Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Tecniche di Deep Learning per problemi di blind deconvolution

Relatore:
Prof.ssa
Elena Loli Piccolomini

Correlatore:
Dottore
Pasquale Cascarano

Presentata da:
Riccardo Statuto

III Sessione
Anno Accademico 2019-20

ai miei compagni di viaggio ...

Introduzione

Durante la fase di acquisizione di un'immagine con strumenti digitali, potrebbe verificarsi il fenomeno detto di "sfocatura", equivalente dell'inglese "blurring", per cui il dato acquisito risulta essere non nitido.
Movimenti rapidi della fotocamera e oscillazioni, granelli di polvere, umidità, errata impostazione della profondità di campo, sono tutti fattori che potrebbero portare ad avere un'immagine sfocata e a causa di questo fenomeno indesiderato preziose informazioni potrebbero andare perse: settori quali astronomia, imaging medico, microscopia ne sono un esempio. Come fare quindi? La deconvoluzione consente di correggere il problema, ricostruendo un'immagine che si avvicina il più possibile a quella reale, attraverso appositi algoritmi. Ne esistono di diversi e lo sviluppo del machine learning ha consentito di crearne di nuovi, ampliando le possibili soluzioni. Tuttavia le tecniche standard di machine learning necessitano di ampi dataset su cui eseguire la fase di addestramento e questo ne limita il loro uso nella pratica. Lo scopo di questa tesi è quello di proporre una tecnica di risoluzione basata sull'uso di reti neurali artificiali che non richiedono alcun tipo di addestramento, apportando modifiche al metodo SelfDeblur, utilizzato come punto di partenza.
Nel primo capitolo verrà illustrato il concetto di deconvoluzione e presentato il problema di blind deconvolution. Nel secondo capitolo saranno analizzate le reti neurali, prestando maggiore attenzione a quelle convoluzionali (CNN), utilizzate in questa tesi, delle quali si descriverà struttura, funzionamento e parametri. Nel terzo capitolo sarà presentato il metodo Deep Image Prior e il suo utilizzo nel SelfDeblur, oggetto di studio di questo lavoro, di cui se ne osserverà struttura e algoritmi e sarà proposta una modifica. Risultati e confronti saranno presentati nel quarto e ultimo capitolo.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Blind deconvolution

In questo primo capitolo verrà presentato il problema di blind deconvolution, osservandolo da un punto di vista matematico e introducendo un possibile metodo risolutivo.

1.1 Definizione del problema di blind deconvolution

Come accennato nell'introduzione, qualsiasi immagine in uscita da un sensore digitale potrebbe essere affetta da sfocatura, generandone così una sua versione sfocata, "blurred image". La deconvoluzione, "deconvolution", si pone come scopo quello di ricostruire un'immagine che rappresenti al meglio la scena inquadrata, rispetto al dato sfocato acquisito. Matematicamente un'immagine sfocata può essere scritta come:

$$y = k \otimes x,$$

dove y è per l'appunto l'immagine sfocata, x è la sua versione nitida che si vuole ricavare, k è una matrice, "kernel" che rappresenta quella che viene definito come "point spread function" (PSF) e $\otimes$ denota l'operatore di convoluzione lineare. Nella blind deconvolution l'unico valore noto è y, mentre k e x sono sconosciuti; questo fa sì che il problema risulti essere mal posto proprio a causa delle infinite possibili soluzioni (k, x). Si ricorda che, un problema per essere "ben posto", come riportato in un documento di Jacques Hadamard pubblicato nel 1902 [2], dovrebbe rispettare queste tre condizioni:

1. La soluzione esiste,

2. La soluzione è unica,

3. La soluzione dipende continuamente dai dati del problema,

e come si è potuto constatare il problema di blind deconvolution non le rispetta. Ricostruire l'immagine x risulta essere quindi un problema complesso alla cui risoluzione va aggiunto un'ulteriore variabile: ogni dispositivo digitale infatti presenta in fase di acquisizione del rumore casuale additivo. Si può allora modellizzare l'acquisizione di y come:

$$y = k \otimes x + \eta,$$

dove η rappresenta il rumore casuale additivo.

Esistono diverse soluzioni al problema di blind deconvolution ma quella su cui ci soffermeremo, visto il suo utilizzo in questo lavoro, sarà quella della risoluzione di un problema di minimo. Basandosi sul metodo MAP, il problema può essere riscritto come:

$$(x, k) = \underset{(x,k)}{argmin} \|k \otimes x - y\|^2 + \lambda\phi(x) + \tau\varphi(k),$$

dove $\lambda\phi(x)$ e $\tau\varphi(k)$ sono regolarizzatori, detti anche "prior" come vedremo più avanti. Per risolvere questo problema faremo uso di reti neurali artificiali, e nello specifico reti neurali convoluzionali.

Capitolo 2

Reti neurali

In questo capitolo verranno presentate le reti neurali, e in dettaglio quelle convoluzionali, di cui ne osserveremo la struttura e il funzionamento.

2.1 Rete neurale: definizione e struttura

Le reti neurali artificiali (ANN), nate intorno agli anni '40 del secolo scorso, sono modelli matematici composti da neuroni artificiali ispirati alle reti neurali biologiche, ovvero quella umana o animale. Svolgono funzioni di predizione, classificazione ed elaborazione su insiemi di dati basandosi sul concetto di connessione tra unità in grado di attivarsi in conseguenza della ricezione di segnali di ingresso attraverso linee d'interconnessione, capaci di veicolare l'attivazione da un'unità all'altra. Il principale ambito di applicazione delle reti neurali è il machine learning, in quanto esse permettono d'implementare un sistema di apprendimento da parte di una macchina e di sfruttarlo per il riconoscimento o la classificazione di dati in problemi futuri.
Una rete neurale presenta uno schema che consiste in un numero elevato di neuroni artificiali, ovvero le unità computazionali di base, collegati tra loro a formare una struttura non lineare. Ogni neurone può elaborare e trasformare un dato in input, in un output tramite l'applicazione di una regola matematica chiamata "funzione di attivazione". Questa funzione viene applicata solo se il valore in entrata del neurone ne supera uno specifico, detto "soglia di attivazione", fissato nel momento dell'implementazione della

rete; altrimenti il nodo resterà inattivo non producendo risultati.
Tutti i neuroni della rete sono collegati tra loro attraverso connessioni orientate e pesate, dove con pesate s'intende che ogni connessione presenta un valore numerico specifico, detto appunto "peso". Per ogni neurone sarà presente almeno una connessione in entrata e una o più in uscita.
Ogni input in uscita viene moltiplicato per il relativo peso della connessione e converge nella sommatoria degli input che passeranno attraverso la funzione di attivazione. Applicare quindi una funzione di attivazione, vuol dire fare una sommatoria tra i prodotti degli input con i relativi pesi a cui può essere aggiunto un valore costante chiamato "bias", che permette di estendere il range di valori accettati dalla funzione. Tutto questo può essere sintetizzato dalla seguente formula:

$$valore\_output = f(\sum(valore\_input * peso\_connessione) + bias),$$

dove f indica la funzione di attivazione scelta.
L'insieme dei neuroni presenti sullo stesso strato compongono il "layer": ogni rete neurale è costituita da almeno due layer, input e output, e in tal caso viene chiamata "percettrone". La sua semplicità e le sue limitazioni hanno portato all'introduzione di reti più complesse, chiamate "multistrato", formate da diversi livelli intermedi, detti "hidden layer". Questo tipo di rete risulta essere più efficiente dal punto di vista della precisione dei risultati ma in molti casi dimostra una notevole lentezza nell'addestramento.
Un'ulteriore distinzione può essere fatta tra reti feedforward e reti feedback:

- le reti feedforward permettono solo collegamenti neuronali tra strati diversi, propagando il segnale in avanti. Questa tipologia di connessione genera dunque un grafo orientato aciclico. Sono il tipo di rete più diffusa e utilizzata per la loro semplicità anche in fase di addestramento e difatti saranno adoperate anche in questo lavoro.

- le reti feedback, al contrario, consentono connessioni tra i neuroni anche sullo stesso strato, permettendo quindi la generazione di cicli che propagano il segnale sia in avanti che indietro nella rete. Questa caratteristica le rende molto dinamiche e potenti ma al tempo stesso rende l'apprendimento molto complicato e dispendioso.

2.2 Tecniche di apprendimento

Il termine apprendimento, nel caso delle reti neurali, indica un processo adattivo e incrementale atto alla modifica dei pesi delle connessioni il cui scopo è quello di minimizzare la differenza tra l'output ottenuto e quello atteso. La differenza è nota come "loss function", o "funzione obiettivo", ed è fondamentale nel processo di apprendimento. Ne esistono di diverso tipo e la scelta è legata alle tipologie di algoritmi utilizzati. Indicativamente, il processo di apprendimento si sviluppa attraverso queste fasi fondamentali e può essere così riassunto:

1. A tutte le connessioni fra i nodi della rete vengono assegnati dei pesi casuali. Sono quindi proposti alla rete dei campioni di dati in input e per ognuno la rete elaborerà un output corrispondente, attraverso tutti i layer che la compongono, applicando la funzione di attivazione scelta. Questa prima fase è definita 'di forward', in quanto il segnale viene propagato in avanti dal primo all'ultimo layer della rete.

2. Una volta completata la fase 1, si utilizza la funzione di errore (loss function) per calcolare la differenza tra l'output ottenuto e quello atteso per ogni input. L'apprendimento mira a minimizzare questa funzione di errore e per fare ciò è necessario calcolarne la derivata parziale rispetto a ciascuno dei pesi, ovvero il "gradiente".

