



SAPIENZA
UNIVERSITÀ DI ROMA

Rilevamento automatico di nuvole tramite segmentazione con tecniche di Deep Learning

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Laurea in Ingegneria Informatica e Automatica

Valerio Ceccarelli

Matricola 1939007

Relatore

Prof. Thomas Alessandro Ciarfuglia

Anno Accademico 2022/2023

Rilevamento automatico di nuvole tramite segmentazione con tecniche di Deep Learning

Tesi di Laurea Triennale. Sapienza Università di Roma

© 2023 Valerio Ceccarelli. Tutti i diritti riservati

Questa tesi è stata composta con L^AT_EX e la classe Sapthesis.

Email dell'autore: valerio.ceccarelli@gmail.com

Sommario

Il rilevamento automatico delle nuvole nelle immagini satellitari è un task cruciale in quanto utile in innumerevoli applicazioni che spaziano dalla meteorologia all'analisi del territorio; per un compito così complesso, software che si basano su algoritmi classici non sono più sufficienti, quindi è necessario ricorrere a soluzioni più avanzate come quelle basate sul Deep Learning come le reti neurali convoluzionali.

Spesso però questi modelli necessitano di hardware costosi perché sono computazionalmente onerosi sia da allenare che da eseguire; in questo elaborato verrà presentato ed analizzato un nuovo modello, in grado di gestire tutte le informazioni spettrali provenienti dal satellite Sentinel-2A, con l'obiettivo di dimostrare che è possibile ottenere dei risultati al pari dell'attuale stato dell'arte anche diminuendo drasticamente il numero di parametri.

Infine, per dimostrare la riuscita di questo progetto, oltre ad esempi visuali, vengono introdotte diverse metriche per produrre dati da confrontare con le altre soluzioni attualmente usate per questo compito; per confermare le sue performance in tutti gli scenari, il modello verrà messo sotto stress con immagini contenenti neve e ghiaccio nello sfondo, dove in alcuni casi la classificazione risulta difficile anche all'occhio umano.

Indice

1	Introduzione	1
2	Il dataset	3
2.1	Le immagini	3
2.2	Note tecniche	6
2.3	Le trasformazioni	7
3	La rete	9
3.1	Panoramica del modello	9
3.1.1	Concatenation and Sum	9
3.1.2	Skip connection e Residual block	10
3.1.3	Convolution layer	11
3.1.4	Deconvolution layer	12
3.1.5	Pooling layer	13
3.1.6	Rectified linear unit	13
3.1.7	Depth-wise separable convolution layer	13
3.1.8	Shared and dilated convolution residual block	14
3.1.9	Mixed depth-wise separable convolution layer	15
3.2	Considerazioni	16
4	L'allenamento	17
4.1	Hardware	17
4.2	Metodo e implementazione	18
4.2.1	Dataset	18
4.2.2	Optimizer	19
4.2.3	Scheduler	20
4.2.4	Scelta degli iperparametri	20
4.2.5	Loss e logits	21
5	Test e risultati	23
5.1	Validation	23
5.1.1	Overfitting	23
5.1.2	Validation set	24
5.2	Metriche di valutazione	25
5.2.1	Accuracy	25
5.2.2	Precision	26
5.2.3	Recall	26

5.2.4	F1-Score	26
5.3	Risultati	27
5.3.1	Confronto con altri modelli	28
5.3.2	Analisi in situazioni di difficoltà	30
6	Conclusioni	33
	Bibliografia	35

Capitolo 1

Introduzione

L'identificazione delle nuvole in un'immagine satellitare costituisce un passo cruciale in numerose indagini scientifiche che spaziano tra diverse discipline. L'abilità di isolare i pixel associati alle nuvole in un'immagine è fondamentale per la conduzione di studi meteo-climatici avanzati, previsioni accurate del tempo e analisi approfondite sul clima.

Le informazioni relative a dati meteo possono contribuire ad attenuare gli effetti derivanti da catastrofi naturali, permettendoci di intervenire tempestivamente ad esempio con alert diramati dalla Protezione Civile; inoltre l'analisi di tali dati consente la pianificazione di sistemi di prevenzione di disastri ambientali di natura idro-meteorologica. Le indagini su fenomeni atmosferici rivestono un ruolo di rilevanza primaria anche nell'ambito dell'energia moderna: prevedere con precisione la potenza generata da fonti energetiche variabili come i pannelli solari, consente una gestione ottimizzata della produzione energetica da immettere nella rete elettrica, creando un vantaggio per l'intera collettività in termini di costi, consumo di risorse ambientali e riduzione dell'inquinamento.

D'altra parte, esistono scenari in cui l'eliminazione delle immagini o delle porzioni di immagini contenenti nuvole è auspicabile; questo si applica soprattutto a contesti in cui altri algoritmi di analisi automatica devono operare, come l'analisi del territorio o il rilevamento dei cambiamenti temporali attraverso la segmentazione delle immagini.

La necessità di automatizzare il processo di identificazione delle nuvole è emersa da tempo: inizialmente, si è tentato di affrontare il problema tramite modelli matematici, ma la complessità e varietà delle forme delle nuvole e del loro contesto (elementi sottostanti come città, neve e fiumi) ha reso difficile trovare una soluzione algoritmica precisa. A questo punto, l'intelligenza artificiale ha rivoluzionato l'approccio al problema: mediante algoritmi di Machine Learning, i modelli apprendono, durante la fase di training, come approssimare la funzione matematica sottostante, consentendo di ottenere risultati notevolmente migliori.

Tuttavia, l'utilizzo di modelli di Machine Learning non è privo di sfide. La grande quantità di parametri da memorizzare e i calcoli computazionalmente intensivi, limitano l'applicabilità di tali modelli a dispositivi di elaborazione di grandi dimensioni e costosi; inoltre, l'allenamento richiede notevole tempo e una vasta quantità di dati di addestramento non sempre facili da reperire.

L'obiettivo di questa tesi è quelli di proporre, implementare e testare un modello

di rete neurale convoluzionale prendendo come riferimento il paper "A lightweight deep learning based cloud detection method for Sentinel-2A imagery fusing multi-scale spectral and spatial features" con il framework PyTorch [1] per analizzare immagini provenienti dalla missione Sentinel[2] dell'ESA.

Il modello proposto sarà progettato per eseguire una segmentazione semantica in due classi: nuvole e sfondo. La segmentazione semantica comporta l'assegnazione di un'etichetta a ciascun pixel dell'immagine in ingresso, generando un'immagine a un solo canale con due possibili stati (bianco e nero). Questa maschera sovrapposta all'immagine originale consentirà di identificare le zone coperte dalle nuvole.

L'implementazione di questo progetto è disponibile su GitHub all'indirizzo <https://github.com/ValerioCeccarelli/CloudDetection-ML> mentre è possibile scaricare il dataset tramite il servizio Zenodo[3]; inoltre, la repository del progetto contiene anche un file "Readme" con le istruzioni per usare gli script Python e per gestire l'ambiente di lavoro.

Capitolo 2

Il dataset

Tutti gli algoritmi di Machine Learning necessitano di un'enorme quantità di dati. La mole rilevante di informazioni può creare un problema soprattutto se si vuole studiare un qualcosa di raro o molto specifico.

2.1 Le immagini

Band No.	Band name	Central Wavelength (μm)	Bandwidth (mm)	Spatial resolution (m)
Band 1	Coastal aerosol	0.443	27	60
Band 2	Blue	0.490	98	10
Band 3	Green	0.665	38	10
Band 4	Red	0.665	38	10
Band 5	Vegetation Red Edge	0.705	19	20
Band 6	Vegetation Red Edge	0.740	18	20
Band 7	Vegetation Red Edge	0.783	28	20
Band 8	NIR	0.842	145	10
Band 8A	Vegetation Red Edge	0.865	33	20
Band 9	Water Vapour	0.945	26	60
Band 10	SWIR-Cirrus	1.375	75	60
Band 11	SWIR	1.610	143	20
Band 12	SWIR	2.190	242	20

Tabella 2.1. Nomi e caratteristiche delle bande prodotte dal satellite

Fortunatamente in questo caso gli esempi non mancano, ed in particolare per questa tesi verranno usate delle immagini provenienti dai satelliti Sentinel 2A e Sentinel 2B che mappano costantemente la superficie terrestre completando un ciclo

di mappatura ogni 5 giorni. I dati prodotti da questi satelliti non sono semplici immagini rappresentabili con i 3 comuni canali dei colori RGB (Rosso, Verde e Blu) ma contengono altre informazioni, provenienti da parti dello spettro luminoso diverse dal sottoinsieme di frequenze dello spettro visivo, per un totale di 13 canali (Tabella 2.1).

Questi dati aggiuntivi sono molto utili in quanto analizzandoli troviamo che differenti oggetti rispondono in modo simile a piccoli range di frequenze; per esempio la neve dei poli o delle montagne riflette la luce solare allo stesso modo delle nuvole, quindi indipendentemente da quale materiale stiamo fotografando, usando solo i canali RGB, troveremo che i pixel di nuvola saranno bianchi esattamente come i pixel di neve. Possiamo già immaginare quindi che una delle problematiche più grandi che affrontano gli algoritmi di Cloud Detection è proprio quella di distinguere i pixel di nuvola dai pixel di neve (il problema si ripropone con tutti gli oggetti bianchi sulla superficie terrestre, tra cui molti prodotti artificiali quali ad esempio i palazzi delle città).

Fisicamente parlando però, gli oggetti in questione potranno anche avere comportamenti simili nella riflessione della luce nello spettro visivo, ma si comporteranno in maniera diversa se stimolati da altre frequenze.

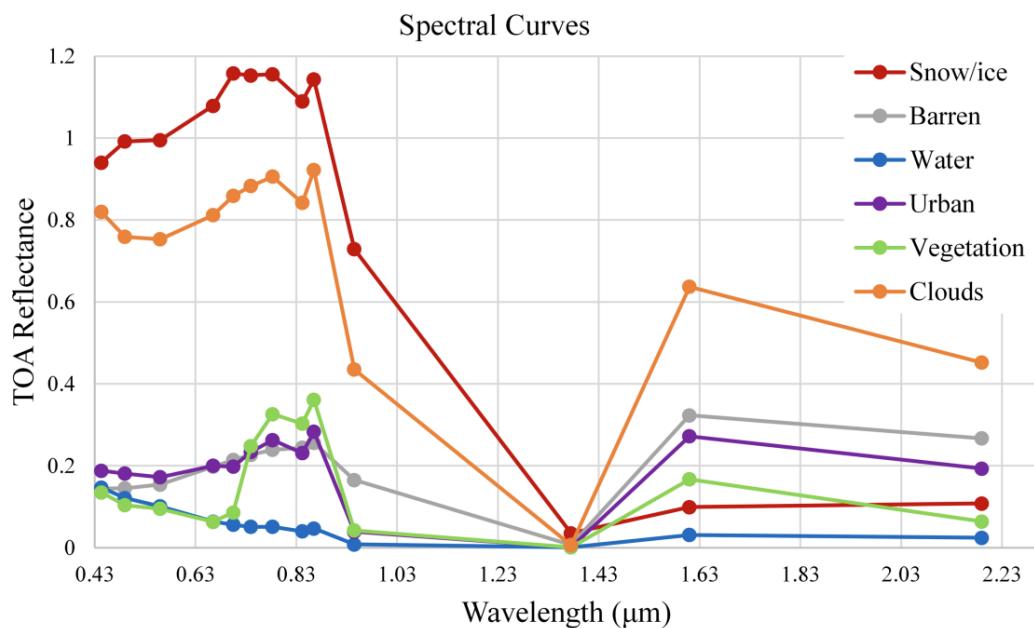


Figura 2.1. Curve di riflessione TOA per 6 comuni tipi di background e nuvole

Campionando i pixel delle immagini è possibile creare il grafico in Figura 2.1 dove sono inseriti i valori della "TOA reflectance" per ogni banda, per le 6 diverse macro categorie che è possibile osservare nelle immagini del dataset. La TOA reflectance (Top of Atmosphere reflectance) è una misura della quantità di luce solare riflessa da un oggetto o da una superficie sulla Terra, misurata alla parte superiore dell'atmosfera terrestre, senza tener conto degli effetti atmosferici; essa ci fa notare che: nelle frequenze dei canali RGB (ovvero da 0.490 a 0.665) sia le nuvole (in arancione) che la neve (in rosso) hanno una risposta estremamente simile, ma

che nelle basse frequenze (ovvero dai 1.6 ai 2.190) denominate SWIR (Short Wave Infrared) la risposta è totalmente diversa.

