



**UNIVERSITÀ DEGLI STUDI DI CATANIA**  
DIPARTIMENTO DI MATEMATICA E INFORMATICA  
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

---

*Alessio Mezzina, Giuseppe Condorelli*

SuperResolution Autoencoder

---

PROGETTO DI FINE CORSO

---

Prof. Sebastiano Battiato  
Prof. Francesco Guarnera

---

Anno Accademico 2023 - 2024

# Contents

<b>1 Problema</b>	<b>1</b>
1.1 Reperibilità del materiale . . . . .	2
<b>2 Dataset</b>	<b>3</b>
2.1 Dataset . . . . .	3
2.2 Padding delle Immagini . . . . .	4
2.3 Dati di training, dati di validation . . . . .	4
2.4 Organizzazione cartelle del dataset . . . . .	6
<b>3 Metodi</b>	<b>7</b>
3.1 DataLoader . . . . .	7
3.2 Seed . . . . .	7
3.3 Autoencoder . . . . .	7
3.4 U-Net . . . . .	8
3.5 Meccanismi di attenzione e Laplacian sharpening kernel . . . . .	9
3.6 Training . . . . .	9
<b>4 Valutazione</b>	<b>10</b>
4.1 Metriche e grafici . . . . .	10
<b>5 Esperimenti</b>	<b>12</b>
5.1 Immagini usate per la validation . . . . .	13
5.2 Prima parte . . . . .	14
5.2.1 Primo esperimento . . . . .	14
5.2.2 Secondo esperimento . . . . .	16
5.2.3 Terzo esperimento . . . . .	18
5.2.4 Quarto esperimento . . . . .	20
5.2.5 Quinto esperimento . . . . .	23

5.2.6	Sesto esperimento . . . . .	27
5.2.7	Settimo esperimento . . . . .	30
5.2.8	Ottavo esperimento . . . . .	33
5.2.9	Nono esperimento . . . . .	36
5.2.10	Decimo esperimento . . . . .	39
5.2.11	Undicesimo esperimento . . . . .	42
5.2.12	Dodicesimo esperimento . . . . .	45
5.3	Seconda Parte . . . . .	47
5.3.1	Fixing white layer . . . . .	47
5.3.2	U-net . . . . .	48
5.3.3	Attention U-net . . . . .	51
5.3.4	Attention U-net & sharpening kernel . . . . .	55
5.4	Conclusione . . . . .	59
<b>6</b>	<b>Codice</b>	<b>60</b>
6.1	Notebook Super Resolution.ipynb . . . . .	60
6.1.1	Preprocessing immagini . . . . .	60
6.1.2	Utility functions . . . . .	60
6.1.3	Esperimenti . . . . .	61
<b>7</b>	<b>Conclusioni</b>	<b>62</b>
7.1	Considerazioni riguardanti gli esperimenti . . . . .	62
7.2	Lezioni apprese . . . . .	62
7.3	Sviluppi futuri . . . . .	63
	<b>Bibliography</b>	<b>64</b>

# Chapter 1

## Problema

Il presente progetto si propone di affrontare la sfida dell'incremento della risoluzione delle immagini, nota come super resolution, mediante un'architettura innovativa che combina autoencoder e reti neurali convoluzionali (CNN). Questa combinazione è particolarmente efficace nell'apprendere e migliorare le caratteristiche visive complesse, rendendola ideale per aumentare i dettagli visivi delle immagini ingrandite e migliorarne la qualità generale, approcci simili sono stati testati e hanno portato a risultati molto validi [1, 2].

L'obiettivo primario è sviluppare un sistema automatico e preciso in grado di migliorare la risoluzione delle immagini, aumentando i dettagli visivi senza introdurre artefatti.

La scelta di utilizzare reti neurali convoluzionali per la super resolution è motivata dalla loro capacità di catturare e amplificare le caratteristiche visive sottili, essenziali per ripristinare dettagli ad alta risoluzione. Questa tecnica è stata impiegata con successo in molteplici applicazioni, mostrando ottimi risultati nel miglioramento della qualità delle immagini, ad esempio gli autori in [3, 4] trattano il task di image denoising con architetture simili a quelle che si intendono sviluppare in questo progetto. Dopo la preparazione del dataset e la definizione dell'architettura, l'attenzione si è spostata verso lo sviluppo di diversi esperimenti per testare l'efficacia del modello, discussi approfonditamente nei capitoli 3 e 5. Inoltre, una descrizione dettagliata degli strumenti software utilizzati durante il progetto è disponibile nel capitolo 6.

## 1.1 Reperibilità del materiale

Tutto il materiale sviluppato, incluso il dataset, è disponibile nella seguente repo [5] su GitHub.

# Chapter 2

## Dataset

In questo capitolo intende descrivere in dettaglio il dataset utilizzato per addestrare e valutare la bontà dell'autoencoder.

### 2.1 Dataset

Il dataset utilizzato per il nostro progetto di computer vision è costituito dal dataset Flickr2K, utilizzato nella challenge [6], disponibile su GitHub al seguente indirizzo: <https://github.com/LimBee/NTIRE2017>. La scelta di utilizzare questo dataset è stata motivata da diverse ragioni:

- **Qualità delle Immagini:** Il dataset Flickr2K contiene immagini di alta qualità, essenziali per il training di modelli di super resolution. Immagini di buona qualità assicurano che il modello possa apprendere i dettagli fini necessari per migliorare la risoluzione delle immagini degradate.
- **Varietà di Contenuti:** Il dataset include una vasta gamma di immagini che coprono molteplici categorie. Questa varietà consente di addestrare il modello su un dataset eterogeneo, migliorando la sua capacità di generalizzare su immagini diverse.
- **Rilevanza per il Compito:** Essendo stato progettato specificamente per competizioni di super resolution, il dataset Flickr2K è particolarmente adatto per addestrare e valutare modelli in questo ambito.
- **Accessibilità:** Il dataset è facilmente accessibile tramite GitHub, permettendo di costruire dataset su misura in base alle esigenze specifiche del progetto.

## 2.2 Padding delle Immagini

Per garantire che tutte le immagini avessero una dimensione minima uniforme e per preservare la proporzione originale durante il resize, abbiamo implementato una classe chiamata `PadIfNeeded`. Questa classe aggiunge padding simmetrico alle immagini che non soddisfano le dimensioni minime richieste, evitando così distorsioni durante il ridimensionamento successivo. Il padding è cruciale per mantenere l'integrità visiva delle immagini non quadrate, poiché il ridimensionamento può altrimenti schiacciare le immagini, perdendo la struttura originale dell'immagine.

### ***Processamento e Salvataggio delle Immagini:***

Abbiamo creato una funzione `process_and_save_images` che processa le immagini presenti in una cartella di input applicando la trasformazione definita dalla classe `PadIfNeeded` e salva le immagini trasformate in una cartella di output. La funzione itera su tutte le immagini nella cartella di input, applica la trasformazione e salva le immagini risultanti.

### ***Resize delle Immagini:***

Abbiamo inoltre ridimensionato tutte le immagini del dataset a una dimensione fissa di 256x256 e 64x64 pixel, l'immagine 256x256 si utilizza come *Ground Truth*, mentre quella a 64x64 rappresenta l'immagine a bassa risoluzione. Questa scelta è stata fatta per andare incontro alle nostre disponibilità hardware, in quanto processare immagini a risoluzioni 2K sarebbe stato troppo oneroso in termini di risorse. Il ridimensionamento delle immagini a una dimensione uniforme di 256x256 pixel è stato eseguito prima del training per garantire la consistenza e l'uniformità del dataset, ciò è stato fatto per ottimizzare l'efficacia del processo di training del modello.

Queste operazioni hanno permesso di standardizzare il dataset, facilitando il training del modello e migliorando la qualità delle immagini utilizzate.

## 2.3 Dati di training, dati di validation

Nel processo di preparazione del dataset per il training e la validazione del modello di super resolution, non è stata adottata la pratica di separare fisicamente le immagini in cartelle distinte per training e validation. Al contrario, è stata impiegata una funzione specifica della libreria PyTorch denominata `random_split` [7]. Questa funzione consente una divisione casuale del dataset, facilitando un approccio più flessibile e

automatizzato nella gestione dei dati. La funzione `random_split` è stata configurata per dividere il dataset in due subset distinti: il 80% delle immagini è stato assegnato al set di training, mentre il restante 20% è stato destinato al set di validation. Questa proporzione è stata scelta per garantire che il modello avesse a disposizione una quantità sufficientemente ampia di dati per l'apprendimento, pur mantenendo una quota adeguata di dati per testare l'accuratezza e la generalizzazione del modello su immagini non utilizzate durante la fase di training. Utilizzare la funzione `random_split` offre diversi vantaggi:

1. Semplicità e Automazione: La divisione del dataset viene gestita internamente dalla funzione, eliminando la necessità di manipolazioni manuali dei file di dati e riducendo il rischio di errori umani.
2. Riproducibilità: Specificando un seed per il generatore di numeri casuali, la divisione del dataset può essere riprodotta esattamente, un aspetto cruciale per la convalida sperimentale e la comparazione dei risultati di ricerca.
3. Flessibilità: È facile modificare la proporzione di split o riapplicare la divisione per sperimentare con diverse configurazioni, senza la necessità di ristrutturare fisicamente il dataset.

Questo metodo di divisione assicura che ogni esecuzione del processo di training e validation possa beneficiare di una distribuzione equilibrata e casualmente variata delle immagini, contribuendo a un'apprendimento robusto e a una valutazione accurata della capacità del modello di super resolution.

## 2.4 Organizzazione cartelle del dataset

La struttura del dataset è organizzata come segue:

- **Flickr2K:** Questa cartella contiene le immagini originali ad alta risoluzione, prive di alterazioni.
- **Flickr2K\_Padded:** Include immagini originali dalla cartella Flickr2K che sono state modificate aggiungendo un padding simmetrico. Questo processo è necessario per garantire che tutte le immagini abbiano dimensioni uniformi prima di essere sottoposte a ulteriori trasformazioni o ridimensionamenti. Il padding preserva le proporzioni originali delle immagini, evitando distorsioni durante il processo di ridimensionamento.
- **Flickr256\_Padded:** Contiene immagini dalla cartella Flickr2K\_Padded che sono state ridimensionate a una dimensione uniforme di 256x256 pixel. Servono come riferimento ideale, o "ground truth", per la valutazione delle prestazioni del modello durante le fasi di training e validazione.
- **Flickr64\_Padded:** Comprende immagini dalla cartella Flickr2K\_Padded che sono state ridimensionate a 64x64 pixel. Queste immagini, rappresentano le immagini dalle quali partire per applicare il processo di SuperResolution da parte dei modelli sviluppati

# Chapter 3

## Metodi

### 3.1 DataLoader

Per caricare il set di immagini su python al fine di essere utilizzati dall’architettura si è utilizzato la funzione `DataLoader` [8] fornita dalla libreria torch. La libreria permette di dividere il dataset in *batch* che vengono poi utilizzare dagli strati implementati nell’architettura. In particolare, si è impostato il parametro *batch\_size* a 64 fino all’esperimento 7, successivamente è stato portato a 32. Rendendo possibile una fluida esecuzione anche con scarsità di memoria nell’hardware a disposizione.

### 3.2 Seed

Al fine di rendere gli esperimenti ripetibili si è utilizzato un *seed* nelle funzioni randomiche utilizzate, in particolar modo nella fase di splitting del dataset in training e validation [9, 10], in maniera tale da garantire la riproducibilità.

### 3.3 Autoencoder

Per affrontare il problema proposto, si è utilizzato l’architettura *Convolutional Autoencoder*, che, data la sua natura, si è ritenuta più opportuna. Al fine di migliorare le prestazioni dell’architettura, si è adottato una strategia *bottom-up*, partendo da un’architettura molto semplice contenente pochi layer per poi progressivamente aumentarne la complessità. Questa approccio ha permesso di valutare varie strategie e tecniche sulla base dei risultati per scegliere quella più vantaggiosa, come proposto in diversi recenti articoli [11, 12]. Ogni architettura proposta presenta diversi strati

convoluzionali seguiti spesso da procedure di *down-sample* e *up-sample*, quali *AvgPool2d*, *MaxPool2d* e *ConvTranspose2d*, che hanno permesso di realizzare la struttura di encoding e decoding tipica degli autoencoder. Altri layers importanti sono anche rappresentati dall'utilizzo di funzioni di attivazioni *SiLu* o *LeakyReLu*, nonché layer di *BatchNormalization*, utili per permettere all'archittetura di cogliere le caratteristiche dei dati non entrando in overfitting. Per l'apprendimento del modello, si è scelto di utilizzare l'algoritmo di ottimizzazione *AdamW*, che si basa sulla discesa del gradiente, insieme alla funzione *MSE loss*, particolarmente adatta per training supervisionato.

## 3.4 U-Net

E' stata impiegata anche l'architettura U-Net nelle fasi finali del progetto, in particolare per affrontare il problema della segmentazione delle immagini. La U-Net è nota per la sua capacità di fornire risultati accurati ed efficienti in compiti di segmentazione grazie alla sua struttura simmetrica che consiste in un percorso di contrazione (encoder) e un percorso di espansione (decoder).

Il percorso di contrazione della U-Net è simile a quello di una classica rete convoluzionale, composto da ripetute applicazioni di convoluzioni  $3 \times 3$  non lineari, seguite da un'operazione di pooling  $2 \times 2$ . Ogni step di downsampling raddoppia il numero di feature channels, consentendo alla rete di apprendere rappresentazioni sempre più astratte.

Nel percorso di espansione, ogni step di upsampling è composto da un'operazione di upsampling del tipo  $2 \times 2$  seguito da una convoluzione  $3 \times 3$ . Inoltre, i layer corrispondenti nel percorso di contrazione e di espansione sono collegati tramite concatenazione, permettendo alla rete di mantenere alta risoluzione spaziale durante l'intero processo di segmentazione.

Un aspetto cruciale dell'architettura U-Net è l'uso di skip connections che trasferiscono informazioni dettagliate dalle parti iniziali della rete (encoder) alle parti finali (decoder). Queste connessioni aiutano a preservare i dettagli di basso livello e migliorano significativamente la precisione delle ricostruzioni.

### 3.5 Meccanismi di attenzione e Laplacian sharpening kernel

L'integrazione di meccanismi di attenzione consente alla rete di focalizzarsi su caratteristiche significative delle immagini, migliorando la discriminazione tra le classi e affinando la qualità delle segmentazioni. Gli strati di attenzione Channel operano selezionando le feature più rilevanti lungo il canale delle caratteristiche, mentre gli strati di attenzione Spatial enfatizzano aree significative dell'immagine, contribuendo così a una rappresentazione più informativa e localizzata.

In aggiunta, abbiamo implementato un Laplacian sharpening kernel nel processo di ricostruzione delle immagini. Questo kernel agisce sui canali RGB dell'immagine, permettendo di migliorare i dettagli e la nitidezza dell'immagine finale.

### 3.6 Training

Per la fase di training si è optato nell'utilizzo di un *EarlyStopping*, un oggetto fornito da pytorch lightning [13] che si è rivelato particolarmente utile. Quest'ultimo infatti monitora il miglioramento di un valore durante la fase di training e controlla se quest'ultimo migliora nel corso delle epoche. Si è usato questo oggetto per controllare che la *"val loss"* migliorasse di almeno 0.005 entro 10 epoche, se così non fosse, la fase di training viene interrotta dall'oggetto segnalando al *trainer* di fermarla. Si è deciso di optare per questa soluzione per fermare in automatico il training nel momento in cui l'architettura smettesse di convergere così da ottimizzare l'uso delle risorse e del tempo. **E' importante inoltre specificare che il max\_epoch del trainer utilizzato è 50.** Nell'ultimo esperimento, abbiamo esteso la durata dell'allenamento impostando il parametro di EarlyStopping a 0.00005. Questa modifica ha portato a osservabili miglioramenti nelle prestazioni del modello.

