ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea in Informatica

Declouding per Immagini Satellitari: Un approccio 2D-2.5D-3D

Relatore:
Chiar.mo Prof.
Davide Evangelista

Presentata da:
Gianluca Sperti

Correlatrice:
Prof.ssa
Elena Loli Piccolomini

Sessione Ottobre 2025
Anno Accademico 2024/2025

Abstract

Questa tesi ha per obiettivo la rimozione delle nubi (*declouding*) da immagini satellitari ottiche e si propone di analizzare comparativamente architetture di deep learning 2D, 2.5D e 3D per questo compito.

Il contesto è quello del monitoraggio continuo del territorio, in cui la copertura nuvolosa interrompe le serie temporali e degrada le metriche di analisi. L'assenza di coppie perfettamente allineate "nuvoloso/pulito" rende l'addestramento supervisionato non banale: per questo prevediamo di impiegare dati sintetici con nubi generate algoritmicamente per indurre nei modelli capacità di ricostruzione coerenti sia strutturalmente sia fotometricamente.

In via previsionale, ipotizziamo che le diverse capacità spaziotemporali portino a compromessi distinti: i modelli 3D dovrebbero massimizzare la fedeltà ricostruttiva grazie alla modellazione temporale; i modelli 2.5D rappresenterebbero il miglior compromesso tra qualità e costo computazionale sfruttando brevi sequenze temporali; i modelli 2D fungerebbero da baseline leggera e robusta. Ci aspettiamo che una componente specifica per il controllo degli artefatti (es. penalizzazione degli aloni ai bordi) contribuisca a preservare le transizioni nuvolasuolo.

Il piano di valutazione prevede metriche oggettive (PSNR/SSIM) e analisi qualitative su aree di interesse per verificare coerenza radiometrica e leggibilità delle ricostruzioni. L'obiettivo applicativo è selezionare l'architettura più adatta all'impiego operativo: verosimilmente 2.5D per il monitoraggio ordinario e 3D per scenari critici che richiedono la massima fedeltà, al fine di garantire maggiore continuità temporale e affidabilità nelle mappe di frana.

Indice

Abstract **i**

Introduzione **v**

1 Metodi **1**

1.1 Reti Neurali Convoluzionali 1

1.2 Architetture U-Net 2

1.2.1 U-Net 2D 3

1.2.2 U-Net 2.5D 5

1.2.3 U-Net 3D 5

1.3 Componenti Avanzati 8

1.3.1 Convoluzioni Residuali 8

1.3.2 Attention Gates 9

1.4 Adam Optimizer 11

1.5 Funzioni di Loss 13

1.5.1 SSIM e PSNR 13

1.5.2 Anti-Halo Loss 13

1.5.3 Perceptual Loss 15

1.6 Perlin Noise 17

2 Esperimenti **19**

2.1 Dataset 19

2.1.1 Composizione e origine 19

2.1.2 Preprocessing 20

2.1.3 Split dei dati 20
2.1.4 Augmentation 20
2.1.5 Generazione del dataset 21
2.1.6 Generazione della maschera di nuvola 22
2.1.7 Finestra temporale e step per 2.5D/3D 25
2.2 Implementazione dei modelli 26
2.2.1 Modello 2D 26
2.2.2 Modello 2.5D 27
2.2.3 Modello 3D 29
2.3 Training 30
2.3.1 Schema di training multi-task 30
2.3.2 Early Stopping e Checkpointing 31
2.3.3 Configurazione Training 2D 32
2.3.4 Configurazione Training 2.5D 33
2.3.5 Configurazione Training 3D 33
2.4 Test sperimentali 35

3 Risultati numerici **37**

4 Conclusioni **49**

Ringraziamenti **57**

Introduzione

In questa tesi affronto la segmentazione delle frane da immagini satellitari, una sfida cruciale per il monitoraggio del territorio e la prevenzione dei rischi geologici. Il problema è particolarmente rilevante in regioni come l'Emilia Romagna, dove morfologia e condizioni climatiche rendono il territorio suscettibile a fenomeni franosi.

La risoluzione di questo problema dipende da molteplici fattori: la risoluzione spaziale, temporale e spettrale delle immagini satellitari, la topografia del terreno, e soprattutto la qualità e pulizia delle immagini acquisite. In particolare, le condizioni atmosferiche avverse, come la presenza di corpi nuvolosi, rappresentano uno degli ostacoli principali che compromettono significativamente le prestazioni dei modelli di segmentazione.

Le nuvole non solo oscurano porzioni del territorio, ma introducono anche variazioni radiometriche che possono confondere gli algoritmi di analisi. Questo problema è particolarmente critico quando si lavora con immagini satellitari ottiche, dove la presenza di nuvole può rendere completamente inutilizzabili ampie porzioni di territorio, compromettendo la continuità temporale necessaria per il monitoraggio delle frane.

Il mio obiettivo principale è sviluppare un algoritmo completo per la rimozione delle nuvole dalle immagini satellitari, primo tassello per una successiva fase di segmentazione delle frane nella regione dell'Emilia Romagna.

In particolare, mi propongo di:

- Sviluppare algoritmi per la generazione artificiale di nuvole realistiche per l'addestramento dei modelli.
- Esplorare approcci multi-dimensionali (2D, 2.5D e 3D) per catturare diverse prospettive del problema.
- Progettare e implementare modelli di deep learning per la rimozione delle nuvole da immagini satellitari.
- Validare i modelli su dati reali della regione Emilia Romagna.

Per affrontare il problema del declouding, in questa tesi esploro tre approcci complementari basati su reti neurali profonde. L'idea centrale è che diversi livelli di complessità spaziale possono catturare aspetti diversi del problema della rimozione delle nuvole.

Un aspetto innovativo del mio lavoro è l'uso di metodi algoritmici per generare nuvole artificiali. Questa strategia mi consente di creare dataset sintetici controllati, in cui densità, forma e distribuzione delle nuvole sono modulabili in modo preciso. Nella pratica non esistono coppie perfettamente appaiate dello stesso identico dato con e senza nuvole: per un limite fisico non è possibile acquisire la medesima scena, nello stesso istante, in entrambe le condizioni. I dati sintetici colmano questo vuoto e permettono al modello di apprendere pattern nuvolosi realistici e diversificati.

L'approccio 2D tratta le immagini come mappe piane, dove ogni pixel viene analizzato in relazione ai suoi vicini immediati. Questo approccio è particolarmente efficace per catturare pattern locali e texture delle nuvole, beneficiando della varietà di nuvole sintetiche generate algoritmicamente.

L'approccio 2.5D introduce una dimensione temporale limitata, considerando sequenze di immagini per catturare la dinamica temporale delle nuvole e migliorare la ricostruzione delle aree oscurate. La generazione artificiale permette di creare sequenze temporali coerenti con evoluzioni nuvolose realistiche.

L'approccio 3D, infine, considera l'intera sequenza temporale come un volume tridimensionale, permettendo al modello di comprendere la struttura spaziale e temporale completa delle nuvole e del territorio sottostante. Gli algoritmi di generazione artificiale permettono di creare volumi 3D con nuvole che seguono leggi fisiche realistiche.

Tutti questi approcci si basano su architetture di reti neurali profonde, in particolare varianti della U-Net e modelli basati su Transformer, che hanno dimostrato eccellenti capacità nel trattamento di immagini e nella comprensione di pattern complessi. La combinazione di dati sintetici generati algoritmicamente e dati reali permette di ottenere modelli più robusti e generalizzabili.

Il problema del declouding di immagini satellitari si inserisce nel più ampio campo del remote sensing e dell'elaborazione di immagini geospaziali.

I metodi tradizionali si basano principalmente su tecniche di interpolazione spaziale e temporale[27, 28], ma spesso falliscono nel catturare la complessità delle strutture nuvolose e delle loro interazioni con il territorio sottostante. I metodi basati su deep learning, invece, hanno mostrato risultati promettenti[29, 30], ma la maggior parte dei lavori si concentra su approcci 2D o su dataset limitati.

Il mio lavoro si distingue per l'esplorazione sistematica di approcci multidimensionali (2D, 2.5D, 3D) e per l'applicazione specifica al contesto dell'Emilia Romagna, utilizzando dati sintetici generati algoritmicamente per superare il limite fisico dell'assenza di coppie perfettamente appaiate.

Capitolo 1

Metodi

1.1 Reti Neurali Convoluzionali

Le reti neurali convoluzionali (CNN, Convolutional Neural Networks) sono un'architettura di deep learning progettata per elaborare dati che presentano una struttura spaziale, come le immagini. Introdotte inizialmente per il riconoscimento di cifre manoscritte[1] e successivamente rese popolari dall'applicazione a larga scala in ImageNet[2], le CNN hanno rivoluzionato il campo della computer vision grazie alla loro capacità di apprendere automaticamente rappresentazioni gerarchiche dei dati.

Il principio alla base di una CNN è l'utilizzo di strati convoluzionali, che applicano piccoli filtri (kernel) sull'immagine per estrarre caratteristiche locali, come bordi, texture e forme semplici. Questi vengono seguiti da funzioni di attivazione non lineari (tipicamente ReLU) e da operazioni di pooling, che riducono la dimensionalità mantenendo le informazioni essenziali. Combinando più strati, la rete costruisce progressivamente rappresentazioni sempre più astratte, fino a includere concetti complessi.[5]

Le CNN presentano due vantaggi principali: la condivisione dei pesi, che riduce il numero di parametri da apprendere rispetto a una rete completamente connessa, e l'equivarianza traslazionale, cioè la capacità di riconoscere una stessa caratteristica indipendentemente dalla sua posizione nell'immagine.

Nel contesto del declouding di immagini satellitari, queste proprietà risultano fondamentali. Le CNN sono infatti in grado di catturare automaticamente i pattern caratteristici delle nuvole, come la loro texture e distribuzione spettrale, e di differenziarli dalle strutture del territorio sottostante. Questa capacità di modellare relazioni spaziali locali è il motivo per cui le CNN costituiscono la base delle architetture encoder–decoder, come la U-Net, oggi largamente utilizzate nei problemi di segmentazione e ricostruzione di immagini in remote sensing[3].

1.2 Architetture U-Net

L'architettura U-Net, introdotta da Ronneberger et al. (2015)[4] per la segmentazione di immagini biomediche, si è rapidamente affermata come uno degli approcci più efficaci per compiti che richiedono una mappatura image-to-image. La sua popolarità deriva dalla capacità di combinare informazioni contestuali globali con dettagli spaziali locali, caratteristica fondamentale per attività come la segmentazione semantica, l'analisi di immagini mediche e più in generale in molti ambiti dell'elaborazione di immagini.

La struttura prende il nome dalla sua forma a "U", che riflette la disposizione simmetrica di due componenti principali:

- **Encoder (contracting path)**: riduce progressivamente la risoluzione spaziale tramite strati convoluzionali e operazioni di pooling. Questo processo permette di catturare rappresentazioni ad alto livello semantico, sacrificando la risoluzione in favore della comprensione del contenuto.

- **Decoder (expanding path)**: ricostruisce l'immagine alla risoluzione originale tramite operazioni di upsampling e convoluzioni. In questa fase vengono recuperati i dettagli spaziali grazie alle connessioni di salto (skip connections), che concatenano le feature map dell'encoder con quelle corrispondenti del decoder.

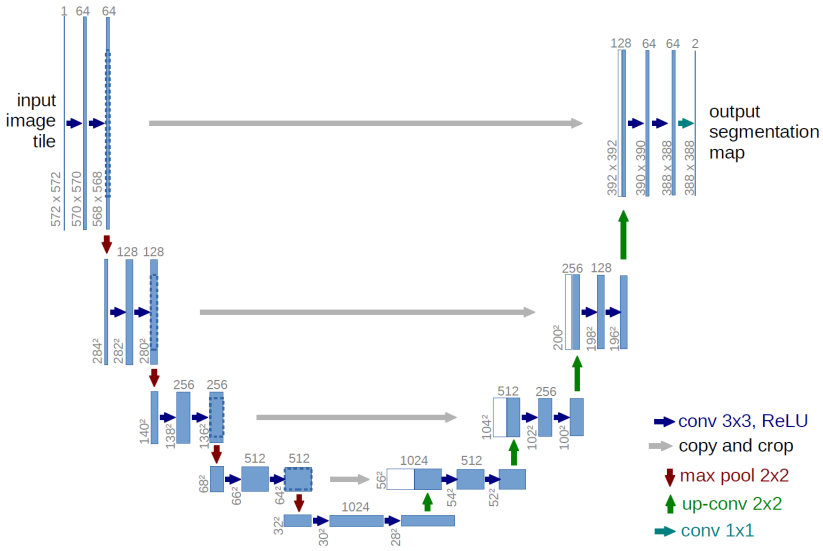


Figura 1.1: Architettura U-Net: struttura encoder-decoder con skip connections[4]

1.2.1 U-Net 2D

La versione classica della U-Net, denominata **U-Net 2D** processa le immagini come mappe piane, in cui ogni pixel viene analizzato in relazione ai suoi vicini immediati attraverso convoluzioni 2D standard. L'encoder della rete riduce progressivamente la risoluzione spaziale, aumentando al contempo la profondità semantica delle feature, mentre il decoder ricostruisce l'output alla risoluzione originale. Le *skip connections* tra layer simmetrici dell'encoder e del decoder consentono di preservare i dettagli fini, fondamentali per compiti in cui l'accuratezza a livello di singolo pixel è critica. Questa architettura è particolarmente efficace nel catturare **pattern locali e texture**, come ad esempio le forme delle nuvole, le loro bordature e le transizioni di intensità, rendendola adatta ad applicazioni di segmentazione in contesti di osservazioni satellitari e meteorologiche

Convoluzioni 2D: formulazione e esempio

Una convoluzione 2D applica un filtro (*kernel*) di dimensione $k_H \times k_W$ sulle feature di input $x \in \mathbb{R}^{H \times W \times C_{in}}$, producendo mappe di uscita $y \in \mathbb{R}^{H_{out} \times W_{out} \times C_{out}}$. Con stride s_H, s_W, padding p_H, p_W e dilatazione d_H, d_W, l'operazione su un canale di uscita c è:

$$y[i,j,c] = b_c + \sum_{\Delta h=0}^{k_H-1} \sum_{\Delta w=0}^{k_W-1} \sum_{c'=0}^{C_{in}-1} W[c,c',\Delta h,\Delta w] \cdot x\big(i' + d_H\,\Delta h,\ j' + d_W\,\Delta w,\ c'\big)$$

dove $i' = i\,s_H - p_H$ e $j' = j\,s_W - p_W$.
Le dimensioni di uscita sono determinate da[16]:

$$H_{out} = \left\lfloor \frac{H + 2p_H - d_H\,(k_H - 1) - 1}{s_H} + 1 \right\rfloor,$$
$$W_{out} = \left\lfloor \frac{W + 2p_W - d_W\,(k_W - 1) - 1}{s_W} + 1 \right\rfloor.$$

dove $\lfloor \cdot \rfloor$ indica la funzione parte intera (floor), che restituisce il massimo intero minore o uguale all'argomento; è usata perché le espressioni tra parentesi possono risultare non intere a causa di stride, padding e dilatazione.

