

Rete Convoluzionale per Image Deblurring

Alessio Tanzi, Giovanni Tortia, Francesco Novia

AA2024/2025

Sommario

L'obiettivo del progetto è l'applicazione di una U-Net Convoluzionale al fine di migliorare la qualità dell'immagine in input rimuovendo il Blur causato dal moto del soggetto acquisito (*Motion Blur*) o causato dalla messa a fuoco dell'obiettivo (*Focus Blur*).

1 Introduzione

Il restauro delle immagini è un compito fondamentale della visione artificiale, utile in campi come telerilevamento, imaging medico e fotografia. Il compito consiste nel ricostruire un'immagine pulita da una versione degradata, un problema noto per essere "mal posto".

Tradizionalmente, si sono usate tecniche basate su caratteristiche manuali, ma oggi il deep learning — in particolare CNN e Transformer — ha rivoluzionato l'approccio. L'obiettivo di questo lavoro è esplorare la architettura CNN "ConvIR" [1] capace di egualizzare o superare le prestazioni dei modelli Transformer, mantenendo un numero minore di parametri e complessità computazionale minore. Di seguito le caratteristiche principali della rete in questione:

Architettura multi-scala potenziata: ConvIR imita i meccanismi multi-stage dei Transformer all'interno di una classica struttura a U (U-Net), gestendo i blur da grossolani a fini.

Modulo di attenzione multi-forma (MSA): Un nuovo modulo che aggrega informazioni in regioni quadrate e rettangolari, usando anche *convoluzioni dilatate* per aumentare il campo ricettivo.

Modulazione della frequenza: A differenza di altri metodi che trasformano le immagini nel dominio delle frequenze (es. Fourier), ConvIR regola direttamente l'importanza dei segnali ad alta frequenza tramite pesi di attenzione.

Efficienza computazionale: ConvIR raggiunge prestazioni simili o superiori ai Transformer con meno parametri e FLOPs.

Versioni del modello [1]: Sono proposte tre varianti (Small, Base, Large), le quali determinano la lunghezza del primo blocco residuale all'interno del *CNNBlock*.

Note su tecnologie correlate

- CNN vs Transformer: I Transformer offrono vantaggi nel cogliere relazioni a lungo raggio ma sono pesanti computazionalmente.
- Moduli di attenzione: Varianti avanzate mirano a focalizzarsi su zone importanti dell'immagine.
- Elaborazione spettrale: ConvIR usa strategie alternative ai trasformatori di Fourier e wavelet,

risparmiando tempo computazionale.

2 Modello teorico e approcci tradizionali

Una immagine con blur è modellata matematicamente come convoluzione tra immagine ground-truth latente e kernel di blur, dove quest'ultimo si assume essere *shift-invariant*. In questo caso, l'estrazione dell'immagine sharp è un problema di *Image Deconvolution*, la quale è suddivisa in *Non-blind-deconvolution* e *Blind-deconvolution*.

Formulazione Matematica:

$$\mathbf{b} = \mathbf{i} * \mathbf{k} + \mathbf{n}$$

Dove:

\mathbf{b} : Immagine con blur

\mathbf{i} : Immagine *ground-truth* latente

\mathbf{k} : Blur Kernel

\mathbf{n} : Rumore presente nell'immagine per contare imperfezioni causate dall'acquisizione (quantizzazione, saturazione del colore, risposta non lineare della camera, ...) (Esempio: rumore gaussiano)

Non-Blind Deconvolution In questa metodologia tradizionale, il blur kernel è noto a priori (Esempio: Point Spread Function Gaussiana per Blur senza direzione, Linea con direzione e lunghezza per Blur con direzione).