Per questo motivo, per avere una migliore classificazione, la rete neurale discussa in questa tesi accetterà come input tutti questi canali.

Ammesso di avere una quantità di dati sufficienti per l'allenamento, nel caso in cui si usino algoritmi di Supervised learning, è necessario anche che questi dati siano accompagnati da delle "label", ovvero la "ground truth" su cui l'algoritmo si baserà per correggere le sue predizioni.

In questo elaborato verrà usato lo stesso dataset proveniente dal paper [4] di riferimento, che gli autori hanno chiamato WHUS2-CD+. Questo è composto da 36 immagini, ciascuna con 13 bande a differenti risoluzioni; come mostrato nell'ultima colonna della Tabella 2.1 infatti le bande possono essere divise in 3 gruppi a seconda della risoluzione del sensore che le ha prodotte.

Queste immagini non sono state scelte a caso, anzi sono state selezionate fotografie di diverse zone della Cina, scattate in diversi periodi dell'anno, per avere la certezza di possedere dati sufficienti ad allenare e testare un algoritmo che sia in grado di ottenere buoni risultati indipendentemente dalle condizioni presenti al momento dello scatto.

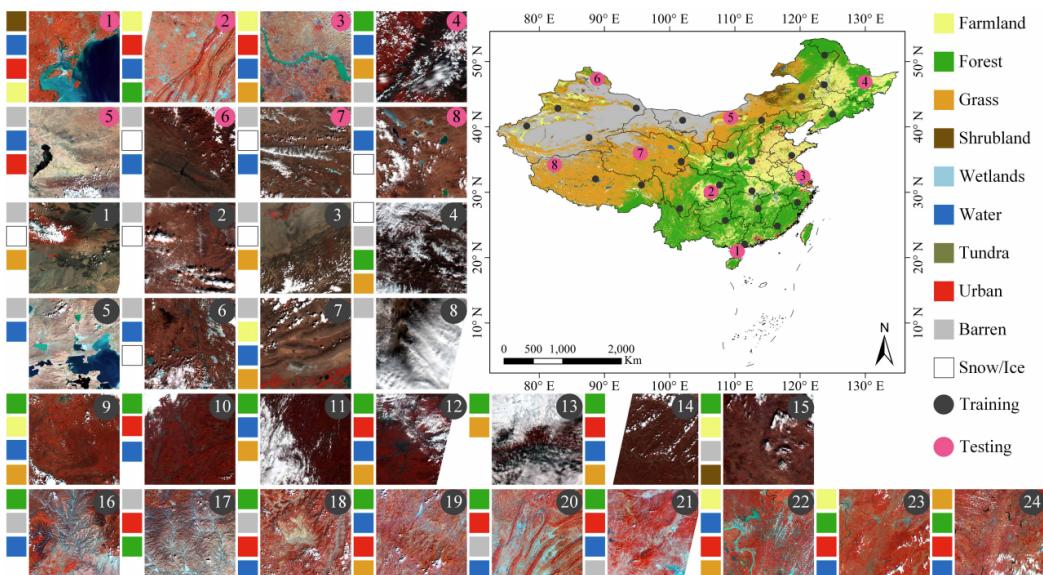


Figura 2.2. WHUS2-CD+ dataset: posizione delle 36 immagini sul territorio cinese

In particolare, queste foto sono state scattate a partire dall'aprile 2018 fino a maggio 2020 e coprono circa 427.500 km^2 di superficie terrestre; come mostrato nella Figura 2.2 si è deciso di identificare 10 macro categorie (farmland, forest, grass, shrubland, wetlands, water, tundra, urban, barren e snow/ice) in modo tale da selezionare delle immagini che siano il più possibile variegate. Questa classificazione è stata utilizzata anche per dividere il dataset in 2 parti, il train set e il test set. Tale scelta ha permesso di selezionare 8 immagini da utilizzare per il test, con delle caratteristiche tali da comprendere tutte le categorie, con un'abbondanza di zone di

neve e zone urbane; questi infatti saranno i casi in cui l'algoritmo verrà messo più in difficoltà quindi è necessario dedicargli un'attenzione maggiore.

Inoltre gli autori hanno creato per ogni immagine una "maschera", ovvero un'immagine in scala di grigi a singolo canale, dove il valore 255 è usato nei pixel corrispondenti alle nuvole, il valore 128 è associato ai pixel di background, e lo 0 marca i pixel senza informazioni. Queste immagini sono chiamate lables, e rappresentano la verità di fondo (ground truth) che il modello dovrà imitare. La cosa fondamentale da notare di queste immagini e che sono state costruite "a mano": i creatori del dataset, con l'utilizzo di programmi di editing delle immagini che aumentano la produttività, hanno dovuto manualmente identificare ogni pixel di nuvola all'interno del dataset. Quindi la scarsità delle immagini satellitari non è un problema di per sé, ma lo è la creazione delle rispettive lables che richiede tempo perché è un lavoro lento e non automatizzabile; questo crea dei vincoli e pone un limite al numero di immagini utilizzabili.

2.2 Note tecniche

Le immagini sono inizialmente memorizzate in un formato poco comodo e non utile per poter essere usate nella rete neurale, in particolare ci sono i seguenti problemi:

- sono troppo grandi (100 MegaByte per ogni banda a $10m$), il che le rende scomode da gestire e da tenere in memoria, specialmente se si parla di VRAM ovvero la memoria delle schede video, in quanto queste ultime hanno delle capacità computazionali ottime per questi task, ma una memoria decisamente minore rispetto al sistema;
- l'assenza di informazioni in alcune parti dell'immagine, il che è dovuto al fatto che il satellite che mappa la superficie terrestre esegue delle scansioni in obliquo rispetto all'equatore, e nel momento in cui si vanno ad estrarre delle immagini quadrate è possibile che i contorni di quest'ultima non siano compresi dalla scansione del satellite.

La soluzione usata per risolvere questi problemi è stata quella di dividere il dataset in immagini molto più piccole (chiamate crop), dove le bande a $10m$ diventano immagini quadrate di 380 pixel di lato; queste immagini poi vengono salvate in memoria in modo tale che possano essere recuperate facilmente nel momento opportuno. Inoltre, durante il processo è possibile scartare i crop che contengono il vuoto, e così facendo si preserva la maggior parte dell'immagine originale e si eliminano i pixel di vuoto.

Per questa occasione, ho creato uno script Python dedicato all'estrazione dei file dal dataset, il quale organizza i crop nel file system memorizzandoli in modo tale che per ogni immagine originale venga creata una cartella con lo stesso nome, contenente molte sottocartelle numerate a scacchiera, all'interno delle quali ci sono 3 immagini, ottenute mettendo insieme le bande della stessa risoluzione. Questo sistema permette un facile e rapido accesso ai singoli crop, e all'occorrenza anche la capacità di filtrare le immagini per nome.

Le immagini del dataset vengono memorizzate nel formato TIF (o TIFF, Tagged Image File Format), ovvero un tipo di file estremamente versatile ed adatto a

memorizzare immagini scientifiche per le successive elaborazioni; è un formato di tipo raster(bitmap) che in questo caso contiene immagini non compresse o compresse con tecniche "lossless" (ovvero senza perdita di informazione), oltre a svariati altri meta-dati (come le coordinate dell'immagine), i quali però vengono ignorati perché non utili ai fini del task di cloud detection. Per interagire con questo formato ho usato la libreria GDAL[5] (Geospatial Data Abstraction Library) che permette di leggere le immagini come delle matrici Numpy, le quali sono estremamente versatili.

2.3 Le trasformazioni

Il problema delle poche immagini (con label) a disposizione riguarda gran parte dei progetti di Machine Learning, e nel tempo si sono trovati numerosi metodi e sistemi per ovviare al problema. Una delle tecniche più usate è quella della "data augmentation", ovvero aumentare artificialmente la grandezza del dataset aggiungendo immagini generate a partire dai dati già esistenti; in particolare per le immagini ho attuato una serie di semplici trasformazioni quali: inversione orizzontale, inversione verticale e rotazione di 90 gradi. L'idea base è che un'immagine specchiata, su uno dei due o su entrambi gli assi, o ruotata può essere vista dal modello come una sequenza di feature totalmente nuova, e questo aiuta il modello a riconoscere gli oggetti nelle immagini indipendentemente dalla loro posizione o rotazione.

Queste tre trasformazioni possono ingrandire il dataset di un fattore 8, perché ognuna di esse di fatto duplica il numero di immagini presenti nello step precedente (l'ordine delle trasformazioni non ha importanza) e come scelta implementativa ho deciso di non memorizzare le trasformazioni sul disco, ma di eseguirle solo nel momento in cui fosse richiesto di accedere ad una data immagine, questo perché:

1. Le nuove immagini non hanno nuove feature, quindi non aggiungono informazioni al modello se non l'ordine spaziale;
2. Il nuovo dataset sarebbe troppo grosso (circa 256 GigaByte);
3. Queste operazioni non rappresentano un collo di bottiglia quindi non c'è bisogno di memorizzare per avere una sorta di cache.

Esistono anche molti altri tipi di tecniche, le quali però non sono state implementate in questo progetto perché non utili o controproducenti per questo specifico caso d'uso; tra queste vi sono:

- Estrazione di crop in posizioni casuali: per poter estrarre dinamicamente dei crop bisognerebbe avere l'immagine originale già interamente caricata in RAM, ma questo processo è da evitare perché crea dei colli di bottiglia durante l'allenamento; per questo, al contrario, ho optato per una divisione statica delle immagini.
- Zoom-in dei crop: questa tecnica prevede l'estrazione di una porzione di un'immagine in modo da avere più visioni dello stesso oggetto con e senza un ampio contesto intorno; se pur potenzialmente utile nel caso delle nuvole, i sotto-crop andrebbero poi usati insieme ad algoritmi di up-scaling che creerebbero delle

feature (quindi dei pixel) artificiali o addirittura feature che non combaciano più con la maschera nella label.

- Rotazioni per tutti gli angoli: le rotazioni per 180° e 270° sono facilmente implementabili come quella da 90° , ma combinate alle 2 inversioni non aggiungerebbero nuove combinazioni, quindi sarebbero inutili (per esempio, è possibile ottenere una rotazione di 180° anche semplicemente usando una volta entrambe le inversioni); le rotazioni per angoli totalmente casuali invece non sono applicabili perché ruotare una scacchiera di un angolo non retto, porterebbe ad dover generare la nuova scacchiera interpolando quella precedente, quindi ancora una volta generando feature non esistenti.
- Filtri di luce: è possibile modificare le immagini in modo da cambiarne i valori caratteristici come il contrasto o la saturazione; questa operazione è in generale utilissima per quei modelli che operano su immagini provenienti da origini diverse (macchine fotografiche con diverse tecnologie), per le quali è importante che il riconoscimento di un oggetto non dipenda da "sole" variazioni nella gradazione dei colori. In questo caso specifico però l'algoritmo è pensato per l'elaborazione di immagini provenienti esclusivamente dai satelliti della missione Sentinel, quindi non è necessario rendere il modello resiliente alle variazioni di diversi sensori.

