

Implementazione di una rete GAN applicata ad un task di Image Inpainting

Luigi Maggipinto

s319874@studenti.polito.it

Fabrizio Pisani

s305391@studenti.polito.it

Matteo Zulian

s310384@studenti.polito.it

July 2, 2025

Abstract

L'obiettivo di questo progetto è ricostruire immagini con parti mancanti o offuscate da maschere. Per farlo, abbiamo realizzato una rete GAN costituita in questo modo: il generatore è una rete Encoder-Decoder, riceve come input l'immagine "mascherata", e il suo obiettivo è quello di ricostruire l'immagine originale; il discriminatore, invece, è un semplice classificatore che riceve una immagine e stabilisce se si tratta di una immagine reale o ricostruita (fake). La creazione delle "maschere" è stata realizzata in maniera automatica nel codice. La posizione della maschera all'interno dell'immagine è casuale, ma uguale per tutte le immagini dello stesso batch. Nella fase di training sono state impiegate due loss: la Adversarial loss semplice e una combinazione pesata di Adversarial Loss e Reconstruction Loss. Il dataset contiene più di 4000 immagini di paesaggi di varia natura: montagne, mare, città, boschi, ecc. Abbiamo valutato il modello attraverso queste metriche: Fréchet inception distance (FID) e il Peak signal-to-noise ratio (PSNR).

1 Introduzione

Le immagini digitali sono diventate fondamentali per la diffusione delle informazioni, ma possono subire danni o presentare parti mancanti. L'*Image inpainting* è una tecnica che mira a restaurare visivamente tali aree e può essere applicata in vari contesti. Il problema è complesso e

non ha soluzioni uniche, ma i progressi del deep learning hanno permesso di ottenere risultati soddisfacenti. Dallo studio della letteratura, è emerso che le *Generative Adversarial Network* (GAN) sono comunemente utilizzate per l'image inpainting. Questo approccio prevede l'addestramento di due reti: un generatore G e un discriminatore D . Il compito di G è generare immagini realistiche, mentre D svolge un ruolo avversario, distinguendo tra le immagini generate da G e quelle reali. A partire dal lavoro descritto nel paper [1], il nostro gruppo ha deciso di sviluppare una rete GAN con lo scopo di produrre sinteticamente immagini "mascherate". Inoltre, ci siamo basati sulla review di Xiang et al. [2] per trarre tutte le informazioni teoriche necessarie per realizzare il progetto.

Il nostro progetto si è sviluppato nel seguente modo: abbiamo deciso di focalizzarci sulla scelta dell'architettura migliore possibile. Una volta trovata un'architettura adatta, abbiamo svolto sette training differenti della stessa rete, variando iperparametri e ottimizzatore.

2 Dataset

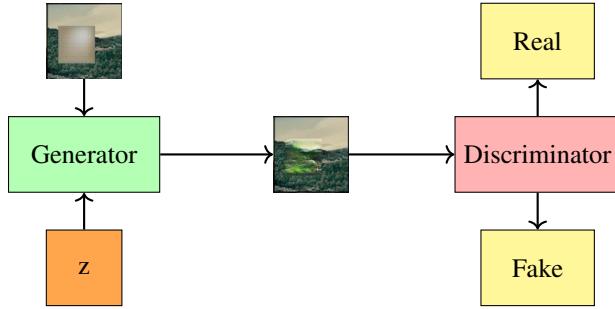
Abbiamo considerato vari domini nel contesto dell'image inpainting, come mare, paesaggi, foto di persone e quadri. La scelta finale è ricaduta sul dominio di paesaggi. La scelta è stata motivata da un buon equilibrio fra somiglianza semantica tra le immagini e varietà cromatica. Per questi motivi, il dataset impiegato è stato ottenuto

da Kaggle al link <https://www.kaggle.com/datasets/arnaud58/landscape-pictures>. Il dataset contiene 4319 immagini, che sono state suddivise in 70% di training, 15% di validation e 15% di test, suddivise in maniera casuale. Le immagini di training e di validation sono state suddivise in batch. Il tuning del *Batch Size* è stato svolto nella fase sperimentale 4.