In U-Net 2D, piccoli kernel (ad es. 3×3) con padding preservano la risoluzione, mentre gli strati di pooling/stride riducono progressivamente H, W, consentendo la cattura di contesto. Le *skip connections* ricombinano dettagli locali ad alta risoluzione con contesto profondo[4, 15].

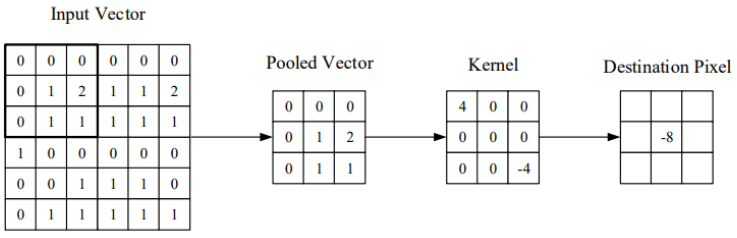


Figura 1.2: Schema di una convoluzione 2D con kernel 3×3, stride e padding[5].

1.2.2 U-Net 2.5D

La **U-Net 2.5D** rappresenta un'estensione intermedia tra l'approccio puramente bidimensionale e quello tridimensionale. In questo caso, viene introdotta una dimensione temporale limitata, in modo da fornire alla rete non solo l'informazione spaziale di una singola immagine, ma anche l'evoluzione di una sequenza di frame. L'input è costituito da **molteplici canali** (ottenuti combinando il numero di frame consecutivi, ciascuno con 3 canali RGB), che permettono al modello di catturare la **dinamica temporale** dell'input. Mantenendo la convoluzione di tipo 2D, questo approccio bilancia l'accuratezza temporale con la complessità computazionale, risultando meno oneroso rispetto a una U-Net 3D, ma più potente di una U-Net 2D tradizionale.

1.2.3 U-Net 3D

La **U-Net 3D** generalizza ulteriormente il concetto, trattando l'intera sequenza temporale come un vero e proprio **volume tridimensionale**, in cui le dimensioni spaziali (altezza e larghezza) sono affiancate dalla dimensione temporale. In questo modo, il modello è in grado di apprendere contemporaneamente la struttura spaziale delle nuvole e la loro evoluzione temporale. Le convoluzioni utilizzate in questa architettura sono di tipo **3D**, operanti su blocchi volumetrici di dati, e consentono di catturare pattern complessi che si estendono nel tempo. Un aspetto critico della U-Net 3D è il costo computazionale, che cresce sensibilmente rispetto alle varianti 2D e 2.5D.

Convoluzioni 3D: formulazione e esempio

La convoluzione 3D estende il filtro a tre dimensioni (k_T, k_H, k_W) su input volumetrici/temporali $x \in \mathbb{R}^{T \times H \times W \times C_{in}}$, producendo $y \in \mathbb{R}^{T_{out} \times H_{out} \times W_{out} \times C_{out}}$.

Con stride s_T, s_H, s_W, padding p_T, p_H, p_W e dilatazione d_T, d_H, d_W:

$$y[t,i,j,c] = b_c + \sum_{\Delta t=0}^{k_T-1}\sum_{\Delta h=0}^{k_H-1}\sum_{\Delta w=0}^{k_W-1}\sum_{c'=0}^{C_{in}-1} W[c,c',\Delta t,\Delta h,\Delta w] \cdot x\big(t'+d_T\,\Delta t,\ i'+d_H\,\Delta h,\ j'+d_W\,\Delta w,\ c'\big)$$

con $t' = t\,s_T - p_T$, $i' = i\,s_H - p_H$, $j' = j\,s_W - p_W$.

Le dimensioni di uscita seguono l'analogo 3D della formula 2D[16].

In U-Net 3D, questo permette di modellare congiuntamente dinamica temporale e struttura spaziale, mostrando vantaggi in compiti volumetrici e video[17, 18].

Dimensioni di uscita. Per ciascuna dimensione si ha:

$$T_{\text{out}} = \left\lfloor \frac{T + 2p_T - d_T\,(k_T-1) - 1}{s_T} + 1 \right\rfloor$$

$$H_{\text{out}} = \left\lfloor \frac{H + 2p_H - d_H\,(k_H-1) - 1}{s_H} + 1 \right\rfloor$$

$$W_{\text{out}} = \left\lfloor \frac{W + 2p_W - d_W\,(k_W-1) - 1}{s_W} + 1 \right\rfloor$$

Numero di parametri (con bias).

$$\text{N\_parametri} = k_T\,k_H\,k_W\,C_{in}\,C_{out} \;+\; C_{out}.$$

Costo computazionale

Indicando con N_MACs il numero di moltiplicazioni-accumuli:

$$\text{N\_MACs} = T_{out}\,H_{out}\,W_{out}\,C_{out}\,\big(k_T\,k_H\,k_W\,C_{in}\big), \quad \text{FLOPs} \approx 2 \cdot \#\,\text{MACs}.$$

Campo recettivo effettivo (RF)

Per una pila di strati 3D, lungo ciascuna dimensione $d \in \{T, H, W\}$ con stride $s_d^{(\ell)}$, dilatazione $d_d^{(\ell)}$ e kernel $k_d^{(\ell)}$:

$$R_d^{(0)} = 1$$

$$J_d^{(0)} = 1$$

$$R_d^{(\ell)} = R_d^{(\ell-1)} + \big(k_d^{(\ell)} - 1\big)\,d_d^{(\ell)}\,J_d^{(\ell-1)}$$

$$J_d^{(\ell)} = J_d^{(\ell-1)}\,s_d^{(\ell)}$$

Dove $R_d^{(\ell)}$ indica quanto è ampia la porzione di input (lungo la dimensione d) che influenza un singolo neurone allo strato ℓ (campo recettivo), mentre $J_d^{(\ell)}$ è il *passo effettivo* (o salto) sull'input tra due neuroni adiacenti allo strato ℓ. Stride maggiori fanno crescere J_d, e ogni nuovo strato aumenta R_d di una quantità proporzionale a $(k_d - 1)$ e al passo accumulato fino allo strato precedente.

Confronto 2.5D vs 3D (parametri per strato)

Se in 2.5D si impilano F frame come canali d'ingresso, allora:

$$\text{2.5D: N\_parametri} = k_H\, k_W\, (F\, C_{in})\, C_{out} + C_{out},$$
$$\text{3D: N\_parametri} = k_T\, k_H\, k_W\, C_{in}\, C_{out} + C_{out},$$

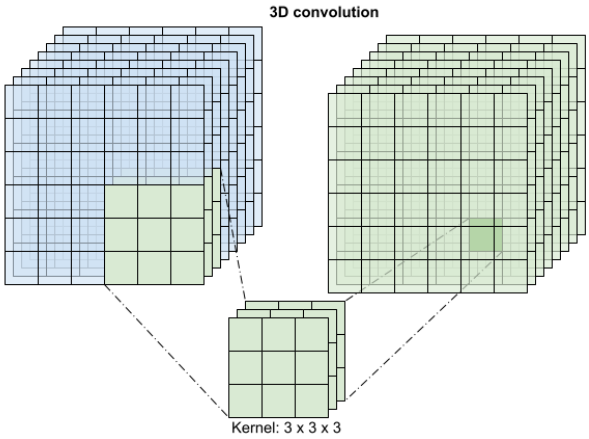


Figura 1.3: Schema di una convoluzione 3D su sequenze/volumi con kernel $(3, 3, 3)$. [24]

1.3 Componenti Avanzati

1.3.1 Convoluzioni Residuali

Le **convoluzioni residuali**, introdotte da He et al. (2016)[6] con l'architettura *ResNet* (*Residual Network*), hanno rappresentato una svolta nella progettazione di reti neurali profonde. Uno dei problemi principali nell'addestramento di architetture molto profonde è il *vanishing gradient*, fenomeno per cui il gradiente si attenua progressivamente durante la retropropagazione, ostacolando l'aggiornamento dei pesi negli strati più lontani dall'output. L'idea alla base dei blocchi residuali è di introdurre delle **connessioni di salto** (*skip connections*) che permettono al segnale di flusso sia in avanti che all'indietro di bypassare uno o più strati convoluzionali. In questo modo, invece di apprendere una mappatura diretta $H(x)$ tra input ed output, la rete apprende una funzione residua $F(x) = H(x) - x$, cosicché l'output del blocco diventa:

$$H(x) = F(x) + x$$

dove x rappresenta l'input e $F(x)$ la trasformazione non lineare appresa dal blocco convoluzionale.

Questa formulazione ha due vantaggi fondamentali:

- **Facilitazione dell'apprendimento**: è più semplice per la rete apprendere una correzione residua rispetto a dover apprendere una trasformazione completa.
- **Stabilità e profondità**: le skip connections mitigano i problemi di vanishing e exploding gradients, consentendo di addestrare reti con centinaia o migliaia di strati senza degrado significativo delle prestazioni.

Dal punto di vista pratico, le convoluzioni residuali hanno aperto la strada a un nuovo paradigma di architetture profonde, influenzando numerose varianti successive, tra cui DenseNet[21], ResNeXt[22] e UNet++[23] nel dominio della segmentazione. La loro efficacia si riflette sia nella rapidità di

convergenza in fase di training, sia nelle prestazioni superiori su benchmark complessi di classificazione e segmentazione.

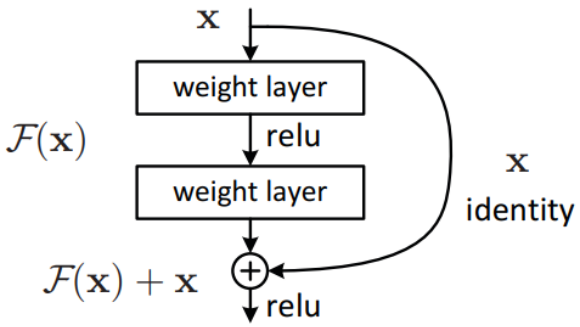


Figura 1.4: Schema di un blocco residuale (ResNet) con skip connection.[6]

1.3.2 Attention Gates

Gli **Attention Gates** (AGs)[7] sono un meccanismo di attenzione spaziale introdotto per potenziare le prestazioni delle architetture di tipo U-Net in compiti di segmentazione. A differenza dei meccanismi di attenzione globale dei *Transformer*, che modellano relazioni a lungo raggio tra tutte le posizioni di un input, gli Attention Gates sono progettati per operare in maniera locale sulle **connessioni di salto** (*skip connections*) tra encoder e decoder. Il principio alla base è quello di filtrare dinamicamente le feature provenienti dall'encoder, pesando solo le regioni rilevanti per il compito di segmentazione. In particolare, viene utilizzato un **gating signal** proveniente dal decoder, che contiene informazioni contestuali di alto livello, per generare una **maschera di attenzione**. Questa maschera modula le feature dell'encoder, sopprimendo quelle irrilevanti o rumorose e potenziando le aree più informative.

Formalmente, l'operazione di attenzione può essere descritta come:

$$\alpha(x) = \sigma\left(W_x x + W_g g + b\right)$$

dove x rappresenta le feature provenienti dall'encoder, g il gating signal dal decoder, W_x e W_g sono matrici di proiezione apprese, b è il bias e σ una

funzione sigmoide che restituisce pesi normalizzati compresi tra 0 e 1. Le feature modulate risultano quindi:

$$\tilde{x} = \alpha(x) \cdot x$$

così da trasmettere al decoder solo l'informazione ritenuta rilevante.