Capitolo 3

La rete

In questo progetto è stato usato un modello di rete neurale convoluzionale; la quale ha come obiettivo quello di essere leggera, sia da un punto di vista computazionale che di spazio in memoria, ma allo stesso tempo riesca a gestire tutte le 13 bande prodotte dai satelliti.

La prima difficoltà che si incontra nella progettazione di un modello con queste caratteristiche riguarda la gestione dei 3 gruppi di bande a diversa risoluzione: se si scegliesse di usare modelli classici come U-Net [6] bisognerebbe scalare tutte le immagini ad una sola delle 3 dimensioni, il che porta dei problemi sia nel caso in cui si faccia un down-scaling (in quanto si avrebbe perdita di informazioni su alcuni canali) che in quello in cui si fa up-scaling (verrebbero introdotte delle feature artificiali che non rispecchiano le vere analisi). Una soluzione migliore quindi implica la creazione di un nuovo modello (il CD-FM3SF [4]) che sarà in grado di accettare separatamente i 3 gruppi in diversi step della rete a risoluzione nativa. Specularmente, per migliorare la capacità di allenamento del modello, si è scelto di collezionare anche 3 diversi output (che rispecchiano le stesse dimensioni delle rispettive immagini in ingresso), in quanto accedendo ad output intermedi si può avere più controllo su i parametri posti in profondità nella rete.

3.1 Panoramica del modello

Come si può vedere in Figura 3.1 il modello è composto da differenti moduli che verranno analizzati nei prossimi paragrafi.

3.1.1 Concatenation and Sum

Queste due operazioni sono molto semplici e al contempo molto utili, ed entrambe sono usate per aggregare più feature provenienti da diversi layer o processi (ammesso che gli oggetti abbiano la stessa dimensione, ad esempio un'immagine 100x100 può essere sommata solo ad un'altra 100x100). L'operazione di concatenazione è usata in molte architetture avanzate come U-Net[6], Inception[7], e DenseNet[8] perché permette di concatenare insieme più layer per formarne uno solo con dimensioni maggiori in modo da preservare le feature prodotte da differenti layer dall'algoritmo. L'operazione di somma invece ha lo scopo di combinare insieme diverse feature in un

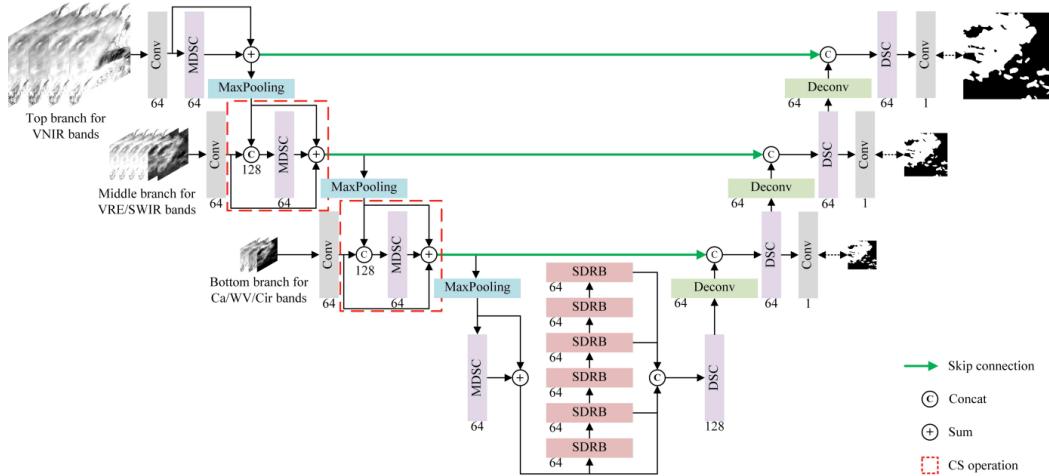


Figura 3.1. Struttura del modello CD-FM3SF. Il numero sotto ogni componente indica il numero di feature map che esso produce

unico risultato, e nel farlo, riduce la quantità dei dati necessari per rappresentarle: questa sarà quindi una delle operazioni fondamentali per avere un modello leggero ma completo. Entrambe le operazioni di concatenazione e di somma sono utili per implementare altri concetti e parti dell'algoritmo molto importanti quali "Skip connections" e "Residual block".

3.1.2 Skip connection e Residual block

Le connessioni di salto, o skip connections, sono un metodo sofisticato e intuitivo che permette all'input di uno specifico strato in una rete neurale di "evitare" uno o più strati e di essere successivamente sommato o concatenato al loro output; quindi questo rappresenta un importante cambio rispetto al classico flusso base per cui l'informazione passa in modo sequenziale attraverso tutti i layer della rete. Questo concetto è fondamentale nella creazione dei blocchi residui, o residual block (visibili in Figura 3.2) che sono aggregazioni di strati, convoluzionali e funzioni di attivazione, attraversati da una connessione di salto.

In una normale situazione la rete dovrebbe riuscire ad approssimare la funzione F che lega l'input x alla ground truth $H(x)$, mentre nei blocchi residui abbiamo che

$$H(x) = F(x) + x \implies F(x) = H(x) - x$$

Quindi il modello, cercando di approssimare F si sta allenando per trovare la funzione che descrive la differenza tra l'input e l'output atteso. Questo approccio è stato sviluppato per la prima volta nella rete ResNet[9] (da cui prende il nome), e si è rivelato molto efficace nel risolvere il problema della degradazione del gradiente ovvero quella problematica che si crea in reti neurali molto profonde dove l'allenamento dei layer più profondi è molto più difficile.

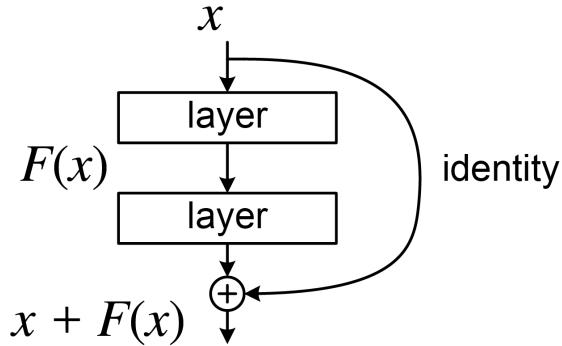


Figura 3.2. Esempio di un semplice residual block

3.1.3 Convolution layer

Questo layer è indicato nella Figura 3.1 con il nome di Conv (blocchi in grigio) ed è l'elemento base delle reti neurali convoluzionali. In sostanza viene applicata l'operazione matematica chiamata convoluzione tra le immagini di input e i filtri (chiamati anche kernel); l'obiettivo è quello di estrarre informazioni spaziali dalle feature. Questo è il concetto fondamentale che differenzia le CNN dalle reti neurali “fully connected”, in quanto queste ultime trattano le feature come vettori unidimensionali e non come matrici, quindi perdono il concetto di spazialità. Al contrario, una CNN è in grado di trovare pattern locali quali angoli o lati, mettendo in relazione i pixel più vicini tra loro, e questo fa sì che le reti convoluzionali siano la prima scelta quando il task da eseguire prevede l'analisi di immagini.

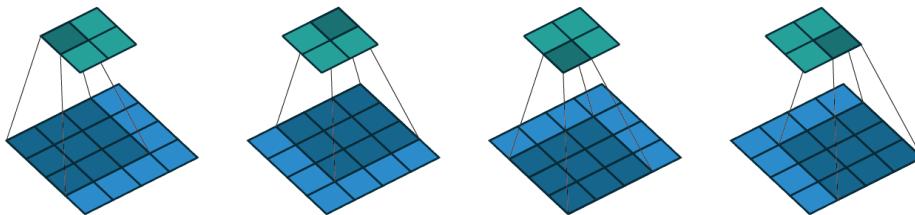


Figura 3.3. Esempio di convoluzione su di un'immagine $4 \cdot 4$ con un kernel $3 \cdot 3$ che ha come output un'immagine $2 \cdot 2$

Usare questo layer però fa aumentare la complessità del modello, in particolare se considerassimo un input di M canali, un output di N e un kernel quadrato di dimensione $K \cdot K$, allora un solo layer convoluzionale aggiungerebbe

$$\#Parametri = M \cdot K \cdot K \cdot N + M \cdot N$$

Questo perché ogni output è formato da N canali derivanti dalla somma di M convoluzioni sui rispettivi input (quindi $M \cdot N$ operazioni), ed ognuna di queste convoluzioni ha bisogno del suo filtro composto dai $K \cdot K$ parametri. Inoltre è

presente un ulteriore parametro per ogni convoluzione che è il bias, il quale aggiunge un ulteriore grado di libertà e fa sì che la convoluzione non sia solo una semplice trasformazione lineare: il che porta a dei vantaggi come una più veloce convergenza e ad una migliore espressività del modello (tra cui un migliore adattamento alle threshold imposte ad esempio dalle relu). Altri 2 parametri molto importanti per questa operazione sono lo stride e la dilatazione:

- Lo "stride" è il passo con cui il filtro convoluzionale si muove sull'input. Uno stride di 1 significa che il filtro si muove di una unità per volta, così ogni pixel dell'input viene considerato; mentre uno stride maggiore di 1 fa sì che alcuni pixel vengano saltati, riducendo quindi la dimensione dell'output, il che può essere utile per diminuire la complessità del modello;
- La "dilatazione" invece modifica il modo in cui il filtro convoluzionale interagisce con l'input; in una convoluzione standard (dilatazione 1), i pixel del filtro sono adiacenti tra loro. Ma con una dilatazione maggiore di 1, vengono introdotti degli spazi vuoti tra i pixel del filtro, permettendo ad esso di coprire un'area più ampia dell'input. Le convoluzioni dilatate sono particolarmente utili quando si vuole aumentare il campo ricettivo del filtro senza aumentare il numero di parametri del filtro stesso.

3.1.4 Deconvolution layer

Questo layer è indicato in Figura 3.1 con il nome “Deconv” (blocchi in verde) ed è implementato con la classe ConvTranspose2d di Pytorch; come ricordato anche nella documentazione ufficiale [10] però, questo layer non sta veramente facendo l'operazione matematica inversa della convoluzione, ma più semplicemente è l'implementazione di un up-sample con convoluzione: si parte inserendo degli zero tra i pixel di un'immagine (il numero varia a seconda del parametro stride) e facendo poi una normale convoluzione, che attraverso gli zeri aggiunti porta ad avere un'immagine di output più grande.

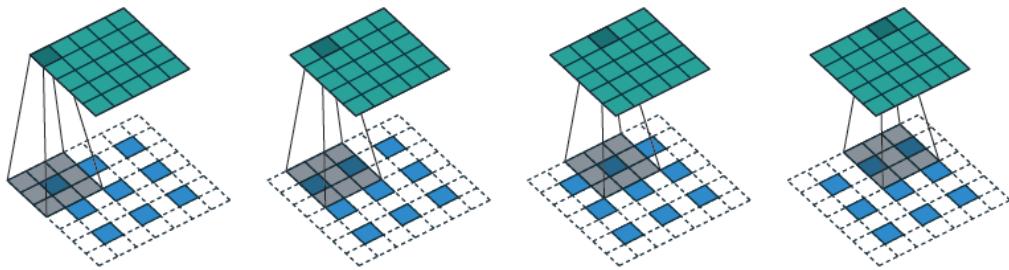


Figura 3.4. Esempio di deconvoluzione su di un'immagine 3×3 aumentata a 7×7 con un kernel 3×3 che ha come output un'immagine 5×5

Per quanto riguarda l'analisi della complessità, tolta la parte dello aggiungere zeri che non è un'operazione che necessita di parametri da allenare, rimane solo la

parte della convoluzione, la quale, come abbiamo visto nel capitolo precedente porta ad avere altri

$$\#Parametri = M \cdot K \cdot K \cdot N + M \cdot N$$

Per gli stessi motivi della convoluzione classica.