3 Metodi

3.1 Architettura

Per risolvere il problema dell'*Image Inpainting*, abbiamo deciso di adottare l'uso delle GAN. Il generatore è una rete encoder-decoder, mentre il discriminatore è un classificatore. L'obiettivo era bilanciare la complessità tra generatore e discriminatore, mantenendo un compromesso tra le risorse computazionali richieste e la qualità del risultato. Quindi, la prima fase del nostro progetto è stata spesa a valutare attentamente come raggiungere questi risultati.



3.1.1 Architettura iniziale

L'architettura iniziale del generatore era basata su un encoder-decoder che riceveva immagini di dimensioni variabili, ridimensionate a $128 \times 128 \times 3$. Ad essa veniva sovrapposta una *patch* nera, creando un input di $128 \times 128 \times 4$. Ogni layer convolutorio comprendeva convoluzioni 2D, attivazioni ReLU e *Max Pooling*, con l'output dell'encoder che alimentava il decoder. Quest'ultimo comprendeva layer così composti: una operazione di de-convoluzione, e una ReLU come attivazione. Uno schema del generatore è riportato in figura 1. Invece,

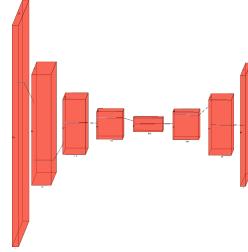


Figure 1: Architettura generatore

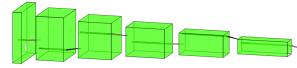


Figure 2: Architettura discriminatore

il discriminatore era un classificatore, che riceveva in input una immagine ridimensionata ($128 \times 128 \times 3$) e produceva in output il suo giudizio sull'immagine, ovvero se essa fosse vera o falsa. Ogni blocco convolutorio del discriminatore coinvolge una convoluzione 2D e una *Leaky ReLU* come funzione di attivazione. L'immagine 2 illustra l'architettura del discriminatore.

Il *training* di questa rete ha prodotto dei risultati promettenti, ma di qualità non ancora sufficiente.

3.1.2 Aggiunta del *Contextual Attention*

Prendendo spunto dall'articolo [3], abbiamo pertanto aggiunto al generatore il modulo di *Contextual Attention* per rendere più efficace e complessa la generazione. Questo approccio si basa sull'idea che, per ripristinare con successo le aree mancanti, sia necessario considerare il contesto circostante all'area da completare, in modo che il contenuto generato sia coerente con il resto dell'immagine. La rete si concentra quindi su queste parti per generare contenuti che si integrino in modo naturale nella zona vuota. Inoltre, abbiamo aggiunto ad ogni layer un *Dropout* di 0.3, nel tentativo di rendere il *training* più veloce e meno complesso. Il discriminatore è analogo a quello descritto in 3.1.1.

Seppur questo approccio fosse decisamente più interessante e complesso del precedente, abbiamo riscontrato che per completare l'allenamento di questa nuova rete

fossero necessario risorse computazioni ben più potenti di quelle impiegate in precedenza. Infatti, per completare un’epoca è stato speso un tempo esponenzialmente maggiore rispetto al *training* della rete iniziale, comportando quindi un tempo enorme per concludere tutto l’allenamento. Inoltre, a parità di epoche di allenamento (300), i risultati di questa rete erano nettamente inferiori di quelli ottenuti con il primo *training*. Questo probabilmente è causato dalla complessità globale della rete, necessitando di più epoche per ottenere risultati di buona qualità. Quindi, a causa di questi impedimenti, abbiamo deciso di abbandonare l’idea di realizzare una rete GAN dotata di *Contextual Attention*.

3.1.3 Architettura finale

Dopo aver scartato l’idea del *Contextual Attention*, abbiamo modificato le reti descritte in 3.1.1. La struttura di base del generatore è rimasta pressoché analoga. Rispetto alla rete illustrata in figura 1, abbiamo aggiunto in ogni layer convolutorio una *batch normalization* (subito dopo l’operazione di convoluzione), e un *Residual block* (in coda al *Max pooling*), per rendere più veloce l’apprendimento. Il discriminatore è rimasto invariato.

Questa rete ha prodotto i risultati sperati in termini di equilibrio fra i due attori avversari, e di carico computazionale-quality. Nel seguito, tutte le altre valutazioni (come scelta *loss* o valutazione degli iperparametri) e gli esperimenti sono stati basati su questa architettura.