Uno dei primi metodi utilizzati in questa categoria, implementato come comparazione, è la *Wiener Deconvolution*, il cui obiettivo è la ricerca di un filtro \mathbf{g} tale che, tramite convoluzione con l'immagine blurred \mathbf{b} . Espresso nel dominio di Fourier:

$$\hat{\mathbf{I}} = \mathbf{G}\mathbf{B} \quad (1)$$

$$\mathbf{G} = \frac{|\mathbf{K}|^2}{|\mathbf{K}|^2 + \frac{1}{\text{SNR}}} \frac{1}{\mathbf{K}} \quad (2)$$

Dove:

\mathbf{G} e \mathbf{K} : trasformate di Fourier di \mathbf{g} e \mathbf{k}

SNR: Signal to noise ratio (infinitamente alto se rumore assente)

Un’implementazione di tale metodo di Deblurring si basa su un metodo di ottimizzazione convessa chiamato *Alternating Direction Method of Multipliers* (ADMM)¹

Blind Deconvolution In questa metodologia, il blur kernel è ignoto², dunque parte dell’algoritmo è la *PSF estimation*, modellata come stima di una stima di densità di probabilità.

3 Architettura del modello

L’architettura utilizzata in [1] si basa, come detto, sulla struttura tipica di una U-Net con encoder-decoder convoluzionali, ed estrazione delle feature effettuata a risoluzioni multiple, quindi effettuando downsampling all’immagine di partenza. Di rilevante importanza in tal senso è il ruolo dei *Multi-Scale Modules* (*MSM*), che hanno l’obiettivo di implementare meccanismi di attenzione di diversa forma (quadrata e rettangolare) e dimensione.

In figura 1 è riportata l’architettura completa. La prima caratteristica interessante è che l’immagine in input viene elaborata non solo alla risoluzione primaria (256x256) proveniente dal dataloader, ma viene ulteriormente sottocampionata (rispettivamente alla metà e a un quarto della risoluzione) e reinserita nella rete come input combinato dei layer successivi. L’obiettivo di questa configurazione multi-input multi-output è di analizzare l’immagine degradata secondo diversi livelli di dettaglio, e quindi di individuare pattern e feature più variegati e di diversa intensità.

L’encoder e il decoder hanno una struttura pressoché speculare, con tre skip connection che collegano i due rami, corrispondenti alle tre differenti risoluzioni a cui viene elaborata l’immagine. Come avviene di consueto nelle reti convolutive, al ridursi della dimensione del tensore in larghezza e altezza, cresce il numero di canali. La feature extraction passa infatti attraverso i seguenti moduli:

- (a) layer convolutivo semplice (blocco *Conv* in verde in figura);
- (b) *ConvS*, blocco utilizzato solo per le versioni sottocampionate dell’immagine in input: consiste in una sequenza di quattro layer convolutivi che mantengono costanti le dimensioni di larghezza e altezza;
- (c) *CNNBlock*, blocco costituito da una serie di layer convolutivi raggruppati in $n+1$ blocchi residuali; nell’ultimo di questi blocchi viene inserito anche il *MSM*;
- (d) *MSM* (*Multi-Scale Module*): fonde l’elaborazione di tre moduli *MSA* (*Multi-Scale Attention*), che operano appunto su tre scale dimensionali gradualmente decrescenti. Ogni *MSA* combina di fatto l’output di un *DSA* e un *DRA*;

¹https://stanford.edu/class/ee367/reading/lecture6_notes.pdf

²I metodi con neural network rientrano in questa categoria

- (e) *DSA* (*Dilated Square Attention*): produce prima una attention map concentrandosi sulle aree quadrate del tensore in input, attraverso layer convolutivi e di pooling, e in seguito la elabora attraverso un filtro passa alto con parametri allenabili che sottrae unicamente la componente continua e tende ad esaltare quelle a più alta frequenza, tipicamente responsabili del blur;
- (f) *DRA* (*Dilated Rectangle Attention*): modulo analogo al precedente ma focalizzato su pattern di forma rettangolare, combina attention map in senso verticale e orizzontale.

L’architettura originale proposta in [1] è stata utilizzata dagli autori in tre diversi formati: *S* (*Small*), *B* (*Base*) e *L* (*Large*), ognuna caratterizzata da un numero variabile di blocchi residuali all’interno di ciascun modulo *CNNBlock*. Le configurazioni prevedevano $n = 3$, $n = 7$ e $n = 15$ rispettivamente per le varianti *S*, *B* e *L*.