3. Il calcolo del gradiente associato ad ogni nodo avviene attraverso l'applicazione dell'algoritmo di backpropagation: esso permette di propagare l'errore all'indietro verso il layer di input. In sintesi, viene calcolato il contributo di ogni peso all'errore finale di computazione dell'output.

4. In ultima fase è possibile applicare un algoritmo di ottimizzazione per aggiornare i pesi della rete in base al valore del gradiente calcolato per ogni nodo. Di fondamentale importanza, in questa fase, risulta la scelta del coefficiente η, detto "learning rate", che moltiplica il gradiente e determina quanto l'errore influisca sull'aggiornamento dei pesi.

Prima d'iniziare la fase di apprendimento, è doveroso determinare la tecnica da utilizzare. Si potrebbe scegliere di sfruttare un set di dati, "training set", di dimensione variabile

formato da coppie input/output correttamente classificate. Questo dataset viene applicato alla rete che adatta i suoi parametri per stabilire un corretto rapporto tra input e output: in questo caso si parla di apprendimento "supervisionato".
Quando, invece, i dati di output non sono noti e si lascia che la rete apprenda in autonomo una logica di classificazione basandosi solo sui dati di input allora si parla di apprendimento "non supervisionato". In ultima analisi si osservi l'apprendimento per rinforzo, "reinforcement learning", l'ultimo dei tre paradigmi principali dell'apprendimento automatico, in cui la logica viene generata sulla base di una "ricompensa" data in caso di scelta corretta nell'associazione tra input e output, tecnica con la quale si suggerisce la direzione da seguire per sviluppare al meglio l'apprendimento.

2.3 Reti neurali convoluzionali

Una particolare categoria di reti neurali è quella delle reti definite "convoluzionali" e saranno proprio queste ad essere utilizzate in questa tesi. Le reti convoluzionali (CNN) sono molto note nel deep learning e sono largamente impiegate nel campo delle applicazioni di Visione Artificiale, meglio conosciuta come "Computer Vision", trovando la loro specializzazione nella classificazione di immagini, video, testi e altri formati multimediali. La loro capacità di riconoscere dettagli e pattern visivi all'interno di un'immagine, attraverso l'apprendimento automatico tramite dataset, ha fatto di queste reti uno strumento largamente usato nella risoluzione di problemi di ricostruzione inversa di immagini quali "denoising"[5, 1], "super-resolution"[4] e "deblurring"[7, 11], il problema preso in studio per questa tesi. In questo specifico caso, la rete sarà utilizzata per generare due immagini che rappresentino relativamente la migliore approssimazione al kernel e all'immagine "deblurrata", prendendo in input la sola immagine sfocata. In questa ottica può, quindi, essere definita rete generativa, o "Autoencoder": dato in input un valore x viene generato un output $\tilde{x}$, ossia una sua ricostruzione.

Come visto in precedenza, una rete neurale è composta da layer e in particolar modo quelli che compongono una rete CNN, svolgono ciascuno particolari funzioni ben definite che consente di identificarli in modo preciso. Nei layer, i neuroni sono organizzati in strutture tridimensionali, detti "tensori" e sono connessi solamente ad un ristretto

numero di neuroni dello strato precedente. I layer fondamentali, che troviamo in qualunque rete CNN sono quelli chiamati "convoluzionali" e sono quelli responsabili del riconoscimento di feature nell'immagine. Gli altri sono di supporto alla convoluzione e possono servire ad aggiornare, ad esempio, i pesi delle connessioni o a ridimensionare i filtri di convoluzione. Saranno ora descritti quelli utilizzati in questo lavoro per costruire la CNN.

2.3.1 Layer convoluzionali

Come detto in precedenza e da come si può intuire dal suo nome questo è il principale tipo di layer: l'utilizzo di uno o più di questi in una rete neurale convoluzionale è indispensabile. La sua costruzione si basa su filtri allenabili, spazialmente piccoli lungo le dimensioni di larghezza e altezza, ma che si estendono per l'intera profondità dell'input a cui vengono applicati. Durante la forward propagation si trasla, o più precisamente si convolve, ciascun filtro lungo la larghezza e l'altezza del volume di input, producendo una activation map (o feature map) bidimensionale per quel filtro delle dimensioni dello stesso. Man mano che lo si sposta lungo l'area dell'input, si effettua un'operazione di prodotto membro a membro, cioè un prodotto scalare, fra i valori del filtro e quelli della porzione di input al quale è applicato. Lavorando principalmente su immagini, bisogna ricordare che se l'immagine di input presenta un solo un canale, ad esempio è in scala di grigio, il risultato della convoluzione sarà una matrice bidimensionale, altrimenti lo stesso filtro dovrà essere utilizzato per ogni canale e l'output sarà dato dalla somma dei risultati per tutti i canali. Classicamente ogni filtro è associato ad una funzione, "feature", ovvero una specifica caratteristica dell'immagine che si vuole identificare, che può essere di basso livello, come il riconoscimento di linee, curve o comunque elementi semplici, oppure di alto livello, parliamo del riconoscimento di elementi complessi come ad esempio interi oggetti.
Nel definire un layer convoluzionale è fondamentale tener conto di quattro parametri di costruzione, detti anche iperparametri: kernel size, depth, stride e padding.

- il kernel size, o dimensione del filtro, rappresenta le dimensioni della matrice del filtro di convoluzione. É buona regola scegliere sempre valori dispari.

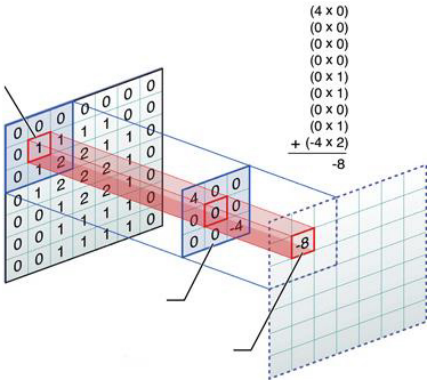


Figura 2.1: Operazione di convoluzione

- il parametro di depth, o profondità, è quello che sostanzialmente controlla il numero di neuroni nel layer convoluzionale che sono connessi alla stessa regione dell'input, ovvero la regione che di volta in volta viene considerata durante l'operazione di convoluzione, la "receptive field".

- il parametro di stride serve per specificare in che maniera traslare il filtro sull'input. Può essere visto anche come il metodo con cui individuare la receptive field lungo lo spazio dell'input. Ad esempio, se si ha uno stride pari a 1, ciò significa che il filtro sarà mosso di un pixel alla volta. Questo potrebbe portare ad un intenso overlapping delle regioni ed anche ad un incremento notevole del volume di output. L'utilizzo invece di uno stride maggiore può portare ad una riduzione dell'overlapping fra le regioni e dunque a dei volumi spazialmente più piccoli.

- l'ultimo parametro è il padding, che indica il numero di pixel da aggiungere all'immagine originale come bordo esterno. Con valori tutti uguali a 0 (si parla infatti di zero-padding) questa procedura rende possibile mappare correttamente e con precisione anche i dettagli che stanno ai bordi dell'immagine. Infatti, in condizioni normali le regioni più esterne non vengono sottoposte al filtro tante volte quanto quelle più interne durante la sua traslazione, che invece possono essere considerate molteplici volte a seconda del passo scelto. La specifica del padding rende inoltre possibile poter controllare artificialmente le dimensioni dell'output finale modicando quelle dell'input: ad esempio se si volesse ottenere un'immagine delle

stesse dimensioni di quella di input, sarà sufficiente usare come numero di padding il valore pari a $(kernelsize - 1)/2$.

2.3.2 Layer di attivazione

Un layer di attivazione è paragonabile alle funzioni di attivazione dei neuroni nelle reti neurali classiche: solo le feature che vengono attivate passano al layer successivo. All'interno di una rete convoluzionale, il layer di attivazione viene inserito dopo ogni layer convoluzionale al fine di introdurre un fattore di non linearità, in quanto le operazioni svolte dal convoluzionale sono del tutto lineari, ricordiamo essere semplici moltiplicazioni tra valori del filtro e del receptive field. La linearità di queste operazioni porterebbe a equiparare una rete con molti layer ad una rete con un solo layer lineare, che di per sé non sarebbe un problema, ma un modello lineare non permetterebbe di ottenere risultati soddisfacenti per decisioni complesse, come il riconoscimento di caratteristiche di immagini.

Sono presenti numerose funzioni di attivazioni e storicamente la più utilizzata è la sigmoide ma, questa funzione come altre, in reti profonde può portare al problema della scomparsa del gradiente [8]. Per questo motivo si è iniziato ad utilizzare e a preferire altre funzioni che aggirano il problema. Nel modello di rete realizzata per questo lavoro è stata utilizzata esclusivamente la funzione di attivazione chiamata ReLU che sarà ora presentata.

Rectified Linear Unit

ReLU è ormai uno standard e risulta essere uno dei layer di attivazione più usato in assoluto, data la sua semplicità e la sua efficienza. Questo tipo di layer non ha nessun parametro impostabile e semplicemente esegue una funzione fissa:

$$f(x) = max(0, x),$$

che banalmente elimina tutti i valori negativi assunti dai pixel delle feature map provenienti dal layer precedente e sostituendoli con 0. Oltre a non avere nessun parametro settabile, non ha alcun parametro allenabile e questo facilita molto la fase di backpropagation.

