Rete Convoluzionale per Image Deblurring

Gruppo

June 14, 2025

Abstract

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 Introduction

[**convir**]

2 Theory and Traditional Approach

Una immagine con Blur è modellata matematicamente come convoluzione tra ground-truth image latente e blur kernel, dove si quest'ultimo 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:
$\boldsymbol{b} = \boldsymbol{i} * \boldsymbol{k} + \boldsymbol{n}$
Dove:
$\boldsymbol{b}$: Immagine con blur
$\boldsymbol{i}$: Immagine *ground-truth* latente
$\boldsymbol{k}$: Blur Kernel
$\boldsymbol{n}$: Rumore presente nell'immagine per contare imperfezioni causate dall'acquisizione (quantizzazione, saturazione del colore, risposta non linare 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 $\boldsymbol{g}$ tale che, tramite convoluzione con l'immagine blurred $\boldsymbol{b}$. Espresso nel dominio di Fourier:

$$\hat{\boldsymbol{I}} = \boldsymbol{GB} \tag{1}$$

$$\boldsymbol{G} = \frac{|\boldsymbol{K}|^2}{|\boldsymbol{K}|^2 + \frac{1}{\text{SNR}}} \frac{1}{\boldsymbol{K}} \tag{2}$$

Dove:
$\boldsymbol{G}$ e $\boldsymbol{K}$: trasformate di Fourier di $\boldsymbol{g}$ e $\boldsymbol{k}$
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 ignoto[2], 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 corrisponde in buona parte a quella illustrata in [**convir**]. La rete si basa su di una struttura a U (encoder-decoder convoluzionali), con estrazione delle feature effettuata a risoluzioni multiple (downsampled) e con l'aggiunta dei cosiddetti *Multi-Scale Modules* (*MSM*), che hanno l'obiettivo di implementare meccanismi di attenzione di diversa forma (quadrata e rettangolare).

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) generata dal dataloader, ma viene ulteriormente sottocampionata (rispettivamente alla metà e a un quarto della risoluzione) e reinserita nella rete come input 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 nu-