La loss function utilizzata corrisponde alla somma pesata di un contributo calcolato nel dominio spaziale (\mathcal{L}_1) e uno in frequenza (\mathcal{L}_{freq}), in modo da considerare adeguatamente i diversi apporti dovuti alla presenza del blur:

$$\mathcal{L}_1 = \mathcal{L}_{spatial} = \sum_{i=1}^3 \frac{1}{P_i} \left\| \hat{\mathbf{I}}_i - \mathbf{Y}_i \right\|_1, \quad (3)$$

$$\mathcal{L}_{freq} = \sum_{i=1}^3 \frac{1}{S_i} \left\| [\mathcal{R}(\hat{\mathbf{I}}_i), \mathcal{I}(\hat{\mathbf{I}}_i)] - [\mathcal{R}(\mathbf{Y}_i), \mathcal{I}(\mathbf{Y}_i)] \right\|_1, \quad (4)$$

Dove i indicizza gli output multipli a diverse risoluzioni; $\hat{\mathbf{I}}$ e \mathbf{Y} rappresentano rispettivamente l’immagine elaborata dalla rete e il ground truth; P e S indicano il numero totale di elementi dei tensori presi in considerazione, in modo da avere delle metriche normalizzate; gli operatori $\mathcal{R}()$ e $\mathcal{I}()$ estraggono rispettivamente la parte reale e immaginaria della FFT operata sull’immagine.

La funzione di costo complessiva è così calcolata:

$$\mathcal{L}_{tot} = \mathcal{L}_{spatial} + \lambda \mathcal{L}_{freq}, \quad (5)$$

dove λ è un iperparametro impostato di default a 0.01.

4 Sperimentazioni

Si illustrano di seguito i dettagli del setup sperimentale adottato, includendo le alterazioni apportate sull’architettura originale e al processo di training, indicandone motivazioni e risultati riscontrati.

Dataset I dataset usati sono principalmente due: GOPRO³ e RSBlur⁴. GOPRO è un dataset creato applicando un motion blur artificiale ai frame di