3.1.5 Pooling layer

Questo layer è indicato in Figura 3.1 con il nome di MaxPool (rettangoli celesti) ed è usato per fare down-scaling delle feature map senza necessità di ulteriori parametri: consiste nel prendere l'elemento massimo da un sottoinsieme di elementi, generalmente da una finestra di dimensioni fisse, e trasmettere questo valore massimo come rappresentante dell'intera area. Questo processo non solo contribuisce a ridurre la complessità computazionale delle fasi successive della rete, ma migliora anche la tolleranza della CNN alle piccole variazioni o distorsioni nello spazio dell'input.

3.1.6 Rectified linear unit

La Rectified linear unit, abbreviata con ReLU, è una delle più famose funzioni di attivazione utilizzate nelle reti neurali; non è inserita nella Figura 3.1 ma è sempre presente dopo ogni convoluzione nell'encoder e dopo ogni deconvoluzione nel decoder. Matematicamente, è definita come

$$F(x) = \max(0, X)$$

Ciò significa che per valori negativi la funzione restituirà zero, mentre per valori positivi restituirà l'input stesso. In generale, l'utilità di avere una funzione di attivazione non lineare in una rete neurale risiede nella capacità di modellare e apprendere relazioni non lineari tra le variabili di input e output: senza una funzione di attivazione non lineare, indipendentemente dalla profondità o complessità della rete, l'intera architettura potrebbe rappresentare solo trasformazioni lineari, limitando severamente la capacità della rete di generalizzare su dati complessi che necessitano relazioni non lineari. Tra tutte le funzioni di attivazione non lineari, la ReLU ha il vantaggio di essere un'operazione estremamente semplice; essa ha anche dei difetti che si possono risolvere con delle varianti leggermente più complesse come ad esempio la "Leaky ReLU", ma in questo modello viene usata solo la versione base.

3.1.7 Depth-wise separable convolution layer

Questo modulo, detto DSC (rettangoli viola nel decoder in Figura 3.1), è composto da due layer di convoluzioni: il primo, detto depth-wise è un layer che applica M filtri agli M canali in input separatamente, ovvero ognuno degli M canali in output è il risultato del relativo canale in input dopo la convoluzione con il relativo filtro e nient'altro; il risultato di questa operazione viene poi processato da un normale layer convoluzionale, il quale però applica un kernel di dimensione $1 \cdot 1$.

Analizzando i parametri in gioco otteniamo che, per il primo layer esistono M filtri $K \cdot K$, mentre per il secondo layer, riprendendo la formula della convoluzione

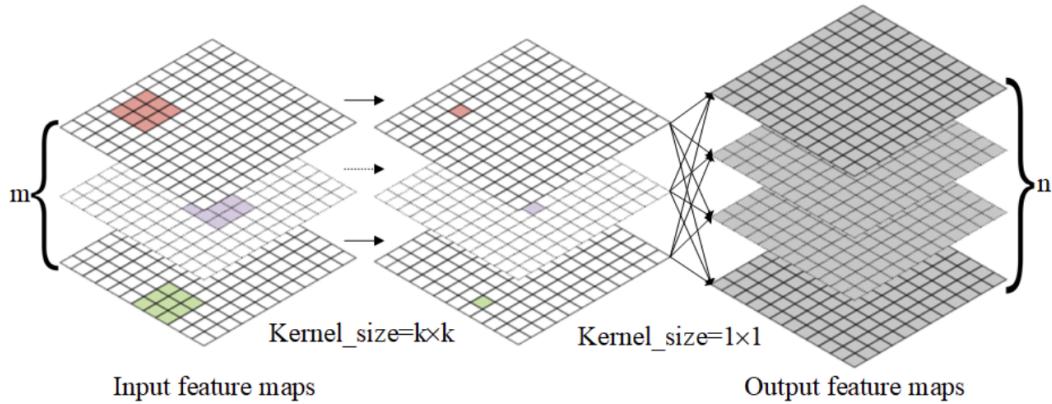


Figura 3.5. Schema convoluzioni in un blocco DSC

semplice $M \cdot K \cdot K \cdot N + M \cdot N$ e sostituendo per $k = 1$ otteniamo $M \cdot N$: quindi in totale i parametri utilizzati da questo modulo sono

$$\#Parametri = M \cdot K \cdot K + M \cdot N$$

In questo caso non si è fatto uso dei bias e possiamo notare che un modulo del genere sta imitando una convoluzione ma con molti meno parametri.

3.1.8 Shared and dilated convolution residual block

Questo modulo chiamato SDRB (rettangolo rosso in Figura 3.1) è un componente più complesso rispetto agli altri presenti in questo modello, e il suo scopo è quello di estrarre informazioni spaziali, a corto e a lungo raggio, usando la tecnica dei blocchi residui. Esso è composto da due layer base che sono delle varianti del normale layer convoluzionale:

1. Shared Convolution Layer, ovvero un componente che usa un solo filtro per fare una convoluzione per ogni canale;
2. Dilated Convolution Layer, che è una normale convoluzione che sfrutta dei filtri dilatati.

Questi due layer combinati formano il blocco base (SDC) per un SDRB, il quale è composto dalla concatenazione di due blocchi base identici usati come residual block, in quanto è presente anche una skip connection direttamente dall'input all'output.

L'analisi dei parametri in gioco per questo blocco non è eccessivamente complicata, infatti partendo dai blocchi base: lo shared layer applica un solo filtro indipendentemente dai canali in ingresso, quindi $K \cdot K$ parametri; mentre la Dilated convolution essendo una normale convoluzione dilatata rispetta comunque la formula del $M \cdot K \cdot K \cdot N + M \cdot N$. Essendo questi due layer usati due volte, di fatto il numero di parametri in gioco è doppio

$$\#Parametri = 2 \cdot (M \cdot K \cdot K \cdot N + M \cdot N + K \cdot K)$$

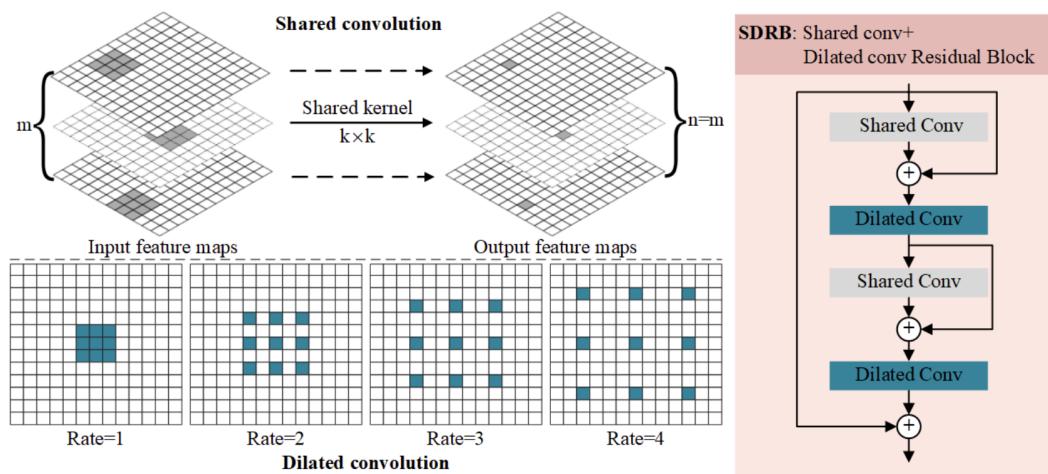


Figura 3.6. Schema convoluzioni e diagramma di un blocco SDRB

In questa rete i blocchi SDRB vengono usati più volte ma con diversi parametri: in particolare, a coppia, varia la dimensione dei kernel e le dilatazioni, con rispettivi valori di $5 \cdot 5$, $7 \cdot 7$ e $9 \cdot 9$ per i kernel, e 2, 3 e 4 per il parametro dilatation.

3.1.9 Mixed depth-wise separable convolution layer

Questo modulo, abbreviato con MDSC (rettangolo viola nell'encoder in Figura 3.1) è pensato per estrarre e fondere insieme le informazioni provenienti da contesti di medio e corto raggio, infatti prevede come primo step l'esecuzione in parallelo di diverse Shared Convolution, di cui una con un kernel $3 \cdot 3$ e l'altra con un kernel $5 \cdot 5$; i loro risultati vengono poi concatenati e passati ad una normale convoluzione di kernel $1 \cdot 1$.

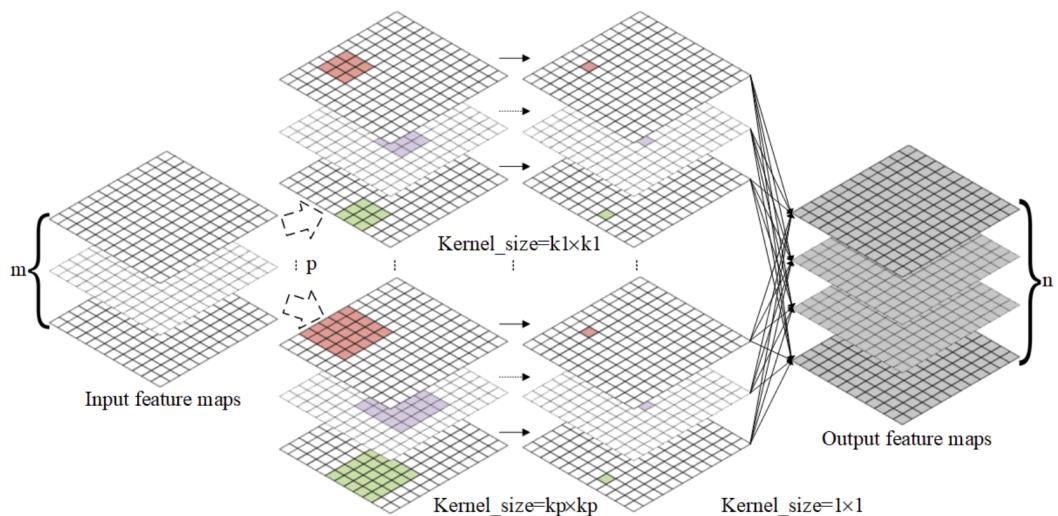


Figura 3.7. Schema convoluzioni in un blocco MDSC

Quindi i parametri usati in tutto sono:

$$\#Parametri = 2 \cdot (M \cdot K \cdot K) + 2 \cdot M \cdot N$$

3.2 Considerazioni

Tutte le attenzioni poste sulla complessità della rete neurale hanno portato di fatto ad avere 1.107.439 parametri, ed a confermarlo è la libreria stessa di PyTorch che tramite il comando

```
1 model = CDFM3SF([4, 6, 3], gf_dim=64)
2 total_params = sum(p.numel() for p in model.parameters()
    if p.requires_grad)
```

ci restituisce il numero totale di parametri allenabili del modello. Questo è un ottimo risultato, infatti prendendo il paper di riferimento [4], i ricercatori hanno creato un modello estremamente simile da 1,01 milioni di parametri, quindi il modello usato in questo progetto è in linea con lo stato dell'arte in questo campo, ed ha solo il 3,95% dei parametri rispetto a modelli usati in vecchie soluzioni come le reti U-Net a 28 milioni di parametri.

Capitolo 4

L’allenamento

4.1 Hardware

Tutto questo interesse verso la complessità del modello è dovuto alla volontà di creare dei sistemi in grado di essere eseguiti in un tempo ragionevole anche su hardware economico. Per il train di questo modello infatti è stato possibile usare un computer con un processore Intel i7 7740x, una scheda video GTX 1060 con 6GB di VRAM, 16 GB di RAM e un disco allo stato solido di Intel.