3.2 Loss

3.2.1 Adversarial Loss

Una delle funzione di *loss* più usata nel contesto delle GAN è la *Adversarial Loss*. Il processo di apprendimento è simile a un gioco tra due giocatori: D riceve come input sia le immagini sintetiche prodotte da G , sia i campioni reali, e cerca di distinguergli. G , invece, cerca di ingannare D generando campioni che siano i più realistici possibile. La *Adversarial Loss* si definisce come:

$$\min_G \max_D (\mathbb{E}_{x \sim P_r} [\log(D(x))] + \mathbb{E}_{\hat{x} \sim P_g} [\log(1 - D(\hat{x}))])$$

dove P_r è la distribuzione reale dei dati e P_g è la distribuzione generata definita da $\hat{x} = G(z)$, con $z \sim p(z)$. Nel contesto dell’inpainting, si impone che il generatore ricostruisca le parti danneggiate delle immagini. Una volta stabilita l’architettura della rete, abbiamo deciso di effettuare i primi training con questa tipologia di *loss*. I risultati sono stati da subito promettenti.

3.2.2 Reconstruction Loss + Adversarial Loss

La *Reconstruction Loss* calcola la distanza pixel-per-pixel tra la previsione della rete I_{out} e l’immagine reale I_{gt} . Come esperimento, abbiamo deciso di combinare, con pesi differenti, le due loss, al fine di bilanciare realismo dell’immagine e fedeltà rispetto a quella originale. I risultati sono stati tuttavia scadenti, anche variando i pesi di bilanciamento fra le due *loss*. Abbiamo pertanto deciso di scartare l’idea della combinazione delle due funzioni di perdita, continuando solo con la *Adversarial Loss*.

4 Esperimenti

Dopo aver stabilito l’architettura finale della rete (descritta in 3.1.3), abbiamo eseguito sette allenamenti della rete, variando ogni volta **learning rate**, la **dimensione del batch** e l’ottimizzatore. Per la valutazione quantitativa dei risultati sono state impiegate le metriche di valutazione FID e PSNR.

4.1 Metriche

Le metriche di valutazione indicano l’efficacia dell’algoritmo considerato. Sono state considerate il rapporto segnale-rumore di picco (PSNR) e la distanza di Frechet di inception (FID).

PSNR è una metrica di valutazione usata nella compressione delle immagini, e misura la qualità della ricostruzione. L’unità di misura del PSNR è il decibel, e valori tipici di PSNR possono essere fra 30 e 50 dB. Un risultato ancora più alto di PSNR indica un’alta qualità delle immagini. Tuttavia, questa metrica potrebbe non riflettere in modo accurato la percezione umana della qualità visiva, pertanto non può essere usata in solitaria.

Il *Frechet Inception Distance* è una metrica usata per valutare la qualità delle immagini generate da modelli

generativi. Utilizza la distanza di Frèchet per confrontare le statistiche dei campioni generati con quelle dei campioni reali. Un punteggio FID più basso indica una maggiore somiglianza tra le distribuzioni delle immagini generate e quelle reali, suggerendo una migliore qualità visiva delle immagini sintetiche. Un punteggio ideale sarebbe 0, che indica che le due distribuzioni sono identiche. Anche in questo caso, FID potrebbe essere in contrasto con la percezione umana della qualità dell'immagine.

4.2 Valutazione Iperparametri

Gli iperparametri considerati sono stati principalmente tre: il *learning rate*, il *batch size* e il numero di epoche.

4.2.1 Numero di epoche

Il numero di epoche indica la durata del *training*. Abbiamo notato empiricamente che un training di circa 300 epoche fosse sufficiente per ottenere risultati buoni. Aumentando il numero di epoche a 500, abbiamo riscontrato risultati pressoché simili, mentre con 1000 epoche il risultato è stato addirittura peggiore. Nel capitolo 4 tutti gli esperimenti si basano su 300 epoche.

4.2.2 Analisi al variare del learning rate

In questo paragrafo sono riportati i *training* al variare del *learning rate*, con i seguenti iperparametri costanti: *Batch size*: 16 e *Ottimizzatore*: *Adams*. Il *learning rate* incide sulla velocità dell'apprendimento. Abbiamo deciso di impiegare 3 diversi valori: $\alpha = 10^{-3}, 10^{-4}, 10^{-5}$.