³<https://www.kaggle.com/datasets/lqzmlaq/gopro-large>

⁴https://drive.google.com/drive/folders/1sS8_qXvF4KstJtyYN1DDHsqKke8qwgnT

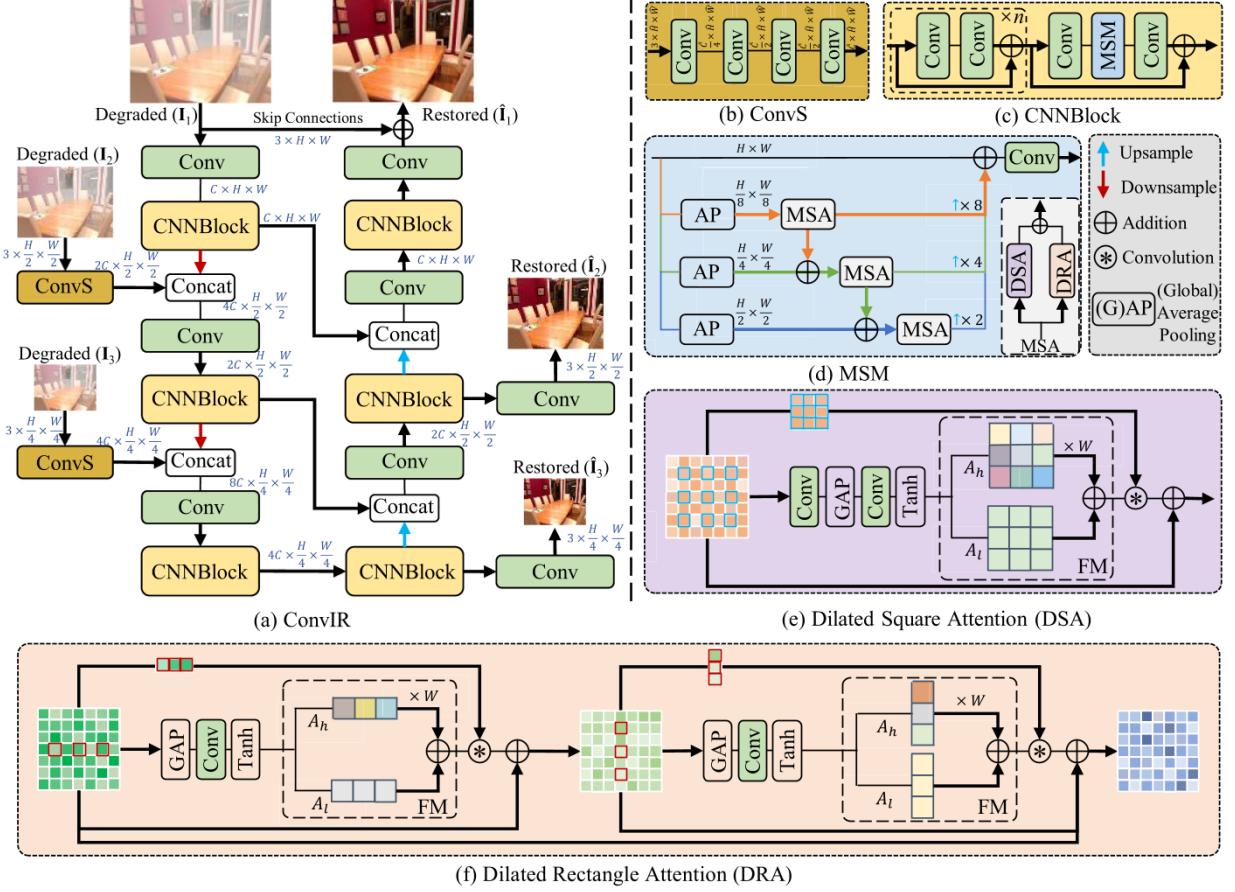


Figura 1: Schema generale dell’architettura completa (ConvIR)

una serie di video girati all’aperto, RSBlur invece usa una speciale fotocamera per creare contemporaneamente una versione sfocata e una nitida della stessa immagine.

Per cercare di ottenere un risultato migliore possibile, abbiamo condotto dei test su entrambi i dataset che risultassero in una singola metrica di performance complessiva, nello specifico:

$$\text{Score} = \frac{1}{2} \sum_{i=1}^2 100 \cdot \text{avg}\left(\frac{\text{PSNR}_i}{33}, \text{SSIM}_i\right)$$

Dove i indica il dataset su cui è stato svolto il test. Questa metrica è stata definita in modo da dare lo stesso peso alle metriche di SSIM e PSNR, oltre che ai due dataset: per ogni dataset si calcolano SSIM e PSNR medio, dopodichè si normalizza il valore di PSNR in modo da riportarlo nell’intervallo $[0, 1]$, si calcola la media tra PSNR e SSIM e la si moltiplica per 100, in modo da ottenere dei valori nell’intervallo $[0, 100]$; ottenuto questo valore quindi lo score è dato dalla media sui dataset.

Inizialmente il training è stato condotto usando GOPRO, ma dal punteggio è emerso come i modelli addestrati su RSBlur ottenessero una performance più bilanciata, mentre invece quelli addestrati su GOPRO tendevano a dare degli ottimi risultati sul test set, ma dei pessimi risultati per quello che riguarda RSBlur.

Nel corso dell’addestramento è stato provato come dataset anche un misto di RSBlur e GOPRO, con scarsi risultati, e alla fine si è deciso di usare esclusivamente RSBlur.

Il training set utilizzato consiste quindi in un sottoinsieme degli esempi di RSBlur, che comprende un totale di 11857 coppie di immagini. Sono stati prodotti diversi sottoinsiemi di RSBlur, compreso uno a risoluzione più bassa dell’originale, ottenuto tramite la funzione `resize(...)`⁵ di OpenCV, ma la versione finale consiste in un training set di 2755 esempi e un test set di 1000 esempi, ottenuto tramite crop delle immagini originale per riportare tutte le immagini alla stessa dimensione (invece che `resize` di OpenCV che interpola).

Oltre ai set citati, abbiamo anche usato un set di 32 immagini sfocate, senza versione nitida, scattato da noi, in modo da ottenere un riscontro più pratico e intuitivo (seppur non molto oggettivo). Nella sezione 5 si possono trovare i risultati su alcune di queste immagini.