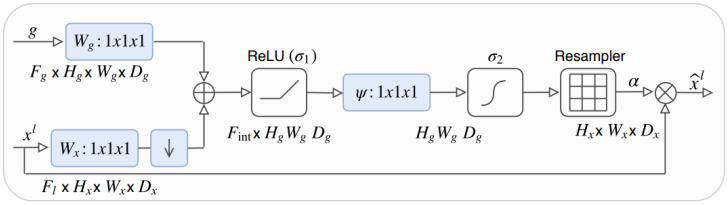


Figura 1.5: Schema dell'Attention Gate (AG)[7]. Le feature di input x_l sono scalate con coefficienti di attenzione α calcolati in AG. Le regioni spaziali sono selezionate analizzando sia le attivazioni sia l'informazione contestuale fornita dal segnale di gating g raccolto a una scala più grossolana. Il ricampionamento a griglia dei coefficienti di attenzione è eseguito tramite interpolazione trilineare.

I principali vantaggi derivanti dall'uso degli Attention Gates sono:

- **Riduzione del rumore**: eliminano l'influenza di regioni non pertinenti, migliorando la qualità della segmentazione.
- **Focalizzazione semantica**: favoriscono l'attenzione su strutture di interesse (ad esempio pattern nuvolosi in immagini satellitari).
- **Efficienza computazionale**: operando solo sulle skip connections, introducono un sovraccarico computazionale ridotto rispetto ai meccanismi di attenzione globale.

1.4 Adam Optimizer

Adam (Adaptive Moment Estimation) è un algoritmo di ottimizzazione stocastica che combina i vantaggi di AdaGrad[12] e RMSProp[13], utilizzando momenti del primo e secondo ordine per adattare dinamicamente il learning rate per ogni parametro[11]. Questo approccio adattivo lo rende particolarmente efficace per l'addestramento di reti neurali profonde su dataset complessi come le immagini satellitari.

Fondamenti Matematici

Adam mantiene due momenti esponenziali mobili dei gradienti:

- **Momento del primo ordine** (media dei gradienti): $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$.
- **Momento del secondo ordine** (media non centrata delle varianze): $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$.

dove g_t rappresenta il gradiente al tempo t, mentre β_1 e β_2 sono i tassi di decadimento esponenziale (tipicamente 0.9 e 0.999 rispettivamente).

Correzione del Bias

Per compensare l'inizializzazione a zero dei momenti, Adam applica una correzione del bias:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{1.1}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{1.2}$$

Regola di Aggiornamento

La regola di aggiornamento dei parametri è:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{1.3}$$

dove α è il learning rate iniziale e ϵ è un termine di regolarizzazione (tipicamente 10^{-8}) per evitare divisioni per zero.

Vantaggi nell'Applicazione

Nel contesto del de-clouding di immagini satellitari, Adam offre diversi vantaggi:

- **Adattività**: Il learning rate si adatta automaticamente per ogni parametro, cruciale per gestire le diverse scale delle caratteristiche nelle immagini satellitari.

- **Robustezza**: La combinazione di momenti del primo e secondo ordine fornisce stabilità durante l'addestramento.

- **Efficienza computazionale**: Richiede solo O(1) spazio aggiuntivo per parametro, rendendolo scalabile per reti profonde.

- **Convergenza rapida**: Particolarmente efficace nei primi stadi dell'addestramento grazie al bias correction.

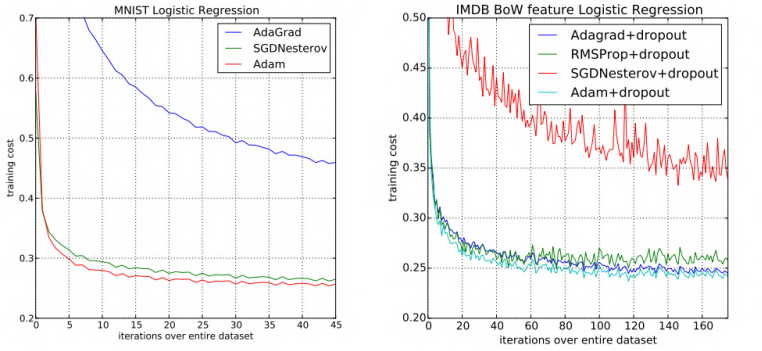


Figura 1.6: La Figura mostra l'andamento della funzione di costo durante l'addestramento di una regressione logistica su due dataset diversi, confrontando Adam con altri ottimizzatori. Si osserva come Adam permetta una convergenza più rapida e stabile rispetto a SGD o AdaGrad, confermando la sua efficacia nella stima adattiva dei momenti dei gradienti[11]

1.5 Funzioni di Loss

1.5.1 SSIM e PSNR

L'**SSIM (Structural Similarity Index)** è una metrica progettata per misurare la similarità percepita tra due immagini, andando oltre il semplice confronto pixel-wise. Essa valuta tre componenti principali: *luminanza*, *contrasto* e *struttura* locale, combinandole in un indice compreso tra 0 e 1, dove 1 indica identità perfetta tra le immagini[9]. Questa metrica è particolarmente utile in applicazioni di *image reconstruction* e *video processing*, poiché è sensibile alle differenze strutturali che influenzano la percezione visiva umana, pur ignorando piccole variazioni locali irrilevanti.

Il **PSNR (Peak Signal-to-Noise Ratio)**, invece, quantifica la qualità di un'immagine ricostruita in termini di rapporto segnale-rumore. Esso confronta l'intensità massima possibile dei pixel con l'errore quadratico medio (MSE) tra l'immagine target e quella generata, fornendo un valore in decibel. Un PSNR più alto indica minori distorsioni e maggiore fedeltà all'immagine originale[10].

1.5.2 Anti-Halo Loss

L'**Anti-Halo Loss** è una funzione di loss sviluppata specificamente per ridurre la formazione di aloni nelle zone di transizione tra nuvole e terreno. Questo fenomeno, noto come "halo effect" o "ghosting"[31, 32], rappresenta uno dei principali problemi nell'elaborazione di immagini satellitari, dove le zone di confine tra nuvole dense e terreno pulito mostrano artefatti di transizione indesiderati[33].

Problema degli Aloni

Gli aloni si manifestano quando il modello di de-clouding produce regioni significativamente più scure rispetto al terreno originale nelle zone adiacenti alle nuvole dense.

Questo effetto è particolarmente problematico perché:

- **Distorsione percettiva**: Crea transizioni artificiali che non esistono nella realtà.
- **Perdita di informazioni**: Maschera dettagli importanti del terreno sottostante.
- **Inconsistenza spaziale**: Genera discontinuità visive nelle immagini ricostruite.

Formulazione Matematica

L'Anti-Halo Loss utilizza un approccio mask-guided che identifica selettivamente le zone problematiche. Sia M la maschera delle nuvole, I_{pred} l'immagine predetta e I_{target} l'immagine target, la loss è definita come:

$$\mathcal{L}_{halo} = \lambda_{halo} \cdot \frac{1}{N} \sum_{i=1}^{N} \max(0, I_{target}^{(i)} - I_{pred}^{(i)}) \cdot M_{dense}^{(i)} \cdot \mathbf{1}_{significant}^{(i)} \tag{1.4}$$

dove:

- $M_{dense}^{(i)} = \mathbf{1}_{M^{(i)}>0.8}$ identifica le nuvole molto dense.
- $\mathbf{1}_{significant}^{(i)} = \mathbf{1}_{(I_{target}^{(i)} - I_{pred}^{(i)})>0.1}$ seleziona solo le differenze significative.
- λ_{halo} è il peso della componente anti-halo (tipicamente 0.03).

Strategia di Applicazione

La loss viene applicata in modo selettivo:

1. **Identificazione delle nuvole dense**: Solo le regioni con densità nuvolosa > 0.8 vengono considerate.
2. **Filtro delle differenze significative**: Solo le differenze > 0.1 tra target e predizione attivano la penalità.
3. **Penalizzazione mirata**: La penalità è proporzionale alla differenza di intensità, concentrandosi sui casi più problematici.

Vantaggi nell'Applicazione

Uno dei principali vantaggi di questa implementazione risiede nella sua selettività, poiché la loss non va a interferire con le regioni dell'immagine dove non si riscontrano effettivamente problemi di aloni. Di conseguenza si ottiene anche una maggiore stabilità, grazie al filtro che esclude le differenze poco significative ed evita così penalizzazioni eccessive o indesiderate. Dal punto di vista pratico, la funzione risulta efficace nel ridurre gli artefatti di transizione generati dal modello, contribuendo a mantenere la qualità generale delle immagini ricostruite. Inoltre, l'introduzione esplicita del parametro λ_{halo} rende facilmente controllabile l'intensità della penalizzazione anti-halo, permettendo di bilanciare opportunamente questa componente rispetto alle altre funzioni di perdita presenti nell'addestramento.

1.5.3 Perceptual Loss

Nel sistema implementato, l'utilizzo di modelli pre-addestrati avviene attraverso l'implementazione di una **Perceptual Loss** basata su VGG19, che sfrutta le rappresentazioni apprese su ImageNet per migliorare la qualità percettiva delle immagini ricostruite[8].

VGGPerceptualLoss: Implementazione

La classe `VGGPerceptualLoss` implementa una loss percettiva che utilizza le feature estratte da VGG19 pre-addestrato su ImageNet:

```
class VGGPerceptualLoss(nn.Module):
    def __init__(self, layers=(2, 7, 12, 21), weight=1.0):
        super().__init__()
        vgg = models.vgg19(weights=models.VGG19_Weights.IMAGENET1K_V1).features.eval()
        for p in vgg.parameters():
            p.requires_grad = False
        self.vgg = vgg
        self.layers = layers
        self.weight = weight
```

Feature Extraction Congelata

L'utilizzo del modello VGG19 come puro estrattore di feature, con i parametri resi non aggiornabili (`requires_grad = False`), garantisce che le rappresentazioni pre-addestrate restino stabili durante tutta la fase di addestramento. In questo modo, si evita ogni aggiornamento dei parametri interni alla rete VGG, con il vantaggio di ridurre il carico computazionale e migliorare l'efficienza complessiva. Inoltre, l'impiego di feature già testate e validate su ampi dataset come ImageNet assicura maggiore affidabilità e solidità alle caratteristiche estratte, rendendo questa soluzione particolarmente adatta per applicazioni complesse come la ricostruzione di immagini satellitari.

Integrazione nella Loss Totale

La Perceptual Loss viene integrata nella funzione di loss complessiva del sistema:

$$\mathcal{L}_{RGB} = (1 - w_{ssim}) \cdot \mathcal{L}_{L1} + w_{ssim} \cdot \mathcal{L}_{SSIM} + \mathcal{L}_{perceptual}$$

La componente percettiva viene bilanciata con le metriche tradizionali (L1 e SSIM) attraverso un peso configurabile che permette di modulare l'influenza delle feature semantiche.

Vantaggi per il De-clouding

La Perceptual Loss aiuta il modello a generare immagini che sono percettivamente simili al target, andando oltre la semplice similarità pixel-wise[14]. Le feature VGG catturano strutture semantiche importanti che devono essere preservate durante la ricostruzione, come bordi di edifici, strade e altre caratteristiche geografiche. Le rappresentazioni pre-addestrate su ImageNet forniscono robustezza alle variazioni di illuminazione, colore e texture tipiche delle immagini satellitari.

1.6 Perlin Noise

Il **Perlin Noise**, introdotto da Ken Perlin nel 1985[25], rappresenta uno dei metodi procedurali più diffusi per la generazione di pattern pseudo-casuali con caratteristiche naturali e organiche. A differenza del rumore puramente casuale, il Perlin Noise è una forma di *gradient noise*, in cui a ciascun nodo di una griglia regolare viene associato un vettore gradiente, successivamente interpolato attraverso funzioni di smoothing (tipicamente polinomi di tipo *fade curve*) per garantire continuità e transizioni fluide tra le celle. Questa costruzione produce un rumore coerente, privo delle discontinuità tipiche del *value noise*, e permette di ottenere strutture visivamente plausibili in ambiti grafici e di simulazione. La somma di più ottave, ottenuta scalando progressivamente frequenza e ampiezza del segnale, consente di generare dettagli a diverse scale, riproducendo l'apparenza gerarchica tipica di fenomeni naturali quali nuvole, montagne o superfici oceaniche. Successivi miglioramenti all'algoritmo originale[26] hanno risolto alcune problematiche legate all'allineamento assiale dei gradienti e alla regolarità delle derivate, fornendo una versione più robusta e adatta a implementazioni efficienti anche su architetture parallele. Grazie alla sua flessibilità, il Perlin Noise è ampiamente utilizzato non solo nella computer grafica e nel procedural modeling, ma anche in applicazioni di simulazione scientifica, ad esempio per la generazione sintetica di texture nuvolose in contesti di osservazione terrestre.

Costruzione matematica

Si consideri una griglia regolare dei nodi (*lattice*). In 2D, dato un punto continuo (x, y), si individuano gli indici di cella $i = \lfloor x \rfloor$, $j = \lfloor y \rfloor$ e le coordinate locali $t_x = x - i$, $t_y = y - j$. A ciascun vertice della cella $(i + \delta_x, j + \delta_y)$, con $\delta_x, \delta_y \in \{0, 1\}$, è associato un vettore gradiente unitario $\mathbf{g}_{\delta_x \delta_y}$, scelto attraverso una funzione di *hash* (tipicamente una tabella di permutazione) per garantire pseudocasualità ma riproducibilità.