# Chapter 4

## Valutazione

### 4.1 Metriche e grafici

Le metriche e grafici utilizzati sono:

- **MSE Loss (Mean Squared Error Loss):** Calcola la media dei quadrati delle differenze tra i valori predetti e quelli reali. Questa metrica è fondamentale per misurare l'accuratezza del modello in termini di errore quantitativo. Un valore più basso di MSE indica una minor discrepanza tra l'immagine ricostruita e quella originale, suggerendo una maggiore precisione del modello.
- **Plot of Training and Validation Loss:** Questo grafico permette di capire se il modello sta riuscendo a convergere col passare delle epoche, dimostrando dunque la sua capacità nell'apprendere e validità.
- **PSNR (Peak Signal-to-Noise Ratio):** Questa metrica, una volta resa disponibile l'immagine originale, consente di valutare la qualità della ricostruzione dell'immagine ottenuta dal modello rispetto a quella di input. In particolare, il PSNR è misurato in decibel (dB) e fornisce un'indicazione della fedeltà della super-risoluzione. Un valore di PSNR più alto indica una maggiore somiglianza con l'immagine originale, suggerendo una migliore performance del modello nel preservare i dettagli e ridurre gli artefatti nella versione ricostruita.
- **SSIM (Structural Similarity Index):** Una metrica molto utile per valutare il livello di somiglianza tra due immagini e dunque valutare la bontà del risultato proposto dal modello rispetto all'immagine originale.

Durante gli esperimenti, osserveremo attentamente tutti i grafici e le metriche citate al fine di valutarne correttamente le capacità e dunque adottare strategie migliori per il miglioramento del modello.

# Chapter 5

## Esperimenti

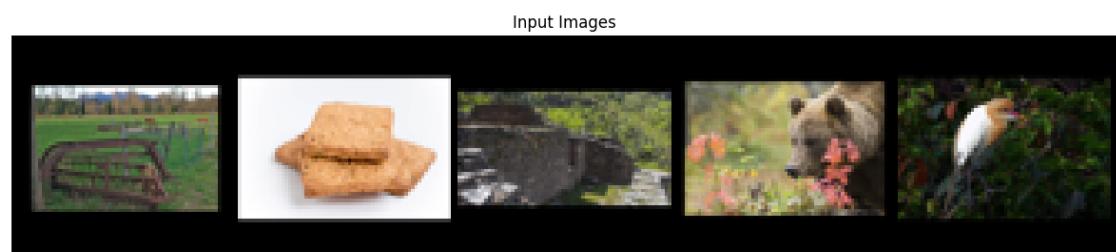
Table 5.1: Risultati degli Esperimenti, in grassetto i risultati migliori

N° Esperimento	PSNR	SSIM	Note
1	15.20 dB	0.2254	
2	15.32 dB	0.2439	
3	13.23 dB	0.2318	Aumentati i layers di CNN e SiLU
4	16.93 dB	0.2815	Sostituiti Avg con Max, SiLU con LeakyReLU, aggiunti BathNorm e tahn
5	17.79 dB	0.3057	Aggiunti layers di Conv, LeakyReLU e BatchNorm
6	18.61 dB	0.3107	Aggiunti ulteriori layers
7	19.06 dB	0.3090	Aggiunti altri layers riducendo il resample
8	19.39 dB	0.3187	Cambiamento disposizione layers
9	18.05 dB	0.3211	Kernel size modificata nei primi ed ultimi layers
10	18.11 dB	0.3165	Diminuita complessità
11	18.17 dB	0.3330	Cambiato ordine Max pooling
12	19.40 dB	0.3609	Aumentato numero epoche, diminuito min_delta (Early stopping)
13	24.24 dB	0.6935	Risolto "layer bianco" ed usato U-Net
14	<b>24.48 dB</b>	0.7082	Aggiunti strati di attenzione
15	24.25 dB	<b>0.7100</b>	Aggiunto kernel di sharpening

## 5.1 Immagini usate per la validation



(a)



(b)

## 5.2 Prima parte

### 5.2.1 Primo esperimento

Il nostro primo approccio di risoluzione del problema è stato quello di costruire una prima architettura che fosse quanto più semplice possibile, al fine anche di valutare quale strada intraprendere,

Com'è possibile evincere dai valori di SSIM e PSNR (Tab.5.3) i risultati sono molto deludenti, ulteriormente confermato dalle immagini in output in Fig.5.1(d).

Layer (type)	Output Shape	Param #
Conv2d-1	[ -1, 8, 128, 128]	224
SiLU-2	[ -1, 8, 128, 128]	0
Conv2d-3	[ -1, 16, 128, 128]	1,168
SiLU-4	[ -1, 16, 128, 128]	0
AvgPool2d-5	[ -1, 16, 64, 64]	0
Conv2d-6	[ -1, 32, 64, 64]	4,640
SiLU-7	[ -1, 32, 64, 64]	0
ConvTranspose2d-8	[ -1, 16, 128, 128]	4,624
SiLU-9	[ -1, 16, 128, 128]	0
ConvTranspose2d-10	[ -1, 8, 256, 256]	1,160
SiLU-11	[ -1, 8, 256, 256]	0
ConvTranspose2d-12	[ -1, 3, 512, 512]	219
<b>Total params:</b>	12,035	
<b>Trainable params:</b>	12,035	
<b>Non-trainable params:</b>	0	

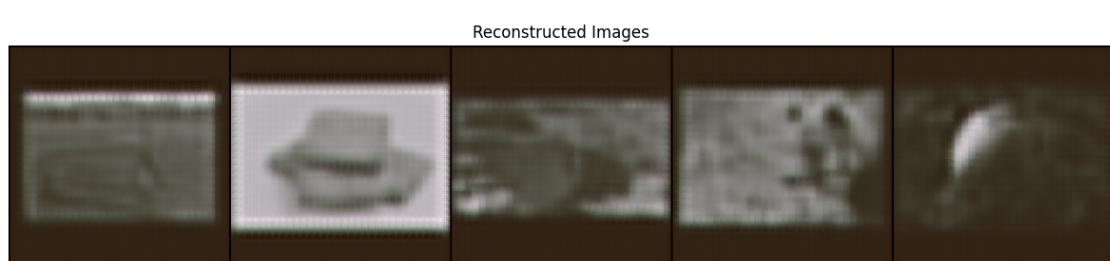
Table 5.2: Modello primo esperimento

Descrizione	Valore
Average PSNR between Sr and reconstructed images	15.20 dB
Average SSIM between Sr and reconstructed images	0.2254

Table 5.3: Valori medi di PSNR e SSIM



(c)



(d)

Figure 5.1: Risultati primo esperimento

Visti i risultati precedenti, i prossimi esperimenti vedranno una sempre più aumentata complessità dell'architettura al fine di ottenere qualche miglioramento.

### 5.2.2 Secondo esperimento

Nonostante i leggeri miglioramenti in metriche (Tab.5.5), le immagini in output risultano decisamente peggiorate a livello percettivo, non permettendo nemmeno di identificare i soggetti e semantica di essa (Fig.5.2(b)).

Layer (type)	Output Shape	Param #
Conv2d-1	[1, 8, 128, 128]	224
SiLU-2	[1, 8, 128, 128]	0
AvgPool2d-3	[1, 8, 64, 64]	0
Conv2d-4	[1, 16, 64, 64]	1,168
SiLU-5	[1, 16, 64, 64]	0
AvgPool2d-6	[1, 16, 32, 32]	0
Conv2d-7	[1, 32, 32, 32]	4,640
SiLU-8	[1, 32, 32, 32]	0
Conv2d-9	[1, 64, 32, 32]	18,496
SiLU-10	[1, 64, 32, 32]	0
Conv2d-11	[1, 128, 32, 32]	73,856
SiLU-12	[1, 128, 32, 32]	0
ConvTranspose2d-13	[1, 64, 64, 64]	73,792
SiLU-14	[1, 64, 64, 64]	0
ConvTranspose2d-15	[1, 32, 128, 128]	18,464
SiLU-16	[1, 32, 128, 128]	0
ConvTranspose2d-17	[1, 16, 256, 256]	4,624
SiLU-18	[1, 16, 256, 256]	0
ConvTranspose2d-19	[1, 8, 512, 512]	1,160
SiLU-20	[1, 8, 512, 512]	0
Conv2d-21	[1, 3, 512, 512]	219
<b>Total params:</b>	196,643	
<b>Trainable params:</b>	196,643	
<b>Non-trainable params:</b>	0	

Table 5.4: Modello secondo esperimento

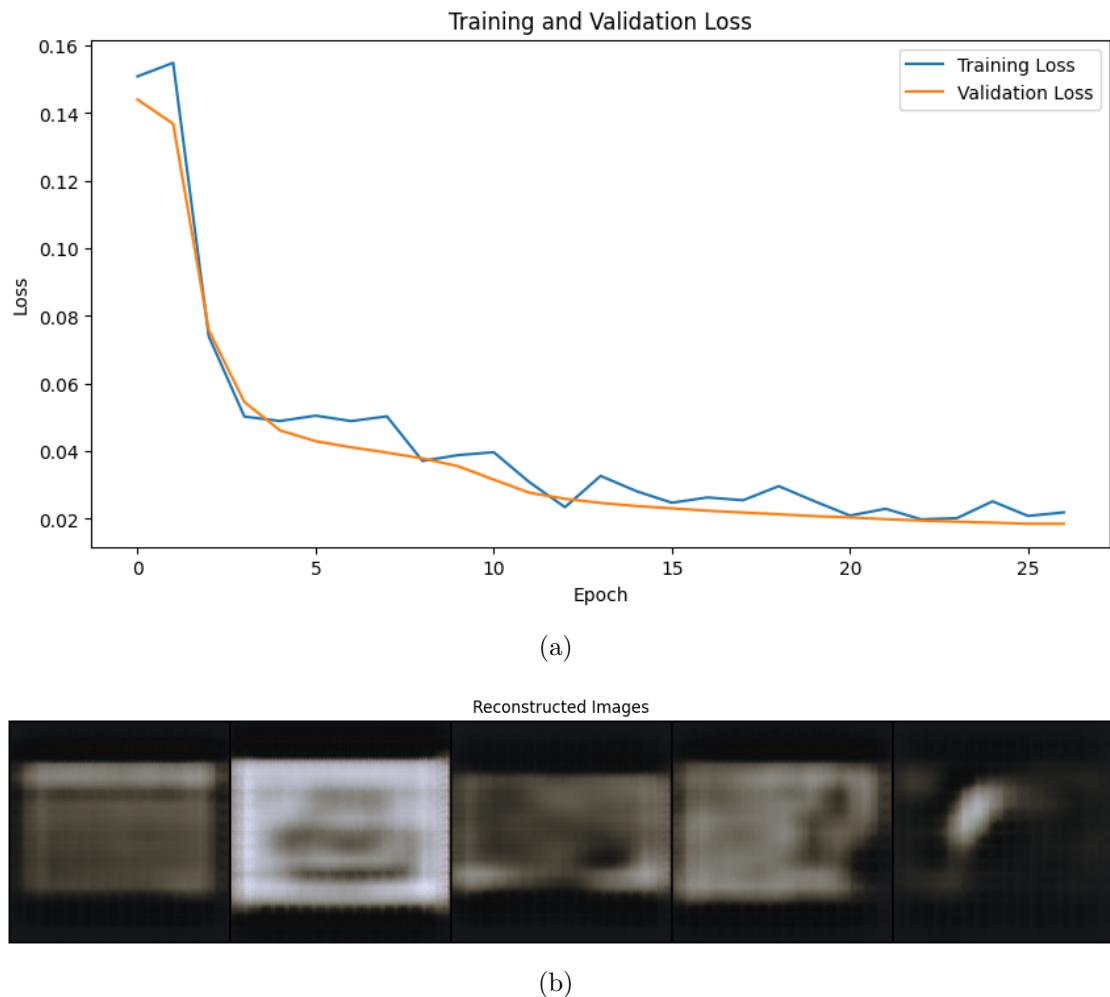


Figure 5.2: Risultati secondo esperimento

Descrizione	Valore
Average PSNR between Sr and reconstructed images	15.32 dB
Average SSIM between Sr and reconstructed images	0.2439

Table 5.5: Valori medi di PSNR e SSIM

### 5.2.3 Terzo esperimento

Nella terza architettura abbiamo ulteriormente aumentato i layers, portando tuttavia ad un ulteriore peggioramento, a livello visivo e percettivo, delle immagini in output, rendendole ancora più indecifrabili e impossibili da ricondurre a quelle originali. (Fig.5.3(b)).

Layer (type)	Output Shape	Param #
Conv2d-1	[ -1, 8, 128, 128]	224
SiLU-2	[ -1, 8, 128, 128]	0
AvgPool2d-3	[ -1, 8, 64, 64]	0
Conv2d-4	[ -1, 16, 64, 64]	1,168
SiLU-5	[ -1, 16, 64, 64]	0
AvgPool2d-6	[ -1, 16, 32, 32]	0
Conv2d-7	[ -1, 32, 32, 32]	4,640
SiLU-8	[ -1, 32, 32, 32]	0
AvgPool2d-9	[ -1, 32, 16, 16]	0
Conv2d-10	[ -1, 64, 16, 16]	18,496
SiLU-11	[ -1, 64, 16, 16]	0
Conv2d-12	[ -1, 128, 16, 16]	73,856
SiLU-13	[ -1, 128, 16, 16]	0
Conv2d-14	[ -1, 256, 16, 16]	295,168
SiLU-15	[ -1, 256, 16, 16]	0
ConvTranspose2d-16	[ -1, 128, 32, 32]	295,040
SiLU-17	[ -1, 128, 32, 32]	0
ConvTranspose2d-18	[ -1, 64, 64, 64]	73,792
SiLU-19	[ -1, 64, 64, 64]	0
ConvTranspose2d-20	[ -1, 32, 128, 128]	18,464
SiLU-21	[ -1, 32, 128, 128]	0
ConvTranspose2d-22	[ -1, 16, 256, 256]	4,624
SiLU-23	[ -1, 16, 256, 256]	0
ConvTranspose2d-24	[ -1, 8, 512, 512]	1,160
SiLU-25	[ -1, 8, 512, 512]	0
Conv2d-26	[ -1, 3, 512, 512]	219
<b>Total params:</b>		786,851
<b>Trainable params:</b>		786,851
<b>Non-trainable params:</b>		0

Table 5.6: Modello terzo esperimento

Descrizione	Valore
Average PSNR between Sr and reconstructed images	13.23 dB
Average SSIM between Sr and reconstructed images	0.2318

Table 5.7: Valori medi di PSNR e SSIM

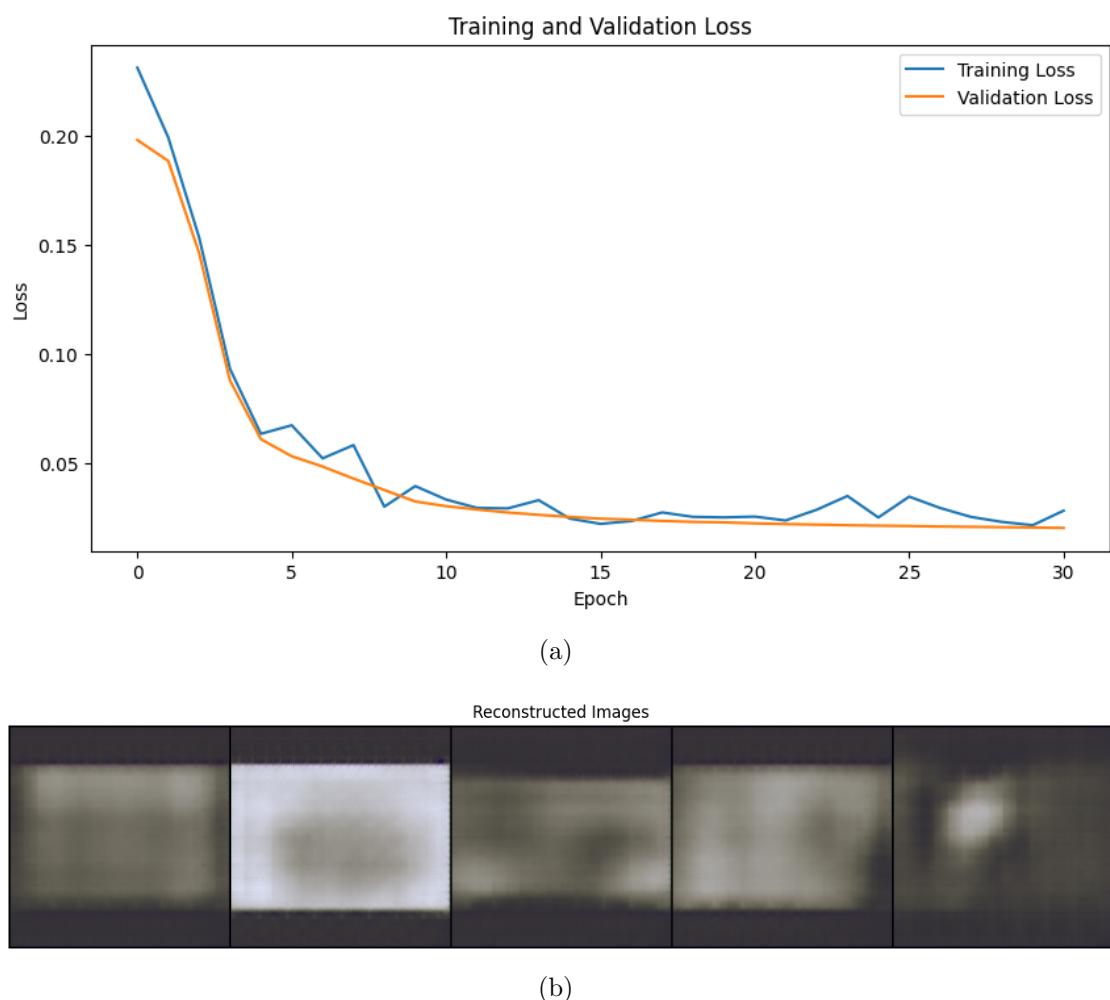


Figure 5.3: Risultati terzo esperimento

Visto che tutte le prove precedenti hanno portato solo a dei peggioramenti, abbiamo deciso di modificare la struttura dell'architettura introducendo layers di Batch-Norm, LeakyReLU ed una funzione tanh come strato finale.