In generale per il processo di train sono necessarie più risorse rispetto al solo compito di esecuzione, e i motivi sono molteplici: il primo problema appare già durante la gestione dei dati per il train, infatti questi occupano molto spazio in memoria, quindi è estremamente improbabile che un computer economico abbia abbastanza ram per mantenere sempre input e label (o almeno, senza che il sistema operativo cominci ad usare algoritmi di swap in e swap out per le pagine in memoria); questo porta a dover memorizzare le immagini su un’unità di memoria interna, la quale può essere un HDD molto economico o un più costoso SSD. Adottare questo sistema però implica che ogni volta che si vuole utilizzare delle immagini si debba fare un accesso al disco, che a causa delle basse prestazioni che offrono rispetto alla RAM, potrebbe rappresentare un collo di bottiglia: per questo la scelta dovrebbe ricadere su di un SSD con delle buone prestazioni (il quale ha comunque un rapporto GB/prezzo più vantaggioso rispetto alla RAM).

Questo problema invece non è rilevante nel momento in cui l’algoritmo entra in esercizio, dove possono accadere più situazioni:

- Analisi massiva di dati, per cui come dice il nome, il task prevede l’analisi di una quantità di dati talmente grande che comunque non potrebbe essere memorizzata nella RAM di nessun tipo di macchina;
- Utilizzo di batch di dimensioni ridotte, come nei casi di sistemi non automatizzati che interagiscono con gli utenti, potrebbe verificarsi un rallentamento del processo a causa della velocità umana, che è generalmente inferiore a quella del computer, a meno che il modello sia leggero a sufficienza.

Sempre parlando della gestione iniziale dei dati, è anche necessario operare tutta quella preparazione del dataset descritta nella sezione 2.2, la quale, nonostante sia

un processo che richiede molto tempo, è necessario farla una sola volta indipendentemente da quanto lungo sia l’effettivo training. C’è da dire però che anche in questo caso un hardware di alta qualità aiuterebbe molto, in quanto parliamo di un processo che accede spesso in lettura e scrittura sul disco, e allo stesso tempo può essere classificato come “cpu intensive”: quindi in questo caso aver avuto a disposizione un processore i7 intel e un performante SSD ha permesso di eseguire queste operazioni in un tempo ragionevole (23 minuti e 50 secondi) in quanto è stato possibile sfruttare l’alta frequenza di clock e i 4 core 8 thread del processore.

Anche se il modello è effettivamente tra i meno complessi in questo ambito per numero di parametri, comunque le operazioni usate sono onerose; infatti come descritto nella sottosezione 3.1.3, le convoluzioni sono effettivamente molto semplici dal punto di vista logico, ma il problema risiede nella quantità di calcoli necessari. Per questo tipo di task quindi si può usare dell’hardware specializzato come le GPU, le quali sono in grado di offrire un alto throughput, soprattutto in calcoli ripetitivi, sfruttando anche la possibilità di parallelizzare il carico di lavoro. Confrontando infatti CPU e GPU nel solo passo di forward con una batch di 4 immagini, notiamo che la CPU completa il task in 2,00 secondi, mentre la GPU impiega solo 0,16 secondi, ovvero un aumento di prestazioni del 1.250%; in questo calcolo è stato ovviamente considerato anche il tempo necessario per trasferire i dati dalla RAM alla VRAM, anche se in questo caso il delay è trascurabile in quanto è la complessità dei calcoli il vero collo di bottiglia. Durante la fase di train però non è sufficiente solo fare il passo di forward, cioè la predizione, ma bisogna applicare anche l’algoritmo di backpropagation per calcolare i gradienti e successivamente aggiornare i pesi: tenendo in considerazione anche questi calcoli, e usando sempre la scheda video (in quanto miglior componente), si raggiungono i 0,25 secondi per ogni batch.

4.2 Metodo e implementazione

Come già citato in precedenza, per l’implementazione del progetto è stata usata la libreria di PyTorch; più correttamente però, bisognerebbe chiamarlo framework e non libreria: infatti, se pur questi concetti vengono comunemente interscambiati, in informatica per framework si intende una struttura o un’infrastruttura di sviluppo più ampia e completa rispetto a una libreria, per la quale il flusso di esecuzione del programma è sotto il controllo di PyTorch; in questi casi, il programmatore deve solo occuparsi dell’implementazione di alcuni blocchi di codice (che appunto verranno usati dal framework nel momento opportuno), e nel farlo deve seguire alcune regole di design imposte dal framework.

4.2.1 Dataset

Come anticipato, il framework definisce un design preciso e il flusso di esecuzione, ma lo fa in modo generico, ovvero in modo che sia riutilizzabile per la maggior parte dei task di machine learning; per questo, come prima cosa è necessario che il programmatore definisca una classe che sia in grado di gestire il dataset. La mia classe MyDataset infatti costituisce un’astrazione in grado di semplificare la gestione del dataset, nascondendo la complessità di accedere al disco e di trasformare le

immagini. Questa espone dei metodi che verranno usati da Pytorch per estrarre i dati:

```

1 def __len__(self):
2     return ...
3
4 def __get__(self, index)
5     return ...

```

che rispettivamente calcolano il numero di immagini nel dataset e restituiscono la coppia input label i -esima dato un certo i come indice. Questa classe verrà poi usata da un DataLoader che organizzerà gli input in batch da 4 elementi, operando la funzione di "shuffle": ovvero quella per la quale il loader sceglie casualmente delle immagini dal dataset per evitare di comporre i batch sempre con le stesse immagini e sempre nello stesso ordine (sapendo che gli step dell'allenamento avverranno sui singoli batch, è meglio comporli nel modo più variegato possibile, in modo da non introdurre bias).

Come spiegato precedentemente il dataset è memorizzato sul disco, quindi la funzione "get" come prima cosa si occuperà di caricare i dati necessari, e successivamente userà l'oggetto memorizzato "transform" per trasformare i dati facendo quello che si chiama "data augmentation". Le trasformazioni che verranno applicate sono elencate dentro la lista passata alla classe di PyTorch "Compose", la quale si occuperà di trasformare l'input passandolo in ordine per ognuna delle trasformazioni elencate nella sezione 2.3.

4.2.2 Optimizer

In generale, un "optimizer" o ottimizzatore è un algoritmo utilizzato nel campo del machine learning per minimizzare o massimizzare una funzione obiettivo, che in questo specifico caso è una "loss function" da minimizzare (la funzione di loss verrà affrontata nella sottosezione 4.2.5).

L'obiettivo di un optimizer è di aggiustare i parametri del modello in maniera tale da migliorare l'efficienza del modello nel risolvere il problema. Lo Stochastic Gradient Descent (SGD) è l'ottimizzatore più semplice e più noto: esso aggiorna i parametri del modello seguendo la direzione opposta al gradiente della funzione obiettivo; tuttavia, SGD è sensibile a vari problemi come la presenza di minimi locali e oscillazioni. Una variante comune di SGD è il "momentum", che cerca di affrontare queste difficoltà considerando anche i gradienti passati nell'aggiornamento dei parametri, fornendo una sorta di "memoria" che attenua le oscillazioni.

In questo progetto invece è stato usato Adam[11] (Adaptive Moment Estimation), che è un altro ottimizzatore che combina le idee di SGD e momentum con tassi di apprendimento adattivi per ciascun parametro. Questo rende Adam particolarmente efficace nell'affrontare una vasta gamma di problemi di ottimizzazione, poiché offre un compromesso ottimale tra la velocità di convergenza e la stabilità numerica; esso infatti è altamente efficace grazie alla sua capacità di mantenere una stima ponderata del momento del primo e del secondo ordine dei gradienti. Questa caratteristica gli consente di adattare dinamicamente la dimensione del passo di aggiornamento per

ciascun parametro durante il processo di addestramento, portando a una convergenza più rapida e stabile.

Una delle caratteristiche fondamentali che rende l'optimizer Adam efficace sono i suoi due parametri, β_1 e β_2 : questi controllano rispettivamente la stima ponderata esponenziale del momento del primo e del secondo ordine dei gradienti, ovvero sia l'effetto di "smorzamento" delle oscillazioni nei gradienti che l'effetto "freno" per evitare aggiornamenti troppo grandi.

4.2.3 Scheduler

Gli scheduler per il tasso di apprendimento rappresentano un complemento fondamentale agli ottimizzatori, offrendo la possibilità di modulare dinamicamente il tasso di apprendimento nel corso dell'addestramento del modello. L'obiettivo è iniziare con un tasso di apprendimento sufficientemente alto per favorire una rapida convergenza nelle fasi iniziali, per poi diminuirlo in maniera controllata, consentendo al modello di raggiungere una soluzione più precisa e stabile vicino a un minimo ottimale della funzione obiettivo.

In questo elaborato, è stata impiegata la classe "ExponentialLR" di PyTorch: questo scheduler aggiorna il tasso di apprendimento dopo ogni epoca, applicando la formula

$$\text{lr aggiornato} = \text{lr}_{\text{iniziale}} \cdot \gamma^{\text{epoca}}$$

dove γ è il fattore di decadimento predefinito, tipicamente compreso tra 0 e 1. Attraverso questa formula, "ExponentialLR" permette una decrescita esponenziale, la quale si è dimostrata la più efficacie nei test fatti durante l'allenamento, soprattutto se comparata ad altre funzioni come quelle lineari.

4.2.4 Scelta degli iperparametri

Nei precedenti capitoli sono stati affrontati gli optimizer e gli scheduler, i quali gestiscono aspetti collegati ma molto diversi; i primi servono ad allenare i parametri del modello, mentre i secondi gestiscono il learning rate che è un iperparametro. Nel contesto del machine learning infatti, i parametri sono le variabili che il modello deve modificare durante la fase di addestramento per fare previsioni più accurate, mentre gli iperparametri sono configurazioni esterne che vengono stabilite a priori e che guidano il processo di addestramento. La cosa importante da sottolineare è che i parametri cambiano in base ai dati, mentre gli iperparametri sono fissati manualmente.

Il primo e più importante iperparametro è il learning rate il quale determina la grandezza dei passi durante l'ottimizzazione del modello; un learning rate elevato può far convergere il modello rapidamente ma rischia di far saltare il minimo globale, mentre uno troppo basso può rendere l'addestramento lento e inefficiente.

Altri iperparametri presenti in questo modello sono:

- Il fattore di decadimento γ di ExponentialLR;
- i parametri β_1 e β_2 di Adam;

- i rapporti tra le loss dei tre output (che verranno affrontati nella sottosezione 4.2.5);
- i parametri della struttura del modello (che per come è implementato, il numero di convoluzioni interno è controllato da un parametro configurabile nel costruttore della classe).

Come scelta implementativa, dovuta anche al fattore tempo, ho deciso di tenere costanti i rapporti tra le loss e la struttura del modello, mentre per la scelta degli altri iperparametri ho usato l'algoritmo chiamato Random Search; questo, è una variante del Grid Search, il quale è un metodo molto semplice che prevede l'allenamento del modello per ogni combinazione di iperparametri.

La problematica più grande di questo approccio è che appunto un'operazione già lenta e complessa come l'allenamento deve essere ripetuta più volte, e il numero delle volte cresce esponenzialmente rispetto al numero di iperparametri da testare. In questo caso, volendo testare tre iperparametri avremo n^3 allenamenti, dove n è il numero di valori da poter testare per ogni iperparametro; quindi ipotizzando di provare i valori [0.00025, 0.00050, 0.00075] per il learning rate, e [0.1, 0.5, 0.9] come parametri beta, otteniamo un totale di 27 combinazioni. Dato che ogni allenamento impiega circa 72 ore, non è possibile né fare un studio più accurato, né provare tutte le 27 combinazioni.