Ottave, fBm e turbulence

Un'*ottava* è un singolo livello di rumore Perlin valutato a una certa **scala** s_k (frequenza) e pesato da una **ampiezza** w_k. La **lacunarity** $\lambda > 1$ controlla la crescita di scala tra ottave successive ($s_{k+1} = \lambda\, s_k$), mentre il **gain** $g \in (0, 1)$ controlla la decrescita dell'ampiezza ($w_{k+1} = g\, w_k$). Intuitivamente, ottave più alte aggiungono dettagli più fini con contributo via via minore. Strutture multiscala si ottengono sommando più ottave del segnale con frequenza crescente e ampiezza decrescente. Indichiamo con s_k le scale (tipicamente potenze di due) e con w_k i pesi decrescenti tra le ottave:

$$f_{\text{fBm}}(\mathbf{x}) = \sum_{k=0}^{K-1} w_k \ \text{Perlin}\left(s_k\, \mathbf{x}\right), \qquad f_{\text{turb}}(\mathbf{x}) = \sum_{k=0}^{K-1} w_k \ \left|\, \text{Perlin}\left(s_k\, \mathbf{x}\right)\right|.$$

Nella pratica di questa tesi si adottano pesi $w_k \propto s_k^{\text{decay\_factor}}$ e si normalizza la somma multi-scala per poi applicare una compressione non lineare con tanh, in accordo con la pipeline di generazione della maschera. I dettagli operativi sono approfonditi nella Sezione 2.1.6.

Figura 1.7: Esempi di Perlin Noise a diverse scale, tratti da[25].

Capitolo 2

Esperimenti

2.1 Dataset

2.1.1 Composizione e origine

I dati sono stati forniti dal Dipartimento di Geologia dell'Università di Bologna e consistono in quattro set di immagini **GeoTIFF** derivati da **Sentinel-2** (post-processing a 2 m/px) relativi ai comuni dell'Emilia Romagna: **Predappio**, **Casola Valsenio**, **Brisighella** e **Modigliana**. Le immagini sono a 4 bande in formato `Float32` e non recano una color interpretation RGB esplicita.

- **CRS**: RDN2008 / UTM zone 32N (EPSG:7791).
- **Risoluzione spaziale**: 2 m/pixel (`Pixel Size` = 2.0 m).
- **Formato e interleave**: `GTiff`, interleave per pixel.
- **Bande**: 4 bande `Float32` (ColorInterp non definita).

A titolo esemplificativo, per il file `Sentinel2_post_2m.tif`: `Size` = 4936×7392; `Origin` = (731220.025, 4893966.875); `Corner Coordinates` (long/lat): *UL* (11°53'30.59"E, 44°09'44.29"N), *LR* (12°00'30.17"E, 44°01'34.29"N).

2.1.2 Preprocessing

La pipeline considera il formato `Float32` a 4 bande e la possibile assenza di un valore `NoData` esplicito. Per l'addestramento 2D si utilizzano le prime 3 bande (`read_tif_rgb`): ciascuna banda viene **clippata al 99° percentile** e **normalizzata individualmente** in $[0, 1]$. Le immagini vengono organizzate per Comune e suddivise in patch a risoluzione nativa, preservando il CRS (nessuna riproiezione).

2.1.3 Split dei dati

La suddivisione avviene **per comune**: un comune è riservato al **test** (di default l'ultimo in ordine alfabetico, oppure selezionato esplicitamente), mentre i restanti comuni costituiscono il set di **train/validation**. Per ciascun comune di train, dopo l'estrazione delle patch si riserva il **10%** finale alle **validazioni**.

2.1.4 Augmentation

Le trasformazioni sono applicate **solo** al set di training, in modo **sincronizzato** su input *cloudy*, target *clean* e maschera.

- **Flip orizzontale**: probabilità 0.5.
- **Flip verticale**: probabilità 0.5.
- **Rotazione**: angolo uniforme in $[-15°, +15°]$ (bilineare per immagini, nearest per maschere).

Le immagini sono infine convertite in tensori `Float32` $[0, 1]$. Eventuale **resize** è applicato nel *DataLoader* alla dimensione `patch_size`.

2.1.5 Generazione del dataset

A partire dai GeoTIFF per comune, si estrae un set di patch **clean** e si genera per ciascuna una controparte **cloudy** sintetica e la relativa **mask** di nuvola, salvando i tre insiemi in directory parallele.

- **Estrazione patch**: `patch_size` $= 320$, `stride` $= 160 \Rightarrow$ overlap 50%.
- **Filtro patch nere**: si scartano patch con frazione di pixel non-neri ≤ 0.2 (soglia su media canali ≥ 0.05).
- **Nuvole sintetiche**: Perlin noise scalato, mix additivo; tentativi multipli (≤ 10) finché la coverage è sotto soglia (approfondito nella sezione "Generazione della maschera di nuvola").
- **Coverage**: max 0.6 in train, max 0.5 in val/test.
- **Parametri random**: `cloud_opacity_range` $= [0.5, 1.0]$, `cloud_scale_range` $= [1.5, 3.0]$.
- **Struttura cartelle**: `train/clean|cloudy|mask`, `val/clean|cloudy|mask`, `test/clean|cloudy|mask`.
- **Formato output**: PNG 8-bit in $[0, 1]$ per *clean*/*cloudy*, maschere in scala di grigi.

Brisighella Casola Valsenio Modigliana Predappio

Figura 2.1: Esempi di patch *clean* estratte da diversi comuni.

2.1.6 Generazione della maschera di nuvola

La **cloud mask** è ottenuta tramite *Perlin noise* multi-scala con normalizzazione non lineare, così da riprodurre i pattern gerarchici tipici delle nuvole (strutture grossolane e dettagli fini) e ottenere contrasti fotorealistici.

Rumore di Perlin (singola ottava)

Si dispone una griglia di vettori gradiente casuali $\{\mathbf{g}_{i,j}\}$. Per un punto locale $(x, y) \in [0, 1]^2$ nella cella (i, j), si usa la *fade function*

$$s(t) = 3t^2 - 2t^3, \quad w_x = 1 - s(x), \quad w_y = 1 - s(y).$$

Indicando con

$$\begin{aligned} n_{00} &= \langle \mathbf{g}_{i,j}, (x, y) \rangle, & n_{10} &= \langle \mathbf{g}_{i+1,j}, (x-1, y) \rangle, \\ n_{01} &= \langle \mathbf{g}_{i,j+1}, (x, y-1) \rangle, & n_{11} &= \langle \mathbf{g}_{i+1,j+1}, (x-1, y-1) \rangle, \end{aligned}$$

il valore di Perlin nella cella è la combinazione bilineare pesata

$$P(x, y) = w_x w_y \, n_{00} + (1 - w_x) w_y \, n_{10} + w_x (1 - w_y) \, n_{01} + (1 - w_x)(1 - w_y) \, n_{11}.$$

Sintesi multi-scala

Per riprodurre le strutture a più scale si sommano K ottave con pesi decrescenti:

$$M_{\text{raw}}(u, v) = \sum_{k=1}^{K} w_k \, P_k\big(s_k \, u, \; s_k \, v\big),$$

dove s_k sono le scale (tipicamente potenze di due, eventualmente riscalate per adattarsi a (H, W)) e $w_k \propto s_k^{\text{decay\_factor}}$ controlla l'energia per ottava. L'immagine viene eventualmente estesa alla potenza di due successiva per efficienza e quindi ritagliata a (H, W).

Normalizzazione e compressione non lineare

La somma multi-scala viene normalizzata e compressa per ottenere contrasti realistici:

$$\widehat{M}(u, v) = \tanh\Big(4 \, \frac{M_{\text{raw}}(u, v)}{\max_{u,v} |M_{\text{raw}}(u, v)|}\Big).$$

Questa trasformazione limita gli outlier e accentua le differenze nella zona centrale, rendendo la maschera più coerente con la densità nuvolosa.

Parametri principali

- **opacity** $\in [0, 1]$: fattore moltiplicativo che scala l'intensità finale della maschera. All'aumentare di `opacity` cresce la *densità media* delle nuvole e l'oscuramento dei pixel sottostanti.

- **scale/scales**: controllano la scala spaziale del Perlin; valori maggiori producono pattern a *grana più larga*, valori minori pattern più dettagliati. L'uso multi-scala combina più s_k con pesi w_k.

- **decay_factor**: regola il decadimento dei pesi tra ottave (enfatizza strutture grossolane o fini).

- **coverage_max**: soglia superiore per la media della mask. Se la coverage $c = \frac{1}{HW} \sum_{i,j} \widehat{M}_{ij}$ supera la soglia (es. 0.6 in training, 0.5 in val/test), si rigenera la maschera (fino a 10 tentativi).

Applicazione al frame RGB

Di seguito formalizziamo il procedimento di fusione (*mixing*) nella notazione impiegata in `Cloud_mix.py`

1. **Replicazione maschera sui canali.** Dati $I \in \mathbb{R}^{B\times C\times H\times W}$ (immagine *clean*), $\widehat{M} \in \mathbb{R}^{B\times H\times W}$ (maschera) e opzionale $\mathbf{m} \in \mathbb{R}^{C}$:

$$\mathcal{C} = \text{replicate}_C(\widehat{M}) \in \mathbb{R}^{B\times C\times H\times W}.$$

2. **Ombre.** Se `shadow` è fornita, si applica prima un'attenuazione moltiplicativa senza blur e senza colore nuvola:

$$I \leftarrow I \cdot \big(1 - \mathcal{S}_{\text{clip}}\big), \qquad \mathcal{S}_{\text{clip}} = \min(\max(\text{replicate}_C(\text{shadow}), 0), 1).$$

3. **Blur dipendente dalla densità.** Si calcola un modulatore e il σ per immagine:

$$\overline{M}_b(u,v) = \tfrac{1}{C}\sum_{c=1}^{C} \mathcal{C}_{b,c}(u,v), \qquad \sigma_b = \alpha \cdot \tfrac{1}{HW}\sum_{u,v} \overline{M}_b(u,v),$$

con $\alpha =$`blur_scaling`. Quindi $I_b^{(\text{blur})} = G_{\sigma_b} * I_b$ (kernel gaussiano con riflessione ai bordi; dimensione $\approx 2\lceil 3\sigma_b \rceil + 1$).

4. **Base cromatica della nuvola.** Se `cloud_color=true`:

$$C_{\text{cloud}}(b,c,u,v) = \tfrac{1}{HW}\sum_{x,y} I(b,c,x,y) \quad \text{(replicato su } (u,v)\text{)};$$

altrimenti $C_{\text{cloud}} = \mathbf{1}$. Se $\mathbf{m}$ è fornito: $C_{\text{cloud}} \leftarrow m_c\, C_{\text{cloud}}$.

5. **Composizione finale.** Sia $\mathcal{C}_{\text{clip}} = \min(\max(\mathcal{C}, 0), 1)$ e $L = \max(\|\mathcal{C}\|_\infty, 1)$. Allora

$$I_{\text{out}} = \begin{cases} I^{(\text{blur})} \cdot (1 - \mathcal{C}_{\text{clip}}), & \text{se } \texttt{invert} = \texttt{true}, \\ I^{(\text{blur})} \cdot \left(1 - \frac{\mathcal{C}}{L}\right) + L\, C_{\text{cloud}} \cdot \left(\frac{\mathcal{C}}{L}\right), & \text{altrimenti.} \end{cases}$$

Il primo termine preserva la scena nelle regioni a bassa densità di nuvola; il secondo inietta la componente nuvola (colore o bianco) proporzionalmente alla maschera. La normalizzazione via L evita sovrasaturazioni quando la maschera supera l'unità.

Clean (Predappio)

Cloudy (Predappio)

Figura 2.2: Esempio di patch *Clean* e *Cloudy* dal comune di Predappio.

2.1.7 Finestra temporale e step per 2.5D/3D

Per i modelli 2.5D e 3D si utilizzano sequenze temporali di patch. Dato un volume $X \in \mathbb{R}^{T\times C\times H\times W}$ e uno **step temporale** s_T, si generano finestre $X_{t:t+K-1}$ con lunghezza K e passi s_T:

$$t = 0, s_T, 2s_T, \ldots, T - K$$

- **2.5D**: le K immagini sono concatenate come canali ($C' = 3K$), convoluzione 2D.

- **3D**: le K immagini sono trattate come dimensione temporale esplicita, convoluzione 3D.

- **stride spaziale**: invariato rispetto al caso 2D (`patch_size` $= 320$, `stride` $=$ 160).

Per simulare il **moto temporale** delle nuvole, tra frame consecutivi la maschera viene traslata di uno *step* e una direzione fissi per sequenza. Campioniamo uno step per frame $s \in [8, 16]$ e una direzione discreta $\mathbf{d} = (d_x, d_y) \in \{(-1, 0), (1, 0), (0, 1), (0, -1), (-1, 1), (1, 1), (-1, -1), (1, -1)\}$. Per il frame i gli offset del crop sono:

$$x_i = x_0 + i\, s\, d_x, \quad y_i = y_0 + i\, s\, d_y,$$

con (x_0, y_0) scelto per mantenere il crop all'interno della maschera Perlin ingrandita; se si uscisse dai limiti si riutilizza l'ultimo frame valido. L'opacità della nuvola varia debolmente nel tempo con un fattore sinusoidale per simulare cambi di densità.