Ciascun dataset è stato suddiviso in `train/validation/test` secondo le seguenti modalità:

- Il `test set` è separato nativamente nel dataset scaricato, poiché il `dataloader` si aspetta la pre-

⁵https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html

- senza delle due cartelle `train/` e `test/` nella directory fornita al programma.
- Il **validation set** è stato ottenuto estraendo il **15% finale** delle immagini presenti nella cartella `train/`, ordinate in modo *lessicografico*. Lo split non è casuale, al fine di garantire la **riproducibilità**.

Funzione di Training Ciascun Forward e Backward pass sono eseguiti, a fine di incrementare le prestazioni, con il formato floating point a 16 bit Brain Float, sfruttando il meccanismo di troncamento automatico offerto da Pytorch chiamato *AMP*⁶. Si nota che, in quanto in questo processo si vanno a perdere cifre decimali della mantissa, e dunque i contributi ai vari layer potrebbero risultare nulli, la Loss viene scalata prima di applicare il backward pass con un growth_factor = 2.0

Per quanto riguarda la funzione di costo, abbiamo utilizzato quella descritta in precedenza (5), ma abbiamo aggiunto un termine di gradient penalty, in modo da disincentivare una crescita eccessiva dei gradienti, consentendo quindi una regolarizzazione più efficace del modello.

Il backward pass comprende quindi un primo calcolo dei gradienti, secondo la loss function originale (5), si ottiene poi la penalità, accumulando la norma euclidea di tutti i gradienti appena ricavati, e la si somma alla funzione di costo precedente, infine vengono propagati i gradienti a partire da quest'ultima loss complessiva.

Iperparametri Di seguito gli iperparametri utilizzati assieme al valore selezionato a seguito di una esplorazione trial-and-error:

- Batch Size:**
 - D Numero di Copie di immagini, Example e Label, caricate in un ciclo di Stochastic Gradient Descent
 - V **Valore:** 8. Valori maggiori, eg. 12 tendono a rendere estremamente lento un ciclo di forward e backward propagation, mentre valori superiori o uguali a 32 Esauriscono la memoria del GPU device utilizzato per i test⁷
- Optimizer:** L'ottimizzatore utilizzato è l'algoritmo di Adam[2] con i seguenti parametri
 - $\beta_1 = 0.9$ peso del primo momento
 - $\beta_2 = 0.999$ peso del secondo momento
- Learning Rate (Scheduling)**
 - D Moltiplicatore determinante Step Size dei parametri a partire dalla loss.
 - V **Valore Iniziale:** 10^{-4} . Circa la schedulazione, tra le strategie di schedulazione considerate, in particolare, *Momentum*, *Warmup*, *Decay*, sono state impiegate la strategia di

Gradual Warmup per le prime $N - 3$ epoche, seguita da un *Cosine Decay* nelle epoche rimanenti. Si nota che è stato evitato l'impiego del momento in quanto superfluo con ottimizzatori a learning rate adattivo.⁸ Sono stati testati valori più alti, come ad esempio $1.2 \cdot 10^{-4}$ e $1.5 \cdot 10^{-4}$, ma questi risultavano in differenze minime o addirittura a far divergere la loss.

- Weight Decay:**

D Nell'ambito dell'ottimizzatore Adam implementato da Pytorch, fattore moltiplicativo dei componenti del gradiente da ogni layer. Quando è diverso da zero, ciascun componente è moltiplicato per $1 + \text{weight_decay}$

V 0

- Accumulate Gradient Frequency:**

D Frequenza, in termini di numero di batch processati, di applicazione dei gradienti calcolati dagli ultimi agf backward pass

V 1. Valori Maggiori (sono stati testati 2 e 3) tendono a rendere la curva della loss meno stabilmente decrescente

- λ :**

D Fattore di peso del contributo della Loss nel dominio di Fourier

V 0.1

Data Augmentation Sono mantenute pressoché invariate le trasformazioni usate dagli autori per la data augmentation, ovvero:

- **RandomCrop(256):** Estraie un crop quadrato di lato 256 pixel, al fine di poter allenare la rete su più porzioni dell'immagine, presentando blur pattern diversi alla rete.
- **RandomHorizontalFlip(p=0.5):** Decide casualmente (con una probabilità del 50%) se ruotare intorno all'asse verticale il crop ottenuto al passo precedente.