Il metodo del Random Search invece prevede di estrarre randomicamente k combinazioni dalle n possibili: in questo modo è possibile decidere quante prove fare, con il vantaggio che il generatore randomico prenderà campioni dallo spazio degli iperparametri in modo uniforme (nel senso che, ad esempio, se da un campionamento si ottiene un β_1 piccolo e β_2 grande, il secondo campionamento può dare una combinazione totalmente diversa). Con questo approccio è evidente che non si sta compiendo una ricerca esaustiva degli iperparametri migliori, ma rappresenta il miglior compromesso se consideriamo il tempo impiegato.

4.2.5 Loss e logits

Le funzioni di perdita (o "loss functions" in inglese) sono metriche matematiche utilizzate per quantificare la discrepanza tra le previsioni di un modello e i dati reali forniti come label; tali funzioni sono essenziali per gli Optimizer in quanto forniscono un criterio oggettivo per l'aggiornamento dei parametri del modello.

Idealmente vorremmo che l'output del modello fosse un'immagine in bianco e nero come le label, dove in ogni pixel il valore 0 implica che non ci sono nuvole, mentre 1 identifica un pixel di nuvola nell'immagine reale, in modo da creare una maschera; nella realtà però gli output dei modelli di reti neurali sono delle probabilità, quindi l'output del modello sarà un'immagine in scala di grigi in modo che in ogni pixel ci possa essere un valore compreso tra 0 e 1 che rappresenta la probabilità che quel pixel appartenga ad una nuvola.

In ogni caso, ci troviamo a dover risolvere un problema di classificazione binaria, quindi una funzione di loss ideale è la Binary Cross Entropy: questa funzione misura la divergenza tra la distribuzione vera (i dati osservati) e la distribuzione prevista dal modello, fornendo un indicatore quantitativo dell'accuratezza delle previsioni del modello. Matematicamente è definita come

$$H(y, p) = -[y \log(p) + (1 - y) \log(1 - p)]$$

dove pixel per pixel y è la ground truth (la label composta da 0 o 1), e p è la probabilità predetta di quel pixel di essere nuvola.

Un valore basso della Binary Cross-Entropy indica che il modello ha una buona capacità di prevedere le etichette corrette, mentre un valore elevato suggerisce che le previsioni sono imprecise; questo perché, come si può notare dalla formula, ognuno dei casi si riconduce ad un logaritmo con argomento minore di 0, il quale è sensibile anche alle piccole variazioni e le amplifica, in modo da "consigliare" all'optimizer di quanto modificare i parametri del modello.

Per una questione di efficienza computazionale e stabilità numerica però, la classe di PyTorch per la Binary Cross-Entropy non accetta delle probabilità, bensì dei logits che rappresentano i valori di output del modello prima dell'applicazione della funzione di attivazione sigmoide; per questo motivo la classe che implementa il modello è priva dell'ultima funzione di attivazione, la quale però viene usata manualmente nella fase di test per riprodurre dei risultati più comodi da analizzare.

Questa classe offre anche l'opportunità di impostare un valore chiamato "weight" a cui va fornito il rapporto tra i pixel di nuvola e i pixel di background, in quanto questo servirà a pesare e bilanciare le loss dei due casi: se i pixel di nuvola sono in numero nettamente minoritario rispetto a quelli di background, ma l'algoritmo commette lo stesso numero di errori per entrambe le classi, è evidente che nel caso delle nuvole, la percentuale di errore sia molto maggiore, e quindi maggiore deve essere anche la loss.

Inoltre, per avere più controllo sui layer intermedi della rete e per velocizzare l'allenamento, oltre all'output principale vengono prodotte altre due immagini che provengono da dei layer intermedi, in modo tale da poter calcolare una loss finale con la somma pesata delle loss delle tre immagini, dove il peso maggiore è assegnato al vero output e il peso minore alla loss dell'output del layer più profondo.

Capitolo 5

Test e risultati

La fase di test è estremamente importante perché serve per valutare quanto bene la rete neurale addestrata è in grado di generalizzare a nuovi dati, ovvero dati che non sono stati utilizzati durante la fase di allenamento. Alla fine dell’allenamento porre il modello davanti a nuovi e rappresentativi dati del problema ci permette di valutare oggettivamente le performance che questo modello ha, simulando una situazione in cui l’algoritmo è in esercizio su dati del mondo reale. Un problema non banale che si pone prima della fase di test però è la scelta di quale modello usare, ovvero quale modello è quello che ha delle prestazioni migliori; per trovarlo bisogna tener conto di alcune cose descritte nel prossimo paragrafo.

5.1 Validation

5.1.1 Overfitting

Prendendo una configurazione di iperparametri e facendo allenare il modello per 74 epoche otteniamo apparentemente un andamento ottimo, come mostrato dalla linea blu in Figura 5.1.

Questa linea è formata unendo i valori delle loss medie calcolate sul dataset di train e mostra che la loss media diminuisce dopo ogni epoca, prima in modo vertiginoso, poi con un andamento asintotico tendente ad un intorno di 0.1361755857056327. Questo sembra suggerire che più il modello si allena e più migliora indipendentemente da quante epoche ha già fatto.

Introducendo però un altro gruppo di immagini, le quali non fanno parte del set di allenamento, scopriamo invece una realtà totalmente diversa: come prima cosa si nota un andamento molto meno definito e con grosse oscillazioni, ma soprattutto si distinguono due andamenti opposti a seconda di quante epoche sono state usate per l’allenamento. Analizzando la curva arancione infatti, possiamo osservare che questa ha un andamento decrescente fino alla ventiduesima epoca, pertanto fino a questo punto il modello sta imparando a risolvere il problema di classificazione; ma da quell’epoca in poi la curva tende a risalire, quindi nelle ultime 50 epoche il modello è solo peggiorato.

Questo problema è noto con il nome di overfitting, e si verifica quando il modello si specializza sui dati del test set e nel mentre perde la capacità di generalizzare il problema. Probabilmente nei dati di test c’è un bias a cui noi umani forse non

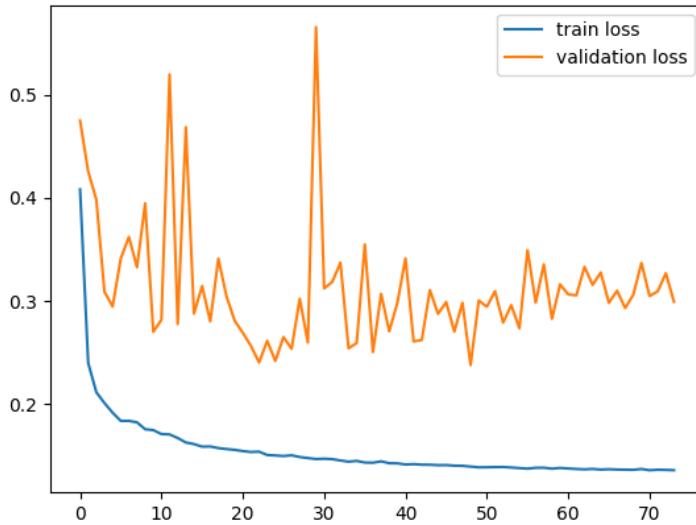


Figura 5.1. Valori di loss medi calcolati per ogni epoca durante l’allenamento syl train set e sul validation set

facciamo caso, ma che la rete neurale ha erroneamente identificato come una feature a cui dare importanza; queste caratteristiche però non fanno parte, in generale, di metriche correlate alle nuvole: per questo nel momento in cui si propone alla rete delle nuove immagini con altri bias, la rete cercherà di fare classificazioni su delle feature sbagliate, peggiorando sempre di più le sue predizioni.

5.1.2 Validation set

Proprio per l’esistenza dell’overfitting è necessario testare più volte il modello in modo da poter selezionare poi quello migliore, il quale come visto in Figura 5.1 non è necessariamente quello che è stato allenato per più tempo. Questo tipo di test però non può essere usato a sua volta come risultato finale, in quanto anche questo gruppo di immagini (per quanto possano essere rappresentative) hanno a loro volta un bias (esattamente come il train set) quindi staremo inconsciamente scegliendo il modello che performa meglio su questo nuovo insieme, e non quello che performa meglio nel problema generale; come detto in precedenza invece il vero test dovrebbe semplicemente essere usato per fare una valutazione secca e finale.

Questo gruppo intermedio di immagini è chiamato validation set e, in estrema sintesi, ha il compito di scegliere il miglior modello alla fine dell’allenamento; questo però non si limita a scegliere il modello che fa riferimento alla loss più bassa tra tutte le epoche, ma permette di scegliere anche il miglior modello tra tante configurazioni di iperparametri. Infatti come spiegato nella sottosezione 4.2.4 sugli iperparametri, per ogni loro combinazione occorre ricominciare ad allenare il modello, in quanto diversi stati di partenza possono portare a notevoli differenze nei modelli finali.

Sia per l'allenamento che per i test sono necessarie grandi moli di dati con label, ma nel momento in cui si divide il dataset in 3 parti (train, validation, test sets) il basso numero di immagini dedicato all'allenamento influisce negativamente sulla possibilità del modello di approssimare il problema, generando dei risultati non soddisfacenti; per questo motivo è una pratica comune dedicare una percentuale maggiore di immagini al train set, il che però implica avere un validation set più piccolo quindi probabilmente con un bias più accentuato. Ovviamente tutto questo è un grosso problema perché è il validation set che decide quale modello scegliere, quindi c'è la necessità di usare delle tecniche per mitigare l'overfitting su di esso.

Una di queste tecniche è la “Shuffle-Split Cross-Validation” in cui inizialmente si divide il dataset solo in train e test set, e solo durante l'allenamento, per ogni combinazione di iperparametri, si divide randomicamente il train set estraendo una piccola porzione di immagini da dedicare al validation set. In questo modo si garantiscono due condizioni importanti:

1. Per ogni combinazione di iperparametri il validation set è sempre diverso, il che permette di isolare i problemi di bias ed overfitting solo all'interno del gruppo di modelli che usano quel set, limitandone (ma non cancellando del tutto) l'effetto.
2. Si continua sempre a seguire il principio per il quale un modello deve essere testato su immagini diverse da quelle su cui è stato allenato per non falsificare i risultati; e questo è vero perché per ogni combinazione di iperparametri la separazione delle immagini viene fatta prima dell'allenamento, quindi qualunque sia il modello scelto alla fine, esso è stato scelto sempre grazie a test indipendenti dalle sue conoscenze.

5.2 Metriche di valutazione

Come citato nella sottosezione 4.2.5, la funzione di loss usata durante l'allenamento rappresenta di per sé una metrica oggettiva per valutare l'errore di predizione del modello; ma, anche se il significato matematico della Binary Cross Entropy è chiaro, questa metrica non ci fornisce tutte le informazioni che vorremmo estrapolare per un'analisi più completa.

5.2.1 Accuracy

La prima e più intuitiva metrica è l'accuracy, che è definita come il rapporto tra i pixel correttamente classificati e il numero totale dei pixel.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

In questa e nelle successive formule verrà usata la seguente notazione:

- TP: "true positive", numero di pixel di nuvola classificati come nuvola;
- FP: "false positive", numero di pixel di background classificati come nuvola;

- TN: “true negative”, numero di pixel di background classificati come background;
- FN: “false negative”, numero di pixel di nuvola classificati come background.