Nel primo esperimento, il *learning rate* è stato impostato a 10^{-3} , un valore relativamente elevato per una rete GAN. I risultati di questo training sono stati insoddisfacenti, caratterizzati da immagini di bassa qualità e da una notevole instabilità nel processo di apprendimento. Già nelle prime 100 epoche, la funzione di perdita, sia del generatore (G) che del discriminatore (D), rimaneva costante, assumendo un valore di 100, un comportamento riscontrato anche nelle *Validation Loss*. Riteniamo che la causa principale di questi risultati fallimentari sia attribuibile a un *learning rate* troppo alto, che ha compiuto aggiornamenti eccessivamente bruschi dei pesi sia per il generatore che per il discriminatore. Questo ha determinato oscillazioni nel processo di ottimizzazione, im-

pedendo una progressione regolare e causando una convergenza scarsa o addirittura assente. Considerando gli scarsi risultati prodotti con l'impiego di questo *learning rate*, abbiamo deciso di scartarlo e non abbiamo applicato le metriche di valutazione.

Nel secondo esperimento, il *learning rate* è stato ridotto a 10^{-4} . In questo caso, i risultati sono stati significativamente migliori, con immagini di alta qualità già dopo 300 epoche, e il training è apparso molto più stabile. Un *learning rate* inferiore ha permesso aggiornamenti più graduali, favorendo un apprendimento più accurato e una convergenza più solida. Questo miglioramento è evidente anche dall'analisi delle curve della *Adversarial Loss* rispetto alla *Validation Adversarial Loss*, sia per il generatore che per il discriminatore, come illustrato nell'immagine 3.

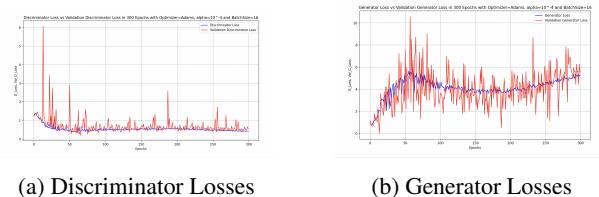


Figure 3: $\alpha = 10^{-4}$

Considerando le metriche di valutazione, il training ha prodotto $FID=95.52$ e un $PSNR=71.28$. Il valore di *FID* è abbastanza alto, considerando che un valore ideale si assesta intorno a 50. D'altro canto, il valore di *PSNR* è ottimo. Pertanto, possiamo considerare che le immagini siano di buona qualità, ma non ottimali. Infatti, a livello visivo, è evidente che la maggior parte delle immagini sia ricostruita in modo ottimo, mentre alcune (una minima parte) sono ricostruite in modo insufficiente. Quindi il training avuto un successo parziale, ma soddisfacente.

Nel terzo esperimento, il *learning rate* è stato ulteriormente ridotto a 10^{-5} . Anche questo training ha prodotto risultati positivi, con immagini di alta qualità e una stabilità notevole durante l'allenamento. Un *learning rate* ancora più basso ha consentito al modello di apprendere con maggiore precisione, riducendo il rischio di aggiornamenti errati e garantendo una convergenza robusta. Tuttavia, rispetto al training con un *learning rate* di 10^{-4} , il processo di apprendimento è risultato leggermente più lento, richiedendo un training di 400 epoche per ottenere

risultati ottimali.

Come evidenziato dall'immagine 4 relativa alla funzione di perdita, la curva è risultata decisamente più stabile, con oscillazioni minime e con un valore assoluto inferiore rispetto al secondo esperimento, dovuto alla riduzione dell'ordine di grandezza del passo di apprendimento.

Per quanto riguarda le metriche, il PSNR è rimasto elevato, con un valore di 71.26, mentre il FID è peggiorato, attestandosi a 157.00, segnalando una qualità complessiva inferiore delle immagini rispetto al secondo esperimento. Questo è probabilmente dovuto alla maggiore lentezza di apprendimento, dovuto ad un learning rate inferiore.

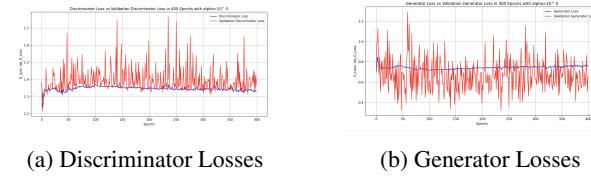


Figure 4: $\alpha = 10^{-5}$

In conclusione, sebbene il training con un learning rate di 10^{-5} abbia mostrato funzioni di perdita più stabili, il **learning rate** di 10^{-4} si è dimostrato il parametro ottimale per ottenere risultati superiori. Questo valore ha garantito un equilibrio ideale tra velocità di apprendimento e stabilità del modello, come evidenziato anche dalle metriche di valutazione, che sono risultate migliori rispetto a quelle ottenute nell'ultimo esperimento.