### 5.2.4 Quarto esperimento

Finalmente, con le modifiche adottate, siamo riusciti ad ottenere dei miglioramenti nelle metriche PSNR e SSIM (Tab.5.9).

Tuttavia, il successo più grande è visibile nelle immagini in output che, a differenza di quelle precedenti, riescono a ricostruire, se ben solo in parte, le immagini in super resolution.

E' infatti possibile distinguere i paesaggi e soggetti, riuscendo anche a cogliere dei dettagli che sono quasi assenti nella loro versione in low resolution.(Fig.5.3(b)).

Layer (type)	Output Shape	Param #
Conv2d-1	[ -1, 16, 128, 128]	448
BatchNorm2d-2	[ -1, 16, 128, 128]	32
LeakyReLU-3	[ -1, 16, 128, 128]	0
Conv2d-4	[ -1, 32, 128, 128]	4,640
BatchNorm2d-5	[ -1, 32, 128, 128]	64
LeakyReLU-6	[ -1, 32, 128, 128]	0
MaxPool2d-7	[ -1, 32, 64, 64]	0
Conv2d-8	[ -1, 64, 64, 64]	18,496
BatchNorm2d-9	[ -1, 64, 64, 64]	128
LeakyReLU-10	[ -1, 64, 64, 64]	0
Conv2d-11	[ -1, 128, 64, 64]	73,856
BatchNorm2d-12	[ -1, 128, 64, 64]	256
LeakyReLU-13	[ -1, 128, 64, 64]	0
Conv2d-14	[ -1, 256, 64, 64]	295,168
BatchNorm2d-15	[ -1, 256, 64, 64]	512
LeakyReLU-16	[ -1, 256, 64, 64]	0
ConvTranspose2d-17	[ -1, 128, 128, 128]	295,040
BatchNorm2d-18	[ -1, 128, 128, 128]	256
LeakyReLU-19	[ -1, 128, 128, 128]	0
ConvTranspose2d-20	[ -1, 64, 256, 256]	73,792
BatchNorm2d-21	[ -1, 64, 256, 256]	128
LeakyReLU-22	[ -1, 64, 256, 256]	0
ConvTranspose2d-23	[ -1, 32, 512, 512]	18,464
BatchNorm2d-24	[ -1, 32, 512, 512]	64
LeakyReLU-25	[ -1, 32, 512, 512]	0
Conv2d-26	[ -1, 16, 512, 512]	4,624
BatchNorm2d-27	[ -1, 16, 512, 512]	32
LeakyReLU-28	[ -1, 16, 512, 512]	0
Conv2d-29	[ -1, 3, 512, 512]	435
Tanh-30	[ -1, 3, 512, 512]	0
<b>Total params:</b>		786,435
<b>Trainable params:</b>		786,435
<b>Non-trainable params:</b>		0

Table 5.8: Model Architecture

Descrizione	Valore
Average PSNR between Sr and reconstructed images	16.93 dB
Average SSIM between Sr and reconstructed images	0.2815

Table 5.9: Valori medi di PSNR e SSIM

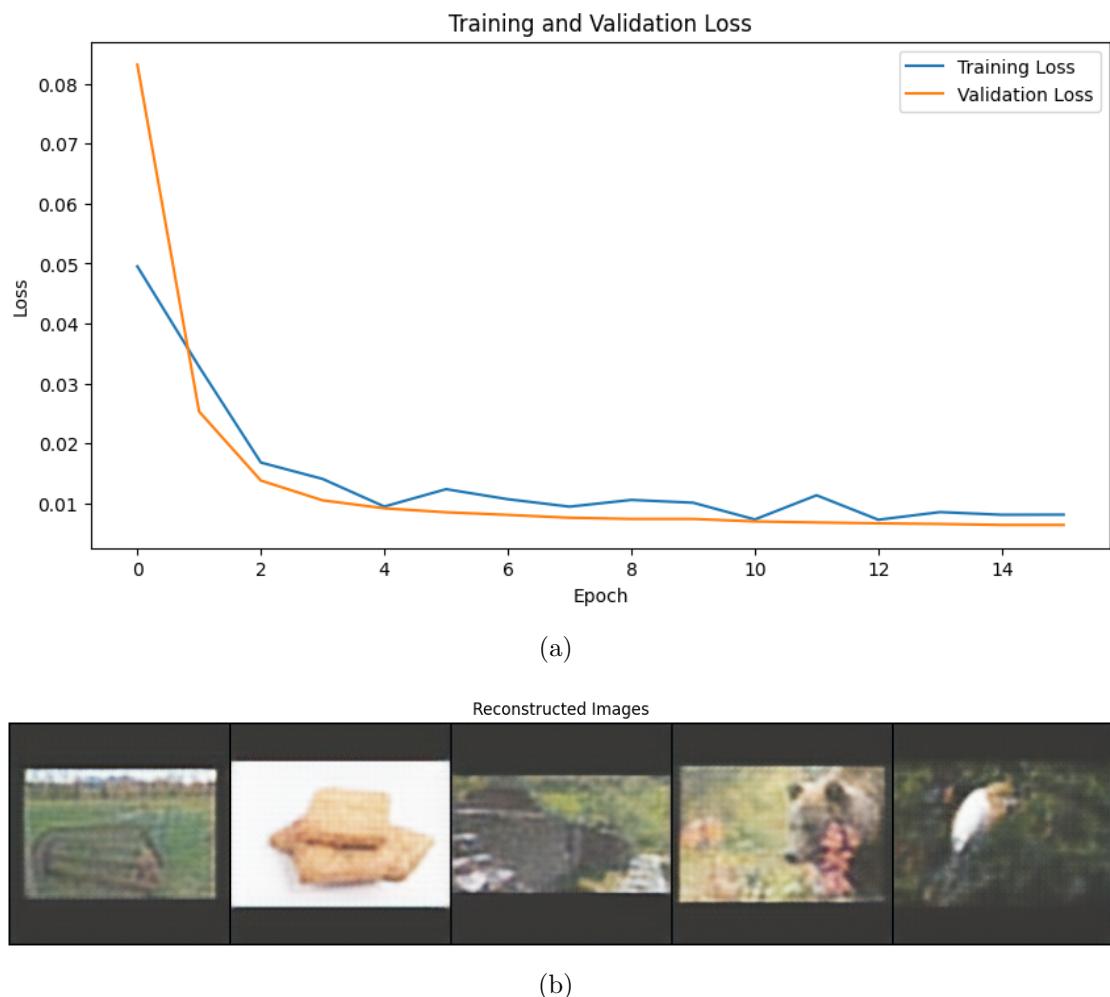


Figure 5.4: Risultati quarto esperimento

Visto il successo delle nuove introduzioni, le prossime architetture seguiranno le stesse orme, portando solo a modifiche a livello di disposizione dei layers o aggiunta di quest'ultimi.

### **5.2.5 Quinto esperimento**

Qui abbiamo aumentato leggermente la complessità.

Il ciò ci ha permesso di migliorare leggermente le metriche (Tab.5.11) senza però avere dei miglioramenti importanti a livello visivo. (Fig.5.5(b)).

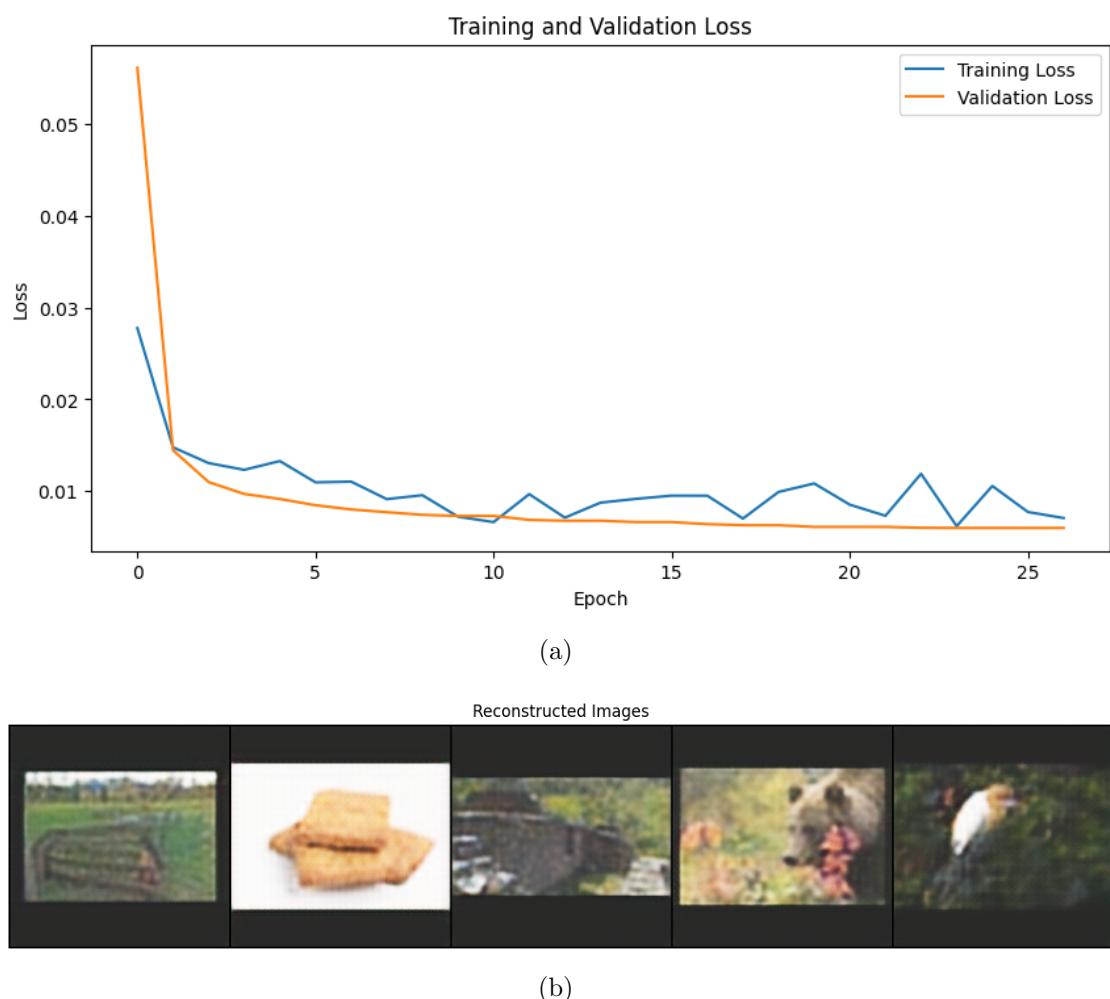


Figure 5.5: Risultati quinto esperimento

Layer (type)	Output Shape	Param #
Conv2d-1	[1, 16, 128, 128]	448
BatchNorm2d-2	[1, 16, 128, 128]	32
LeakyReLU-3	[1, 16, 128, 128]	0
Conv2d-4	[1, 32, 128, 128]	4,640
BatchNorm2d-5	[1, 32, 128, 128]	64
LeakyReLU-6	[1, 32, 128, 128]	0
MaxPool2d-7	[1, 32, 64, 64]	0
Conv2d-8	[1, 64, 64, 64]	18,496
BatchNorm2d-9	[1, 64, 64, 64]	128
LeakyReLU-10	[1, 64, 64, 64]	0
Conv2d-11	[1, 128, 64, 64]	73,856
BatchNorm2d-12	[1, 128, 64, 64]	256
LeakyReLU-13	[1, 128, 64, 64]	0
Conv2d-14	[1, 128, 64, 64]	147,584
BatchNorm2d-15	[1, 128, 64, 64]	256
LeakyReLU-16	[1, 128, 64, 64]	0
Conv2d-17	[1, 256, 64, 64]	295,168
BatchNorm2d-18	[1, 256, 64, 64]	512
LeakyReLU-19	[1, 256, 64, 64]	0
ConvTranspose2d-20	[1, 128, 128, 128]	295,040
BatchNorm2d-21	[1, 128, 128, 128]	256
LeakyReLU-22	[1, 128, 128, 128]	0
Conv2d-23	[1, 128, 128, 128]	147,584
BatchNorm2d-24	[1, 128, 128, 128]	256
LeakyReLU-25	[1, 128, 128, 128]	0
ConvTranspose2d-26	[1, 64, 256, 256]	73,792
BatchNorm2d-27	[1, 64, 256, 256]	128
LeakyReLU-28	[1, 64, 256, 256]	0
ConvTranspose2d-29	[1, 32, 512, 512]	18,464
BatchNorm2d-30	[1, 32, 512, 512]	64
LeakyReLU-31	[1, 32, 512, 512]	0
Conv2d-32	[1, 16, 512, 512]	4,624
BatchNorm2d-33	[1, 16, 512, 512]	32
LeakyReLU-34	[1, 16, 512, 512]	0
Conv2d-35	[1, 3, 512, 512]	435
Tanh-36	[1, 3, 512, 512]	0
<b>Total params:</b>	1,082,115	
<b>Trainable params:</b>	1,082,115	
<b>Non-trainable params:</b>	0	

Table 5.10: Modello quinto esperimento

Descrizione	Valore
Average PSNR between Sr and reconstructed images	17.79 dB
Average SSIM between Sr and reconstructed images	0.3057

Table 5.11: Valori medi di PSNR e SSIM

### 5.2.6 Sesto esperimento

Abbiamo ulteriormente aumentato la complessità (il numero di layers), ottenendo ulteriori miglioramenti in metriche.(Tab.5.13).