2.3.3 Layer Dense

Questo tipo di layer è esattamente uguale ad uno qualsiasi dei layer di una classica rete neurale artificiale con architettura fully connected: semplicemente in un layer Dense, ciascun neurone è connesso a tutti i neuroni del layer precedente e nello specifico alle loro attivazioni ed è per questo anche chiamato layer "Fully Connected" (FC). La computazione svolta è esattamente la stessa, di tipo lineare, che consiste in un prodotto scalare fra matrici. L'unico parametro settabile in modo arbitrario di questo tipo di layer è il numero di neuroni N che lo costituiscono ovvero la dimensione del suo output. Il suo output sarà infatti un singolo vettore 1 x 1 x N, contenente le attivazioni calcolate. Dopo l'utilizzo di un singolo layer FC si passa quindi da un volume di input organizzato in 3 dimensioni, ricordiamoci del tensore, ad un singolo vettore di output in una singola dimensione e questo fa intuire che dopo l'applicazione di un layer dense non si può più utilizzare alcun layer convoluzionale. Per questo i layer FC sono inseriti alla fine della rete. La funzione principale di questo tipo di layer è quello di effettuare una sorta di raggruppamento delle informazioni ottenute fino a quel momento, esprimendole con un singolo numero, il quale servirà nei successivi calcoli per la classificazione finale. Solitamente si utilizza più di un layer FC in serie e l'ultimo di questi ha il parametro N pari al numero di classi presenti nel dataset. Gli N valori finali saranno infine dati in pasto all'output layer, il quale, tramite una specifica funzione probabilistica, effettuerà la classificazione.

Capitolo 3

Deep Image Prior e SelfDeblur

In questo capitolo sarà presentato il metodo Deep Image Prior e il suo utilizzo nel SelfDeblur per risolvere il problema di blind deconvolution. Verrà illustrata quindi una proposta di modifica del SelfDeblur per quanto concerne la rete utilizzata per la creazione del kernel e l'algoritmo di ricerca e di generazione delle immagini.

3.1 Deep Image Prior

Il metodo Deep Image Prior [13], "DIP", nasce per risolvere i problemi di ripristino delle immagini. Come accennato in precedenza, in questo tipo di problemi si vuole recuperare l'immagine originale x, o meglio, la sua migliore approssimazione x^* avendo a disposizione la sola immagine corrotta y. Matematicamente questo problema può essere formulato come un problema di minimo così descritto:

$$x^* = \min_x E(x; y) + R(x),$$

dove $E(x; y)$ è una tipologia di problema da risolvere mentre $R(x)$ è quello che viene definito "image prior" o regolarizzatore, ovvero un'informazione preliminare dell'immagine che può essere usata nella risoluzione del problema. Tipicamente il prior $R(x)$ viene riconosciuto e ricercato all'interno di un dataset con un vasto numero di esempi attraverso reti CNN.

Il metodo DIP basa il suo sviluppo sull'idea che a x corrisponda una funzione parametrica $f_\theta(z)$ e che il regolarizzatore sia incluso nei parametri θ della rete CNN e che

quindi possa essere ritrovato in essa. Il regolarizzatore può essere espresso quindi come:

$$\theta^* = \underset{\theta}{argmin} E(f_\theta(z); y)$$

in cui θ^* è ottenuto, nella pratica, attraverso la discesa del gradiente in una rete in cui i parametri θ sono inizializzati in modo casuale e l'unica informazione a disposizione è l'immagine corrotta y. Da questo risultato si può dedurre quindi che:

$$x^* = f_{\theta^*}(z),$$

Partendo con θ_0 in cui i pesi della rete sono generati in modo del tutto casuale, si

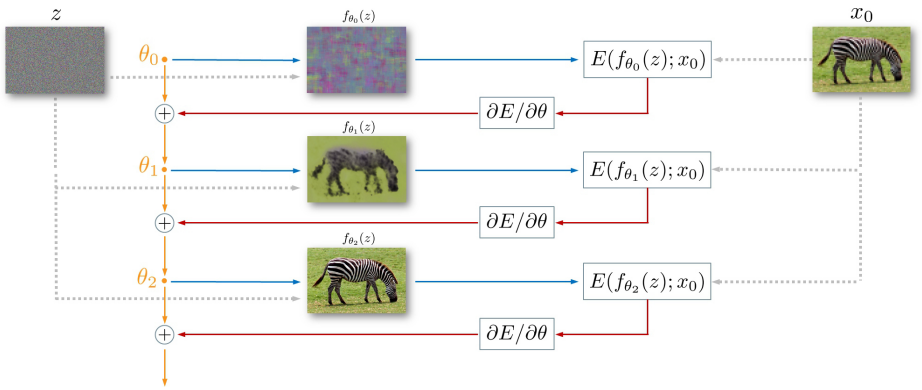


Figura 3.1: Funzionamento del DIP - Immagine tratta da [12]

procede in modo iterativo al loro aggiornamento così da minimizzarne il valore: ad ogni iterazione si utilizzano i parametri della rete per generare una nuova immagine $\tilde{x}$ usando in input un tensore z formato da rumore casuale; questa sarà utilizzata per il calcolo del gradiente con cui si aggiorneranno i pesi della rete. Questa tecnica, grazie alla specifica architettura della rete che consente di convergere verso y più lentamente in presenza di input con rumore casuale, permette di trovare una buona approssimazione di x o per lo meno di andarci vicino.

3.2 SelfDeblur

Come si è potuto osservare il metodo Deep Image Prior [13] mira a risolvere problemi relativi al ripristino delle immagini ma quello della deconvoluzione non viene affrontato.

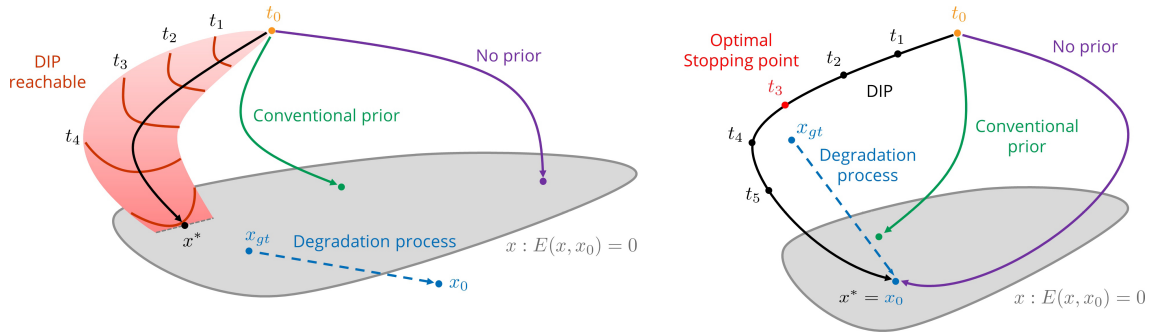


Figura 3.2: Ricostruzione con prior - Immagine tratta da [12]

Ispirati dal lavoro di Ulyanov *et al.*, Dongwei R. *et al.* realizzano un metodo definito "SelfDeblur"[10] che si concentra nella risoluzione del problema di blind deconvolution, con metodo di apprendimento non supervisionato, da cui il nome SelfDeblur. Nel capitolo 1, abbiamo visto che, il problema può essere scritto come:

$$(x, k) = \underset{(x,k)}{argmin} \|k \otimes x - y\|^2 + \lambda\phi(x) + \tau\varphi(k),$$

Utilizzando il modello DIP, x e k sono rimappate da due reti neurali generative portando a riformulare il problema come

$$\underset{(G_x, G_k)}{min} \|G_k(z_k) \otimes G_x(z_x) - y\|^2$$

3.2.1 Struttura del SelfDeblur

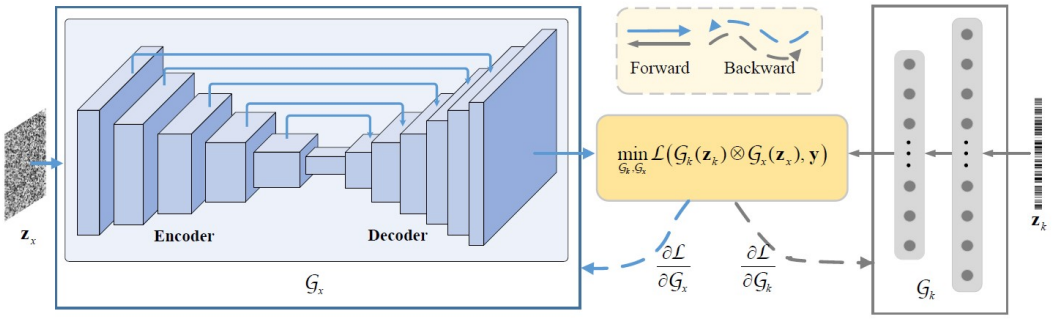


Figura 3.3: Struttura del metodo SelfDeblur - Immagine tratta da [10]

Ripartendo dalla riformulazione è possibile notare due reti generative: G_x e G_k. La prima rete G_x è esattamente la CNN usata nel DIP ideato da Ulyanov *et al.*: come si può vedere dalla Figura 3.3 è composta da un "Autoencoder" asimmetrico con collegamenti diretti tra layer. I primi 5 layer che compongono l'encoder sono direttamente connessi ai rispettivi 5 decoder. L'ultimo layer presente è un layer convoluzionale che genera in output l'immagine ricostruita ricercata. Il suo input Z_x, come detto in precedenza, è fissato e si tratta di rumore casuale.

La seconda rete generativa G_k è una Fully Connected molto semplice, presenta infatti un primo layer di tipo lineare che riceve come input del rumore casuale sotto forma di vettore (1D), la cui dimensione è 200. Il secondo layer è sempre di tipo lineare ed è composto da 1000 nodi, il suo output è un vettore di K^2 elementi, dove K indica la dimensione del lato del kernel. Per garantirne la non-negatività, l'output passa per un layer di attivazione di tipo "SoftMax": questo particolare tipo di layer scala completamente i valori del vettore in modo da essere contenuti nel range $[0, 1]$ la cui come somma è esattamente 1.
Il metodo proposto per la ricerca della soluzione prevede due differenti algoritmi, basati su un diverso metodo di ottimizzazione: uno definito "Alternating Optimization" e uno definito "Joint Optimization".