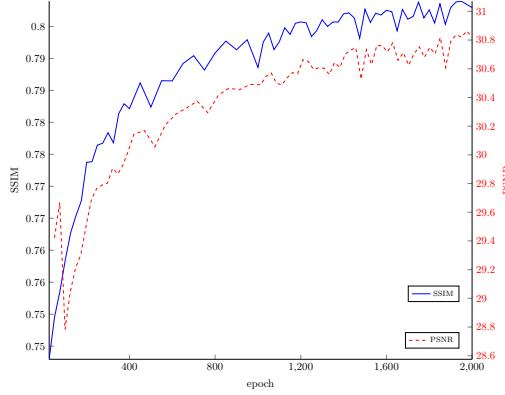
Sono inoltre state introdotte altre trasformazioni alteranti le informazioni sul colore dell'immagine in input, come *ColorJitter*, e *AutoAugment* ([3]) con policy IMAGENET. Queste trasformazioni menzionate sono state in seguito rimosse a seguito di riscontro di performance degradata. Tale risultato trova giustifica nel fatto che modificare il contenuto dei pixel delle immagini si altera l'informazione associata al blur contenuto in esse.

Alterazioni sull'architettura Sono stati effettuati diversi tentativi per ottimizzare la struttura architettonale originale in modo da snellire le tempistiche e il carico computazionale per l'addestramento ma mantenendo comunque delle prestazioni compa-

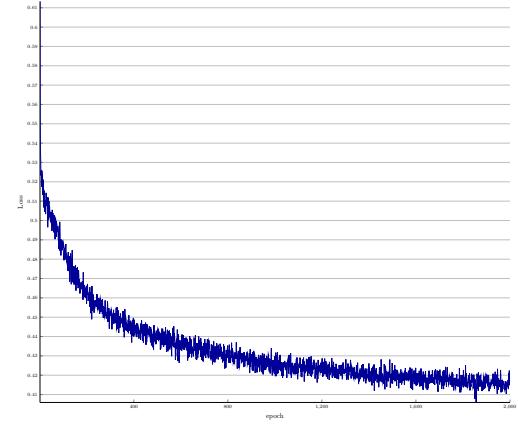
⁶<https://docs.pytorch.org/docs/stable/amp.html>

⁷RTX 3060, VRAM: 8GB

⁸Infatti, il calcolo del momento è incluso nell'Adam Optimizer



(a) Metriche di validazione: SSIM (a sinistra) e PSNR (a destra) su epochs.



(b) Training loss nelle epoch

ribili, per quanto inferiori, all'originale e dei risultati visivamente apprezzabili.

Abbiamo ad esempio provato a sostituire, all'interno del ramo di decoding, l'operazione di deconvoluzione (o convoluzione trasposta) con quella di pixel shuffle (introdotta in [4]), sulla carta più efficiente per operazioni di upsampling; in realtà, questo tentativo non ha portato miglioramenti significativi, per cui non abbiamo mantenuto questa modifica.

In definitiva, abbiamo deciso di intervenire sul modulo *CNNBlock*, impostando numero di blocchi residuali pari a $n = 6$ in ciascun modulo, generando una sorta di variante intermedia tra la *S* e la *B*. Abbiamo inoltre rimosso completamente il modulo *MSM* al suo interno: sebbene si trattasse di uno dei componenti caratterizzanti dell'architettura di partenza, dati i meccanismi di attenzione implementati, abbiamo osservato e valutato, anche grazie agli studi in [1], che in sostanza andava ad appesantire considerevolmente il modello introducendo dei miglioramenti marginali e difficilmente percepibili, soprattutto considerati gli obiettivi fissati per questo progetto.

Un'ultima variazione ha previsto l'introduzione della funzione di attivazione GELU (Gaussian Error Linear Unit) al posto della ReLU per tutti i layer convolutivi, che conferisce una convergenza migliore e riduce la possibilità di scomparsa dei gradienti.

