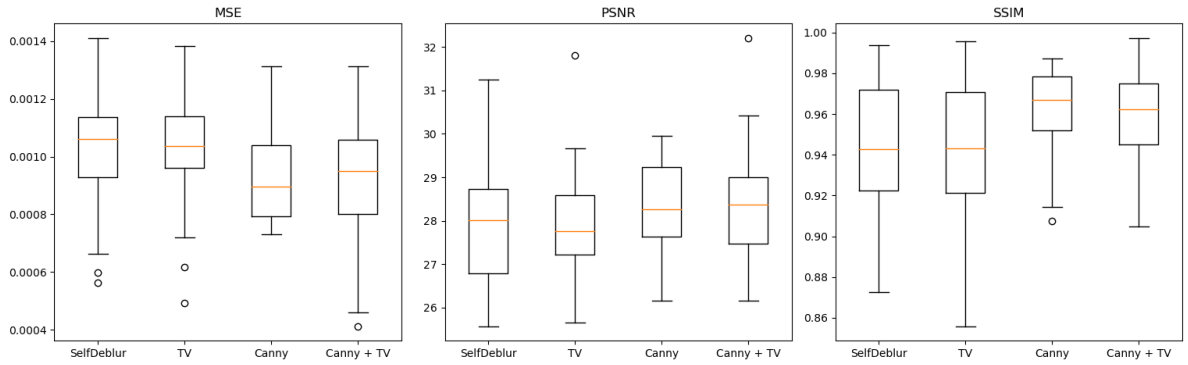



Figura 4.7: Boxplot dei valori delle metriche per ogni metodo testato

Figura 4.8: Artefatti su metodo con edge detection

(a) Input (b) GT

(c) SelfDeblur (d) Selfdeblur$_{tv}$

(e) Selfdeblur$_{ctv}$ (f) Selfdeblur$_{c}$

Figura 4.9: Confronto tra i vari metodi, il kernel in alto a sinistra

(a) Input (b) GT

(c) SelfDeblur (d) Selfdeblur$_{tv}$

(e) Selfdeblur$_{ctv}$ (f) Selfdeblur$_{c}$

Figura 4.10: Confronto tra i vari metodi, il kernel in alto a sinistra

### 4.5.2 Risultati con rumore addittivo sull'immagine

| Metodo | MSE | PSNR | SSIM |
|---|---|---|---|
| SelfDeblur | 0.001469 | 26.225078 | 0.921188 |
| SelfDeblur$_{tv}$ | 0.001523 | 26.064201 | 0.905177 |
| SelfDeblur$_{c}$ | **0.001306** | **26.737152** | **0.954104** |
| SelfDeblur$_{ctv}$ | 0.001331 | 26.657830 | 0.950179 |

Tabella 4.8: Media delle metriche testate su tutto il dataset con rumore gaussiano $\sigma^2 = 1e^{-3}$

| Metodo | MSE | PSNR | SSIM |
|---|---|---|---|
| SelfDeblur | 0.001257 | 26.916152 | 0.940164 |
| SelfDeblur$_{tv}$ | 0.001278 | 26.849210 | 0.936308 |
| SelfDeblur$_{c}$ | **0.000930** | **28.259853** | 0.963017 |
| SelfDeblur$_{ctv}$ | 0.001071 | 27.611504 | **0.964382** |

Tabella 4.9: Media delle metriche testate su tutto il dataset con rumore gaussiano $\sigma^2 = 1e^{-4}$

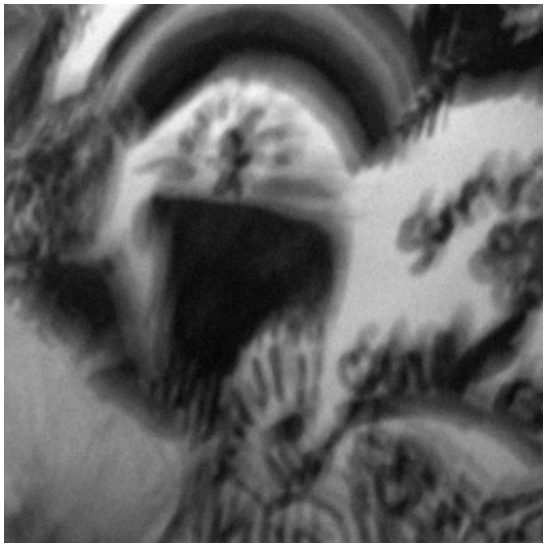
| Metodo | MSE | PSNR | SSIM |
|---|---|---|---|
| SelfDeblur | 0.001067 | 27.668704 | 0.940298 |
| SelfDeblur$_{tv}$ | 0.001093 | 27.548083 | 0.935901 |
| SelfDeblur$_{c}$ | 0.000930 | 28.273531 | 0.961640 |
| SelfDeblur$_{ctv}$ | **0.000924** | **28.287952** | **0.965293** |