Alternating Optimization

Il primo algoritmo che osserviamo è quello definito Alternating per via dell'alternanza nell'ottimizzazione delle reti e può essere riassunto come segue:

Alternating Optimization

Input: Immagine sfocata y
Output: Kernel k e immagine ricostruita x

1. Costruzione di z_x e z_k con rumore causuale
2. k = $G_k^0(z_k)$
3. **for** $t = 1$ to T **do**

4. x = $G_x^{t-1}(z_x)$
5. Calcolo del gradiente della rete G_k
6. Aggiornamento della rete G_k utilizzando come algoritmo di ottimizzazione ADAM [3]
7. k = $G_k^t(z_k)$
8. Calcolo del gradiente della rete G_x
9. Aggiornamento della rete G_x utilizzando come algoritmo di ottimizzazione ADAM [3]
10. **end for**
11. x = $G_x^T(z_x)$, k=$G_k^T(z_k)$

Come si può notare questo primo algoritmo, dopo aver generato una prima stima del kernel k_0 con la rete G_k di tipo FC, prevede un ciclo di T iterazioni, dove in ognuna di essa viene prima calcolata x_{t-1} attraverso la rete G_x di tipo DIP, si passa quindi all'aggiornamento dei pesi[1] della rete G_k risolvendo il problema di minimo con x_{t-1} e k_{t-1}. Eseguito quest'ultimo si genera una nuova k_t e si procede a un nuovo aggiornamento dei pesi di G_x risolvendo nuovamente il problema di minimo ma questa volta con x_{t-1} e k_t. Il procedimento continua fino alla conclusione delle T iterazioni, al termine del quale non resta che produrre l'immagine e il kernel stimati.

Joint Optimization

Il secondo che osserviamo è quello definito Joint in cui l'ottimizzazione delle reti viene eseguito in contemporanea per entrambe. L'algoritmo può essere riassunto come segue:

[1] L'aggiornamento dei pesi include fase 5 e 6

Joint Optimization

Input: Immagine sfocata y

Output: Kernel k e immagine ricostruita x

1. Costruzione di z_x e z_k con rumore causuale
2. **for** $t = 1$ to T **do**
3. k = $G_k^{t-1}(z_k)$
4. x = $G_x^{t-1}(z_x)$
5. Calcolo del gradiente delle reti G_k e G_x
6. Aggiornamento delle rete G_k e G_x utilizzando come algoritmo di ottimizzazione ADAM [3]
7. **end for**
8. x = $G_x^T(z_x)$, k=$G_k^T(z_k)$

Al contrario del precedente, questo algoritmo aggiorna le reti in contemporanea, infatti, nel suo processo di iterazione genera prima k_{t-1} e x_{t-1}, poi usa entrambi per risolvere il problema di minimo e successivamente procede con l'aggiornamento delle reti. Si potrebbe quindi considerare uno sviluppo in parallelo.
Il Joint Optimization risulta convergere a soluzioni migliori rispetto all'Alternating.[9]

3.3 Proposte di modifiche

La fase sperimentale di questa tesi è iniziata analizzando e mettendo alla prova il metodo SelfDeblur con algoritmo di Joint: osservandone il codice è stato possibile notare che la ricerca del kernel avviene per dimensione fissata e che per il dataset di Levin *et al.*[6], utilizzato in fase di test, siano specificate dimensioni diverse in base al kernel usato per sfocare l'immagine. Sono stati quindi modificati i parametri relativi la dimensione

Figura 3.4: Confronto tra metodi Alternating e Joint - Immagine tratta da [10]

per valutarne i risultati in caso di sovrastima e sottostima. Come mostrato in Figura

(a) Dimensione sottostimata (b) Dimensione corretta (c) Dimensione sovrastimata

Figura 3.5: Test dimensione kernel

3.5 in entrambi i casi la soluzione risulta essere molto simile a quella prodotta con dimensione prestabilita, ma si possono osservare delle differenze. Appare evidente che in caso di sottostima, figura 3.5 (a), il kernel non riesce a rappresentare completamente lo shape del blur e nel caso in cui fosse eccessiva porterebbe a soluzioni non ottimali. Al contrario, in caso di sovrastima, figura 3.5 (c), lo shape è contenuto completamente

all'interno del kernel prodotto ma non centrato rispetto l'originale, portando, così, allo "shifting" dell'immagine, ovvero uno slittamento dei pixel pari alla distanza e al verso dello shape rispetto al centro.
Dopo aver verificato la sua efficienza nei casi sopra riportati, escludendo il caso della eccessiva sottostima, la seconda fase di test si è focalizzata nel mettere alla prova il metodo SelfDeblur con una differente tipologia di kernel non presente in alcun dataset utilizzato nel paper[10]: il kernel gaussiano.

3.3.1 Kernel gaussiano

Come accennato nell'introduzione il fenomeno di sfocatura può verificarsi per svariati motivi e in diversi campi della fotografia digitale: quella "gaussiana", in particolare caratterizza la fotografia astronomica e microscopica. Nel campo dell'imaging applicare un kernel gaussiano equivale ad applicare una funzione gaussiana all'immagine provocando una riduzione del rumore e dei dettagli. L'effetto visivo presenta una sfocatura morbida simile a quella della visione dell'immagine attraverso uno schermo traslucido.
I test effettuati sul metodo SelfDeblur hanno reso evidente la sua difficoltà nel rendere

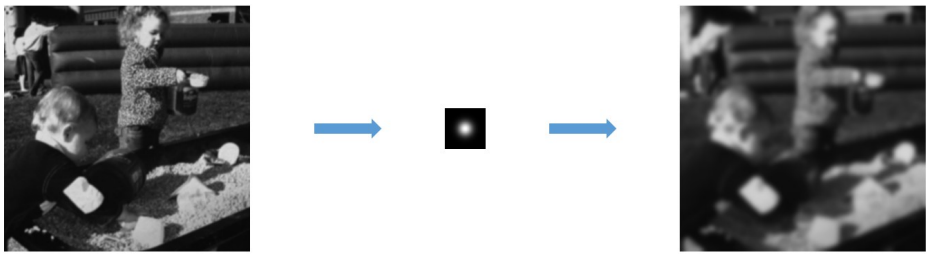
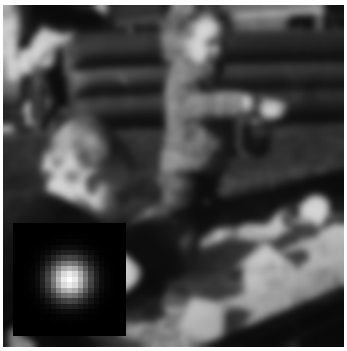


Figura 3.6: Applicazione di un kernel gaussiano

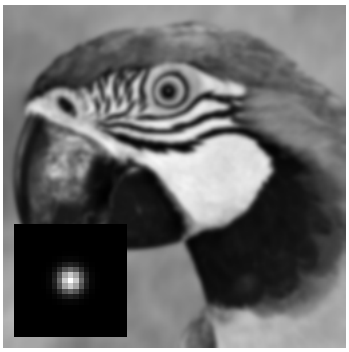
risultati ottimali in caso di kernel gaussiano. Nello specifico la struttura della rete FC G_k riproduce un kernel rumoroso e che non presenta il tipico effetto "smooth" della funzione

gaussiana. Si è deciso, allora, di "specializzare" il metodo, applicando alcune modifiche per migliorarne la qualità delle soluzioni proposte.

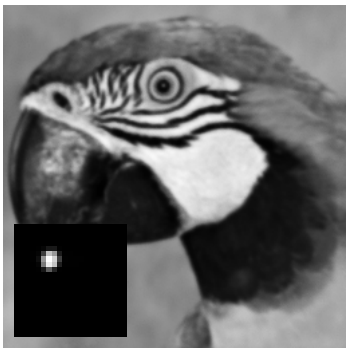


(a) Immagine blurrata

(b) Risultato SelfDeblur



(c) Immagine blurrata



(d) Risultato SelfDeblur

Figura 3.7: Test kernel gaussiano

3.3.2 Modifiche proposte

La prima importante modifica al metodo ha riguardato la rete FC G_k interamente sostituita da una rete CNN. Prima di giungere alla struttura proposta, ne sono state testate diverse variando dalla profondità della rete alla tipologia dei layer di attivazione. Le prime architetture si sono dimostrate, fin da subito, più efficienti nella ricerca di kernel gaussiani e si è, infine, optato per una struttura piramidale con profondità dei layer convoluzionali crescenti-decrescenti, rispettivamente 8-16-32-32-16-8, intervallati da layer di attivazione ReLU, il cui output finale viene convertito in tensore monodimensionale

attraverso una utility chiamata “Flatten”.
Per rendere efficiente la rete G_k anche su kernel di tipo motion, ossia quelli che sono generati dal movimento in fase di cattura, sono stati aggiunti due layer lineari, separati da un ReLU e un layer di attivazione Softmax, attivabili tramite flag al momento dell'esecuzione.