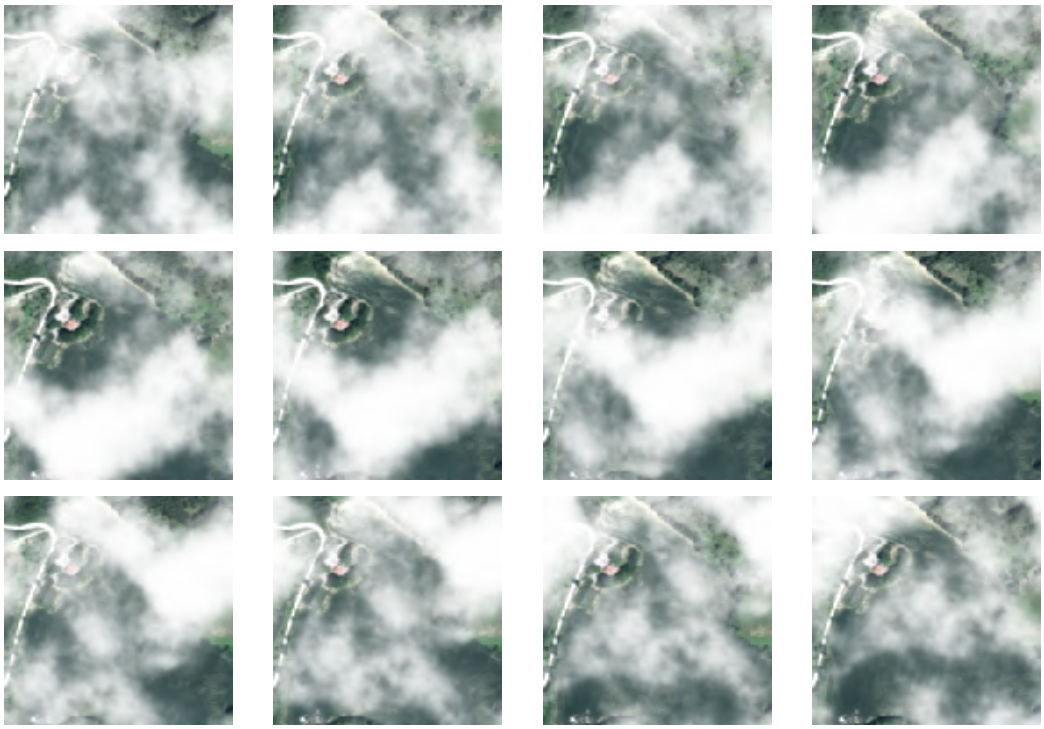


Figura 2.3: Sequenza di 12 frame (Predappio) usati per i modelli 2.5D/3D.

2.2 Implementazione dei modelli

2.2.1 Modello 2D

L'implementazione adotta una U-Net migliorata con blocchi residuali e *Attention Gates*[4, 6, 7] in *impostazione multi-task* (ricostruzione RGB pulita + stima della maschera).

Definizione architetturale

Backbone: `UNetImproved(in_channels=3, out_channels=3, base=64)`.

- **Encoder (4 livelli):** ciascun livello è un `ResidualConv` con due `Conv2d` 3×3, `BatchNorm2d` e `ReLU`, più residuo 1×1 se cambia la dimensione dei canali. Progressione canali: $64 \to 128 \to 256 \to 512$ con `MaxPool2d(2)` tra i livelli.

- **Bottleneck:** un ulteriore `ResidualConv` con base $\times 16 = 1024$ canali.
- **Decoder (4 stadi):** per ogni stadio: `ConvTranspose2d` 2×2 per upsampling, `AttentionBlock` per pesare la skip-connection dell'encoder, concatenazione e `ResidualConv` per la fusione. Le attenzioni sono implementate con proiezioni 1×1 su ramo di gating e ramo skip, non-linearità ReLU e testa sigmoide.
- **Teste multi-task:** due `Conv2d` 1×1 distinte partendo dall'ultimo decoder: `out_rgb` (3 canali) e `out_mask` (1 canale), con attivazione sigmoide e clamp in $[0, 1]$.

Flusso dei tensori e vincoli pratici

- **Formati:** input $I \in \mathbb{R}^{B\times 3\times H\times W}$; le dimensioni H, W devono essere multiple di $2^4 = 16$ per l'architecture con 4 pooling.
- **Skip con attenzione:** per ogni livello del decoder, il gating dal decoder g e la skip dall'encoder x sono proiettati in uno spazio intermedio e combinati per produrre il coefficiente di attenzione $\alpha \in [0, 1]$, applicato elemento per elemento a x. Questo filtra contenuti non utili alla ricostruzione.
- **Output:** la rete produce coppia $(\hat{I}, \hat{M})$ con $\hat{I} \in [0, 1]^{B\times 3\times H\times W}$ e $\hat{M} \in [0, 1]^{B\times 1\times H\times W}$, pronte per le loss definite nella sezione *Training*.

2.2.2 Modello 2.5D

L'implementazione 2.5D estende la 2D elaborando sequenze di **8 frame** concatenate lungo i canali ($24 = 8 \times 3$), mantenendo convoluzioni 2D e skip con *Attention Gates*. Ciò consente di sfruttare il contesto temporale con costi computazionali contenuti rispetto al 3D.

Definizione architetturale

Backbone: `UNetImproved25D(in_channels=24, out_channels=3, base=64)`.

- **Input conv** 1×1**:** proietta i 24 canali (8 frame $\times$ 3) in base $=$ 64 feature tramite mixing lineare per-pixel, fondendo l'informazione temporale senza alterare la risoluzione spaziale e allineando i canali alla larghezza della rete.

- **Encoder (4 livelli):** sequenza di `ResidualConv` con progressione canali $64 \to 128 \to 256 \to 512$, intervallati da `MaxPool2d(2)`.

- **Bottleneck:** `ResidualConv` con base $\times\, 16 = 1024$ canali.

- **Decoder (4 stadi):** `ConvTranspose2d` $2{\times}2$ per upsampling, `AttentionBlock` per pesare la skip dall'encoder, concatenazione e `ResidualConv` di fusione.

- **Teste multi-task:** `out_rgb` 1×1, 3 canali, e `out_mask` 1×1, 1 canale; sigmoide + clamp in $[0, 1]$.

Flusso dei tensori e vincoli pratici

- **Input:** sequenza appiattita $S \in \mathbb{R}^{B\times 24\times H\times W}$ (8 frame RGB concatenati per canali).

- **Compatibilità dimensionale:** H, W multipli di $2^4 = 16$ per le 4 fasi di pooling/upsampling.

- **Output:** $\hat{I} \in [0,1]^{B\times 3\times H\times W}$ e $\hat{M} \in [0,1]^{B\times 1\times H\times W}$. Durante l'addestramento si confronta $\hat{I}$ con il frame pulito di riferimento (tipicamente il centrale della finestra), senza vincolare l'architettura a dipendenze temporali esplicite.

2.2.3 Modello 3D

L'implementazione 3D adotta una U-Net 3D con blocchi residuali e *Attention Gates 3D* in *impostazione multi-task* (ricostruzione RGB + maschera) su volumi (C, D, H, W).

Definizione architetturale
Backbone: `UNet3D(in_channels=3, out_channels=3, base=32, depth=D)`.

- **Encoder (livelli dinamici):** il numero di livelli dipende da D per evitare over-pooling temporale: $D \geq 32 \Rightarrow 4$ livelli, $D \geq 16 \Rightarrow 3$, $D \geq 8 \Rightarrow 2$, altrimenti 1. Ogni livello è un `ResidualConv3D` (due `Conv3d` $3 \times 3 \times 3$ + `BatchNorm3d` + residuo $1 \times 1 \times 1$), separato da `MaxPool3d(2)`.

- **Bottleneck:** `ResidualConv3D` con numero di canali calibrato sul livello encoder più profondo (gestito tramite `bottleneck_out_channels`).

- **Decoder (dinamico):** per ciascun livello disponibile: `ConvTranspose3d` $2 \times 2 \times 2$ per upsampling, `AttentionBlock3D` per pesare la skip dall'encoder (gating 3D), concatenazione e `ResidualConv3D` di fusione.

- **Teste multi-task:** `out_rgb` e `out_mask` sono `Conv3d` $1 \times 1 \times 1$ con sigmoide + clamp in $[0, 1]$, restituendo volumi.

Flusso dei tensori e vincoli pratici

- **Input:** $X \in \mathbb{R}^{B \times 3 \times D \times H \times W}$. La profondità D definisce i livelli encoder attivabili (per non collassare la dimensione temporale).

- **Compatibilità dimensionale:** si preferiscono H, W multipli di $2^4 = 16$ per allineare pooling/upsampling.

- **Output:** $\hat{I} \in [0, 1]^{B \times 3 \times D \times H \times W}$ e $\hat{M} \in [0, 1]^{B \times 1 \times D \times H \times W}$.

2.3 Training

2.3.1 Schema di training multi-task

Tutti e tre i modelli (2D, 2.5D, 3D) sono addestrati in modo **multi-task**, ottimizzando congiuntamente la ricostruzione RGB e la stima della maschera. L'ottimizzazione con *backpropagation congiunta* sfrutta due teste d'uscita (RGB, mask) e una loss composita:

$$\mathcal{L}_{total} = \mathcal{L}_{RGB} + \lambda_{mask}\mathcal{L}_{mask} + \lambda_{halo}\mathcal{L}_{halo}$$

dove λ_{mask} e λ_{halo} governano il bilanciamento tra i compiti. La testa mask agisce da supervisione ausiliaria esplicita sulle regioni nuvolose, mentre la componente $\mathcal{L}_{halo}$ penalizza artefatti scuri in prossimità dei bordi nuvolosi.

Loss RGB

La funzione di costo per la ricostruzione dell'immagine pulita, in continuità con quanto illustrato in precedenza, combina in modo pesato tre contributi:

$$\mathcal{L}_{RGB} = (1 - w_{ssim}) \cdot \mathcal{L}_{L1} + w_{ssim} \cdot \mathcal{L}_{SSIM} + \mathcal{L}_{perceptual}$$

dove i termini sono definiti come segue:

- $\mathcal{L}_{L1}$ (MAE): penalizza discrepanze pixel-wise tra predizione e riferimento.
- $\mathcal{L}_{SSIM} = 1 - \text{SSIM}(y_{pred}, y_{true})$[9]: misura di dissimilarità strutturale (in forma di loss).
- $\mathcal{L}_{perceptual}$: componente percettiva basata su feature VGG19[8].
- $w_{ssim} = 0.2$: iperparametro che controlla il contributo relativo della componente SSIM.

Loss Maschera

La predizione della maschera delle nuvole utilizza Binary Cross-Entropy:

$$\mathcal{L}_{mask} = -\frac{1}{N}\sum_{i=1}^{N}[m_i \log(\hat{m}_i) + (1 - m_i)\log(1 - \hat{m}_i)]$$

con peso $\lambda_{mask} = 1.0$. Questa loss ausiliaria aiuta la rete a identificare esplicitamente le regioni nuvolose.

Anti-Halo Loss

Per prevenire artefatti scuri (halo effect) sotto le nuvole dense, introduco la seguente loss:

$$\mathcal{L}_{halo} = \frac{1}{|\Omega_{dense}|}\sum_{p\in\Omega_{dense}} \max(0, y_{true}(p) - y_{pred}(p)) \cdot \mathbb{1}_{[\text{diff}(p)>\tau]}$$

dove $\Omega_{dense} = \{p : m(p) > 0.8\}$ sono i pixel sotto nuvole molto dense, e $\tau = 0.1$ è la soglia per differenze significative. Il peso è $\lambda_{halo} = 0.03$, volutamente piccolo per non compromettere la ricostruzione generale.

2.3.2 Early Stopping e Checkpointing

Per tutti i modelli, implemento:

- **Validation**: ogni epoca, calcolo loss, PSNR e SSIM sul validation set.
- **Best model**: salvo i pesi quando la validation loss migliora di almeno 10^{-6}.
- **Early stopping**: interrompo il training se non c'è miglioramento per N epoche consecutive (patience).
- **Sample visualization**: salvo griglie di predizioni ogni 5 epoche per monitoraggio qualitativo.

2.3.3 Configurazione Training 2D

Hyperparametri

Per il training 2D adotto l'ottimizzatore Adam[11] ($\beta_1 = 0.9$, $\beta_2 = 0.999$). Il learning rate è inizializzato a 10^{-4} ed è gestito da uno scheduler *ReduceLROnPlateau*[19] con *patience* pari a 8, che riduce il passo di apprendimento in assenza di miglioramenti sulla *validation loss*. L'addestramento è eseguito fino a un massimo di 100 epoche con *early stopping*[20] (patience=10). La configurazione operativa prevede batch size 16 su patch di dimensione 320×320 pixel, con 64 canali base.

Fine-tuning 2D (mask-guided)

Successivamente, ho effettuato un **fine-tuning** enfatizzando il carattere multi-task: la loss anti-halo è *guidata dalla maschera predetta* per colpire selettivamente le zone problematiche (bordi nuvola, aloni cromatici). Ho adottato:

- **LR**: $6 \cdot 10^{-6}$ (ottimizzato per correzioni puntuali).
- λ_{halo}: 0.15, con pesatura spaziale basata su confidenza della maschera predetta e zone di bordo.
- **Percettiva**: peso 0.1; **batch**: 4; **patience**: 8.

Questo approccio sfrutta la sinergia tra i due task: la testa mask non solo migliora la segmentazione, ma guida la regolarizzazione cromatica dell'RGB nelle regioni più a rischio di aloni.

Mixed Precision Training

Utilizzo Automatic Mixed Precision (AMP) opzionale per ridurre l'uso di memoria e accelerare il training, mantenendo la precisione numerica grazie al gradient scaling.

2.3.4 Configurazione Training 2.5D

Il modello 2.5D richiede adattamenti per gestire le sequenze temporali:

Hyperparametri

Anche per il 2.5D utilizzo Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$), con learning rate iniziale 10^{-4} controllato da uno scheduler *on plateau*[19] (patience=8). L'addestramento si estende fino a 100 epoche con *early stopping*[20] (patience=15). Per gestire il carico computazionale delle sequenze temporali impiego batch size 8 su patch 320×320, 64 canali base e uno stride di 160 (overlap 50%).