Tabella 4.10: Media delle metriche testate su tutto il dataset con rumore gaussiano $\sigma^2 = 1e^{-5}$

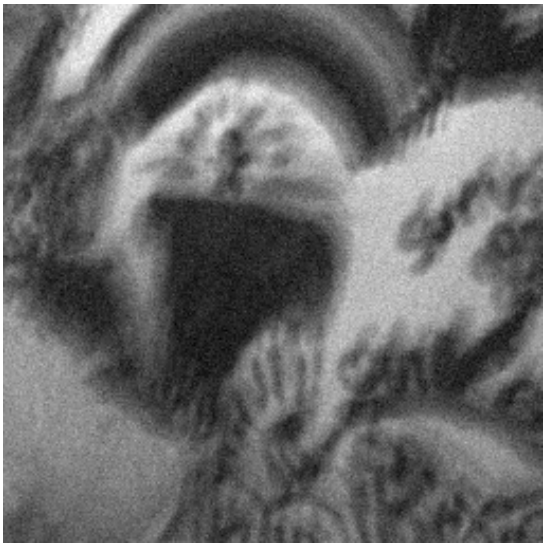
La maggior parte delle immagini reali presentano vari tipi di rumore quindi si rende necessario provare come si comporta la rete in queste situazioni.
E' stato testato l'intero dataset aggiungendo rumore gaussiano impostando la varianza a valori diversi progressivamente più grandi.
Le figure 4.12 e 4.13 mostrano come i metodi con edge detection producano immagini con nettamente meno rumore. La figura 4.13f è uno dei casi in cui la sfocatura non viene completamente rimossa.
Non è stata rilevata la $P_a$ poichè il rumore è presente in quantita significativa soprattutto quando $\sigma^2 = 1e^{-3}$. Da notare però che all'aumentare del rumore i metodi con edge detection presentano minori artefatti mantenendo ottime metriche il che li rende dei candidati perfetti per immagini molto rumorose.