La seconda modifica proposta riguarda invece il dato di input: come visto in precedenza

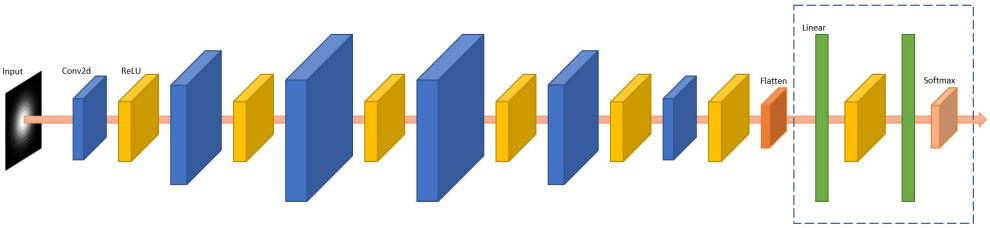
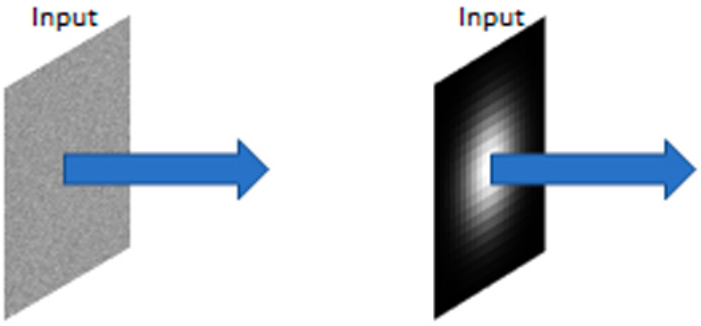


Figura 3.8: Struttura rete CNN proposta

la rete G_k riceve come input del rumore casuale ma volendo specializzare il metodo nel rilevare kernel gaussiano si è pensato di modificarlo direttamente in un kernel gaussiano, rendendo di fatto G_k una rete che trasforma un kernel gaussiano in un altro kernel gaussiano. I test hanno dimostrato la capacità di convergere a soluzioni migliori rispetto al precedente input e in special modo rispetto alla precedente rete FC.

Un'ulteriore proposta di modifica riguarda l'introduzione di una fase preliminare nell'algoritmo Joint Optimization. Esaminando la figura 3.10 (d) è possibile notare un decentramento del kernel gaussiano rispetto al kernel che non si presenta invece in (b). Questo perchè l'algoritmo di Joint lavorando in parallelo, nelle fasi iniziali, aggiorna i pesi della rete del kernel risolvendo il problema di minimo con una x molto distante dalla soluzione reale, rendendo di fatto il centramento del kernel suscettibile all'immagine di input. Prendendo in considerazione l'algoritmo Alternating, visto in precedenza, se ne è sfruttata l'idea eseguendo per primo un aggiornamento della rete G_x, risolvendo il problema di minimo con x_t, ovvero l'immagine x generata ad ogni $t-1$ iterazione dalla rete, e $k = Z_k$, l'input con kernel gaussiano. Questa operazione consente di avvicinarsi ad un



(a) Z_k = Rumore casuale (b) Z_k = Kernel gaussiano

Figura 3.9: Modifica input rete G_k

insieme di pesi che riproducono un'immagine più vicina a quella che si vuole ricostruire e crea una base ottimizzata per l'algoritmo di Joint.
Le proposte di modifica hanno restituito risultati ottimali in termini qualitativi, abbattendo, inoltre, notevolmente i tempi di computazione rispetto al metodo SelfDeblur.

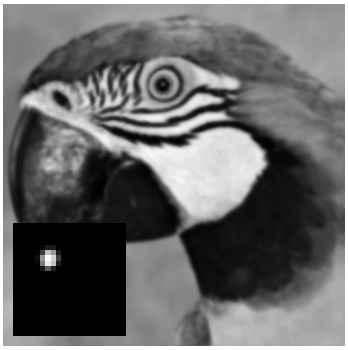
Joint Optimization proposta di modifica

Input: Immagine sfocata y
Output: Blur kernel k e immagine ricostruita x

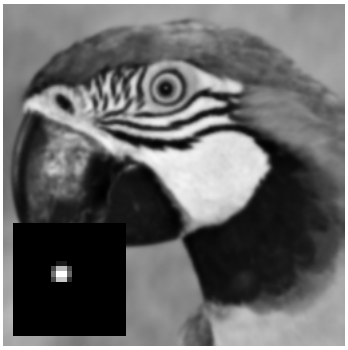
1. Costruzione di z_x con rumore causuale
2. Costruzione di z_k con kernel gaussiano
3. **for** $t = 1$ to T **do**
4. x = $G_x^{t-1}(z_x)$
5. Calcolo del gradiente della rete G_x
6. Aggiornamento della rete G_x utilizzando come algoritmo di ottimizzazione ADAM [3]
7. **end for**

(a) Risultato con rete FC

(b) Risultato con rete CNN



(c) Risultato SelfDeblur



(d) Risultato con rete CNN

Figura 3.10: Confronto FC - CNN

8. **for** $t = 1$ to T **do**
9. k = $G_k^{t-1}(z_k)$
10. x = $G_x^{t-1}(z_x)$
11. Calcolo del gradiente delle reti G_k e G_x
12. Aggiornamento delle rete G_k e G_x utilizzando come algoritmo di ottimizzazione ADAM [3]
13. **end for**
14. x = $G_x^T(z_x)$, k=$G_k^T(z_k)$

Capitolo 4

Risultati numerici

In questo ultimo capitolo saranno esposti i risultati numerici e grafici ottenuti applicando il metodo SelfDeblur e le modifiche proposte, viste nella Sezione 3.3.2 . I risultati riportati faranno leva principalmente sul variare dei risultati per diversi valori della deviazione standard relativa al kernel gaussiano utilizzato, l'intensità del rumore casuale additivo aggiunto e sul confronto fra il metodo SelfDeblur e la proposta suggerita.

4.1 Criteri di valutazione

Quando un'immagine risulta sfocata subisce inevitabilmente, una perdita di informazioni che ne compromette la qualità. Lo scopo del blind deconvolution è quello di ripristinare le feature di questa immagine per riportarla alla sua originaria definizione, recuperandone le informazioni perse e di ricercare il kernel di sfocatura. Si rende quindi necessario stabilire dei criteri per valutare la bontà dell'approssimazione ottenuta. In questo senso è possibile distinguere due classi di metodi di valutazione:

- Metodi soggettivi: si basano sulla percezione soggettiva di chi analizza le immagini e non prevedono nessun riferimento numerico prestabilito.
- Metodi oggettivi: utilizzano formule matematiche e statistiche per ottenere valori numerici che esprimano le caratteristiche dell'immagine, in modo da poter eseguire confronti tra grandezze della stessa unità di misura e giungere così a risultati universalmente comparabili.

Saranno ora definiti i parametri su cui si baserà il confronto tra i risultati: il Mean Squared Error (MSE), il Peak Signal to Noise Ratio (PSNR) e lo Structural Similarity Index Measure (SSIM). Tutti sono comunemente utilizzati nei problemi legati all'imaging e rientrano nella categoria dei metodi oggettivi.

4.1.1 MSE

Il Mean Squared Error (MSE) rappresenta l'errore quadratico medio tra due funzioni f e g e nel caso specifico dell'imaging le due funzioni rappresentano l'immagine originale e quella ricostruita. La sua forma è la seguente:

$$MSE(f,g) = \frac{1}{WH}\sum_{i=1}^{W}\sum_{j=1}^{H}(f_ij - g_ij)^2,$$

con W e H rispettivamente la larghezza e l'altezza delle immagini.

4.1.2 PSNR

Il Peak Signal to Noise Ratio (PSNR) esprime il rapporto tra il massimo valore assunto dai pixel di un'immagine e il rumore ad essa applicato. Considerando ancora una volta f e g, come rappresentazione dell'immagine originale e di quella ricostruita, è possibile calcolare il valore PSNR applicando la formula:

$$PSNR(f,g) = 10\log_{10}\frac{max(f)}{MSE(f,g)}$$

Se il MSE tende a zero, il PSNR tende all'infinito, perciò maggiore sarà il valore PSNR di un'immagine, migliore sarà la sua qualità.

4.1.3 SSIM

Lo Structural Similarity Index Measure (SSIM)[14] misura la somiglianza fra due immagini tenendo in considerazione, a differenza del PSNR, i termini di percezione della qualità del sistema visivo umano. Questo perché, il SSIM considera la distorsione totale di un'immagine come il prodotto di tre fattori variabili: luminosità l, contrasto c e

struttura s. Può essere formulato come:

$$SSIM(f,g) = l(f,g) \cdot c(f,g) \cdot s(f,g)$$

dove f e g indicano, come in precedenza, le immagini da confrontare. Luminosità, contrasto e struttura sono definiti come segue:

$$l(f,g) = \frac{2\mu_f\mu_g + C_1}{\mu_f^2 + \mu_g^2 + C_1}$$

$$c(f,g) = \frac{2\sigma_f\sigma_g + C_2}{\sigma_f^2 + \sigma_g^2 + C_2}$$

$$s(f,g) = \frac{\sigma_{fg} + C_3}{\sigma_f\sigma_g + C_3}$$

in cui μ è la luminosità media, σ è la deviazione standard che misura il contrasto e C_1, C_2, C_3 sono costanti inserite per evitare l'annullamento del denominatore. Come per il PSNR maggiore sarà il valore e migliore sarà la correlazione tra le due immagini ma nel SSIM il range di valori è contenuto nell'intervallo $[0, 1]$.

4.2 Test-set

Per valutare le performance delle modifiche proposte, è stato costruito un test-set contenente immagini provenienti da diversi dataset tra cui quello di Levin *et al.*[6] usato nel paper[10] e alcune "Manmade". Tutte le immagini a colori sono state convertite in scala di grigio per pura scelta implementativa, e ridimensionate per permettere i test sulla macchina in uso. (Figura 4.1)

4.3 Risultati e analisi

Prima di presentare i risultati e le loro analisi, va specificato che in tutti i test eseguiti non è stato effettuato il processo di registrazione dell'immagine sull'output prodotto dai metodi messi a confronto. I risultati numerici potrebbero quindi presentare dei valori inferiori a quelli che si sarebbero potuti ottenere con la registrazione. Il processo non è stato eseguito volutamente per enfatizzare, come, il problema dello shifting dovuto al decentramento del kernel possa produrre risultati negativi in termini numerici.