NO Data Augmentation

A differenza del 2D, **non applico augmentazioni** durante il training 2.5D. La motivazione è preservare la *coerenza temporale* del movimento delle nuvole tra i frame. Flip o rotazioni casuali distruggerebbero la direzione e velocità del movimento, rendendo incoerente la sequenza. La variabilità è già garantita durante la generazione del dataset con:

- 8 direzioni di movimento diverse.
- Step variabile (70% $\in [12, 18]$, 30% $\in [18, 22]$).
- Parametri Perlin noise casuali.
- Opacità variabile nel tempo.

2.3.5 Configurazione Training 3D

Il modello 3D è il più costoso computazionalmente e richiede parametri ridotti; resta **multi-task** con teste volumetriche e usa il frame centrale per le metriche 2D.

Hyperparametri

Nel caso 3D adotto Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$) con learning rate iniziale 10^{-4} e scheduler *ReduceLROnPlateau*[19] (patience=8). Il training è limitato

a 100 epoche con *early stopping*[20] (patience=15). Per contenere memoria e tempi, impiego batch size 8 su patch 320 × 320, profondità temporale 16 frame, 64 canali base e 2 worker per il caricamento dei volumi.

Architettura Adattiva

Come descritto precedentemente, il numero di livelli encoder si adatta automaticamente alla depth:

- Depth 8: 2 livelli encoder (capacità ridotta ma memoria gestibile).
- Depth 16: 3 livelli encoder (compromesso ottimale).
- Depth 32+: 4 livelli encoder (massima capacità).

Loss sul Frame Centrale

Per i modelli 3D, tutte le componenti della loss sono calcolate sul frame centrale del volume predetto:

$$y_{pred}^{center} = y_{pred}[:, :, \lfloor D/2 \rfloor, :, :]$$

Questo permette un confronto diretto con i modelli 2D/2.5D e riduce il costo computazionale della Perceptual Loss (che richiede feature VGG19 2D).

2.4 Test sperimentali

In questa sezione riassumo le scelte sperimentali adottate per esplorare le configurazioni dei modelli.

2.5D

Ho esplorato diverse politiche di **step temporale** tra i frame:

- **Step fisso**: esperimenti con passo costante per l'intera sequenza.
- **Step variabile**: distribuzione mista (ad es. 70% in $[12, 18]$, 30% in $[18, 22]$) per aumentare la diversità di moto nuvoloso.

Inoltre, ho variato il numero di frame utilizzati per sequenza, provando $N\_\text{FRAME} \in \{6, 8, 12\}$, mantenendo input appiattito a canali ($3\,N\_\text{FRAME}$).

3D

Una volta individuata la politica di step più efficace dal 2.5D, l'ho **riutilizzata in 3D** per garantire coerenza del moto; in 3D ho quindi **fissato lo step** e ho esplorato:

- **Profondità temporale**: $D \in \{8, 16\}$.
- **Canali base**: base_ch $\in \{32, 64\}$.

Questo ha permesso di analizzare l'effetto della capacità del modello (base_ch) e della profondità temporale (D) sul comportamento dell'architettura volumetrica a parità di dinamica di moto.

Capitolo 3

Risultati numerici

Andamento 2D

Nel 2D si osserva un incremento rapido delle prestazioni nelle prime epoche, seguito da una fase di plateau. Il *PSNR* cresce stabilmente e l'*SSIM* supera rapidamente valori elevati, mentre la loss di train/val decresce senza segnali evidenti di overfitting.

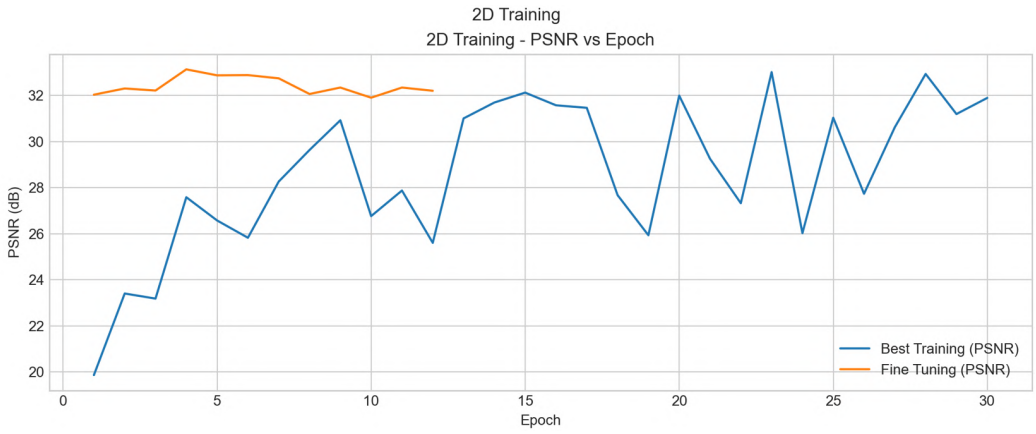


Figura 3.1: 2D: andamento del PSNR per epoca; confronto dei tentativi.

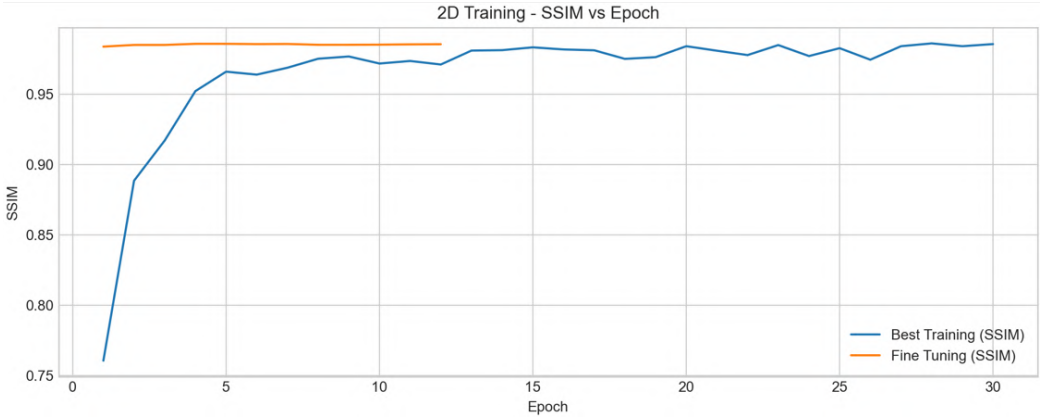


Figura 3.2: 2D: andamento dell'SSIM per epoca; le curve sono ravvicinate nelle ultime epoche.

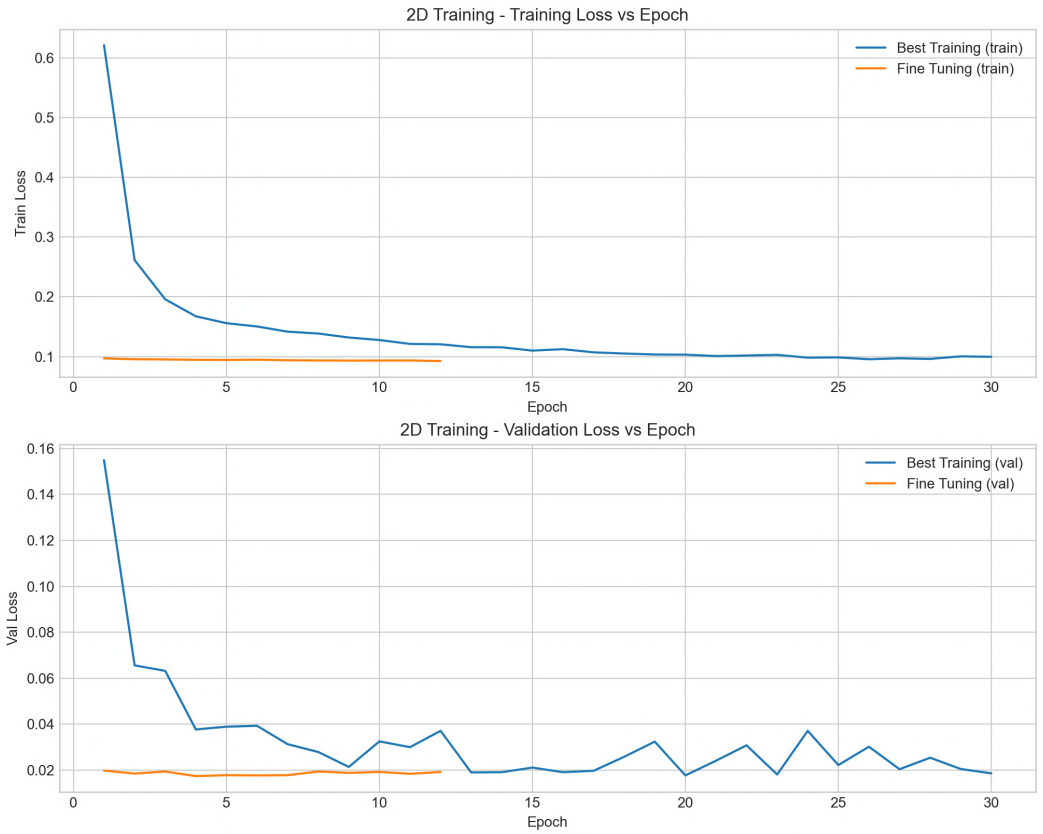


Figura 3.3: 2D: loss di training e validation; calo regolare con stabilizzazione finale.

Andamento 2.5D

Con il 2.5D, l'uso di sequenze temporali porta a un aumento del PSNR e dell'SSIM rispetto al 2D, a fronte di un costo computazionale maggiore. I grafici evidenziano differenze tra i tentativi: il compromesso migliore mostra un miglior rapporto prestazioni/costo e una loss valida più stabile nelle ultime epoche.

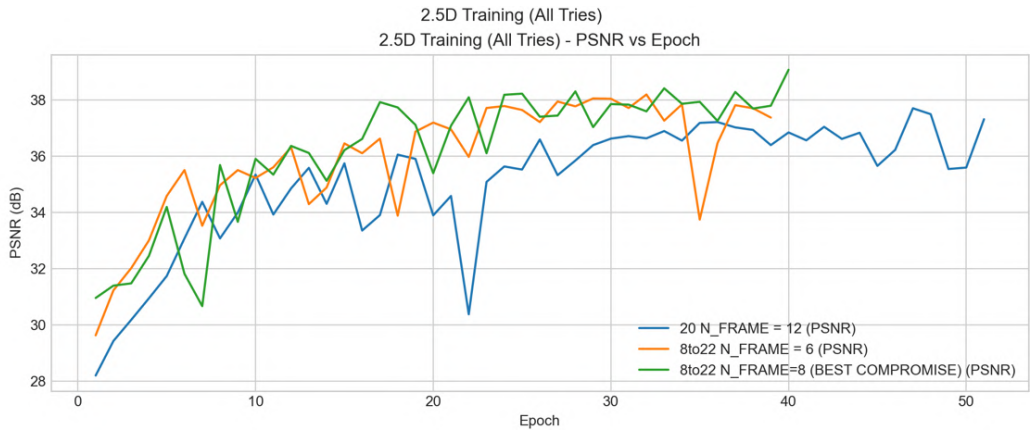


Figura 3.4: 2.5D: PSNR per epoca e confronto tra i diversi tentativi.

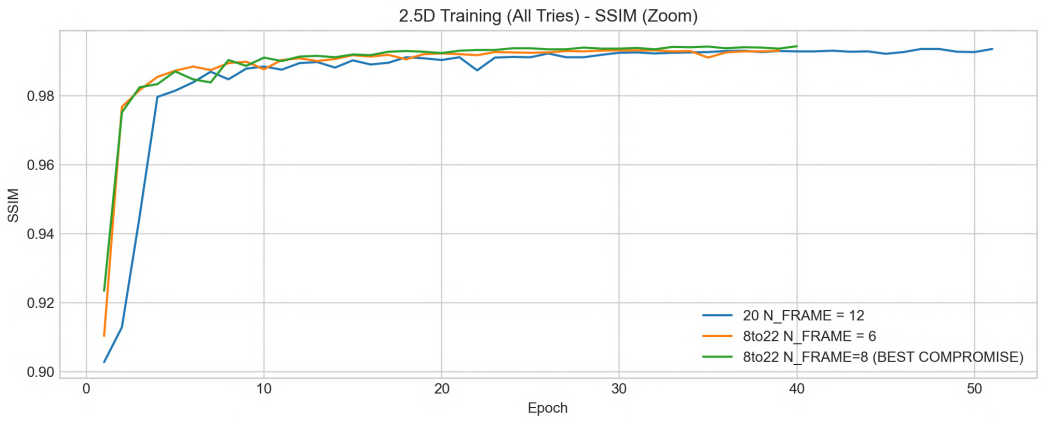


Figura 3.5: 2.5D: SSIM per epoca e confronto tra i diversi tentativi.