Possibili casi critici Nel corso degli esperimenti sono state trovate due criticità: la rete, infatti, tende a lavorare peggio con immagini ad alta risoluzione (e.g. 4080×3060), lasciandole molto più invariate rispetto alla stessa immagine a risoluzione più bassa come ad esempio 1280×720 o 1920×1080 . Il motivo preciso per cui questo accada non è stato individuato, ma sospettiamo che abbia a che vedere con la mancanza dei meccanismi di attenzione implementati nel paper originale: i kernel convolutivi usati infatti potrebbero coprire aree troppo piccole dell'immagine per poter rilevare certe zone sfocate che in immagini ad alta risoluzione sono composte da molti pixel.

5 Risultati e conclusioni finali

Nel presente lavoro, i risultati del modello addestrato sono stati confrontati con quelli della versione originale di ConvIR [1]. Sebbene le prestazioni siano complessivamente inferiori rispetto ai modelli basati su Transformer, il nostro approccio risulta competitivo in alcuni casi, con metriche comparabili (soprattutto SSIM) e una significativa riduzione della complessità architettonale.

La riduzione dei parametri è cruciale: ottenuta intervenendo sul numero di blocchi convoluzionali e rimuovendo moduli onerosi come il Multi-Scale Module. Ulteriori strategie di compressione, come l'uso sistematico di convoluzioni 1×1 in stile NiN (introdotte in GoogLeNet [5]), potrebbero ridurre ulteriormente la dimensione del modello, fungendo da bottleneck tra layer convolutivi.

Per l'upsampling nel decoder, l'uso di *PixelShuffle* [6] rappresenta un'alternativa priva di parametri apprendibili rispetto alle deconvoluzioni, utile per progettare modelli leggeri.

Nel complesso, i risultati mostrano un buon compromesso tra efficienza e qualità dell'output: nonostante una lieve perdita in accuratezza, l'approccio proposto offre vantaggi in termini di tempo di addestramento, risorse e velocità di inferenza.

Sono riportati esempi qualitativi con: (1) immagine originale sfocata, (2) output deblurred, (3) residuo applicato (inteso come il modulo della differenza algebrica tra tensori in input e di output), amplificato per evidenziare le correzioni. Inoltre, vengono presentati:

- grafici di SSIM e PSNR sui dati di validazione;
- grafico della funzione di costo nel tempo;
- tabelle riepilogative delle metriche finali.

I risultati confermano la validità dell'approccio, mostrando che anche modelli leggeri e meno complessi possono garantire prestazioni visivamente e numericamente soddisfacenti rispetto a quelli basati su self-attention.

| Model | RSBlur PSNR | RSBlur SSIM | GOPRO PSNR | GOPRO SSIM | #Parameters |
|----------|-------------|-------------|------------|------------|-------------|
| ConvIR-L | 34.06 dB | 0.868 | 33.28 dB | 0.963 | 14.83M |
| Ours | 30.95 dB | 0.83 | 26.72 dB | 0.85 | 5.35M |