Figura 4.1: Alcune delle immagini usate per il testset:
“im01” e “im2” (Levin), “parrot” e “tiger”, “manmade02” (Manmade).

4.3.1 Confronto con SelfDeblur

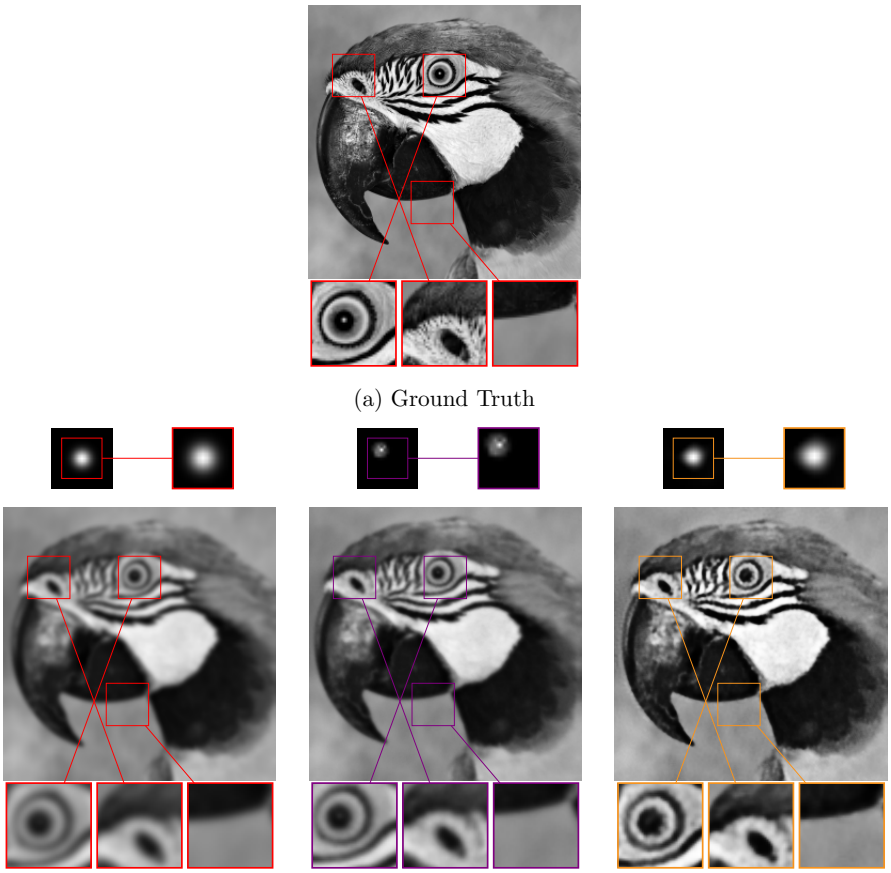
In questa prima sezione sarà mostrato un primo confronto tra i risultati prodotti dal SelfDeblur e la sua versione modificata contenente le proposte osservate in questo lavoro, che indicheremo per semplicità come "SelfDeblur*".
A tale scopo vengono proposte:

- l'immagine originale (a) presente nel test-set, che da qui in avanti sarà definita come "Ground Truth" (GT);
- una sua versione sfocata con kernel gaussiano (b), la cui deviazione standard sarà identificata dalla variabile σ e con l'eventuale aggiunta di rumore casuale additivo, la cui intensità sarà indicata dalla variabile η;
- i due risultati ottenuti con i rispettivi metodi (c,d).

La prima immagine messa a confronto è la "parrot" del test-set che è stata sfocata con kernel gaussiano $\sigma = 2$ e $\eta = 0$. (Figura 4.2)
Visibilmente è possibile notare come le modifiche proposte apportino notevoli miglioramenti a livello qualitativo e i valori riportati in Tabella 4.1 ne danno conferma. Il MSE è stato utilizzato per valutare la qualità del kernel ricostruito, mentre per l'immagine sono stati utilizzati il PSNR e il SSIM.
Un ulteriore test con parametri $\sigma = 2$ e $\eta = 0$ è stato eseguito sull'immagine "manmade02" del test-set per confermare i risultati visti nel primo confronto. In Figura 4.3 è possibile vederne le ricostruzioni e il paragone con l'immagine GT. L'applicazione del SelfDeblur* produce ancora una volta risultati migliori rispetto al SelfDeblur. Si può notare inoltre, che in entrambi i test, a differenza del SelfDeblur*, il SelfDeblur genera un kernel decentrato, sviluppando lo shifting dell'immagine. Nella Tabella 4.2 è possibile

Metodo	MSE	PSNR	SSIM
SelfDeblur	0.0001	16.4693	0.5815
SelfDeblur*	1.9645e-06	23.5723	0.8369

Tabella 4.1: Confronto risultati numerici "parrot"



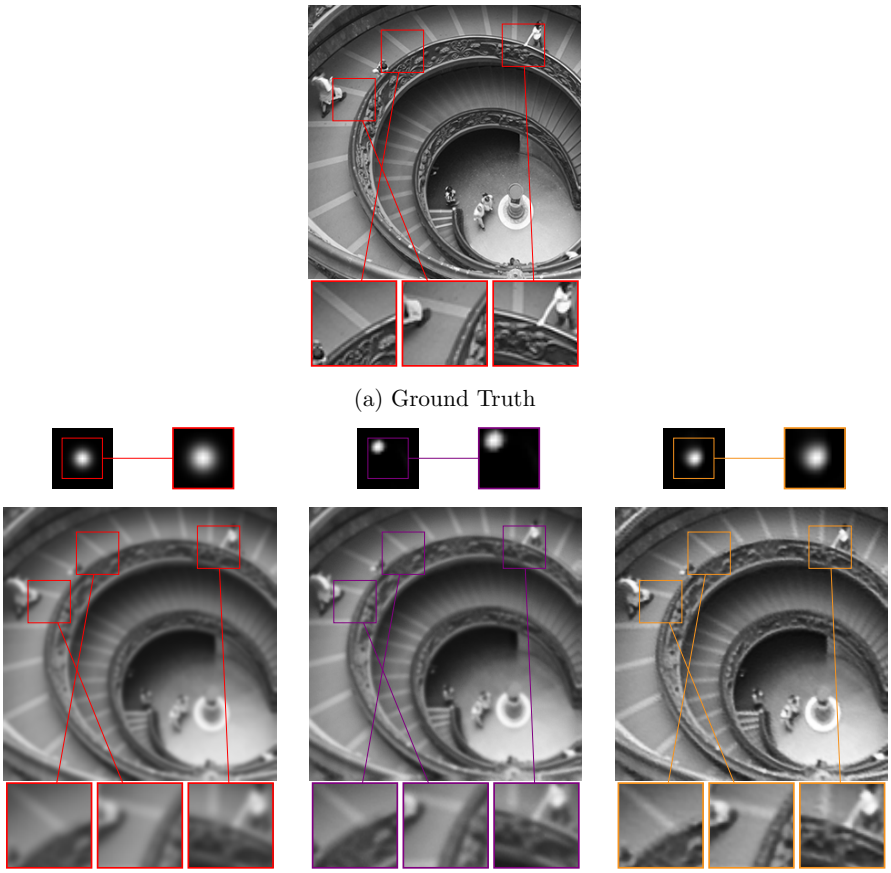
(a) Ground Truth

(b) Sfocata $\sigma = 2$ (c) SelfDeblur (d) SelfDeblur*

Figura 4.2: Confronto SelfDeblur e SelfDeblur* su 07

comparare i risultati numerici.

Per concludere il confronto tra metodi è stato eseguito un ultimo test sull'immagine "tiger" a cui è stato applicato un kernel gaussiano con $\sigma = 1.5$ e del rumore additivo pari a $\eta = 5$. Come vedremo nella prossima sezione 4.3.2, l'aggiunta di rumore additivo può portare all'overfitting e per questo motivo in questo test sono stati considerati solo i migliori risultati ottenuti per entrambi i metodi. Come per i precedenti test vengono mostrati in Figura 4.4 i confronti. Osservando le immagini in Figura 4.4 si potrebbe essere portati a pensare che il metodo SelfDeblur abbia ottenuto un risultato migliore, ma un'attenta analisi porta alla luce la presenza di artefatti causati dall'eccessivo rumore presente in essa (c). E inoltre visibile come il kernel prodotto sia anche in questo caso