(a) Input

(b) Input con $\sigma^2 = 1e^{-5}$

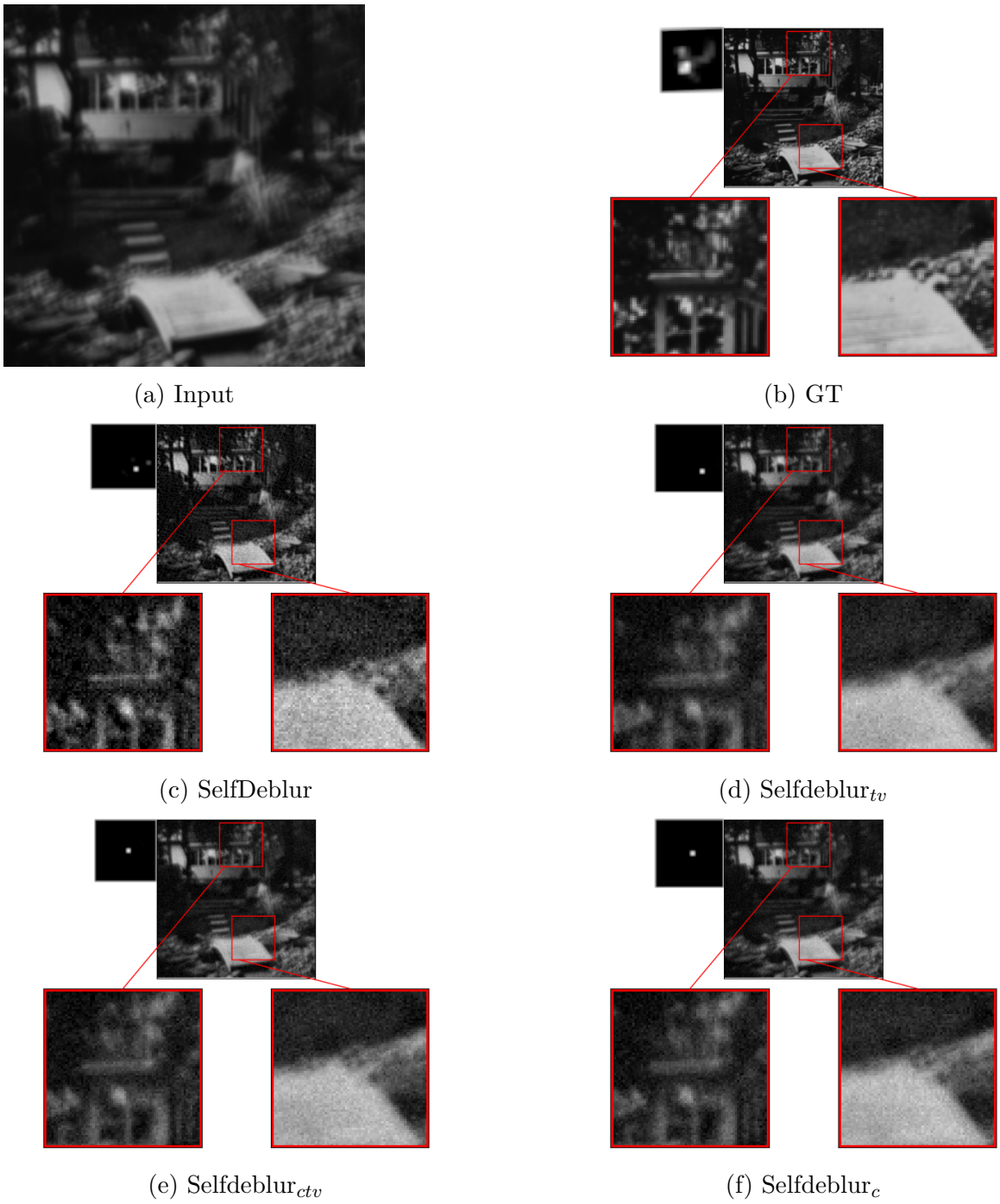


(c) Input con $\sigma^2 = 1e^{-4}$



(d) Input con $\sigma^2 = 1e^{-3}$

Figura 4.11: Immagine test al variare del rumore additivo



(a) Input (b) GT

(c) SelfDeblur (d) Selfdeblur$_{tv}$

(e) Selfdeblur$_{ctv}$ (f) Selfdeblur$_{c}$

Figura 4.12: Confronto tra i vari metodi su immagini con rumore additivo di varianza $\sigma^2 = 1e^{-3}$, il kernel in alto a sinistra

(a) Input

(b) GT

(c) SelfDeblur

(d) Selfdeblur$_{tv}$

(e) Selfdeblur$_{ctv}$

(f) Selfdeblur$_{c}$

Figura 4.13: Confronto tra i vari metodi su immagini con rumore additivo di varianza $\sigma^2 = 1e^{-4}$, il kernel in alto a sinistra

## 4.6 Risultati su immagini reali

(a) Input

(b) Selfdeblur

(a) Input

(b) Selfdeblur

Abbiamo anche provato il funzionamento della rete con immagini reali tratte dal dataset [10], i risultati non sono stati particolarmente buoni poichè in immagini reali potrebbero esserci multipli nuclei di sfocatura di dimensioni differenti e anche sovrapposti. La rete sembra non essere adatta a questo tipo di lavori.

# Conclusioni

In questa tesi si è mostrata una panoramica sommaria delle applicazioni della convoluzione e deconvoluzione lineare, mostrando sia metodi tradizionali che basati sul deep learning per la risoluzione di problemi di blind deconvolution, con particolare attenzione alla rimozione della sfocatura da un'immagine.
Abbiamo poi approfondito il funzionamento della rete SelfDeblur e proposto possibili modifiche.
La prima modifica consiste nell'utilizzare la rete su delle patch per permettere la sua esecuzione su immagini di grandi dimensioni, impossibilitata dal grande dispendio di risorse necessario.
La seconda modifica consiste nell'analisi della correlazione tra rumore in input e immagine finale e l'uso della Canny Edge Detection per migliorare i risultati finali. Abbiamo usato la canny edge detection ma ulteriori tecniche di feature extraction potrebbero funzionare allo stesso modo, se non meglio, ulteriore ricerca è ritenuta necessaria. A causa della portata di questa tesi non è stato possibile analizzare a fondo le implicazioni ma possiamo ipotizzare che la generazione di un rumore in input ad hoc sia utilizzabile anche con altre reti generative convoluzionali. In particolare dovrebbe essere molto efficace in problemi di image restoration poichè abbiamo dei dati degradati dell'immagine originale ed esistono una moltitudine di algoritmi in grado di ottenere delle feature dall'immagine degradata da poter aggiungere all'input della rete per migliorare il risultato.
Riteniamo che una maggiore ricerca sia necessaria in questo ambito poichè la letteratura è pressochè inesistente e le potenzialità sono elevate.