Table 5.12: Modello sesto esperimento

<b>Layer (type)</b>	<b>Output Shape</b>	<b>Param #</b>
Conv2d-1	[ -1, 16, 128, 128]	448
BatchNorm2d-2	[ -1, 16, 128, 128]	32
LeakyReLU-3	[ -1, 16, 128, 128]	0
Conv2d-4	[ -1, 32, 128, 128]	4,640
BatchNorm2d-5	[ -1, 32, 128, 128]	64
LeakyReLU-6	[ -1, 32, 128, 128]	0
MaxPool2d-7	[ -1, 32, 64, 64]	0
Conv2d-8	[ -1, 64, 64, 64]	18,496
BatchNorm2d-9	[ -1, 64, 64, 64]	128
LeakyReLU-10	[ -1, 64, 64, 64]	0
Conv2d-11	[ -1, 128, 64, 64]	73,856
BatchNorm2d-12	[ -1, 128, 64, 64]	256
LeakyReLU-13	[ -1, 128, 64, 64]	0
Conv2d-14	[ -1, 128, 64, 64]	147,584
BatchNorm2d-15	[ -1, 128, 64, 64]	256
LeakyReLU-16	[ -1, 128, 64, 64]	0
Conv2d-17	[ -1, 128, 64, 64]	147,584
BatchNorm2d-18	[ -1, 128, 64, 64]	256
LeakyReLU-19	[ -1, 128, 64, 64]	0
Conv2d-20	[ -1, 256, 64, 64]	295,168
BatchNorm2d-21	[ -1, 256, 64, 64]	512
LeakyReLU-22	[ -1, 256, 64, 64]	0
ConvTranspose2d-23	[ -1, 128, 128, 128]	295,040
BatchNorm2d-24	[ -1, 128, 128, 128]	256
LeakyReLU-25	[ -1, 128, 128, 128]	0
Conv2d-26	[ -1, 128, 128, 128]	147,584

Table 5.12: Modello sesto esperimento (continua)

Layer (type)	Output Shape	Param #
BatchNorm2d-27	[ -1, 128, 128, 128]	256
LeakyReLU-28	[ -1, 128, 128, 128]	0
Conv2d-29	[ -1, 128, 128, 128]	147,584
BatchNorm2d-30	[ -1, 128, 128, 128]	256
LeakyReLU-31	[ -1, 128, 128, 128]	0
ConvTranspose2d-32	[ -1, 64, 256, 256]	73,792
BatchNorm2d-33	[ -1, 64, 256, 256]	128
LeakyReLU-34	[ -1, 64, 256, 256]	0
ConvTranspose2d-35	[ -1, 32, 512, 512]	18,464
BatchNorm2d-36	[ -1, 32, 512, 512]	64
LeakyReLU-37	[ -1, 32, 512, 512]	0
Conv2d-38	[ -1, 16, 512, 512]	4,624
BatchNorm2d-39	[ -1, 16, 512, 512]	32
LeakyReLU-40	[ -1, 16, 512, 512]	0
Conv2d-41	[ -1, 3, 512, 512]	435
Tanh-42	[ -1, 3, 512, 512]	0
<b>Total params:</b>	1,377,795	
<b>Trainable params:</b>	1,377,795	
<b>Non-trainable params:</b>	0	

Descrizione	Valore
Average PSNR between Sr and reconstructed images	18.61 dB
Average SSIM between Sr and reconstructed images	0.3107

Table 5.13: Valori medi di PSNR e SSIM

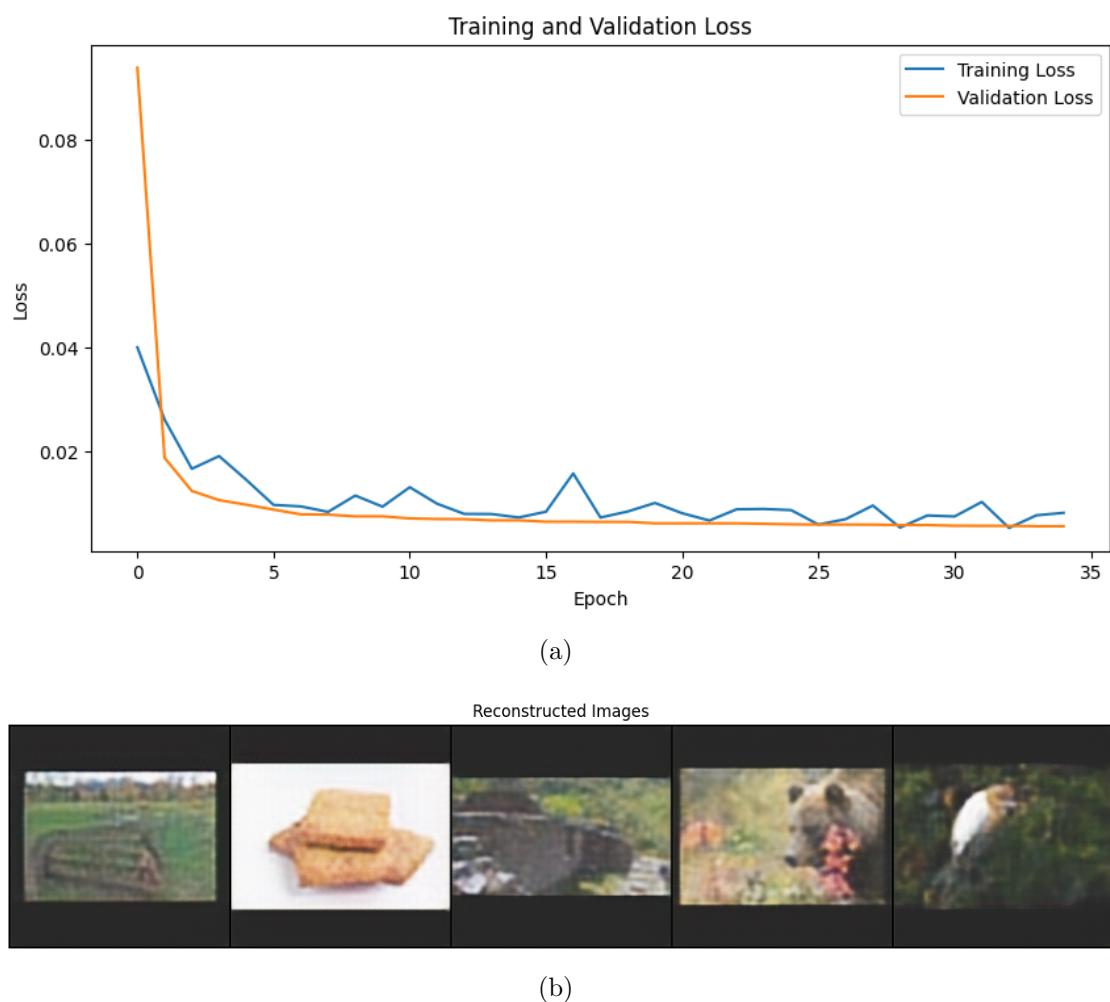


Figure 5.6: Risultati sesto esperimento

### 5.2.7 Settimo esperimento

Dato che, nelle scorse architetture, i risultati erano più o meno stazionari, qui abbiamo deciso di complicare notevolmente l'architettura raddoppiandone i parametri, con l'obiettivo di fare un salto di qualità.

Questa strategia, tuttavia, non ha portato ad alcun miglioramento in metriche (Tab.5.15) e nemmeno a livello visivo (Fig.5.7(b)).

Table 5.14: Modello settimo esperimento

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 128, 128]	448
BatchNorm2d-2	[-1, 16, 128, 128]	32
LeakyReLU-3	[-1, 16, 128, 128]	0
Conv2d-4	[-1, 32, 128, 128]	4,640
BatchNorm2d-5	[-1, 32, 128, 128]	64
LeakyReLU-6	[-1, 32, 128, 128]	0
MaxPool2d-7	[-1, 32, 64, 64]	0
Conv2d-8	[-1, 64, 64, 64]	18,496
BatchNorm2d-9	[-1, 64, 64, 64]	128
LeakyReLU-10	[-1, 64, 64, 64]	0
Conv2d-11	[-1, 128, 64, 64]	73,856
BatchNorm2d-12	[-1, 128, 64, 64]	256
LeakyReLU-13	[-1, 128, 64, 64]	0
MaxPool2d-14	[-1, 128, 32, 32]	0
Conv2d-15	[-1, 128, 32, 32]	147,584
BatchNorm2d-16	[-1, 128, 32, 32]	256
LeakyReLU-17	[-1, 128, 32, 32]	0
Conv2d-18	[-1, 256, 32, 32]	295,168
BatchNorm2d-19	[-1, 256, 32, 32]	512
LeakyReLU-20	[-1, 256, 32, 32]	0
Conv2d-21	[-1, 256, 32, 32]	590,080

Table 5.14: Modello settimo esperimento (continua)

Layer (type)	Output Shape	Param #
BatchNorm2d-22	[-1, 256, 32, 32]	512
LeakyReLU-23	[-1, 256, 32, 32]	0
ConvTranspose2d-24	[-1, 256, 64, 64]	590,080
BatchNorm2d-25	[-1, 256, 64, 64]	512
LeakyReLU-26	[-1, 256, 64, 64]	0
ConvTranspose2d-27	[-1, 128, 128, 128]	295,040
BatchNorm2d-28	[-1, 128, 128, 128]	256
LeakyReLU-29	[-1, 128, 128, 128]	0
ConvTranspose2d-30	[-1, 128, 256, 256]	147,584
BatchNorm2d-31	[-1, 128, 256, 256]	256
LeakyReLU-32	[-1, 128, 256, 256]	0
ConvTranspose2d-33	[-1, 64, 512, 512]	73,792
BatchNorm2d-34	[-1, 64, 512, 512]	128
LeakyReLU-35	[-1, 64, 512, 512]	0
Conv2d-36	[-1, 32, 512, 512]	18,464
BatchNorm2d-37	[-1, 32, 512, 512]	64
LeakyReLU-38	[-1, 32, 512, 512]	0
Conv2d-39	[-1, 16, 512, 512]	4,624
BatchNorm2d-40	[-1, 16, 512, 512]	32
LeakyReLU-41	[-1, 16, 512, 512]	0
Conv2d-42	[-1, 3, 512, 512]	435
Tanh-43	[-1, 3, 512, 512]	0
<b>Total params:</b> 2,263,299		
<b>Trainable params:</b> 2,263,299		
<b>Non-trainable params:</b> 0		

Descrizione	Valore
Average PSNR between Sr and reconstructed images	19.06 dB
Average SSIM between Sr and reconstructed images	0.3090

Table 5.15: Valori medi di PSNR e SSIM

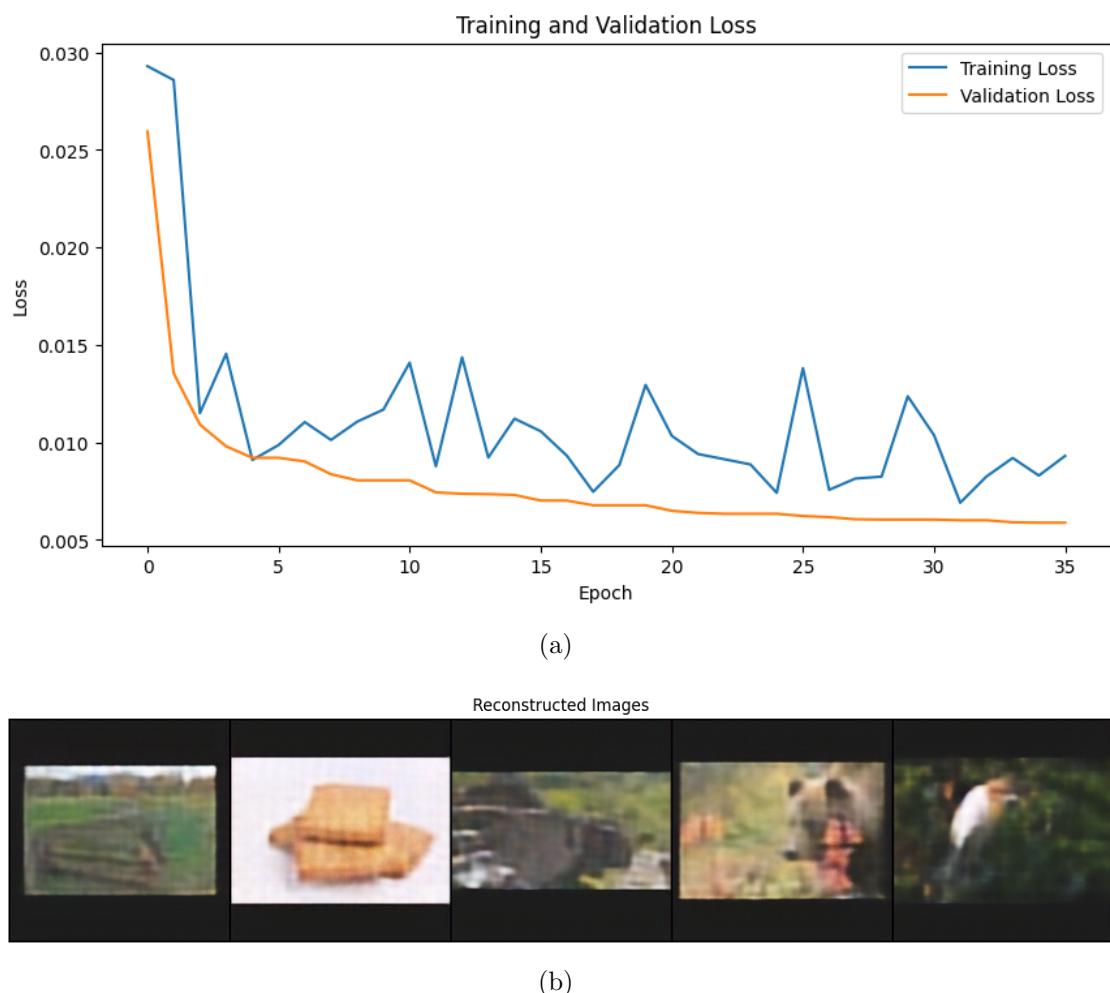


Figure 5.7: Risultati settimo esperimento

### 5.2.8 Ottavo esperimento

Dati i risultati del precedente modello, abbiamo capito che aumentare ancora la complessità non avrebbe portato vantaggi, dunque abbiamo optato per cambiare la disposizione dei layers concentrandoci in quelli di downsampling (maxpooling) ed upsampling(convtranspose).

Questa idea ci ha permesso di ottenere dei leggeri miglioramenti (Tab.5.17), tuttavia restiamo ancora molto lontani dai risultati sperati, infatti le immagini in output presentano ancora molte imperfezioni rispetto a quelle originali, caratterizzate anche dalla presenza di un "layer bianco" sopra di esse (Fig.5.8(b)).