4.2.3 Analisi al variare della BatchSize

In questo paragrafo sono riportati i training al variare del batch size, con i seguenti iperparametri costanti: Learning rate: 10^{-4} e Ottimizzatore: Adams. Il batch size è la dimensione del gruppo di dati che viene elaborato insieme. Usare un batch size maggiore migliora la stima del gradiente, rendendo più lento l'addestramento. Invece, con un batch size minore avviene il contrario. Abbiamo considerato i seguenti valori: 8, 16, 32, 64. Tutti i training sono stati eseguiti per 300 epoch, con l'obiettivo di individuare la configurazione migliore in termini di qualità delle immagini generate e stabilità del processo di apprendimento.

Nel primo esperimento, il batch size è stato impostato a 8. I risultati ottenuti sono stati buoni, sebbene non ottimali. Un batch size ridotto ha permesso aggiornamenti frequenti, ma con un numero limitato di campioni, il che ha determinato un training più rumoroso, data la sua natura stocastica, con oscillazioni più evidenti nelle funzioni di perdita. Tuttavia, la rete è riuscita a generare immagini di qualità discreta. In termini di metriche di valutazione, il training ha prodotto risultati analoghi a quello descritto in precedenza: FID di 154.8 e un PSNR di 69.08. In questo caso, è plausibile che un batch size molto piccolo possa aver causato un andamento molto casuale del training, rallentando un po' il processo di apprendimento.

Nel secondo esperimento, il batch size è stato aumentato a 16, e i risultati sono stati eccellenti. Un batch size moderato ha consentito un buon equilibrio tra la frequenza degli aggiornamenti e la stabilità del training. Le immagini generate hanno mostrato una qualità elevata già dopo circa 250 epoch, con una convergenza ottimale sia per il generatore che per il discriminatore. Le metriche di valutazione hanno confermato la qualità del training, con risultati di buon livello (FID=95.52, PSNR=71.28).

Nel terzo esperimento, con batch size pari a 32, la qualità delle immagini è rimasta accettabile, ma leggermente inferiore rispetto a quella ottenuta con batch size 16. Il training è stato comunque stabile, registrando però un FID elevato di 155.20 e un PSNR di 70.14.

Infine, con batch size 64, il training è risultato instabile, con oscillazioni marcate nelle curve della *Loss* e una qualità delle immagini notevolmente peggiorata. Infatti, le metriche di valutazione (FID=236.32, PSNR=67.00) hanno confermato la scarsa performance di questa configurazione.

Alla luce di questi risultati, il **batch size** di 16 si è dimostrato il valore migliore per ottenere risultati buoni, sia in termini di velocità di apprendimento che di stabilità del modello. Questa conclusione è supportata anche dalle metriche di valutazione, che confermano la validità di questa configurazione. Anche in questo, tuttavia, i risultati sono considerati buoni, ma non ottimali.

4.2.4 Analisi al variare dell'Ottimizzatore

In questo paragrafo sono riportati i training al variare dell'ottimizzatore, con i seguenti iperparametri costanti: Learning rate: 10^{-4} e Batch Size: 16 Nella scelta

dell'ottimizzatore sono stati considerati SGD, RMSProp e ADAM. Il primo aggiorna i parametri basandosi su piccoli sottoinsiemi di dati (per questo è definito stocastico), mentre RMSProp adatta il *learning rate* in base alla media esponenziale dei quadrati dei gradienti recenti. ADAM combina i vantaggi di RMSProp e AdaGrad, adattando i tassi di apprendimento ai momenti del gradiente e risultando più stabile. In base a queste caratteristiche e ai risultati empirici, è stato scelto ADAM per la sua stabilità ed efficienza. In questa analisi, sono stati condotti tre training di una rete GAN, con lo scopo di valutare l'impatto della scelta dell'ottimizzatore sulle prestazioni complessive del modello. Gli esperimenti sono stati eseguiti utilizzando i seguenti ottimizzatori: **Adam**, **SGD**, e **RMSprop**.