# Bibliografia


[1] Michal Irani Assaf Shocher Nadav Cohen. “"Zero-Shot" Super-Resolution using Deep Internal Learning”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2018.

[2] John Canny. “A Computational Approach to Edge Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[3] T.F. Chan e Chiu-Kwong Wong. “Total variation blind deconvolution”. In: *IEEE Transactions on Image Processing* 7.3 (1998), pp. 370–375. DOI: 10.1109/83.661187.

[4] Ian Goodfellow et al. “Generative Adversarial Networks”. In: *Commun. ACM* 63.11 (ott. 2020), pp. 139–144. ISSN: 0001-0782. DOI: 10.1145/3422622. URL: https://doi.org/10.1145/3422622.

[5] Won-Ki Jeong, Hanspeter Pfister e Massimiliano Fatica. “Chapter 46 - Medical Image Processing Using GPU-Accelerated ITK Image Filters”. In: *GPU Computing Gems Emerald Edition*. A cura di Wen-mei W. Hwu. Applications of GPU Computing Series. Boston: Morgan Kaufmann, 2011, pp. 737–749. ISBN: 978-0-12-384988-5. DOI: https://doi.org/10.1016/B978-0-12-384988-5.00046-2. URL: https://www.sciencedirect.com/science/article/pii/B9780123849885000462.

[6] Diederik Kingma e Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (dic. 2014).

[7] Yann Lecun, Sumit Chopra e Raia Hadsell. “A tutorial on energy-based learning”. In: gen. 2006.

[8] Victor Lempitsky, Andrea Vedaldi e Dmitry Ulyanov. “Deep Image Prior”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9446–9454. DOI: 10.1109/CVPR.2018.00984.

[9] Anat Levin et al. “Understanding and evaluating blind deconvolution algorithms”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 1964–1971. DOI: 10.1109/CVPR.2009.5206815.

[10] Seungjun Nah, Tae Hyun Kim e Kyoung Mu Lee. “Deep Multi-Scale Convolutional Neural Network for Dynamic Scene Deblurring”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Lug. 2017.

[11] Manisha Padala, Debojit Das e Sujit Gujar. *Effect of Input Noise Dimension in GANs*. 2020. arXiv: 2004.06882 [cs.LG].

[12] Dongwei Ren et al. “Neural Blind Deconvolution Using Deep Priors”. In: *CoRR* abs/1908.02197 (2019). arXiv: 1908.02197. URL: http://arxiv.org/abs/1908.02197.

[13] Leonid I. Rudin, Stanley Osher e Emad Fatemi. “Nonlinear total variation based noise removal algorithms”. In: *Physica D: Nonlinear Phenomena* 60.1 (1992), pp. 259–268. ISSN: 0167-2789. DOI: https://doi.org/10.1016/0167-2789(92)90242-F. URL: https://www.sciencedirect.com/science/article/pii/016727899290242F.

[14] Christian J. Schuler et al. “Learning to Deblur”. In: *CoRR* abs/1406.7444 (2014). arXiv: 1406.7444. URL: http://arxiv.org/abs/1406.7444.

[15] David Wipf e Haichao Zhang. “Revisiting Bayesian Blind Deconvolution”. In: (2013). arXiv: 1305.2362 [cs.CV].

[16] Xiangyu Xu et al. “Motion Blur Kernel Estimation via Deep Learning”. In: *IEEE Transactions on Image Processing* 27.1 (2018), pp. 194–205. DOI: 10.1109/TIP.2017.2753658.

[17] V. O. Yachnaya et al. “Noise Model Effect upon the GAN-Synthesized Images”. In: *2020 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF)*. 2020, pp. 1–6. DOI: 10.1109/WECONF48837.2020.9131529.

[18] Li Si- Yao, Dongwei Ren e Qian Yin. “Understanding Kernel Size in Blind Deconvolution”. In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2019, pp. 2068–2076. DOI: 10.1109/WACV.2019.00224.