Table 5.16: Modello ottavo esperimento

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 128, 128]	448
BatchNorm2d-2	[-1, 16, 128, 128]	32
LeakyReLU-3	[-1, 16, 128, 128]	0
Conv2d-4	[-1, 32, 128, 128]	4,640
BatchNorm2d-5	[-1, 32, 128, 128]	64
LeakyReLU-6	[-1, 32, 128, 128]	0
MaxPool2d-7	[-1, 32, 64, 64]	0
Conv2d-8	[-1, 64, 64, 64]	18,496
BatchNorm2d-9	[-1, 64, 64, 64]	128
LeakyReLU-10	[-1, 64, 64, 64]	0
Conv2d-11	[-1, 128, 64, 64]	73,856
BatchNorm2d-12	[-1, 128, 64, 64]	256
LeakyReLU-13	[-1, 128, 64, 64]	0
Conv2d-14	[-1, 128, 64, 64]	147,584
BatchNorm2d-15	[-1, 128, 64, 64]	256
LeakyReLU-16	[-1, 128, 64, 64]	0
Conv2d-17	[-1, 256, 64, 64]	295,168
BatchNorm2d-18	[-1, 256, 64, 64]	512

Table 5.16: Modello ottavo esperimento (continua)

Layer (type)	Output Shape	Param #
LeakyReLU-19	[-1, 256, 64, 64]	0
MaxPool2d-20	[-1, 256, 32, 32]	0
Conv2d-21	[-1, 256, 32, 32]	590,080
BatchNorm2d-22	[-1, 256, 32, 32]	512
LeakyReLU-23	[-1, 256, 32, 32]	0
ConvTranspose2d-24	[-1, 256, 64, 64]	590,080
BatchNorm2d-25	[-1, 256, 64, 64]	512
LeakyReLU-26	[-1, 256, 64, 64]	0
ConvTranspose2d-27	[-1, 128, 128, 128]	295,040
BatchNorm2d-28	[-1, 128, 128, 128]	256
LeakyReLU-29	[-1, 128, 128, 128]	0
Conv2d-30	[-1, 128, 128, 128]	147,584
BatchNorm2d-31	[-1, 128, 128, 128]	256
LeakyReLU-32	[-1, 128, 128, 128]	0
Conv2d-33	[-1, 64, 128, 128]	73,792
BatchNorm2d-34	[-1, 64, 128, 128]	128
LeakyReLU-35	[-1, 64, 128, 128]	0
Conv2d-36	[-1, 32, 128, 128]	18,464
BatchNorm2d-37	[-1, 32, 128, 128]	64
LeakyReLU-38	[-1, 32, 128, 128]	0
Conv2d-39	[-1, 16, 128, 128]	4,624
BatchNorm2d-40	[-1, 16, 128, 128]	32
LeakyReLU-41	[-1, 16, 128, 128]	0
Conv2d-42	[-1, 3, 128, 128]	435
Tanh-43	[-1, 3, 128, 128]	0
<b>Total params:</b> 2,263,299		
<b>Trainable params:</b> 2,263,299		
<b>Non-trainable params:</b> 0		

Descrizione	Valore
Average PSNR between Sr and reconstructed images	19.39 dB
Average SSIM between Sr and reconstructed images	0.3187

Table 5.17: Valori medi di PSNR e SSIM

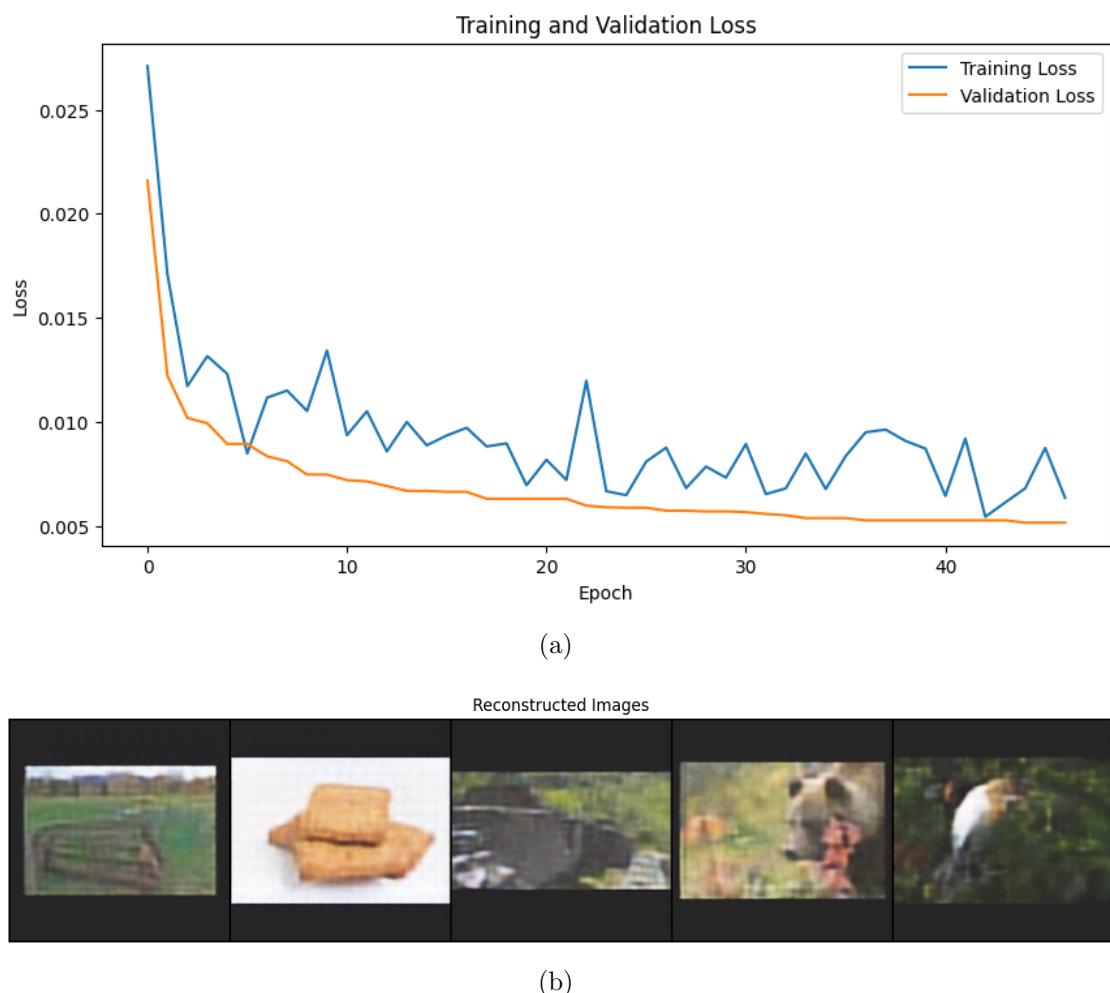


Figure 5.8: Risultati ottavo esperimento

### 5.2.9 Nono esperimento

Questa architettura si distingue dalle altre per l'utilizzo di kernel di dimensioni variabili nei primi due layers convoluzionali (kernel di dimensioni 7 e 5) e negli ultimi due (kernel di dimensioni 5 e 7).

Rispetto alle altre, in questa possiamo notare una capacità del modello di convergere molto più velocemente, tuttavia le metriche mostrano solo un leggerissimo miglioramento rispetto alla precedente e continuiamo ad avere lo stesso artefatto nelle immagini in output (Fig.5.9(b)).

Table 5.18: Modello nono esperimento

<b>Layer (type)</b>	<b>Output Shape</b>	<b>Param #</b>
Conv2d-1	[ -1, 16, 130, 130]	2,368
BatchNorm2d-2	[ -1, 16, 130, 130]	32
LeakyReLU-3	[ -1, 16, 130, 130]	0
Conv2d-4	[ -1, 32, 130, 130]	12,832
BatchNorm2d-5	[ -1, 32, 130, 130]	64
LeakyReLU-6	[ -1, 32, 130, 130]	0
MaxPool2d-7	[ -1, 32, 65, 65]	0
Conv2d-8	[ -1, 64, 65, 65]	18,496
BatchNorm2d-9	[ -1, 64, 65, 65]	128
LeakyReLU-10	[ -1, 64, 65, 65]	0
Conv2d-11	[ -1, 128, 65, 65]	73,856
BatchNorm2d-12	[ -1, 128, 65, 65]	256
LeakyReLU-13	[ -1, 128, 65, 65]	0
Conv2d-14	[ -1, 128, 65, 65]	147,584
BatchNorm2d-15	[ -1, 128, 65, 65]	256
LeakyReLU-16	[ -1, 128, 65, 65]	0
Conv2d-17	[ -1, 256, 65, 65]	295,168
BatchNorm2d-18	[ -1, 256, 65, 65]	512
LeakyReLU-19	[ -1, 256, 65, 65]	0

<b>Layer (type)</b>	<b>Output Shape</b>	<b>Param #</b>
MaxPool2d-20	[ -1, 256, 32, 32]	0
Conv2d-21	[ -1, 256, 32, 32]	590,080
BatchNorm2d-22	[ -1, 256, 32, 32]	512
LeakyReLU-23	[ -1, 256, 32, 32]	0
ConvTranspose2d-24	[ -1, 256, 64, 64]	590,080
BatchNorm2d-25	[ -1, 256, 64, 64]	512
LeakyReLU-26	[ -1, 256, 64, 64]	0
ConvTranspose2d-27	[ -1, 128, 128, 128]	295,040
BatchNorm2d-28	[ -1, 128, 128, 128]	256
LeakyReLU-29	[ -1, 128, 128, 128]	0
Conv2d-30	[ -1, 128, 128, 128]	147,584
BatchNorm2d-31	[ -1, 128, 128, 128]	256
LeakyReLU-32	[ -1, 128, 128, 128]	0
Conv2d-33	[ -1, 64, 128, 128]	73,792
BatchNorm2d-34	[ -1, 64, 128, 128]	128
LeakyReLU-35	[ -1, 64, 128, 128]	0
Conv2d-36	[ -1, 32, 128, 128]	18,464
BatchNorm2d-37	[ -1, 32, 128, 128]	64
LeakyReLU-38	[ -1, 32, 128, 128]	0
Conv2d-39	[ -1, 16, 130, 130]	12,816
BatchNorm2d-40	[ -1, 16, 130, 130]	32
LeakyReLU-41	[ -1, 16, 130, 130]	0
Conv2d-42	[ -1, 3, 128, 128]	2,355
Tanh-43	[ -1, 3, 128, 128]	0
<b>Total params:</b> 2,283,523		
<b>Trainable params:</b> 2,283,523		
<b>Non-trainable params:</b> 0		

Descrizione	Valore
Average PSNR between Sr and reconstructed images	18.05 dB
Average SSIM between Sr and reconstructed images	0.3211

Table 5.19: Valori medi di PSNR e SSIM

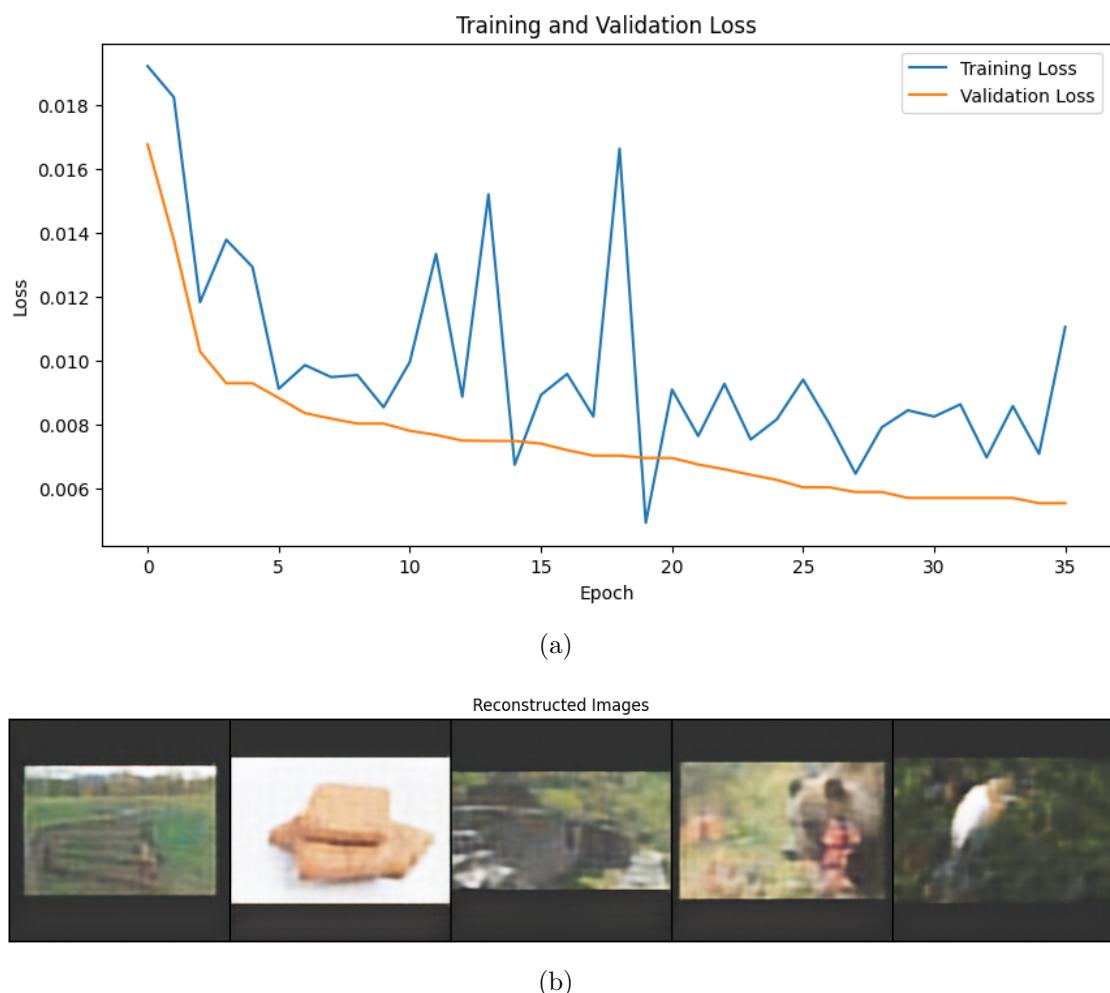


Figure 5.9: Risultati nono esperimento

### 5.2.10 Decimo esperimento

In questa architettura abbiamo deciso di fare qualche passo indietro riducendo la complessità ma mantenendo le modifiche più importanti (dimensioni kernel).

Nonostante i risultati siano più o meno uguali, la velocità di training e leggerezza computazionale di questa soluzione ci ha convinti nel mantenere le future architetture più leggere.

Table 5.20: Modello decimo esperimento

<b>Layer (type)</b>	<b>Output Shape</b>	<b>Param #</b>
Conv2d-1	[1, 16, 128, 128]	2,368
BatchNorm2d-2	[1, 16, 128, 128]	32
LeakyReLU-3	[1, 16, 128, 128]	0
Conv2d-4	[1, 32, 128, 128]	12,832
BatchNorm2d-5	[1, 32, 128, 128]	64
LeakyReLU-6	[1, 32, 128, 128]	0
MaxPool2d-7	[1, 32, 64, 64]	0
Conv2d-8	[1, 64, 64, 64]	18,496
BatchNorm2d-9	[1, 64, 64, 64]	128
LeakyReLU-10	[1, 64, 64, 64]	0
Conv2d-11	[1, 128, 64, 64]	73,856
BatchNorm2d-12	[1, 128, 64, 64]	256
LeakyReLU-13	[1, 128, 64, 64]	0
Conv2d-14	[1, 256, 64, 64]	295,168
BatchNorm2d-15	[1, 256, 64, 64]	512
LeakyReLU-16	[1, 256, 64, 64]	0
ConvTranspose2d-17	[1, 128, 128, 128]	295,040
BatchNorm2d-18	[1, 128, 128, 128]	256
LeakyReLU-19	[1, 128, 128, 128]	0
Conv2d-20	[1, 64, 128, 128]	73,792
BatchNorm2d-21	[1, 64, 128, 128]	128
LeakyReLU-22	[1, 64, 128, 128]	0
Conv2d-23	[1, 32, 128, 128]	18,464
BatchNorm2d-24	[1, 32, 128, 128]	64
LeakyReLU-25	[1, 32, 128, 128]	0
Conv2d-26	[1, 16, 130, 130]	12,816
BatchNorm2d-27	[1, 16, 130, 130]	32
LeakyReLU-28	[1, 16, 130, 130]	0
Conv2d-29	[1, 3, 128, 128]	2,355
Tanh-30	[1, 3, 128, 128]	0
<b>Total params:</b> 806,659		
<b>Trainable params:</b> 806,659		
<b>Non-trainable params:</b> 0		

Descrizione	Valore
Average PSNR between Sr and reconstructed images	18.11 dB
Average SSIM between Sr and reconstructed images	0.3165