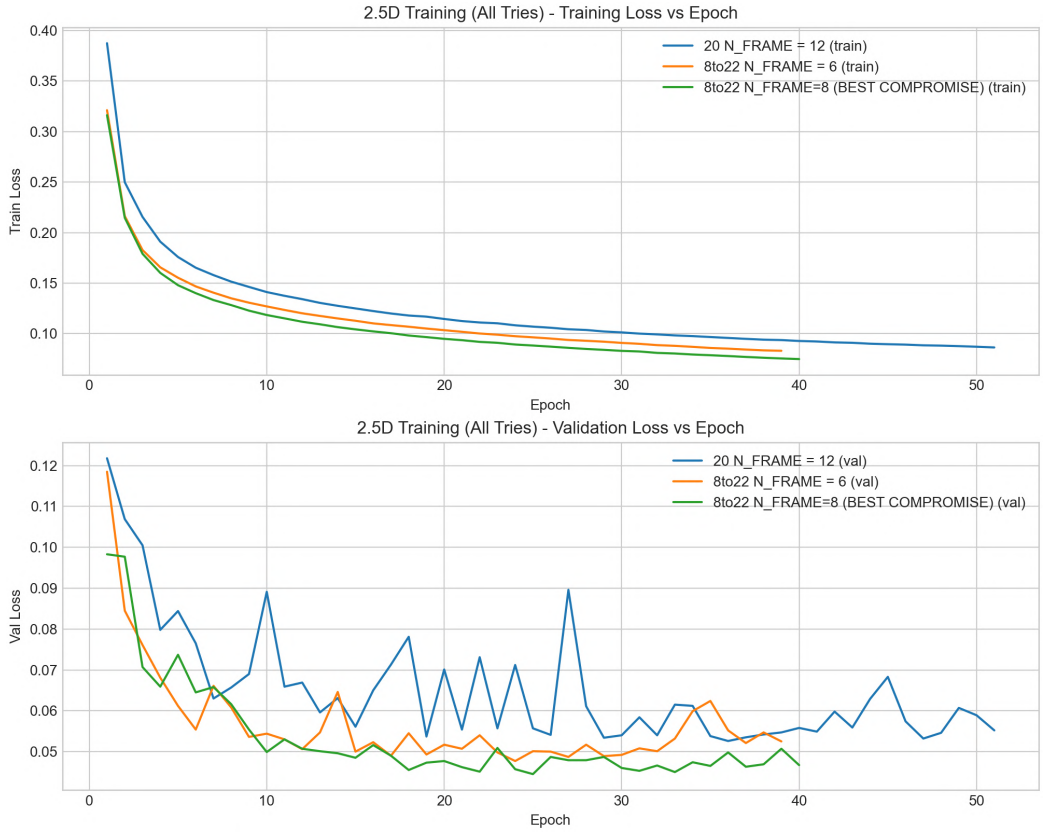


Figura 3.6: 2.5D: loss di training e validation; maggiore stabilità e minore varianza rispetto al 2D.

Andamento 3D

Il modello 3D raggiunge le prestazioni più alte in termini di PSNR/SSIM, con trend regolari e loss di validation che converge su valori inferiori rispetto a 2D/2.5D. L'incremento di performance è accompagnato da un costo per epoca sensibilmente più elevato.

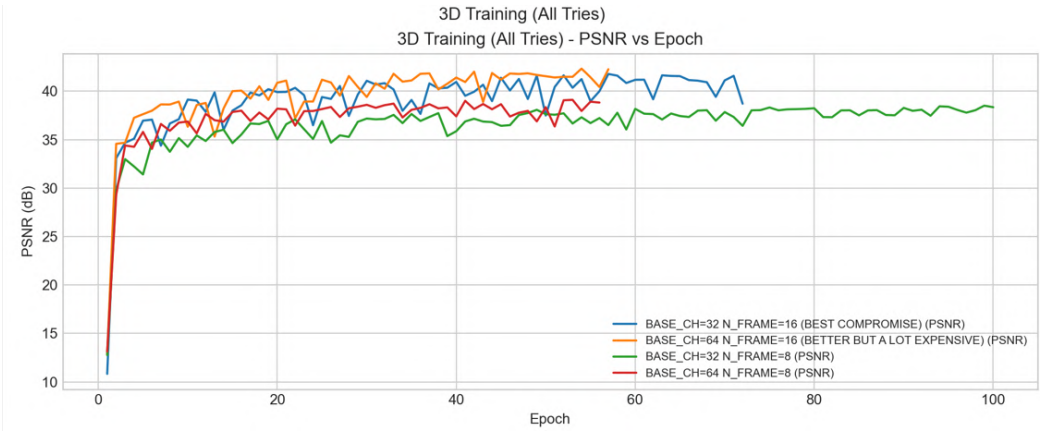


Figura 3.7: 3D: PSNR per epoca per i diversi setting (profondità e canali base).

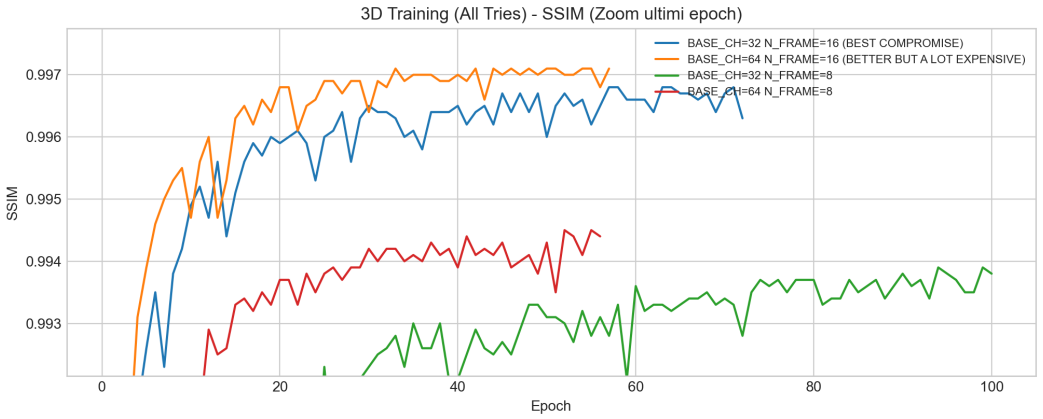


Figura 3.8: 3D: SSIM per epoca; confronto tra i diversi tentativi.

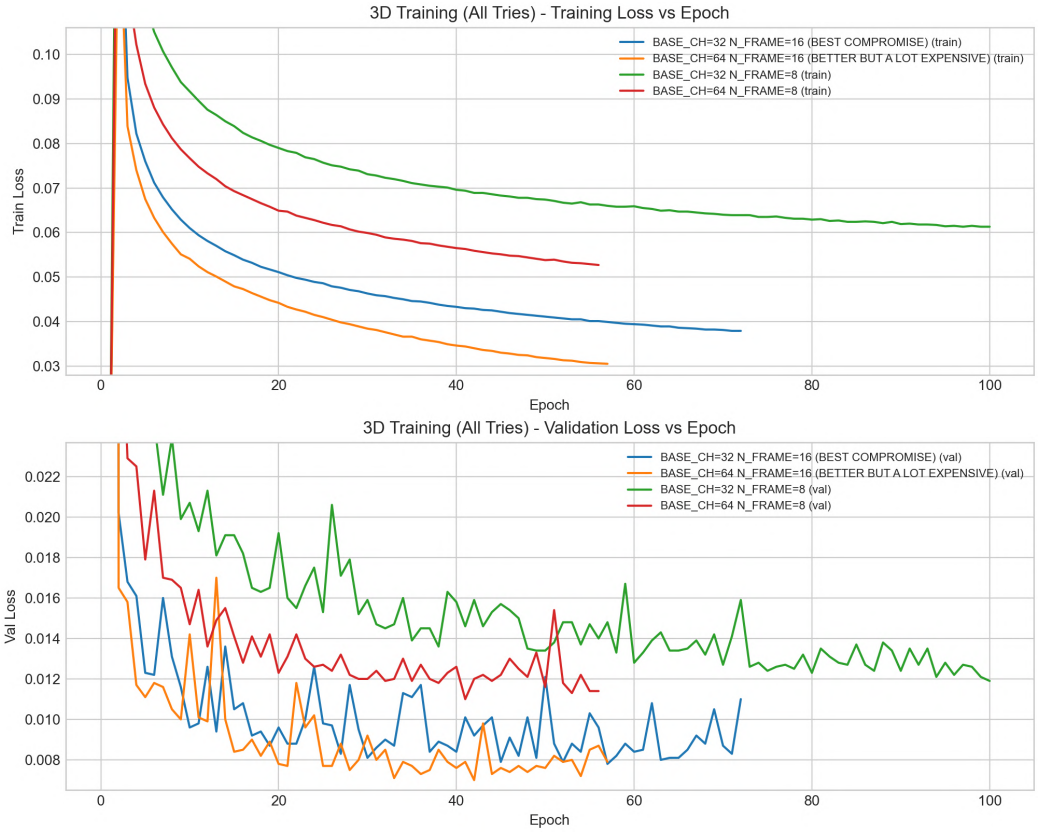


Figura 3.9: 3D: loss di training e validation; convergenza più rapida e plateaux più bassi.

Confronto finale

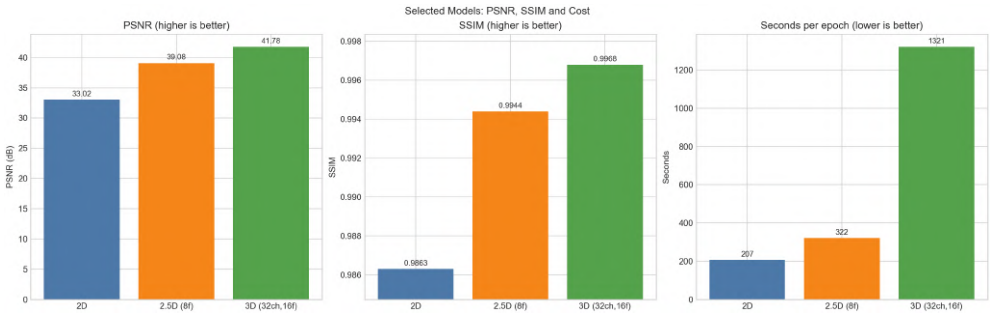


Figura 3.10: Confronto finale tra i migliori modelli: PSNR, SSIM e costo (sec/epoca).

In sintesi:

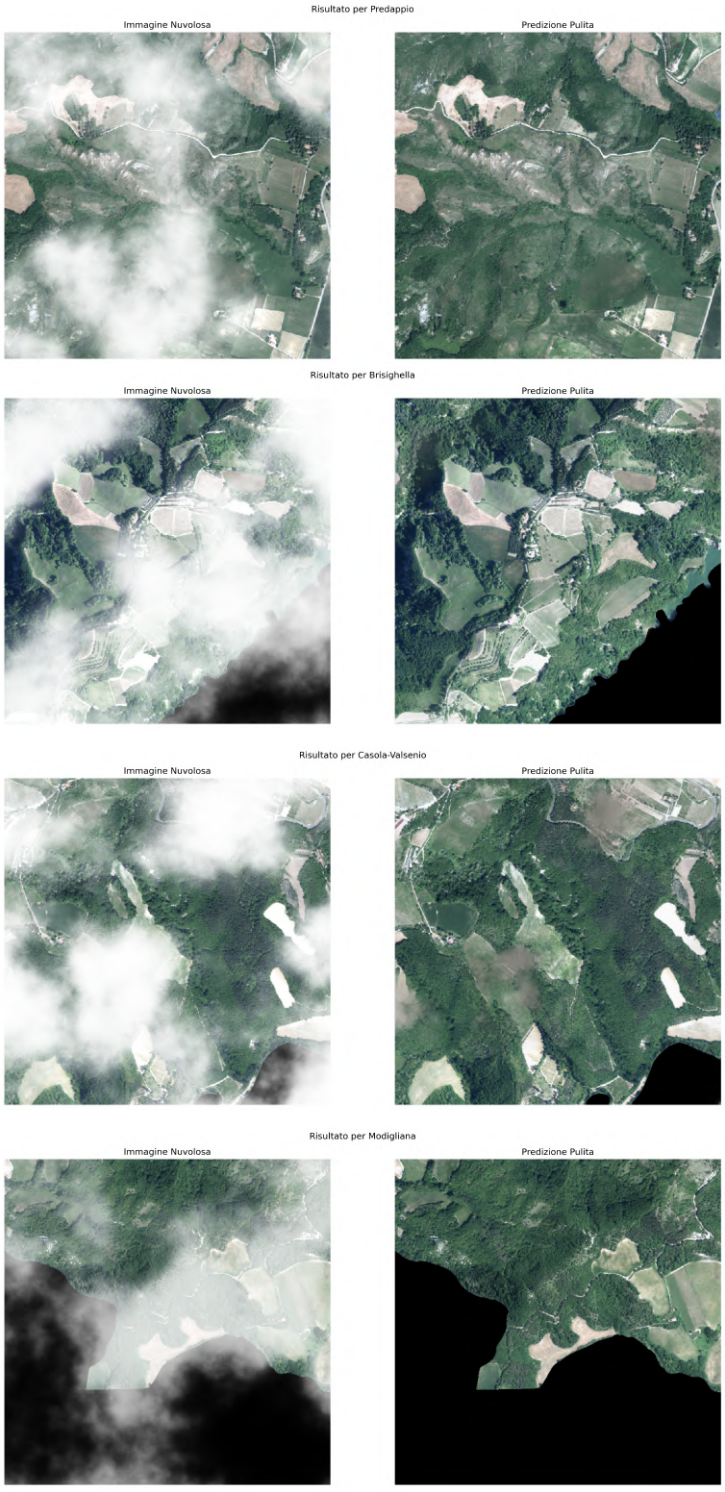
2D: costo per epoca minimo, ma metriche PSNR/SSIM inferiori; buon *baseline*.

2.5D (8 frame): netto salto in PSNR/SSIM a fronte di un incremento di costo moderato; *miglior compromesso* qualità/costo.