Nel primo esperimento, è stato utilizzato l'Adam come ottimizzatore, ed i risultati ottenuti sono stati buoni. Adam si è dimostrato particolarmente efficace nel bilanciare la velocità di apprendimento e la stabilità del modello. Le immagini generate hanno mostrato una qualità elevata già dopo circa 250 epoch, con una convergenza stabile sia per il generatore che per il discriminatore.

Le curve della *Loss* e della *Validation Loss* hanno seguito un andamento regolare e conforme al comportamento previsto per una GAN ben allenata, senza oscillazioni significative. Questo comportamento ha permesso alla rete di apprendere in modo efficace e continuo, portando a una convergenza robusta e a immagini realistiche.

Come già analizzato in precedenza, le metriche di valutazione hanno confermato la qualità del training, con buoni risultati (FID: 95.52, PSNR: 71.28).

Nel secondo esperimento, è stato utilizzato l'SGD (Stochastic Gradient Descent) come ottimizzatore, ma i risultati sono stati insoddisfacenti.

Il training è stato caratterizzato da un comportamento instabile, con oscillazioni anomale nelle curve della *Loss* e della *Validation Loss* sia per il generatore che per il discriminatore. Ciò ha impedito sia al generatore che al discriminatore di migliorare in modo consistente, producendo immagini di bassa qualità e rendendo il processo di apprendimento inefficiente. Infatti, possiamo notare dalla metrica di valutazione corrispondente al FID un valore di 270.3, il più alto di tutti gli esperimenti condotti finora, classificandosi come il peggior risultato.

Nel terzo esperimento, è stato utilizzato l'RMSprop come ottimizzatore. Anche in questo caso, i risultati sono

stati scadenti. L'RMSprop ha prodotto un training estremamente rumoroso, con forti oscillazioni sia nella *Loss* che nella *Validation Loss*, specialmente nel generatore. Queste oscillazioni hanno impedito alla rete di convergere in modo stabile, compromettendo significativamente la qualità delle immagini generate, come possiamo notare anche dalle metriche di valutazione (FID: 170.9, PSNR: 68.03).

In conclusione, l'**ottimizzatore Adam** è stato identificato come il migliore, garantendo una maggiore stabilità nelle funzioni di perdita e una superiore accuratezza nelle metriche di valutazione rispetto agli altri ottimizzatori testati. Anche in questo caso, il training 1 si conferma come il migliore.

4.3 Sommario dei risultati

Nella tabella 1 riportiamo un sommario di tutti i risultati che abbiamo ottenuto dai vari esperimenti svolti. Nel seguito di questo capitolo, analizziamo nel dettaglio i valori delle metriche durante il *tuning*.

Rif	α	Opt	BS	Epoche	FID	PSNR
1	10^{-4}	Adams	16	300	95.52	71.28
2	10^{-4}	Adams	8	300	154.8	69.08
3	10^{-4}	Adams	32	300	155.2	70.14
4	10^{-4}	Adams	64	300	236.3	68.71
5	10^{-4}	RMSprop	16	300	141.9	68.03
6	10^{-4}	SGD	16	300	270	68.03
7	10^{-5}	Adams	16	400	157.0	71.26
8	10^{-3}	Adams	16	300	-	-

Table 1: Risultati ottenuti. α è learning rate, **Opt** è ottimizzatore, **BS** è batch size, **FID** è Frechet Inception Distance e **PSNR** è Peak signal-to-noise ratio.

4.4 Best Model

Dopo un'attenta analisi, il modello 1 con **learning rate** di 10^{-4} , **batch size** di **16**, e **ottimizzatore Adam** è stato identificato come il migliore per la sua efficacia complessiva. Il learning rate di 10^{-4} ha permesso una convergenza rapida e stabile, producendo immagini di buona qualità, come confermato dalle metriche FID e PSNR. Il

batch size di 16 ha bilanciato bene la frequenza degli aggiornamenti e la stabilità, evitando i problemi di rumorosità dei batch più piccoli e la lentezza di quelli più grandi. L’ottimizzatore Adam ha garantito un training stabile, con curve di *Loss* regolari e una gestione efficace del gradiente, superando alternative come SGD e RMSprop.

Questa combinazione ha garantito i migliori risultati ottenuti in termini di stabilità e qualità delle **immagini generate** come possiamo chiaramente denotare da questo set di immagini di testing 5.