Table 5.21: Valori medi di PSNR e SSIM

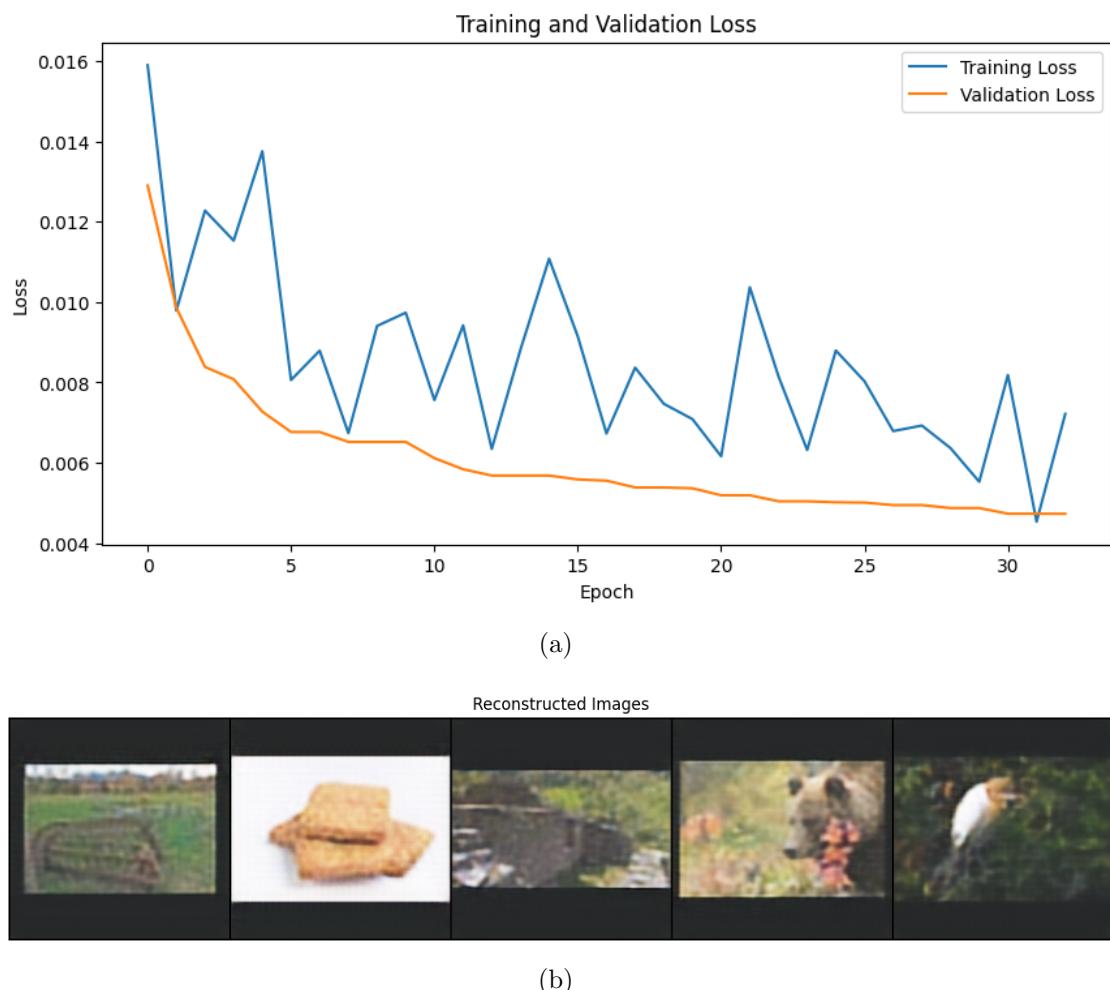


Figure 5.10: Risultati decimo esperimento

### 5.2.11 Undicesimo esperimento

La nuova architettura si distingue, rispetto alla precedente, per una disposizione ottimizzata dei layer di convoluzione, concentrandosi particolarmente sulla fase di encoding. In questa fase, abbiamo implementato un downsampling più strategico verso la parte finale dell'encoding, migliorando così l'estrazione delle caratteristiche e riducendo la dimensionalità in modo più efficace.

Queste modifiche ci hanno permesso di ottenere, dopo i momenti di stallo, un miglioramento nella SSIM (Tab.5.23), risultando anche in delle immagini con delle leggere migliorie a livello visivo (Fig.5.11(b)).

Table 5.22: Modello undicesimo esperimento

<b>Layer (type)</b>	<b>Output Shape</b>	<b>Param #</b>
Conv2d-1	[1, 16, 128, 128]	2,368
BatchNorm2d-2	[1, 16, 128, 128]	32
LeakyReLU-3	[1, 16, 128, 128]	0
Conv2d-4	[1, 32, 128, 128]	12,832
BatchNorm2d-5	[1, 32, 128, 128]	64
LeakyReLU-6	[1, 32, 128, 128]	0
Conv2d-7	[1, 64, 128, 128]	18,496
BatchNorm2d-8	[1, 64, 128, 128]	128
LeakyReLU-9	[1, 64, 128, 128]	0
Conv2d-10	[1, 128, 128, 128]	73,856
BatchNorm2d-11	[1, 128, 128, 128]	256
LeakyReLU-12	[1, 128, 128, 128]	0
MaxPool2d-13	[1, 128, 64, 64]	0
Conv2d-14	[1, 256, 64, 64]	295,168
BatchNorm2d-15	[1, 256, 64, 64]	512
LeakyReLU-16	[1, 256, 64, 64]	0
ConvTranspose2d-17	[1, 128, 128, 128]	295,040
BatchNorm2d-18	[1, 128, 128, 128]	256
LeakyReLU-19	[1, 128, 128, 128]	0
ConvTranspose2d-20	[1, 64, 256, 256]	73,792
BatchNorm2d-21	[1, 64, 256, 256]	128
LeakyReLU-22	[1, 64, 256, 256]	0
ConvTranspose2d-23	[1, 32, 512, 512]	18,464
BatchNorm2d-24	[1, 32, 512, 512]	64
LeakyReLU-25	[1, 32, 512, 512]	0
Conv2d-26	[1, 16, 514, 514]	12,816
BatchNorm2d-27	[1, 16, 514, 514]	32
LeakyReLU-28	[1, 16, 514, 514]	0
Conv2d-29	[1, 3, 512, 512]	2,355
Tanh-30	[1, 3, 512, 512]	0
<b>Total params:</b> 806,659		
<b>Trainable params:</b> 806,659		
<b>Non-trainable params:</b> 0		

Descrizione	Valore
Average PSNR between Sr and reconstructed images	18.17 dB
Average SSIM between Sr and reconstructed images	0.3330

Table 5.23: Valori medi di PSNR e SSIM

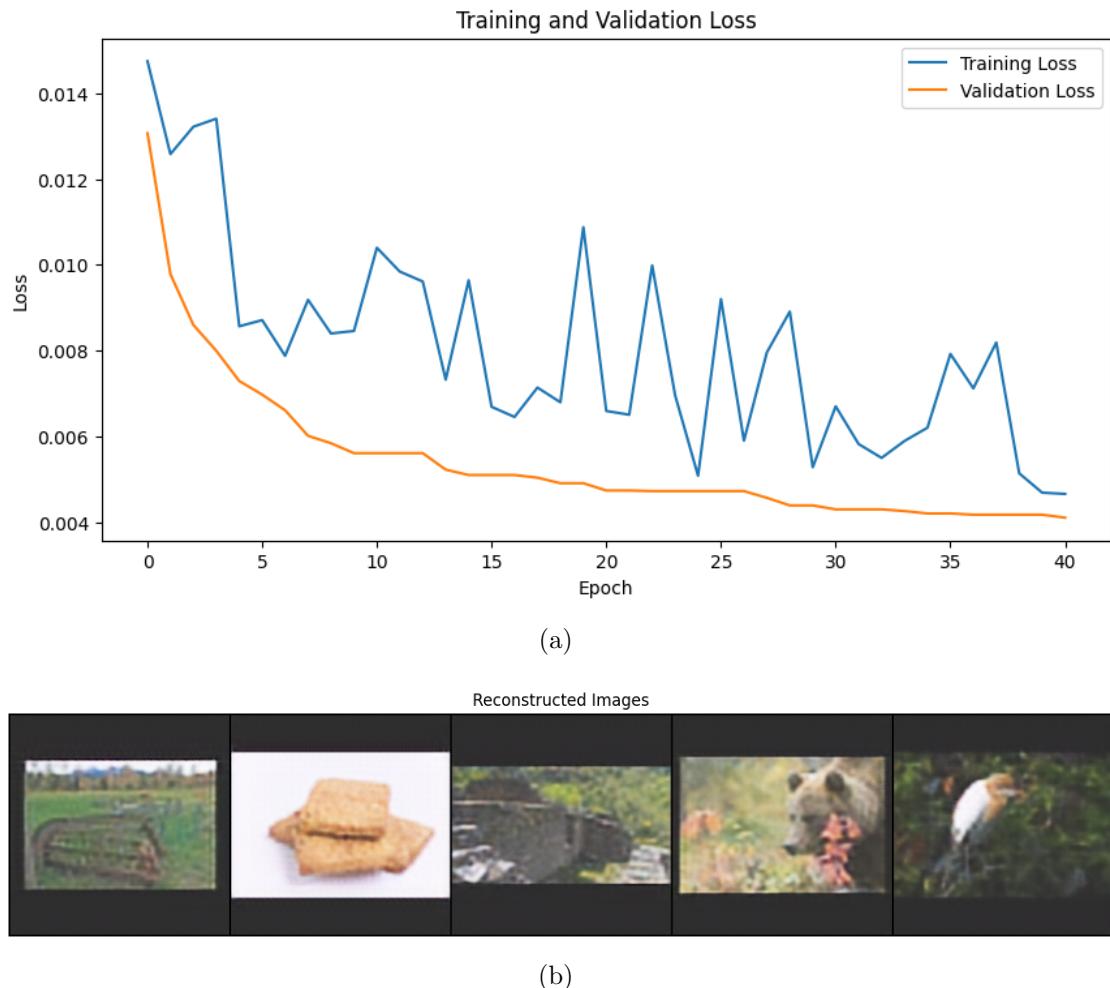


Figure 5.11: Risultati undicesimo esperimento

### 5.2.12 Dodicesimo esperimento

Visto il successo della precedente architettura, in questo esperimento abbiamo semplicemente deciso di addestrarlo per molte più epoche (arrivando ad un totale di 100), riuscendo ad ottenere i migliori risultati in metriche e visivi tra tutti gli esperimenti condotti.

Descrizione	Valore
Average PSNR between Sr and reconstructed images	19.40 dB
Average SSIM between Sr and reconstructed images	0.3609

Table 5.24: Valori medi di PSNR e SSIM

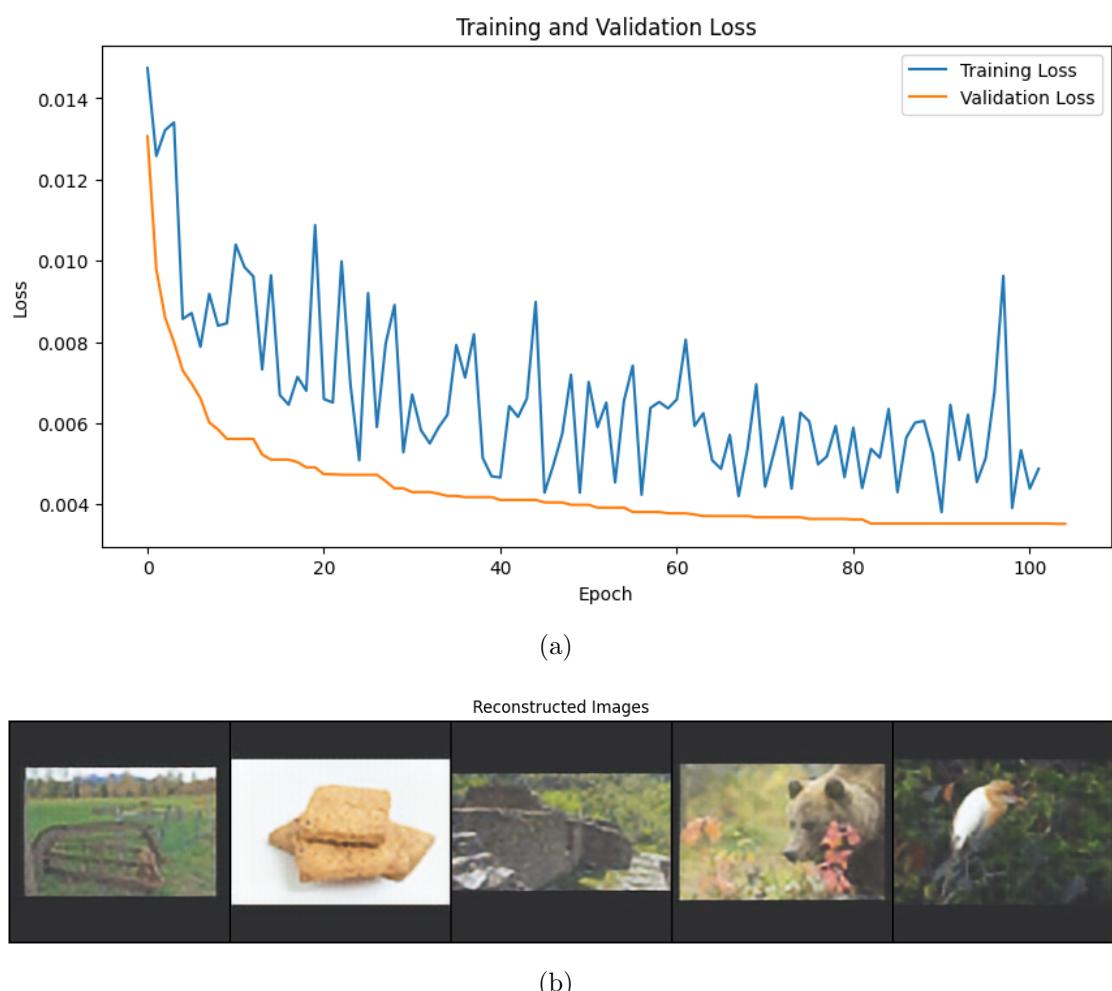


Figure 5.12: Risultati dodicesimo esperimento

## 5.3 Seconda Parte

### 5.3.1 Fixing white layer

In questa sezione, il nostro interesse si è spostato nel "layer bianco" presente in tutte le immagini di output di quasi tutti gli esperimenti da noi condotti.

In particolare, abbiamo iniziato a pensare di aver gestito male la fase di data loading o training, introducendo dunque qualche errore che portasse a questi risultati.

Al fine di trovarlo, abbiamo dunque deciso di analizzare dettagliatamente le funzioni di dataloading e di attivazione usate nelle architetture.

Questo ci ha permesso di trovare il problema: ovvero non gestivamo correttamente la normalizzazione dei dati in ingresso rispetto alla funzione di attivazione finale delle architetture.

Infatti, le immagini in ingresso venivano normalizzate nel range  $[0,1]$ , mentre in output, per via della funzione tahn, avevamo immagini nel range  $[-1,1]$ . Questa discrepanza ha portato la presenza del cosiddetto "layer bianco", creando dunque problemi nei risultati.

Per risolvere il problema, potevamo scegliere tra sistemare la normalizzazione iniziale o cambiare funzione di attivazione. Abbiamo deciso di optare per la seconda, adottando non più la tahn ma la sigmoide (che ha range  $[0,1]$ ).

### 5.3.2 U-net

Risolto la presenza dell'artefatto, abbiamo deciso di cambiare approccio utilizzando un'architettura U-net.

Questa scelta l'abbiamo fatta per via delle capacità di U-Net di cogliere molti più dettagli in più fasi dell'up e down sampling rispetto al semplice autoencoder.

L'architettura è stata comunque costruita basandosi sui successi dei precedenti esperimenti, in particolare abbiamo usato l'11° modello come base dell'U-net.

Com'è possibile vedere dalle metriche (Tab.5.26) e dalle immagini in output (Fig.??), sia per l'utilizzo della sigmoide che della u-net, siamo riusciti ad avere un enorme miglioramento, riuscendo a rimuovere l'artefatto più grande e riuscendo a ricostruire molti più dettagli in alta risoluzione che, con nessuno dei precedenti esperimenti, siamo riusciti ad ottenere.