Questa metrica è in grado di darci una prima semplice valutazione sul modello, in quanto è chiaro che un accuracy bassa, ad esempio del 60%, implica che il modello predice correttamente solo 6 pixel ogni 10; inoltre in questo caso specifico in cui la predizione è binaria, possiamo fare anche dei confronti con un classificatore totalmente randomico la cui accuracy sarà mediamente del 50%, in modo da capire quando il modello si discosti da un semplice generatore casuale. Come specificato per la loss, anche l'accuracy risente del problema dello squilibrio tra le due label, infatti in questo caso essendo le nuvole solo il 19% del dataset, un classificatore che predice sempre background otterrebbe un accuracy del 81%; il che senza contesto può sembrare un buon risultato ma in realtà un modello del genere è totalmente inutile.

5.2.2 Precision

La precision è il rapporto tra il numero di pixel correttamente classificati come positivi e il numero totale di pixel classificati positivamente.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Questa metrica è importante nei casi in cui il costo di una classificazione positiva errata è molto alto, ad esempio nel task di cloud removal classificare un edificio bianco come nuvola implicherebbe la rimozione di un dato importante del background. Per come è definita la formula infatti, per massimizzare la precisione bisogna minimizzare il numero di falsi positivi.

5.2.3 Recall

La recall, detta anche “sensitivity”, è il rapporto tra il numero di pixel correttamente classificati come positivi e i pixel realmente positivi.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Questa metrica misura quindi l'abilità del modello di classificare correttamente i pixel di nuvola, in quanto per massimizzare questo valore bisogna ridurre al minimo il numero di falsi negativi; nel task di cloud removal è infatti importante riuscire a non tralasciare eventuali pixel di nuvola in quanto significherebbe ottenere ancora delle nuvole alla fine del processo.

5.2.4 F1-Score

Come scritto nella sottosezione 4.2.5 l'ultimo layer di attivazione della rete neurale è una sigmoide che restituisce una probabilità, quindi un numero continuo da 0 a 1, ma per fare classificazione è necessario inserire una threshold (o soglia) che divida i due stati; in questo modello è stata scelta genericamente una soglia al 50% per

dividere equamente i casi, ma in realtà questo iperparametro è molto importante in quanto, a parità di modello, è in grado di modificare sia la precision che la recall.

Purtroppo però questa modifica potrebbe migliorare una di queste metriche a scapito dell'altra: ad esempio aumentare la threshold significherebbe diminuire il numero di FP (in quanto verrebbero classificati come nuvole solo i pixel con una probabilità alta di appartenere a quella classe) aumentando quindi la precision; ma al contempo aumenterebbero i FN (in quanto tutti i pixel di nuvola classificati come nuvola con una probabilità bassa ora sarebbero classificati come background) quindi una minore recall. Per questi motivi è necessario introdurre anche l'F1-Score, che è una metrica composita in grado di riassumere i risultati di precision e recall.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

5.3 Risultati

Il modello proposto ha dimostrato un'eccezionale capacità nel risolvere il problema di segmentazione semantica delle immagini satellitari, e questo è dimostrato dall'elevato valore di Accuracy del 98,81%. Per quanto riguarda l'F1-Score invece, il modello raggiunge il 93,87% che, nonostante rappresenti pure questo un valore ottimo in generale, ci costringe ad indagare meglio sui valori di precision e recall che sono rispettivamente 96,98% e 90,95%.

L'alta accuratezza e precisione indicano che il modello è altamente affidabile, ma la recall leggermente più bassa suggerisce che il modello è più propenso a mancare alcune nuvole piuttosto che a identificare erroneamente altri oggetti come tali. Questo potrebbe essere un problema in applicazioni dove la rilevazione completa delle nuvole è critica, ad esempio, in studi climatici o applicazioni di previsione del tempo, mentre è ottimo per compiti dove preservare i dettagli in background è fondamentale, in quanto raramente il modello li classificherà come nuvole.

Per renderci conto visivamente di cosa rappresentino queste metriche e questi risultati raggiunti, in Figura 5.2 sono mostrate quattro immagini prese casualmente dal dataset di test, e per ogni riga abbiamo:

1. l'immagine in input di cui vengono mostrati solo i canali RGB;
2. l'output del modello sotto forma di maschera a cui è già stata applicata la threshold;
3. la label originale come confronto;
4. un'immagine in cui sono riportate le differenze tra output e ground truth.

In particolare, le immagini dell'ultima colonna sono fatte in modo da mostrare in nero i pixel correttamente classificati come background, in bianco i pixel correttamente classificati come nuvola, in rosso i pixel classificati erroneamente come background e in blu quelli classificati erroneamente come nuvola. Queste immagini rendono evidente il significato di recall e precision, mostrando come la presenza di rosso è maggiore rispetto al blu, ma comunque la maggior parte dell'immagine (il

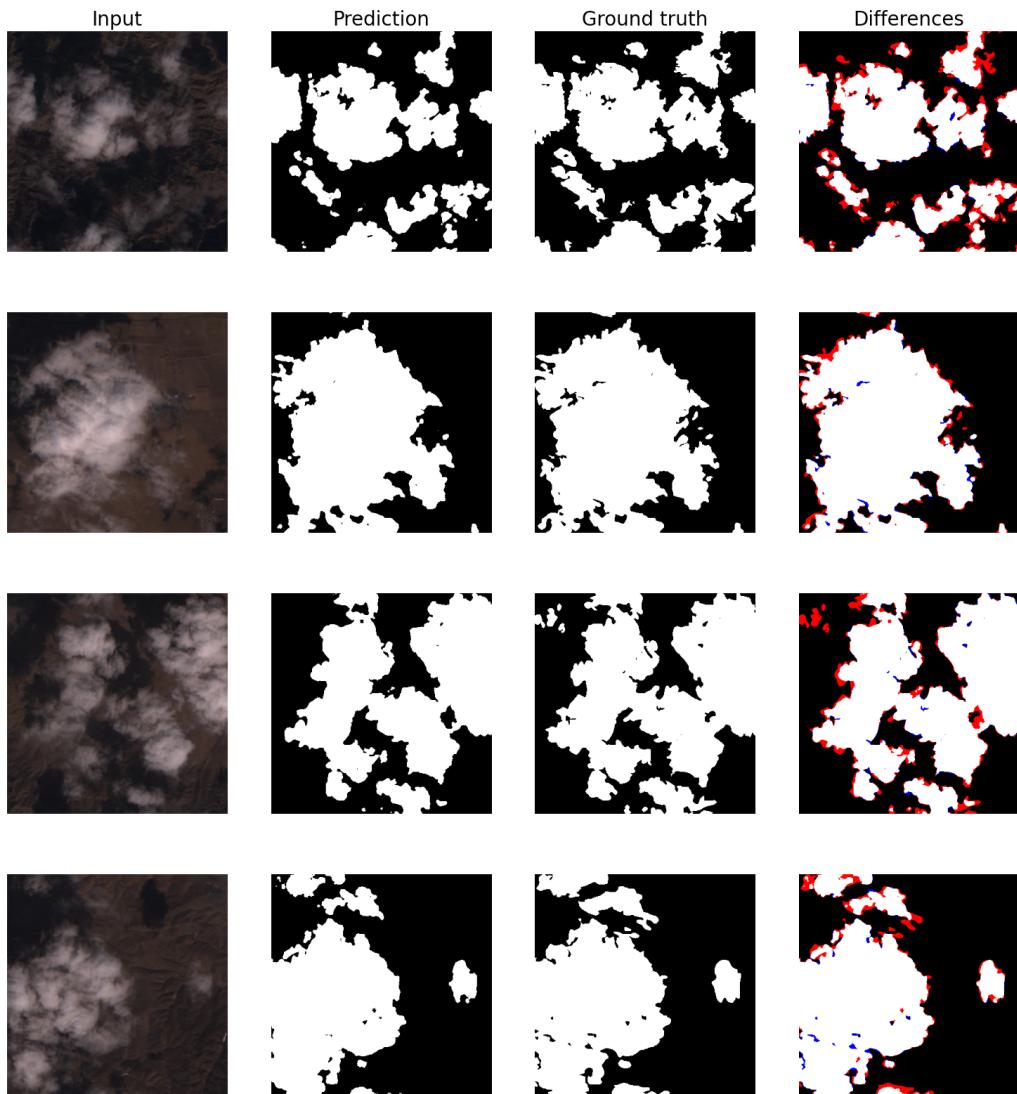


Figura 5.2. 4 esempi di segmentazione dal dataset di test

98,81% appunto) è o bianca o nera; ed è per questo che senza la quarta immagine che enfatizza le differenze con colori accesi, confrontando ad occhio nudo la predizione e la ground truth le troveremo estremamente simili.

5.3.1 Confronto con altri modelli

Una valutazione più estesa richiede anche il confronto con altre soluzioni per valutare come questo nuovo modello si posiziona rispetto agli altri algoritmi; per fare ciò, vengono usati i dati forniti dal paper di riferimento, il quale confronta i seguenti modelli:

- Fmask, ovvero uno degli algoritmi più vecchi e comunemente usati per la classificazione automatica delle nuvole in immagini satellitari; la sua caratteristica

è quella di non essere una rete neurale, ma un algoritmo classico che usa una combinazione di bande spettrali e indici di vegetazione per classificare i pixel.

- Sen2Cor, che è un algoritmo di correzione atmosferica e classificazione sviluppato principalmente per i dati del satellite Sentinel-2, parte del programma Copernicus dell'Unione Europea.
- MSCFF (Multi-Scale Convolutional Feature Fusion), una rete neurale convoluzionale progettata per segmentare immagini satellitari attraverso l'utilizzo di una fusione di caratteristiche a più scale. La fusione di queste feature a diverse scale consente al modello di catturare dettagli a diversi livelli, migliorando l'accuratezza e la precisione della segmentazione.
- RS-Net, ovvero una rete neurale progettata specificamente per la segmentazione in immagini da telerilevamento (Remote Sensing).

Inoltre sono presenti delle varianti di MSCFF e RS-Net che sono rispettivamente MSCFF-13 e RS-Net-13, dove il 13 sta a significare che queste varianti prendono in ingresso tutte e 13 le bande prodotte dal satellite, come il modello in esame, e non semplicemente le bande VNIR (visible and near-infrared). Infine, è presente anche il confronto tra il modello originale del paper di riferimento e il nuovo modello, in quanto sono state apportate modifiche, ed è stato fatto un diverso studio degli iperparametri che ha portato a risultati lievemente diversi.

Model	Accuracy	Precision	Recall	F1-Score
Fmask	92,62%	59,90%	95,14%	70,98%
Sen2Cor	90,89%	64,20%	69,36%	62,52%
MSCFF	98,40%	96,42%	78,77%	86,37%
MSCFF-13	98,78%	96,22%	86,48%	91,02%
RS-Net	98,03%	88,77%	79,90%	83,72%
RS-Net-13	98,59%	93,63%	85,06%	89,00%
Old CD-FM3SF	98,86%	96,50%	87,75%	91,86%
New CD-FM3SF	98,81%	96,98%	90,95%	93,87%

Tabella 5.1. Nomi e caratteristiche delle bande prodotte dal satellite

Come si può vedere nella Tabella 5.1, il nuovo modello è secondo solo all'originale CD-FM3SF[12] per quanto riguarda l'accuracy generale, e la differenza è solo di 5 centesimi di percentuale; ma al contempo supera tutti gli altri modelli nell'F1-Score, il che implica un eccellente equilibrio tra precisione e recall.

Infatti, andando a guardare altre metriche come la Recall, notiamo che Fmask supera di molto tutti gli altri, ma performa malissimo in Precision; quindi non considerando Fmask in quanto troppo instabile, il nuovo modello ottiene i risultati migliori anche per Precision e Recall.

Come si vede dal grafico in Figura 5.3, tolte le soluzioni Fmask e Sen2Cor, tutti gli altri modelli di reti neurali convoluzionali hanno comunque delle performance molto simili, quindi il vero vantaggio offerto dai due modelli CD-FM3SF è il fatto che sono più leggeri in quanto spazio in memoria e per performance su hardware più

economico rispetto agli altri modelli, e nonostante questo, performano alla pari se non meglio.