3D (BASE_CH=64, 16 frame): metriche migliori (PSNR/SSIM) ma costo sensibilmente maggiore; indicato quando la massima qualità è prioritaria rispetto al tempo di addestramento.

Inferenze finali per modello

2D



Nelle regioni con copertura nuvolosa elevata il modello fatica a ricostruire il contenuto sottostante: tende a lasciare velature residue e perdita di texture nelle aree più dense, con bordi talvolta sfumati e leggere dominanti cromatiche. Il degrado è più evidente dove la copertura è spessa e continua, coerentemente con un calo locale delle metriche (SSIM/PSNR).

2.5D

Predappio

Nuvolosa *Predizione* *Clean*

Brisighella

Nuvolosa *Predizione* *Clean*

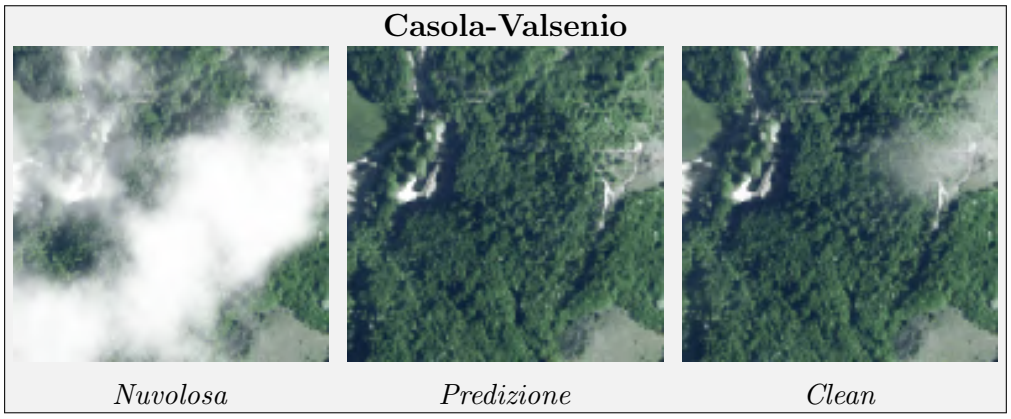
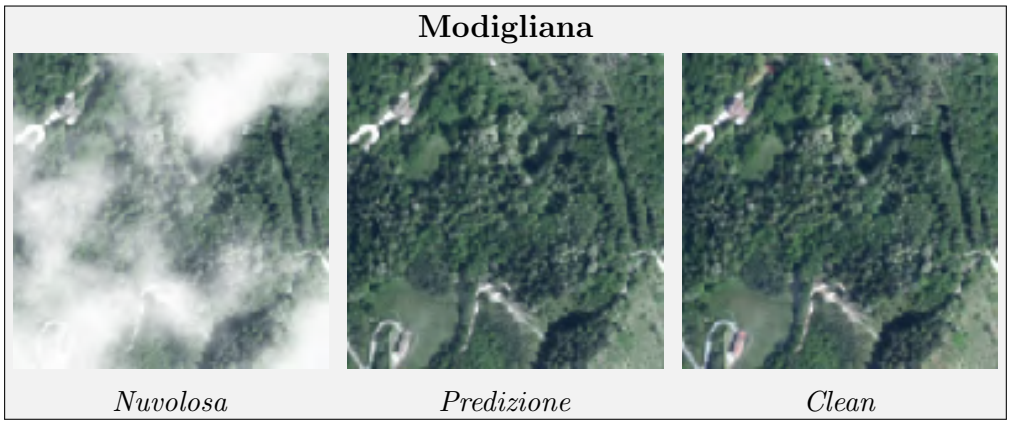


Figura 3.11: esempio di leggere tracce di *ghosting* nelle transizioni rapide della nuvola e smoothing in presenza di occlusioni persistenti.



L'uso di contesto temporale migliora la coerenza tra frame e il recupero dei dettagli sottili rispetto al 2D. Rimangono critiche le occlusioni persistenti su più frame (nubi molto lente/spesse), dove possono comparire leggere tracce di *ghosting* o smoothing nelle transizioni rapide della nuvola (Figura 3.11). In generale, le velature residue risultano attenuate.

3D

Predappio

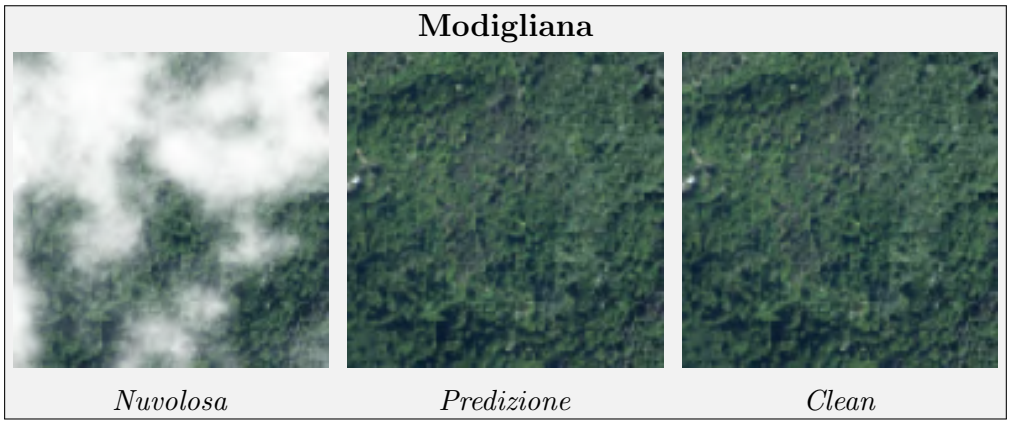
Nuvolosa *Predizione* *Clean*

Brisighella

Nuvolosa *Predizione* *Clean*

Casola-Valsenio

Nuvolosa *Predizione* *Clean*



La modellazione spazio–temporale con convoluzioni 3D gestisce meglio le nubi dense e produce ricostruzioni più stabili ai bordi, con coerenza temporale superiore. In assenza di informazione osservabile per molti frame consecutivi può comparire un lieve smoothing o allucinazioni di dettaglio. Il costo computazionale e di memoria è maggiore, ma il guadagno qualitativo è evidente nei casi di copertura estesa.

Capitolo 4

Conclusioni

In questa tesi è stato affrontato il problema della rimozione delle nubi da immagini ottiche satellitari, con l'obiettivo di abilitare un monitoraggio più continuo e affidabile delle frane in EmiliaRomagna. In coerenza con gli obiettivi introdotti, è stata progettata e valutata una pipeline di declouding che combina dati sintetici con nubi generate algoritmicamente e tre famiglie di modelli di deep learning (UNet 2D, UNet 2.5D su sequenze brevi, UNet 3D volumetrica), adottando funzioni di loss mirate (SSIM/PSNR, Perceptual Loss) e una componente AntiHalo.

Nel quadro sperimentale è stata condotta un'analisi comparativa sistematica:

- **Configurazione 2.5D**: esplorazione del passo temporale e del numero di frame (N_FRAME = 6, 8, 12).
- **Configurazione 3D**: valutazione delle profondità temporali (D = 8, 16) e dei canali base (32, 64)

La valutazione, in termini di PSNR/SSIM e analisi qualitative, ha evidenziato che:

- **2D**: baseline efficace e computazionalmente parsimoniosa
- **2.5D**: miglior compromesso tra prestazioni e costo computazionale

- **3D**: prestazioni massime (PSNR/SSIM) a fronte di un costo per epoca superiore

Le visualizzazioni su casi reali (Predappio, Brisighella, CasolaValsenio, Modigliana) confermano la coerenza radiometrica delle ricostruzioni e la mitigazione degli artefatti lungo i margini nuvolosi.

In sintesi, i risultati conseguiti attestano che la configurazione 2.5D rappresenta il miglior compromesso tra qualità e costo, la 3D massimizza le metriche quando la fedeltà è prioritaria, mentre la 2D costituisce una baseline affidabile ed efficiente. L'integrazione del declouding nella pipeline di analisi genera ricostruzioni radiometricamente coerenti, riduce gli artefatti e abilita analisi temporali più continue. La soluzione proposta soddisfa l'obiettivo di rimozione delle nubi e fornisce un fondamento solido per l'impiego operativo nel monitoraggio delle frane in EmiliaRomagna.

Bibliografia

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.

[2] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 1097-1105.

[3] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern recognition*, 77, 354-377.

[4] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015)*, LNCS 9351, pp. 234–241. Springer.

[5] O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks. *arXiv preprint arXiv:1511.08458*.

[6] He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep Residual Learning for Image Recognition*. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778.

[7] Oktay, O., Schlemper, J., Le Folgoc, L., Lee, M. C. H., Heinrich, M. P., Misawa, K., Mori, K., McDonagh, S., Hammerla, N. Y., Kainz, B.,

Glocker, B., & Rueckert, D. (2018). Attention U-Net: Learning Where to Look for the Pancreas. *arXiv preprint arXiv:1804.03999*.

[8] Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In: *European Conference on Computer Vision (ECCV)*, pp. 694-711. Springer.

[9] Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4), 600-612.

[10] Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing*. 4th Edition. Pearson.

[11] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[12] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2121-2159.

[13] Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 26-31.

[14] Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y., Dong, C., ... & Change Loy, C. (2018). ESRGAN: Enhanced super-resolution generative adversarial networks. In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 63-79.

[15] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. (Cap. 9: Convolutional Networks).

[16] Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

[17] Ji, S., Xu, W., Yang, M., & Yu, K. (2013). 3D Convolutional Neural Networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1), 221-231.

[18] Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., Brox, T., & Ronneberger, O. (2016). 3D U-Net: Learning dense volumetric segmentation from sparse annotation. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 424–432. Springer.

[19] Keras Documentation. ReduceLROnPlateau (Learning Rate Scheduler). Disponibile online: `https://keras.io/api/callbacks/reduce_lr_on_plateau/`.

[20] Prechelt, L. (1998). Early stopping — but when? In: *Neural Networks: Tricks of the Trade*, pp. 55–69. Springer.

[21] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708.

[22] Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). Aggregated Residual Transformations for Deep Neural Networks (ResNeXt). In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1492–1500.

[23] Zhou, Z., Siddiquee, M. M. R., Tajbakhsh, N., & Liang, J. (2018). UNet++: A Nested U-Net Architecture for Medical Image Segmentation. In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pp. 3–11. Springer.

[24] TensorFlow Tutorials. Video classification with a 3D convolutional neural network. Disponibile online: `https://www.tensorflow.org/tutorials/video/video_classification`.

[25] Perlin, K. (1985). An Image Synthesizer. In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '85)*, pp. 287–296. ACM, San Francisco, CA, USA.

[26] Perlin, K. (2002). Improving Noise. *ACM Transactions on Graphics*, 21(3), 681–682. ACM, New York, NY, USA.

[27] Shen, H., Li, X., Cheng, Q., Zeng, C., Yang, G., Li, H., & Zhang, L. (2015). Missing information reconstruction of remote sensing data: A technical review. *IEEE Geoscience and Remote Sensing Magazine*, 3(3), 61-85.

[28] Zhang, Q., Yuan, Q., Zeng, C., Li, X., & Wei, Y. (2018). Missing data reconstruction in remote sensing image with a unified spatial–temporal–spectral deep convolutional neural network. *IEEE Transactions on Geoscience and Remote Sensing*, 56(8), 4274-4288.

[29] Singh, P., & Komodakis, N. (2017). Cloud-GAN: Cloud removal for Sentinel-2 imagery using a cyclic consistent generative adversarial networks. In: *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, pp. 1772-1775. IEEE.

[30] Li, J., Wu, Z., Hu, Z., Zhang, J., Li, M., Mo, L., & Molinier, M. (2022). Thin cloud removal in optical remote sensing images based on generative adversarial networks and physical model of cloud distortion. *ISPRS Journal of Photogrammetry and Remote Sensing*, 166, 373-389.

[31] Xu, M., Jia, X., Pickering, M., & Plaza, A. J. (2020). Cloud removal based on sparse representation via multitemporal dictionary learning. *IEEE Transactions on Geoscience and Remote Sensing*, 54(5), 2998-3006.

[32] Zhang, H., Patel, V. M. (2018). Densely connected pyramid dehazing network. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3194-3203.

[33] Ebel, P., Meraner, A., Schmitt, M., & Zhu, X. X. (2021). Multisensor data fusion for cloud removal in global and all-season Sentinel-2 imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 59(7), 5866-5878.

Ringraziamenti

Nel corso di questa tesi desidero esprimere la mia sincera gratitudine a tutte le persone che hanno contribuito, in modi diversi, a rendere possibile questo percorso.

Un sentito ringraziamento va al Prof. Davide Evangelista, mio relatore, per la sua costante disponibilità, per i preziosi consigli scientifici e per avermi trasmesso entusiasmo e passione per la ricerca nel campo delle immagini e dell'intelligenza artificiale.

Ringrazio inoltre la Prof.ssa Elena Loli Piccolomini, correlatrice di questa tesi, per il supporto fondamentale e gli utili confronti che hanno arricchito e migliorato il mio lavoro.

Un grazie va anche a tutti i colleghi e amici del Dipartimento di Informatica — in particolare a chi ha condiviso con me le sfide, le difficoltà e le soddisfazioni di questi anni universitari.

Un pensiero speciale ai miei genitori e alla mia famiglia, per la fiducia, l'affetto e il continuo sostegno morale che non mi è mai mancato.

Infine, grazie a tutte le persone che, nei modi più diversi, mi sono state vicine durante questo percorso.