Table 5.25: Modello U-Net

Layer (type)	Output Shape	Param #
Conv2d-1	[ -1, 16, 128, 128]	2,368
BatchNorm2d-2	[ -1, 16, 128, 128]	32
LeakyReLU-3	[ -1, 16, 128, 128]	0
Conv2d-4	[ -1, 32, 128, 128]	12,832
BatchNorm2d-5	[ -1, 32, 128, 128]	64
LeakyReLU-6	[ -1, 32, 128, 128]	0
Conv2d-7	[ -1, 64, 128, 128]	18,496
BatchNorm2d-8	[ -1, 64, 128, 128]	128
LeakyReLU-9	[ -1, 64, 128, 128]	0
Conv2d-10	[ -1, 128, 128, 128]	73,856
BatchNorm2d-11	[ -1, 128, 128, 128]	256
LeakyReLU-12	[ -1, 128, 128, 128]	0
MaxPool2d-13	[ -1, 128, 64, 64]	0
Continua nella pagina successiva		

**Table 5.25 – continuazione dalla pagina precedente**

<b>Layer (type)</b>	<b>Output Shape</b>	<b>Param #</b>
Conv2d-14	[1, 256, 64, 64]	295,168
BatchNorm2d-15	[1, 256, 64, 64]	512
LeakyReLU-16	[1, 256, 64, 64]	0
ConvTranspose2d-17	[1, 128, 128, 128]	295,040
BatchNorm2d-18	[1, 128, 128, 128]	256
LeakyReLU-19	[1, 128, 128, 128]	0
ConvTranspose2d-20	[1, 64, 256, 256]	73,792
BatchNorm2d-21	[1, 64, 256, 256]	128
LeakyReLU-22	[1, 64, 256, 256]	0
ConvTranspose2d-23	[1, 32, 512, 512]	18,464
BatchNorm2d-24	[1, 32, 512, 512]	64
LeakyReLU-25	[1, 32, 512, 512]	0
Conv2d-26	[1, 16, 514, 514]	12,816
BatchNorm2d-27	[1, 16, 514, 514]	32
LeakyReLU-28	[1, 16, 514, 514]	0
Conv2d-29	[1, 3, 512, 512]	2,355
Sigmoid-30	[1, 3, 512, 512]	0
<b>Total params</b>		<b>806,659</b>
<b>Trainable params</b>		<b>806,659</b>
<b>Non-trainable params</b>		<b>0</b>

<b>Descrizione</b>	<b>Valore</b>
Average PSNR between Sr and reconstructed images	24.24 dB
Average SSIM between Sr and reconstructed images	0.6935

Table 5.26: Valori medi di PSNR e SSIM

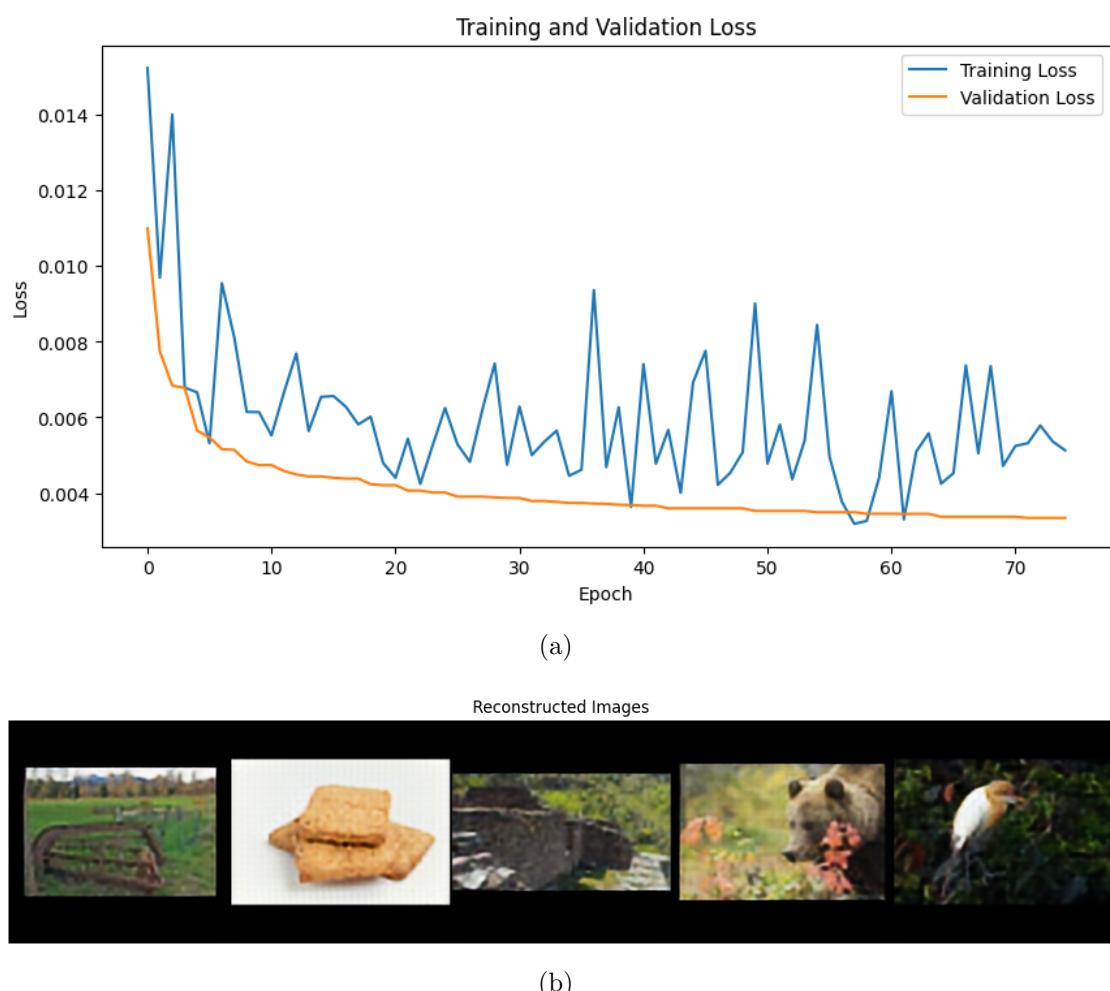


Figure 5.13: Risultati U-Net

### 5.3.3 Attention U-net

Visti gli ottimi risultati della U-net e dell'utilizzo della sigmoide, abbiamo deciso di apportare ulteriori migliorie all'architettura aggiungendo strati di attenzione Channel [14] e Spatial [15]. Gli strati di attenzione Channel sono stati scelti per la loro capacità di enfatizzare le caratteristiche più rilevanti attraverso l'adattamento del peso di ogni canale, migliorando così la rappresentazione dei dettagli più importanti [14]. Gli strati di attenzione Spatial, invece, aiutano a identificare le regioni salienti dell'immagine, migliorando ulteriormente la capacità del modello di concentrarsi sulle parti più significative [15].

Le modifiche adottate hanno permesso di ottenere un leggero miglioramento in termini di metriche, tuttavia il cambiamento più importante che abbiamo notato è stata la capacità del modello di arrivare a convergenza in maniera molto più stabile, come è possibile confermare osservandone il grafico (Fig. ??).

Table 5.27: Modello attention u-net

Layer (type)	Output Shape	Param #
Conv2d-1	[ -1, 16, 128, 128]	2,368
BatchNorm2d-2	[ -1, 16, 128, 128]	32
LeakyReLU-3	[ -1, 16, 128, 128]	0
MaxPool2d-4	[ -1, 16, 64, 64]	0
Conv2d-5	[ -1, 32, 64, 64]	12,832
BatchNorm2d-6	[ -1, 32, 64, 64]	64
LeakyReLU-7	[ -1, 32, 64, 64]	0
MaxPool2d-8	[ -1, 32, 32, 32]	0
Conv2d-9	[ -1, 64, 32, 32]	18,496
BatchNorm2d-10	[ -1, 64, 32, 32]	128
LeakyReLU-11	[ -1, 64, 32, 32]	0
MaxPool2d-12	[ -1, 64, 16, 16]	0
Conv2d-13	[ -1, 128, 16, 16]	73,856
BatchNorm2d-14	[ -1, 128, 16, 16]	256

Continued on next page

**Table 5.27 – continued from previous page**

<b>Layer (type)</b>	<b>Output Shape</b>	<b>Param #</b>
LeakyReLU-15	[ -1, 128, 16, 16]	0
MaxPool2d-16	[ -1, 128, 8, 8]	0
Conv2d-17	[ -1, 256, 8, 8]	295,168
BatchNorm2d-18	[ -1, 256, 8, 8]	512
LeakyReLU-19	[ -1, 256, 8, 8]	0
AdaptiveAvgPool2d-20	[ -1, 256, 1, 1]	0
Conv2d-21	[ -1, 32, 1, 1]	8,192
ReLU-22	[ -1, 32, 1, 1]	0
Conv2d-23	[ -1, 256, 1, 1]	8,192
AdaptiveMaxPool2d-24	[ -1, 256, 1, 1]	0
Conv2d-25	[ -1, 32, 1, 1]	8,192
ReLU-26	[ -1, 32, 1, 1]	0
Conv2d-27	[ -1, 256, 1, 1]	8,192
Sigmoid-28	[ -1, 256, 1, 1]	0
ChannelAttention-29	[ -1, 256, 1, 1]	0
Conv2d-30	[ -1, 1, 8, 8]	98
Sigmoid-31	[ -1, 1, 8, 8]	0
SpatialAttention-32	[ -1, 1, 8, 8]	0
ConvTranspose2d-33	[ -1, 128, 16, 16]	295,040
BatchNorm2d-34	[ -1, 128, 16, 16]	256
LeakyReLU-35	[ -1, 128, 16, 16]	0
ConvTranspose2d-36	[ -1, 64, 32, 32]	147,520
BatchNorm2d-37	[ -1, 64, 32, 32]	128
LeakyReLU-38	[ -1, 64, 32, 32]	0
ConvTranspose2d-39	[ -1, 32, 64, 64]	36,896
BatchNorm2d-40	[ -1, 32, 64, 64]	64
LeakyReLU-41	[ -1, 32, 64, 64]	0
ConvTranspose2d-42	[ -1, 16, 128, 128]	9,232
BatchNorm2d-43	[ -1, 16, 128, 128]	32
LeakyReLU-44	[ -1, 16, 128, 128]	0
ConvTranspose2d-45	[ -1, 16, 512, 512]	4,624

Continued on next page

**Table 5.27 – continued from previous page**

<b>Layer (type)</b>	<b>Output Shape</b>	<b>Param #</b>
BatchNorm2d-46	[ -1, 16, 512, 512]	32
LeakyReLU-47	[ -1, 16, 512, 512]	0
Conv2d-48	[ -1, 3, 512, 512]	2,355
Sigmoid-49	[ -1, 3, 512, 512]	0
<b>Total params:</b>	<b>932,757</b>	
<b>Trainable params:</b>	<b>932,757</b>	
<b>Non-trainable params:</b>	<b>0</b>	

<b>Descrizione</b>	<b>Valore</b>
Average PSNR between Sr and reconstructed images	24.48 dB
Average SSIM between Sr and reconstructed images	0.7082

Table 5.28: Valori medi di PSNR e SSIM

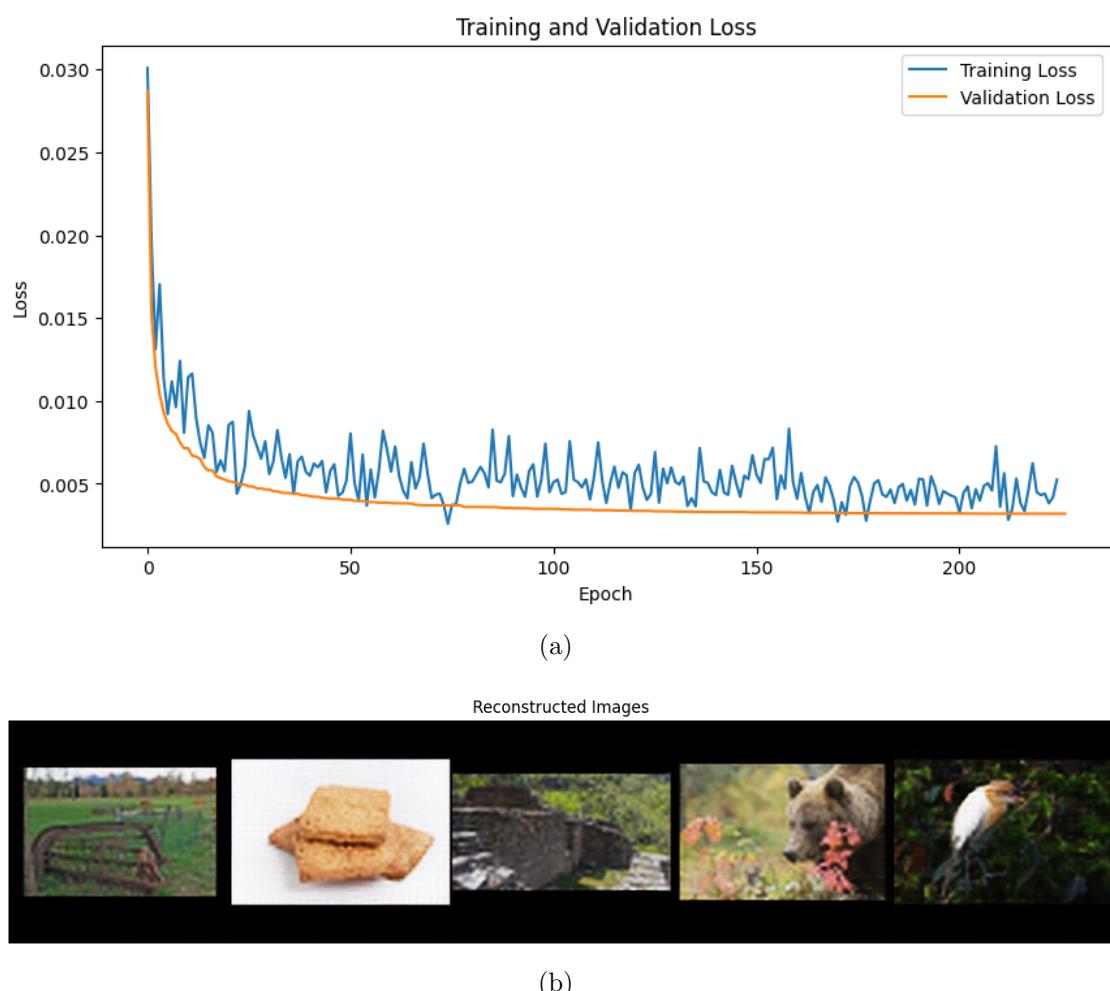


Figure 5.14: Risultati Attention U-Net

### 5.3.4 Attention U-net & sharpening kernel

Nonostante gli enormi successi ottenuti con le ultime architetture, le immagini ricostruite in super-resolution continuano ad apparire sfocate, non permettendo di raggiungere la qualità sperata.

Per cercare di porre rimedio a ciò abbiamo deciso di aggiungere, nella fase finale dell'architettura, un kernel laplaciano con l'obiettivo di applicare un filtro di sharpening all'immagine finale e rimuovere l'effetto "blur" ottenuto con le scorse architetture.

Tuttavia, benchè il leggero miglioramento in SSIM, l'applicazione del kernel non ha portato ad un miglioramento visivo degno di nota, continuando a lasciare le immagini in output sfocate.