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

[2] I metodi con neural network rientrano in questa categoria

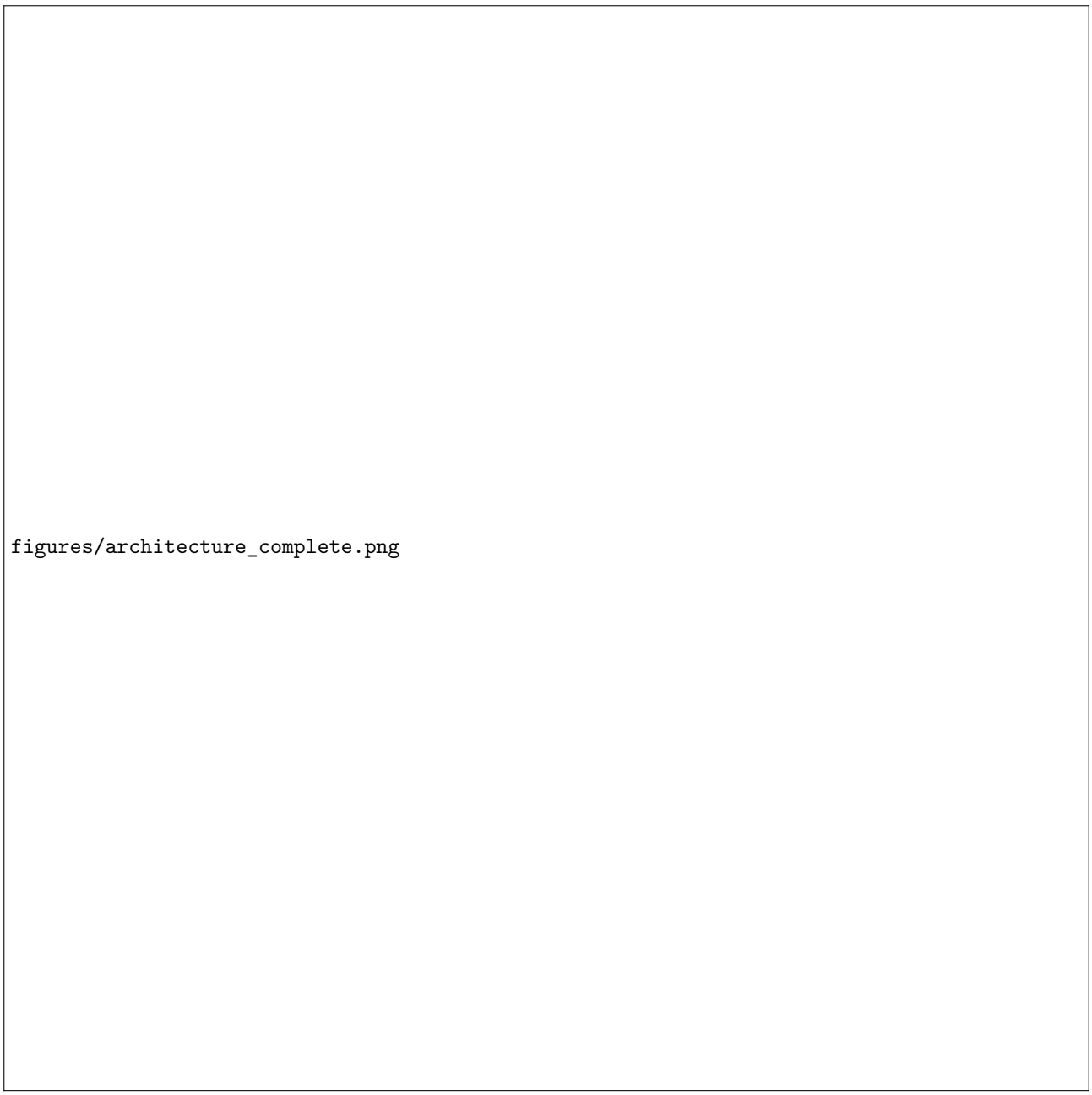


Figure 1: Schema generale dell'architettura completa (ConvIR)

mero 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*;
- **(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 [**convir**] 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'in-

terno di ciascun modulo *CNNBlock*. Le configurazioni prevedevano $n = 3$, $n = 7$ e $n = 15$ rispettivamente per le varianti S, B e L. La nostra scelta è ricaduta in una variante intermedia tra la S e la B, con $\mathbf{n = 6}$ blocchi residuali. Abbiamo inoltre modificato il *CNNBlock* rimuovendo completamente il modulo *MSM* al suo interno. In questo modo, il training è risultato molto più rapido ed efficiente con solo una minima perdita di qualità del risultato finale.

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:

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

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

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:

$$\mathcal{L}_{tot} = \mathcal{L}_{spatial} + \lambda \mathcal{L}_{freq}, \tag{5}$$

dove λ è un iperparametro impostato di default a 0.01.

4 Observations

Si illustrano di seguito le alterazioni apportate sull'architettura e processo di training, dettagliandone ragione e risultato.

Dataset I dataset usati sono principalmente due: GOPRO e RSBlur. GOPRO è un dataset creato applicando un motion blur artificiale ai frame di una serie di video girate 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 che tenesse conto della performance su entrambi i dataset, nello specifico:

$$\text{Score} = \frac{1}{2} \sum_{i=1}^{2} 100 \cdot avg(\frac{PSNR_i}{33}, SSIM_i)$$

Dove i indicizza 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, composto da 2500 immagini per il training set (di cui il 15% è dedicato alla validazione) e 1000 per il test set. Sono stati prodotti diversi sottoinsiemi di RSBlur (con un numero di immagini simile se non uguale a quelli citati in precedenza), compreso un otunetto

Funzione di Training Alla funzione di costo descritta in precedenza (**??**) abbiamo aggiunto un termine di gradient penalty, che appunto disincentivasse una crescita eccessiva dei gradienti, consentendo quindi una regolarizzazione più efficace del modello. La penalità è stata calcolata accumulando

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*[3]. 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

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

Batch Size

- `D` Numero di Coppie di immagini, Example e Label, caricate in un ciclo di Stochastic Gradient Descent
- `V` **Valore**: 8. Valori maggiori, eg. 12 tendono a rallentare un ciclo di forward e backward propagation, mentre valori superiori o uguali a 32 Esauriscono la memoria del GPU device utilizzato per i tests[4]

Optimizer
L'ottimizzatore utilizzato è l'algoritmo di Adam con i seguenti parametri

[3] https://docs.pytorch.org/docs/stable/amp.html
[4] RTX 3060, VRAM: 8GB

$\beta_1 = 0.9$ 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.[5]

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 + weight_decay

V 0

Accumulate Gradient Frequency

D Frequenza, in termini di numero di batches processati, di applicazione

V

λ

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

V 0.1

Image Distance Metric

D Funzione utilizzata per comparare una coppia di immagini nel calcolo della Loss.