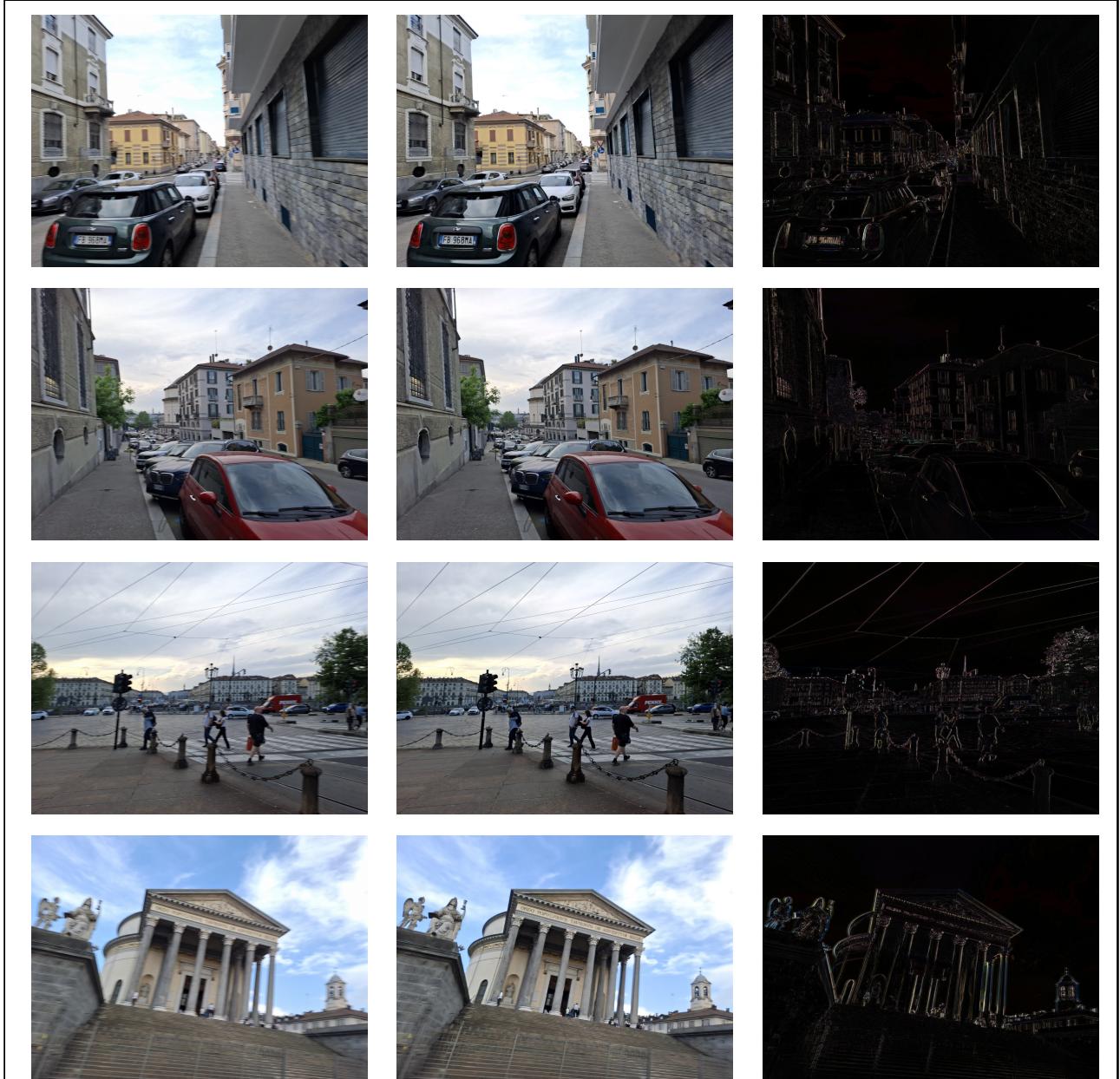


Figura 3: Una serie di immagini prodotte dal modello: nella prima colonna è presente l'immagine originale (sfocata), nella seconda l'output della rete, e nella terza il residuo che viene sommato all'immagine originale per ottenere la versione nitida, il cui valore dei pixel è stato raddoppiato in modo da rendere più visibili le zone interessate dalla correzione dell'immagine

Riferimenti bibliografici

- [1] Y. Cui, W. Ren, X. Cao e A. Knoll, “Revitalizing Convolutional Network for Image Restoration”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, n. 12, pp. 9423–9438, 2024. DOI: 10.1109/TPAMI.2024.3419007.
- [2] D. P. Kingma e J. Ba, *Adam: A Method for Stochastic Optimization*, 2017. arXiv: 1412 . 6980 [cs.LG]. indirizzo: <https://arxiv.org/abs/1412.6980>.
- [3] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan e Q. V. Le, *AutoAugment: Learning Augmentation Policies from Data*, 2019. arXiv: 1805 . 09501 [cs.CV]. indirizzo: <https://arxiv.org/abs/1805.09501>.
- [4] W. Shi et al., *Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network*, 2016. arXiv: 1609 . 05158 [cs.CV]. indirizzo: <https://arxiv.org/abs/1609.05158>.
- [5] C. Szegedy et al., *Going Deeper with Convolutions*, 2014. arXiv: 1409 . 4842 [cs.CV]. indirizzo: <https://arxiv.org/abs/1409.4842>.
- [6] O. Zamzam, *PixelShuffler: A Simple Image Translation Through Pixel Rearrangement*, 2025. arXiv: 2410 . 03021 [cs.CV]. indirizzo: <https://arxiv.org/abs/2410.03021>.