(a) Ground Truth

(b) Sfocata $\sigma = 2$ (c) SelfDeblur (d) SelfDeblur*

Figura 4.3: Confronto SelfDeblur e SelfDeblur* su manmade

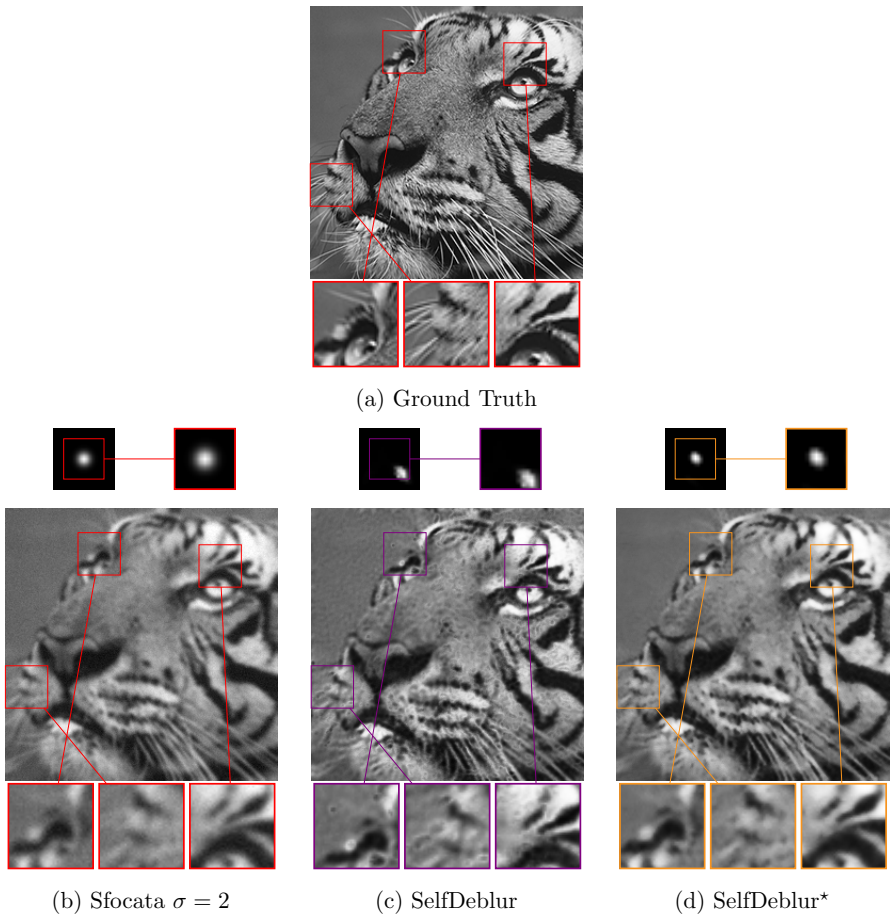
decentrato. Presentiamo ancora una volta la tabella comparativa relativa al MSE, PSNR e SSIM (Tabella 4.3).

4.3.2 Overfitting

Mostriamo in questa sezione come per immagini molto rumorose il metodo SelfDeblur* soffra del problema di overfitting, ovvero all'aumentare delle iterazioni l'output prodotto dalla rete converge all'input sfocato y.
Per questo test è stata utilizzata l'immagine "parrot" del test-set a cui è stato applicato un kernel gaussiano con $\sigma = 1.5$ e rumore additivo $\eta = 10$. Osservando la Figura 4.5 è possibile notare che lo stop ottimale per un miglior risultato sia intorno alle 700 iterazio-

Metodo	MSE	PSNR	SSIM
SelfDeblur	0.0001	18.7147	0.4865
SelfDeblur⋆	1.6471e-05	24.2277	0.8007

Tabella 4.2: Confronto errori “manmade”



(a) Ground Truth

(b) Sfocata $\sigma = 2$ (c) SelfDeblur (d) SelfDeblur⋆

Figura 4.4: Confronto SelfDeblur e SelfDeblur⋆ su tiger

ni. Andando avanti infatti si produrrà un’immagine sempre più simile a quella iniziale fino a convergere completamente a y.

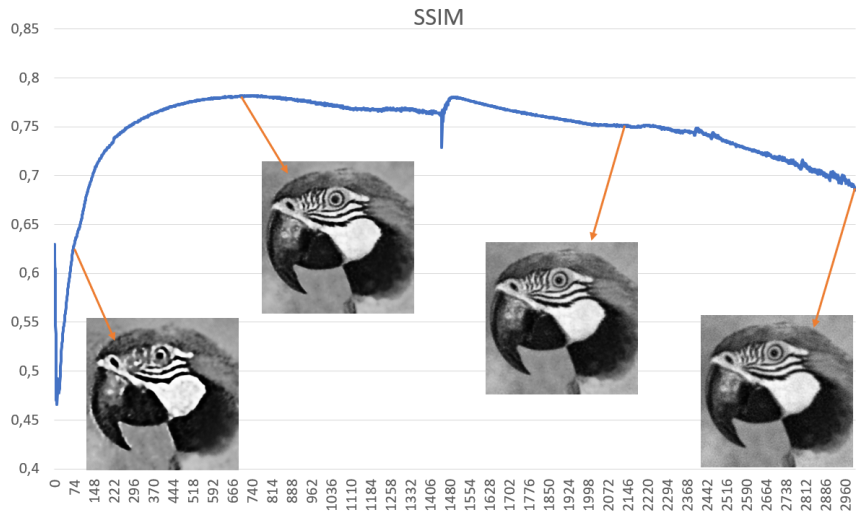


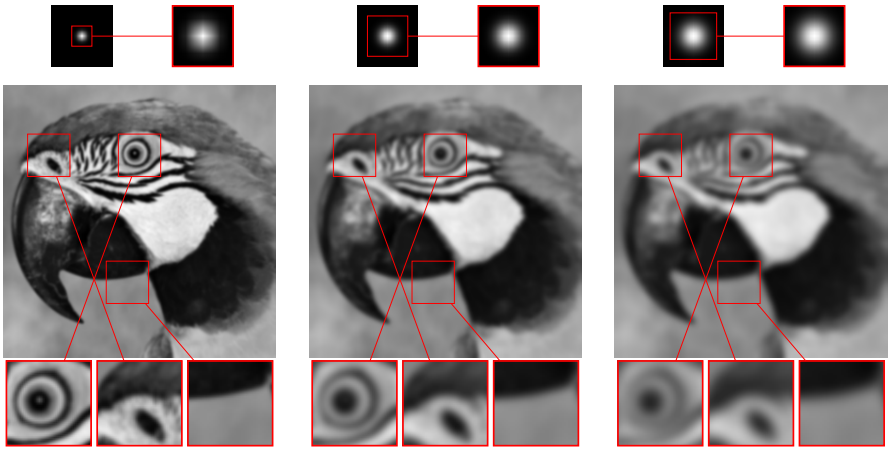
Figura 4.5: Grafico andamento SSIM e problema overfitting

4.3.3 Sfocatura σ

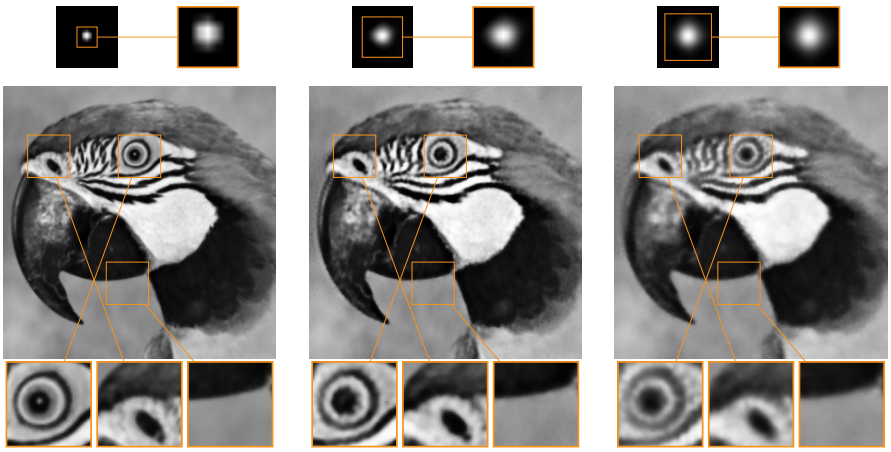
Saranno ora analizzati i risultati ottenuti dal SelfDeblur* al variare del σ, variabile che indica la deviazione standard del kernel gaussiano applicato all'immagine GT. Di seguito, in Figura 4.6 vengono riportate le immagini di confronto, sfocate/ricostruite, con relativi kernel. Sarà possibile notarne i dettagli per un miglior paragone visivo. La tabella 4.4 mostra i risultati numerici degli errori: come in precedenza il MSE è stato utilizzato per valutare la qualità del kernel ricostruito e il PSNR e il SSIM per valutare le immagini ricostruite. Per il test è stata utilizzata l'immagine "parrot" del test-set e si è variato il σ con valori rispettivamente di [1,2,3].

Metodo	MSE	PSNR	SSIM
SelfDeblur	0.0001	14.6370	0.2191
SelfDeblur*	3.0919e-05	19.8051	0.6236

Tabella 4.3: Confronto errori "tiger"



(a) Sfocata $\sigma = 1$ (b) Sfocata $\sigma = 2$ (c) Sfocata $\sigma = 3$



(d) Ricostruita $\sigma = 1$ (e) Ricostruita $\sigma = 2$ (f) Ricostruita $\sigma = 3$

Figura 4.6: Confronto con variazione σ.