Table 5.29: Attention unet con sharpening kernel

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 128, 128]	2,368
BatchNorm2d-2	[-1, 16, 128, 128]	32
LeakyReLU-3	[-1, 16, 128, 128]	0
MaxPool2d-4	[-1, 16, 64, 64]	0
Conv2d-5	[-1, 32, 64, 64]	12,832
BatchNorm2d-6	[-1, 32, 64, 64]	64
LeakyReLU-7	[-1, 32, 64, 64]	0
MaxPool2d-8	[-1, 32, 32, 32]	0
Conv2d-9	[-1, 64, 32, 32]	18,496
BatchNorm2d-10	[-1, 64, 32, 32]	128
LeakyReLU-11	[-1, 64, 32, 32]	0
MaxPool2d-12	[-1, 64, 16, 16]	0
Conv2d-13	[-1, 128, 16, 16]	73,856
BatchNorm2d-14	[-1, 128, 16, 16]	256
LeakyReLU-15	[-1, 128, 16, 16]	0
<i>Continued on next page</i>		

Table 5.29 – *Continued from previous page*

<b>Layer (type)</b>	<b>Output Shape</b>	<b>Param #</b>
MaxPool2d-16	[-1, 128, 8, 8]	0
Conv2d-17	[-1, 256, 8, 8]	295,168
BatchNorm2d-18	[-1, 256, 8, 8]	512
LeakyReLU-19	[-1, 256, 8, 8]	0
AdaptiveAvgPool2d-20	[-1, 256, 1, 1]	0
Conv2d-21	[-1, 32, 1, 1]	8,192
ReLU-22	[-1, 32, 1, 1]	0
Conv2d-23	[-1, 256, 1, 1]	8,192
AdaptiveMaxPool2d-24	[-1, 256, 1, 1]	0
Conv2d-25	[-1, 32, 1, 1]	8,192
ReLU-26	[-1, 32, 1, 1]	0
Conv2d-27	[-1, 256, 1, 1]	8,192
Sigmoid-28	[-1, 256, 1, 1]	0
ChannelAttention-29	[-1, 256, 1, 1]	0
Conv2d-30	[-1, 1, 8, 8]	98
Sigmoid-31	[-1, 1, 8, 8]	0
SpatialAttention-32	[-1, 1, 8, 8]	0
ConvTranspose2d-33	[-1, 128, 16, 16]	295,040
BatchNorm2d-34	[-1, 128, 16, 16]	256
LeakyReLU-35	[-1, 128, 16, 16]	0
ConvTranspose2d-36	[-1, 64, 32, 32]	147,520
BatchNorm2d-37	[-1, 64, 32, 32]	128
LeakyReLU-38	[-1, 64, 32, 32]	0
ConvTranspose2d-39	[-1, 32, 64, 64]	36,896
BatchNorm2d-40	[-1, 32, 64, 64]	64
LeakyReLU-41	[-1, 32, 64, 64]	0
ConvTranspose2d-42	[-1, 16, 128, 128]	9,232
BatchNorm2d-43	[-1, 16, 128, 128]	32
LeakyReLU-44	[-1, 16, 128, 128]	0
ConvTranspose2d-45	[-1, 16, 512, 512]	4,624

*Continued on next page*

Table 5.29 – *Continued from previous page*

Layer (type)	Output Shape	Param #
BatchNorm2d-46	[-1, 16, 512, 512]	32
LeakyReLU-47	[-1, 16, 512, 512]	0
Conv2d-48	[-1, 3, 512, 512]	2,355
Sigmoid-49	[-1, 3, 512, 512]	0
<b>Total params:</b>		932,757
<b>Trainable params:</b>		932,757
<b>Non-trainable params:</b>		0

Descrizione	Valore
Average PSNR between Sr and reconstructed images	24.25 dB
Average SSIM between Sr and reconstructed images	0.7100

Table 5.30: Valori medi di PSNR e SSIM

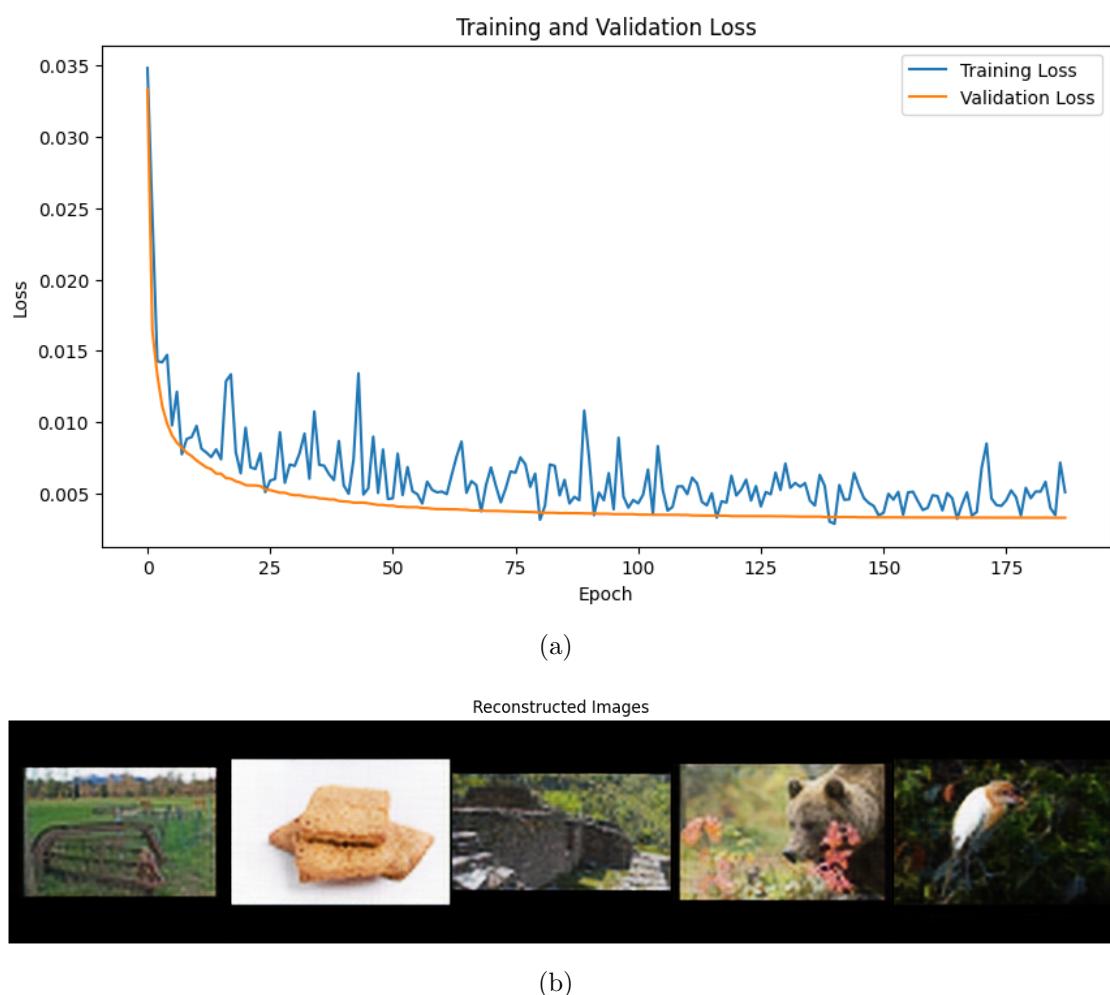


Figure 5.15: Risultati Attention U-Net con sharpening kernel

## 5.4 Conclusione

Con U-net si concludono i nostri tentativi di risolvere il problema di super resolution, tuttavia ogni singolo esperimento ci ha permesso di cogliere l'importanza di fallire per migliorare, permettendoci di migliorare le nostre capacità di analisi.

L'utilizzo delle metriche si sono rilevate di notevole importanza, in particolar modo la SSIM, che ci ha permesso di avere, ancor prima di vederle visivamente, un quadro completo dei risultati ottenuti. Si è infatti rilevata spesso veritiera ed in linea con gli output visivi.

Nonostante una prima fase di scoraggiamento, per via degli scarsi risultati, ci riteniamo soddisfatti di aver risolto i problemi che affliggevano le architetture e che ci hanno poi permesso di ottenere delle immagini in output ricostruite molto più gradevoli all'occhio e migliori rispetto alla loro versione in low resolution.

E' infatti possibile apprezzare come, nella loro versione ricostruita in super resolution, le immagini, specialmente dell'architettura U-Net (con e senza uso dei meccanismi di attenzione e kernel laplaciano), si presentano molto meno "pixellate" e più definite, permettendo di cogliere dettagli altrimenti nascosti.

# Chapter 6

## Codice

In questo capitolo non si mostrerà il codice completo ma si mostrerà com'è organizzato il notebook ed il funzionamento. Il notebook completo resta comunque disponibile alla seguente repository Rep.<sup>[5]</sup>

### 6.1 Notebook Super Resolution.ipynb

Il notebook super Resolution.ipynb contiene tutti gli esperimenti fatti, con relativi modelli e risultati, e alcune funzioni e procedure utili per il preprocessing del dataset e calcolo di metriche e plot ai fini della valutazione.

#### 6.1.1 Preprocessing immagini

La parte di preprocessing contiene importanti fasi della creazione del dataset.

In ordine:

- Padding
- Resizing

#### 6.1.2 Utility functions

Qui si definisce una serie di funzioni utilizzate durante la fase di sperimentazione.

- **calculate\_psnr & calculate\_ssim**: due funzioni utili per il calcolo delle due metriche
- **show\_images**: usata per mostrare, a fine training, le prime 5 immagini dell'ultima fase di validation del modello, mostrando inoltre anche le metriche PSNR e SSIM
- **ImageDataset**: questa è una classe molto utile che permette di caricare in batches con il DataLoader offerto da pytorch le coppie di immagini originali con la loro versione con risoluzione più bassa.

### 6.1.3 Esperimenti

In questa sessione è possibile trovare tutti gli esperimenti descritti in Cap. 5. Tutte le architetture proposte sono state implementate sfruttando Pytorch Lightning al fine di averle più strutturate ed organizzate. Ogni architettura è stata implementata definendo una lista contenente i risultati dell'ultima fase di validation ed estendendo le seguenti funzioni fornite dal framework:

- **training\_step & validation\_step**: queste funzioni le abbiamo adattate per usare batch contenenti sia l'immagine originale che con risoluzione più bassa, entrambe utili per calcolare la MSE loss
- **on\_train\_epoch\_end & on\_validation\_epoch\_end**: abbiamo sfruttato queste due funzioni per conservare le loss ottenute
- **on\_train\_start**: questa parte l'abbiamo cambiata per stampare una summary più dettagliata dell'architettura
- **on\_train\_end**: qui abbiamo aggiunto la fase di stampa dei plot di train e val loss nonché il salvataggio dello stato del modello

# Chapter 7

## Conclusioni

Il progetto proposto ha comportato diverse fasi, dell’addestramento alla valutazione del modello. Nonostante una prima fase negativa con risultati deludenti, siamo riusciti a risollevarci risolvendo alcuni problemi di gestione dei dati ed introducendo un’architettura U-Net, che ci ha permesso di ottenere delle immagini ricostruite, se ben lontane da quelle originali, molto gradevoli all’occhio, fornendo molti più dettagli rispetto alla loro controparte low resolution.

### 7.1 Considerazioni riguardanti gli esperimenti

Durante lo sviluppo del progetto, sono stati creati e testati diversi modelli di deep learning utilizzando come esempio gli autoencoder. Sono stati effettuati diversi esperimenti con architetture diverse, introducendo strati di *convoluzione*, *pooling*, *batch normalization*, *attention*, *laplacian kernel*. Sono stati utilizzati diversi strumenti per la valutazione dei modelli, come la *MSE loss* ed i grafici di train e validation loss nonché metriche molto diffuse in computer vision quali *PSNR* e *SSIM*. Questi strumenti hanno aiutato a capire come il modello si sia comportato durante il training evidenziando la difficoltà di ottenere di immagini di buona qualità da immagini di qualità più bassa.

### 7.2 Lezioni apprese

Lo sviluppo del progetto e la conseguente stesura della relazione ha fornito diverse lezioni:

1. **Regolazione dei modelli:** Si è provato ad affrontare il problema con diverse architetture, modificandone la complessità e aggiungendo o rimuovendo strati. Da questo si è sicuramente appreso che più complesso non implica necessariamente un miglioramento delle prestazioni. Spesso, infatti, i migliori risultati sono stati ottenuti con architetture più semplici.
2. **Importanza delle metriche:** si è sicuramente apprezzato l'importanza delle metriche per valutare correttamente le prestazioni del modello. In particolare la SSIM ha permesso di capire quanto distante fossero, a livello visivo e percettivo, le immagini ricostruite da quelle originali
3. **Hardware:** È emersa l'importanza di disporre di risorse computazionali potenti per affrontare efficacemente questo tipo di problemi. La scarsità di risorse ha limitato la fase di sperimentazione, causando lunghe attese dovute ai tempi di training. Inoltre, un'altra limitazione significativa è stata la necessità di tenere le proprie macchine accese continuamente, anche di notte, comportando un elevato consumo energetico, logorando l'hardware e generando calore, un problema particolarmente accentuato durante i mesi estivi.
4. **Normalizzazione dei dati:** E' importante controllare e normalizzare i dati in modo tale che siano consoni con ciò che il modello si aspetta e dà in output.

### 7.3 Sviluppi futuri

Infine sono stati tratti diversi punti di miglioramento:

1. **Tuning degli iperparametri:** Esplorare una gamma più ampia di iperparametri potrebbe aiutare a trovare una configurazione migliore per il modello.
2. **U-net:** Esplorare più in dettaglio la U-Net, nonchè i vari meccanismi di attenzione presenti, potrebbe portare ad ulteriori miglioramenti.

# Bibliography

- [1] A. Paul, S. Das, and U. S. MUKHERJEE. *Image Super-Resolution with Autoencoders*. <https://github.com/Sukrit-py/Image-Super-Resolution-with-Autoencoders>. 2024.
- [2] D. Chira, I. Haralampiev, O. Winther, A. Dittadi, and V. Liévin. “Image Super-Resolution with Deep Variational Autoencoders”. In: *Computer Vision – ECCV 2022 Workshops*. Ed. by L. Karlinsky, T. Michaeli, and K. Nishino. Cham: Springer Nature Switzerland, 2023, pp. 395–411. ISBN: 978-3-031-25063-7.
- [3] L. Gondara. “Medical Image Denoising Using Convolutional Denoising Autoencoders”. In: *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. 2016, pp. 241–246. DOI: [10.1109/ICDMW.2016.0041](https://doi.org/10.1109/ICDMW.2016.0041).
- [4] K. Bajaj, D. K. Singh, and M. A. Ansari. “Autoencoders Based Deep Learner for Image Denoising”. In: *Procedia Computer Science* 171 (2020). Third International Conference on Computing and Network Communications (Co-CoNet’19), pp. 1535–1541. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2020.04.164>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050920311431>.
- [5] A. Mezzina and G. Condorelli. *Super-Resolution-Computer-Vision-2024*. <https://github.com/Gpp23/Super-Resolution-Computer-Vision>.
- [6] Z. Chen, Z. Wu, E. Zamfir, K. Zhang, Y. Zhang, R. Timofte, X. Yang, H. Yu, C. Wan, Y. Hong, et al. “Ntire 2024 challenge on image super-resolution (x4): Methods and results”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 6108–6132.
- [7] PyTorch Contributors. *torch.utils.data.random\_split*. PyTorch Documentation. 2024. URL: [https://pytorch.org/docs/stable/data.html#torch.utils.data.random\\_split](https://pytorch.org/docs/stable/data.html#torch.utils.data.random_split).

- [8] PyTorch Contributors. *Data Loading and Processing Tutorial*. PyTorch Documentation. 2024. URL: [https://pytorch.org/tutorials/beginner/data\\_loading\\_tutorial.html](https://pytorch.org/tutorials/beginner/data_loading_tutorial.html).
- [9] NumPy Developers. *Random sampling (numpy.random)*. NumPy Documentation. 2024. URL: <https://numpy.org/doc/stable/reference/random/index.html>.
- [10] PyTorch Contributors. *Reproducibility in PyTorch*. PyTorch Documentation. 2024. URL: <https://pytorch.org/docs/stable/notes/randomness.html>.
- [11] C. Frenkel, D. Bol, and G. Indiveri. “Bottom-up and top-down approaches for the design of neuromorphic processing systems: tradeoffs and synergies between natural and artificial intelligence”. In: *Proceedings of the IEEE* 111.6 (2023), pp. 623–652.
- [12] E. Giffard, P. Jannin, and J. S. H. Baxter. “A preliminary exploration into top-down and bottom-up deep-learning approaches to localising neuro-interventional point targets in volumetric MRI”. In: *International Journal of Computer Assisted Radiology and Surgery* 19.2 (2024), pp. 283–296. ISSN: 1861-6429. DOI: 10.1007/s11548-023-03023-9. URL: <https://doi.org/10.1007/s11548-023-03023-9>.
- [13] W. Falcon and the PyTorch Lightning team. *PyTorch Lightning*. <https://github.com/Lightning-AI/lightning>. Accessed: [Insert Date Here]. 2019.
- [14] J. Hu, L. Shen, and G. Sun. “Squeeze-and-Excitation Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [15] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon. “CBAM: Convolutional Block Attention Module”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.