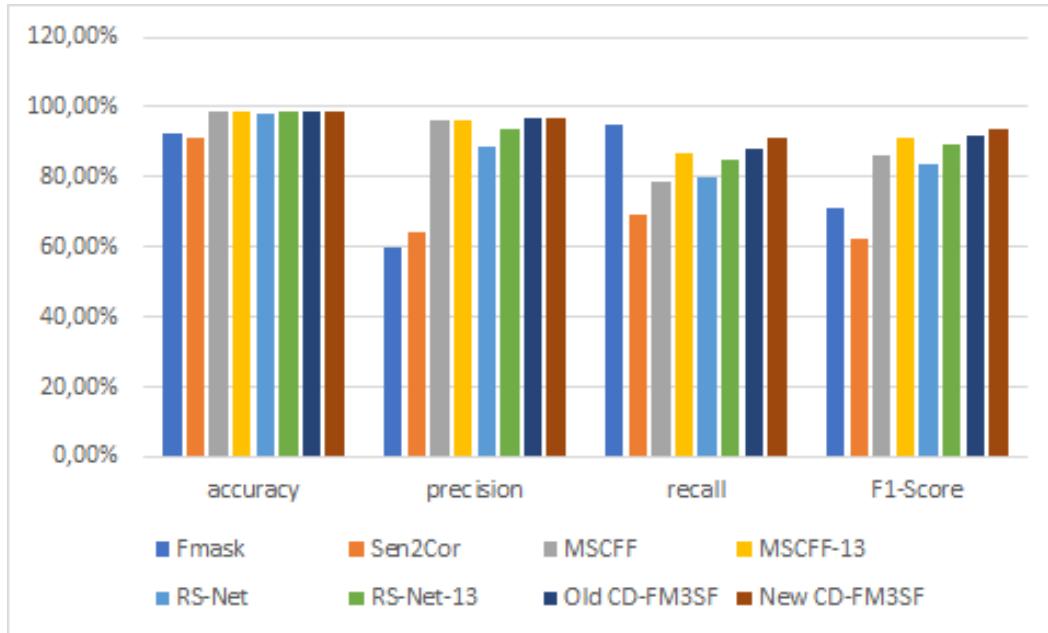


Figura 5.3. Grafico a barre per il confronto dei risultati tra modelli

5.3.2 Analisi in situazioni di difficoltà

Le immagini usate per il test set sono state selezionate appositamente in modo tale da contenere tutti i possibili ambienti, infatti come si può vedere in Figura 2.2 queste immagini sono equamente distribuite su tutto il territorio cinese; questo fa sì che i risultati esposti nei capitoli precedenti diano una visione generale delle performance di classificazione indipendentemente dall'ambiente fotografato.

Ci sono però delle situazioni specifiche in cui il modello può avere più difficoltà, in quanto il 98,91% è solo una media, la quale potrebbe essere il risultato di una performance eccellente su delle immagini e di una performance insufficiente su altre: in un algoritmo che accetta solo immagini VNIR, come mostrato nella Tabella 2.1 del TOA, ci si può aspettare che le differenze, ad esempio, tra Clouds e Vegetation siano talmente ampie che non ci siano difficoltà a distinguere le due classi, ma al contrario, la differenza tra Cloud e Snow/Ice è molto sottile, il che potrebbe portare a numerosi errori.

Anche in questo caso, per dare una migliore idea e per comprendere a pieno i risultati, in Figura 5.4 vengono mostrate quattro immagini contenenti diverse quantità di nuvole e neve. La prima mostra una situazione in cui la neve copre quasi tutta l'immagine, ma non sono presenti delle nuvole, ed infatti l'algoritmo predice correttamente classificando tutto come background. Mentre nelle immagini seguenti c'è sempre meno neve e più nuvole, e in particolare il modello è messo sotto stress nella seconda immagine, dove il mix tra nuvole e neve ogni tanto rende difficile

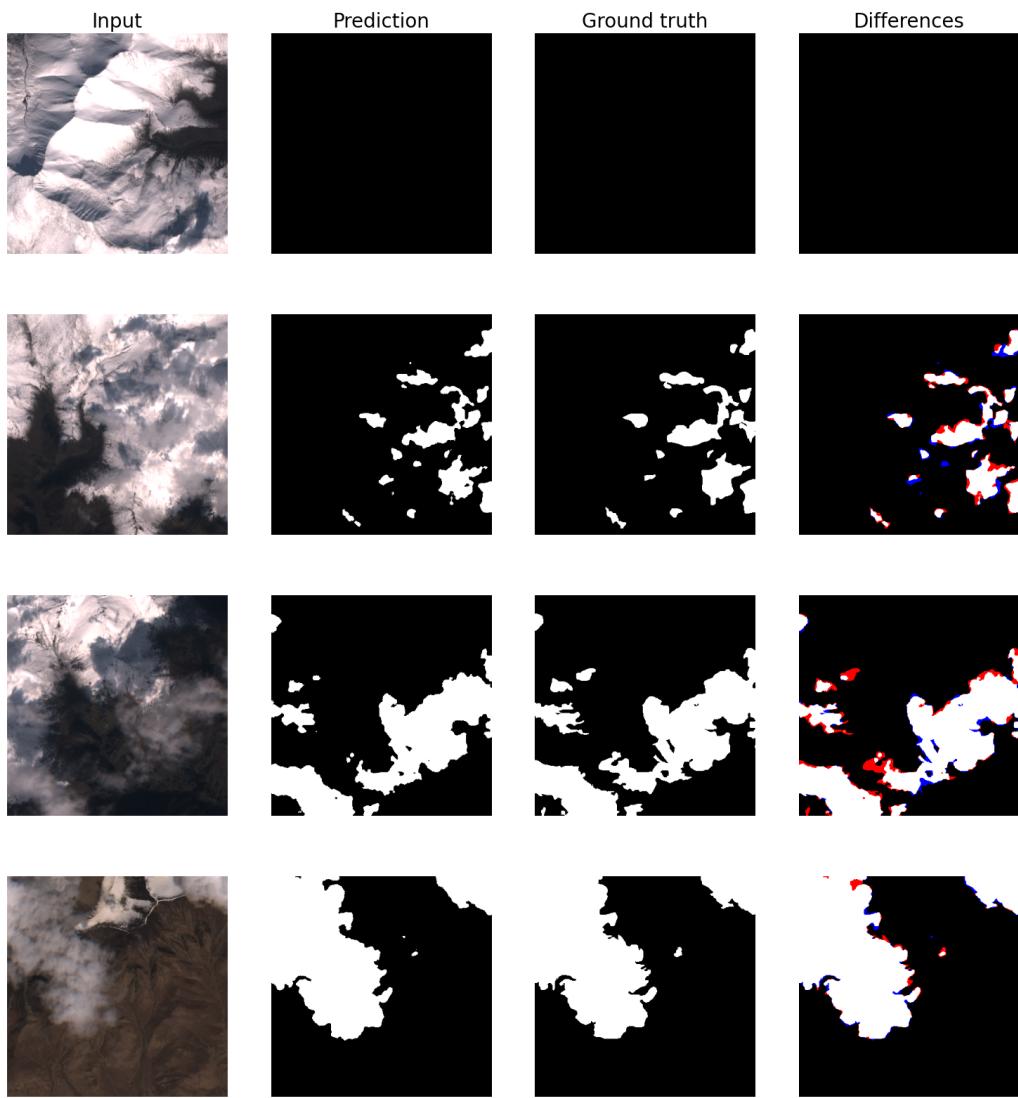


Figura 5.4. 4 esempi di segmentazione su immagini con diverse quantità di neve

pure all'occhio umano distinguere le due classi, ed è proprio in questo caso che la possibilità di usare tutte e 13 le bande garantisce una performance più elevata.

I dati infatti confermano quello che si vede nella figura di esempio in quanto l'accuracy media è di 98.58% quindi (a parte piccole variazioni insignificanti) del tutto in linea con i risultati delle altre immagini negli altri contesti; purtroppo però delle differenze di 1 o 2 punti percentuali si notano nelle metriche precision e recall, che scendono rispettivamente a 95,21% e 88,38%, quindi ovviamente anche l'FI-Score si abbassa a 91,16%. In conclusione, si può dire che il modello rimane comunque valido anche in queste situazioni e mantiene sempre il rapporto tra false positive e false negative più o meno costante senza sbilanciarsi.

Capitolo 6

Conclusioni

Il modello di rete neurale presentato offre un metodo efficace per la segmentazione semantica di immagini satellitari. L'efficacia del modello è stata confermata attraverso metriche di valutazione come l'accuratezza, la recall, la precisione e l'F1-Score, che si sono rivelate in linea con le soluzioni di machine learning più avanzate nel campo. Tuttavia, quello che rende particolarmente interessante il lavoro è l'efficienza del modello proposto: nonostante abbia solamente un venticinquesimo dei parametri rispetto a soluzioni più pesanti come RS-Net, il modello ha mantenuto un'alta qualità nei risultati, dimostrando così che è possibile raggiungere un eccellente compromesso tra efficienza computazionale e precisione.

Un altro aspetto distintivo è stato l'utilizzo di tutte e 13 le bande prodotte dal satellite Sentinel-2A alla loro definizione nativa; ogni banda fornisce un insieme unico di informazioni che ha contribuito all'alta accuratezza del modello, permettendo di distinguere efficacemente tra nuvole e altre formazioni come la neve nelle aree montuose, un compito che spesso risulta essere più difficile per algoritmi tradizionali.

Per quanto riguarda l'implementazione, è stato usato il framework PyTorch, la cui architettura modulare ha reso semplice sia la costruzione della rete neurale che le fasi di addestramento e test. Lo stesso stile è stato usato infatti anche per costruire dei componenti più avanzati, quali ad esempio lo "Shared and dilated convolution residual block" e il "Mixed depth-wise separable convolution layer" che sono il cuore del modello, in modo che potessero poi essere usati con la stessa facilità con cui si usano delle semplici convoluzioni.

In termini di casi d'uso, il modello potrebbe trovare applicazione in diversi ambiti, come il monitoraggio ambientale, la meteorologia e l'agricoltura di precisione. Va notato, tuttavia, che il modello tende a commettere più falsi negativi che falsi positivi; questo potrebbe limitare la sua efficacia in applicazioni dove la rilevazione completa delle nuvole è cruciale, ma potrebbe essere adatto per applicazioni dove minimizzare i falsi positivi è più importante.

Bibliografia

- [1] PyTorch Foundation. PyTorch main page. <https://pytorch.org/>, 2023.
- [2] European Space Agency. Sentinel-2 mission. <https://sentinel.esa.int/web/sentinel/missions/sentinel-2>.
- [3] European Organization For Nuclear Research and OpenAIRE. Zenodo, 2013.
- [4] Jun Li, Zhaocong Wu, Zhongwen Hu, Canliang Jian, Shaojie Luo, Lichao Mou, Xiao Xiang Zhu, and Matthieu Molinier. A lightweight deep learning-based cloud detection method for sentinel-2a imagery fusing multiscale spectral and spatial features. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–19, 2022.
- [5] Open Source Geospatial Foundation. OSGeo repository. <https://github.com/OSGeo>.
- [6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.
- [8] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.
- [9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [10] PyTorch Foundation. PyTorch ConvTranspose2d documentation. <https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html>, 2023.
- [11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [12] Jun Li, Zhaocong Wu, Zhongwen Hu, Canliang Jian, Shaojie Luo, Lichao Mou, Xiao Xiang Zhu, and Matthieu Molinier. Original CDFM3SF. <https://github.com/Neooolee/WHUS2-CD/blob/master/CDFM3SF.py>.