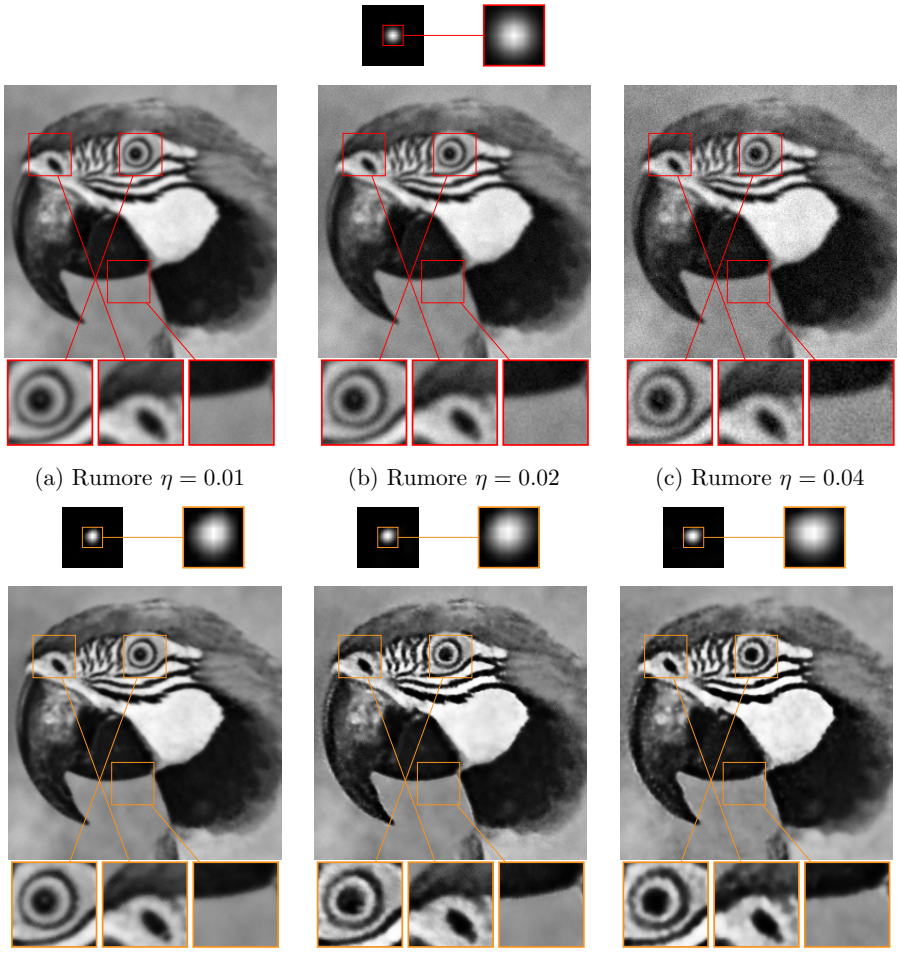
4.3.4 Intensità rumore additivo

Vengono ora mostrati i risultati ottenuti variando l'intensità del rumore casuale additivo. Per questo test è stata utilizzata nuovamente l'immagine "parrot" del test-set a cui è stata applicata una sfocatura gaussiana con valore $\sigma = 1.5$. Successivamente nell'immagine è stato introdotto del rumore casuale additivo che ha subito la variazione con valori rispettivamente di [0.01,0.02,0.04]. Considerando il problema di overfitting visto in sezione 4.3.2, per ogni η saranno presi in considerazione i migliori risultati ottenuti, che non equivalgono necessariamente con l'output prodotto al termine delle iterazioni e

Sfocatura	MSE	PSNR	SSIM
$\sigma = 1$	5.2865e-05	28.6661	0.9194
$\sigma = 2$	1.9645e-06	23.5723	0.8369
$\sigma = 3$	1.3296e-06	21.6674	0.7686

Tabella 4.4: Valori di MSE, PSNR e SSIM al variare di σ

mostrati in Tabella 4.5. In Figura 4.7 è possibile visionare il confronto.



(a) Rumore $\eta = 0.01$ (b) Rumore $\eta = 0.02$ (c) Rumore $\eta = 0.04$

(d) Ricostruita $\eta = 0.01$ (e) Ricostruita $\eta = 0.02$ (f) Ricostruita $\eta = 0.04$

Figura 4.7: Confronto per variazione η.

Rumore	MSE	PSNR	SSIM
$\eta = 0.01$	9.2218e-06	24.4054	0.8250
$\eta = 0.02$	7.9693e-06	23.3774	0.7954
$\eta = 0.04$	6.5682e-06	23.9606	0.7820

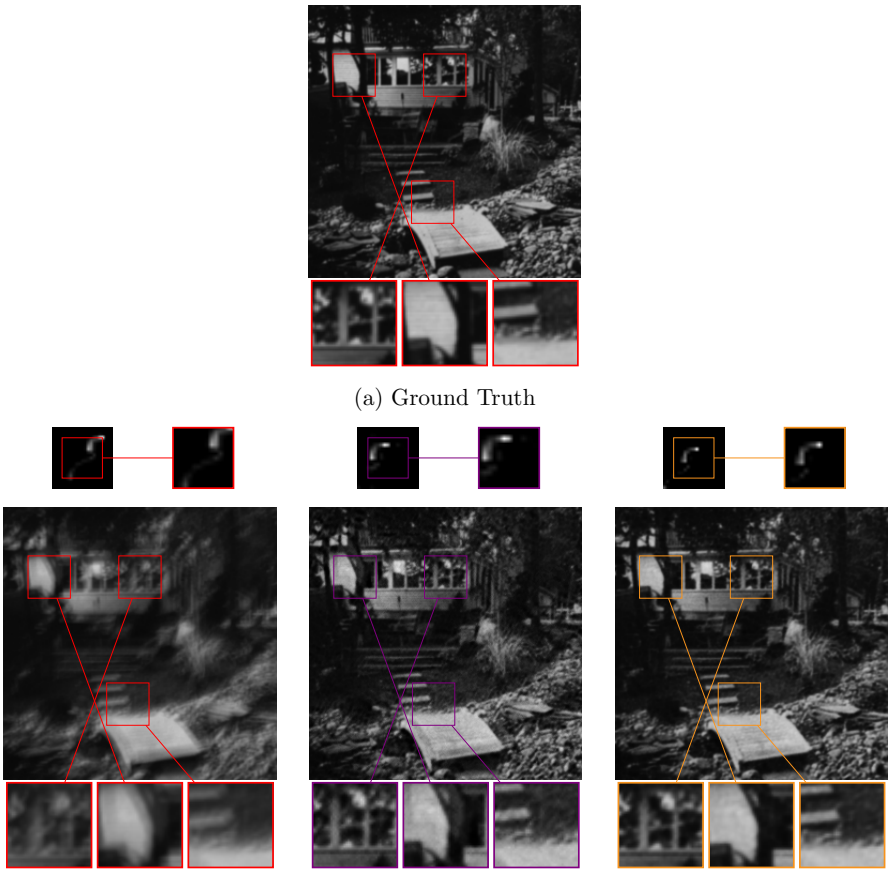
Tabella 4.5: Valori di MSE, PSNR e SSIM al variare di η

4.3.5 Kernel Motion

In ultima analisi il SelfDeblur* è stato messo alla prova con un kernel di tipo motion, per verificarne la sua efficacia anche su tipi diversi dal gaussiano. A questo scopo è stata utilizzata l'immagine "im2" del test-set a cui è stato applicato il "kernel6" del dataset di Levin *et al.*[6].
Il confronto con il SelfDeblur è più che positivo; visionando la Figura 4.8 non risultano differenze sostanziali tra i due metodi, nonostante il SelfDeblur* risulti ottimizzato per un'altra tipologia di kernel. Anzi, come riportato in Tabella 4.6 i valori PSNR e SSIM presentano leggere differenze a vantaggio del metodo proposto, in quanto l'immagine prodotta mostra minor shifting e minor rumore, avvicinandosi maggiormente a quella GT.

Rumore	PSNR	SSIM
SelfDeblur	19.7194	0.5119
SelfDeblur*	20.1816	0.5452

Tabella 4.6: Confronto valori PSNR e SSIM su kernel motion



(a) Ground Truth

(b) Sfocata kernel motion (c) SelfDeblur (d) SelfDeblur⋆

Figura 4.8: Confronto SelfDeblur e SelfDeblur⋆ su im2

Conclusioni

In questo lavoro di tesi si è cercato di fornire una panoramica sul problema di blind deconvolution e sulla sua risoluzione con l'intento finale di proporre una tecnica risolutiva che non preveda alcun tipo di addestramento apportando modifiche al metodo SelfDeblur, utilizzato come punto di partenza.
Grazie all'introduzione di una nuova rete neurale convoluzionale, che ha sostituito quella contenuta nel SelfDeblur, sono stati raggiunti risultati migliori nella ricostruzione di immagini sfocate con kernel gaussiani e sicuramente equiparabili in caso di tipo motion. L'inserimento di una nuova fase preliminare nell'algoritmo di Joint ne ha permesso una maggiore stabilità e un decremento dei tempi computazionali nella ricerca di una soluzione, arrivando anche a dimezzarli.
Sicuramente il problema di blind deconvolution è risultato complesso e questo offre lo spunto per ampi margini di miglioramento. In particolare, ulteriori test potrebbero essere eseguiti sulla variazione degli iperparametri della rete e dello stesso algoritmo per valutarne comportamenti e risultati. Interessante sarebbe anche ricercare ottimizzazioni per quanto concerne la dimensione del kernel che in questo lavoro è stato mantenuto sempre costante ma che come si è visto in caso di sottostima produrrebbe risultati non ottimali.

Bibliografia

[1] HC. Burger, CJ. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with bm3d? pages 2392 – 2399, June 2012.

[2] J. Hadamard. *Sur les problèmes aux dérivées partielles et leur signification physique*, page 49–52. Bull. Univ. Princeton, 1902.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

[5] Stamatios Lefkimmiatis. Non-local color image denoising with convolutional neural networks, 2017.

[6] A. Levin, Y. Weiss, F. Durand, and W. T. Freeman. Understanding and evaluating blind deconvolution algorithms. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1964–1971, 2009.

[7] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dynamic scene deblurring. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[8] Michael A. Nielsen. What's causing the vanishing gradient problem? unstable gradients in deep neural nets. In *Neural Networks and Deep Learning*, 2015.

[9] Dongwei Ren, Kai Zhang, Qilong Wang, Qinghua Hu, and Wangmeng Zuo. Alternating optimization vs. joint optimization. In *Neural Blind Deconvolution Using Deep Priors*, page 7, 08 2019.

[10] Dongwei Ren, Kai Zhang, Qilong Wang, Qinghua Hu, and Wangmeng Zuo. Neural blind deconvolution using deep priors, 08 2019.

[11] Jian Sun, Wenfei Cao, Zongben Xu, and Jean Ponce. Learning a convolutional neural network for non-uniform motion blur removal, 2015.

[12] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. *arXiv:1711.10925*, 2017.

[13] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[14] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, Student Member, Eero P. Simoncelli, and Senior Member. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13:600–612, 2004.