V Norma L1 della differenza (invariata rispetto all'architettura di riferimeento)

Data Augmentation Seguendo il codice originale, abbiamo mantenuto pressoché invariate le trasformazioni usate dagli autori per la data augmentation, ovvero:

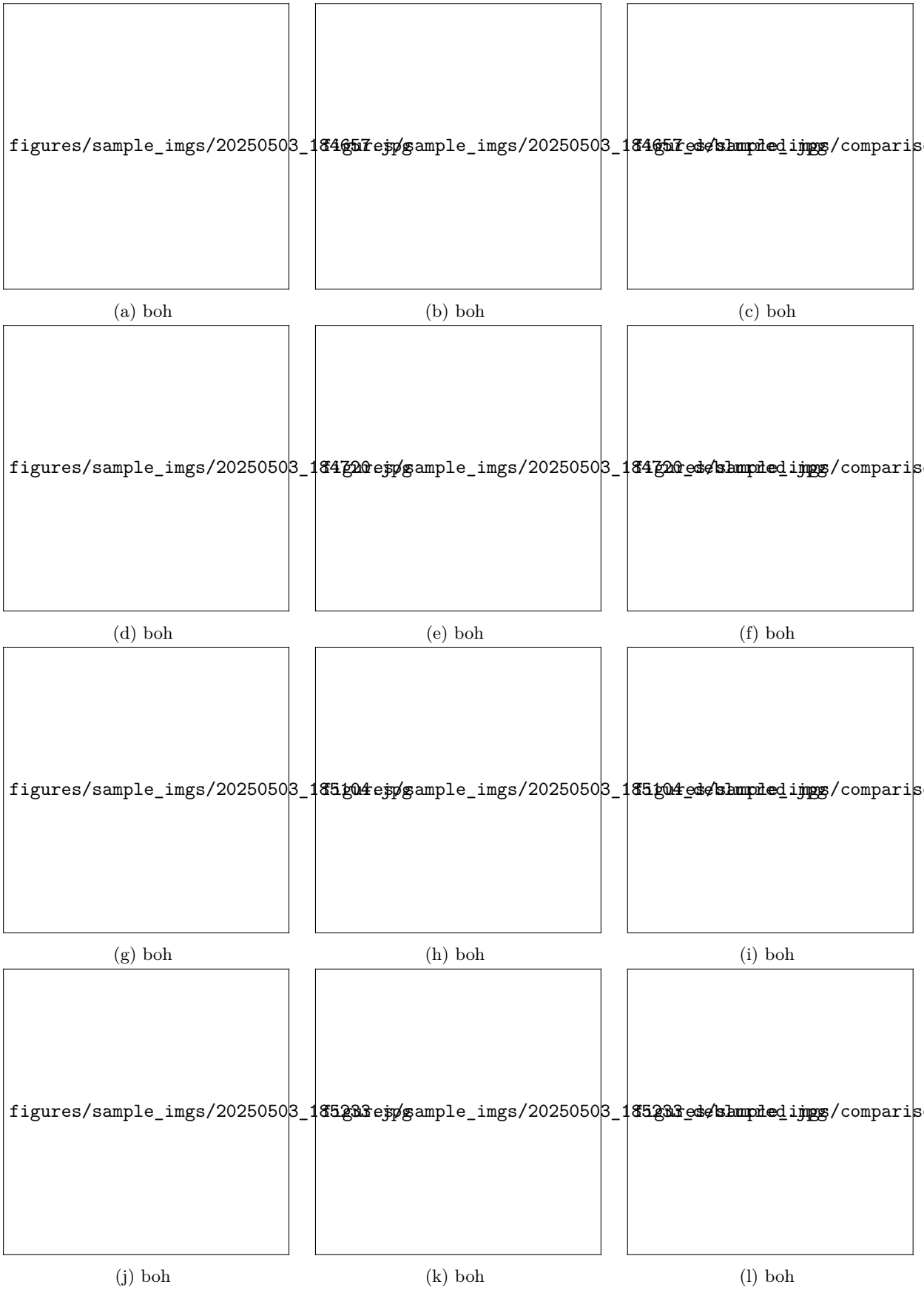
- `RandomCrop(256)`: Estrae un crop quadrato di lato 256 pixel, rendendo il training estremamente più veloce
- `RandomHorizontalFlip(p=0.5)`: Decide casualmente (con una probabilità del 50%) se ruotare intorno all'asse verticale il crop ottenuto al passo precedente
- `ToDtype(torch.get_default_dtype())`: Cast degli elementi del tensore a `torch.float32`
- `NormalizeRange()`: Normalizza i valori del tensore nell'intervallo [0, 1]

Queste trasformazioni sono volte ad aumentare artificialmente la dimensione del dataset, senza però andare ad alterare l'informazione associata al blur: inizialmente abbiamo provato altre trasformazioni come ad esempio `ColorJitter`, ma è risultato evidente che

Alterazioni sull'Architettura

[5] Infatti, il calcolo del momento è incluso nell'Adam Optimizer

5 Results



(a) boh (b) boh (c) boh

(d) boh (e) boh (f) boh

(g) boh (h) boh (i) boh

(j) boh (k) boh (l) boh

Figure 2: roba