Segnaliamo che l’allenamento 7 ha prodotto risultati alquanto validi, nonostante la valutazione quantitativa sia peggiore. Come precedentemente accennato, questo è probabilmente causato dall’insufficiente numero di epoche. Riportiamo anche le immagini prodotte da questa rete 6.

In conclusione, sebbene PSNR abbia sempre raggiunto valori accettabili, e la qualità percettiva dell’immagine sia ottima, segnaliamo che FID non ha mai raggiunto un valore inferiore a 50, indicando che i risultati non sono comunque considerabili ottimali. Questo è essere causato sicuramente dalle limitate risorse computazionali a disposizione. Riteniamo comunque, che le immagini ricostruite dal modello 1 siano di buona qualità.

5 Conclusioni

In questo progetto, abbiamo sviluppato una rete GAN per il task di Image Inpainting su immagini di paesaggi, permettendo al nostro gruppo di imparare a destreggiarsi in problemi propri del *Deep Learning* e delle GAN, quali la definizione dell’architettura, del tuning degli iperparametri, della scelta dell’ottimizzatore. Questo ci ha permesso di imparare come si allena nel pratico una rete neurale. La versione finale della nostra rete ha prodotto risultati buoni in termini di equilibrio tra qualità dell’immagine e carico computazionale, non raggiungendo comunque risultati eccellenti. Effettuando vari tuning degli iperparametri, abbiamo preso atto che la scelta di un learning rate moderato (10^{-4}), un batch size di 16, e l’ottimizzatore Adam, fosse la migliore. Le difficoltà principali riscontrate durante l’addestramento sono state la stabilità del training e la capacità del generatore di creare immagini realistiche. L’introduzione del Contextual Attention, ha portato a un’elevata complessità com-

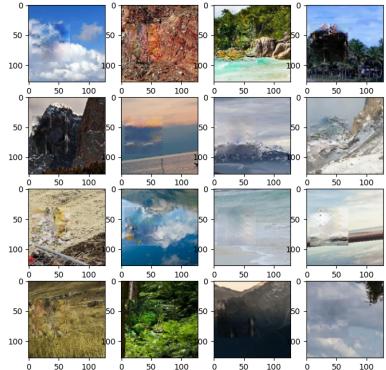
putazionale e a un peggioramento delle prestazioni, evidenziando la sfida di bilanciare complessità del modello e risorse disponibili. Per il futuro, sarebbe interessante poter realizzare questo tipo di architettura con l’uso di risorse computazionali più potenti, migliorando ulteriormente la qualità dell’inpainting.

References

- [1] Irina-Mihaela Ciortan, Sony George, and Jon Yngve Harderberg. Colour-balanced edge-guided digital inpainting: Applications on artworks. *Sensors*, 21(6), 2021.
- [2] Hanyu Xiang, Qin Zou, Muhammad Ali Nawaz, Xianfeng Huang, Fan Zhang, and Hongkai Yu. Deep learning for image inpainting: A survey. *Pattern Recognition*, 134:109046, 2023.
- [3] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. Generative image inpainting with contextual attention, 2018.

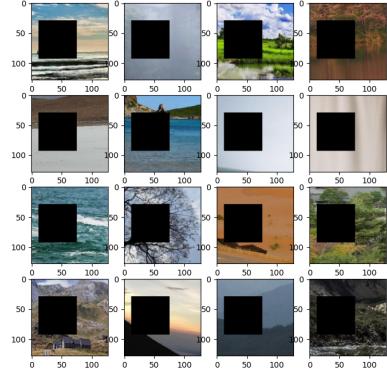


(a) Immagini di Testing prima del training

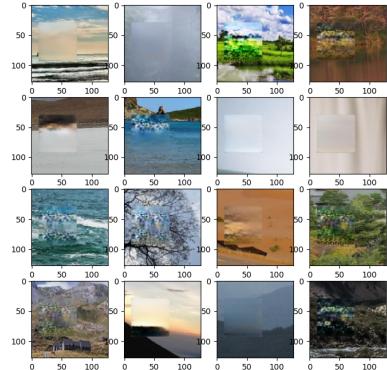


(b) Immagini di Testing dopo il training

Figure 5: $\alpha = 10^{-4}$



(a) Immagini di Testing prima del training



(b) Immagini di Testing dopo il training

Figure 6: $\alpha = 10^{-5}$