

Cloud Detection nelle immagini satellitari del Dataset 38-Cloud mediante la rete U-Net

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

Francesco Fabrizi
Matricola 1883509

Relatore
Prof. Thomas Alessandro Ciarfuglia

Sommario

Il rilevamento delle nuvole nelle immagini satellitari rappresenta una sfida cruciale per diverse applicazioni scientifiche e gestionali, come il monitoraggio del cambiamento climatico, gli studi del suolo e la gestione delle risorse naturali. Negli ultimi anni ci si è concentrati sul rilevamento delle nuvole nelle immagini satellitari, usando molte tecniche di rilevamento basate principalmente sulle reti neurali, ottenendo modelli sempre più efficaci grazie all'uso di varie bande di frequenza, tra cui la NIR (banda vicino infrarosso).

Le reti neurali convoluzionali (CNN) hanno rivoluzionato il campo del rilevamento delle nuvole, offrendo un approccio più sofisticato e robusto rispetto ai metodi tradizionali basati sullo studio delle forme nuvolose o su alcune loro proprietà.

In questo studio, utilizzerò come modello la rete U-Net, che risulta leggero, ma allo stesso tempo molto efficiente per il rilevamento delle nuvole nelle immagini satellitari.

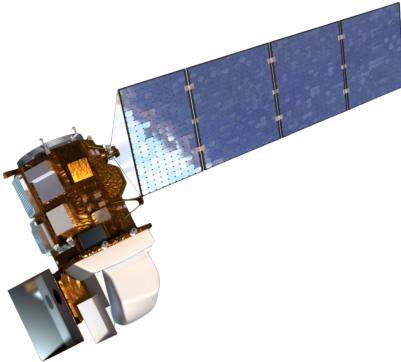
La relazione è divisa in capitoli, partiremo introducendo il problema del Cloud Segmentation. Parlerò brevemente del dataset, Sentinel-2 Cloud Dataset, un insieme di immagini satellitari ad alta risoluzione con etichette di nuvole segmentate manualmente, della sua struttura e di come ho proceduto alla sua gestione per usarlo nella CNN. Introdurrò la rete U-Net e il suo funzionamento. Entrerò poi nel dettaglio del lavoro descrivendo i vari test che ho effettuato per migliorare il mio modello. Alla fine verranno visualizzati alcuni risultati cercando di capire i punti deboli del modello e alcuni possibili miglioramenti.

Indice

| | |
|---|-----------|
| Capitolo 1 – Introduzione | 8 |
| 1.1 Task | 8 |
| 1.2 Reti Neurali Convoluzionali | 9 |
| 1.2.1 Architettura di una CNN | 9 |
| 1.2.2 Applicazioni delle CNN nel Rilevamento delle Nuvole | 11 |
| 1.3 Strumenti | 11 |
| Capitolo 2 – Dataset 38-Cloud | 13 |
| 2.1 Struttura del Dataset | 13 |
| 2.2 Suddivisione e Organizzazione File | 14 |
| 2.3 Cloud Dataset | 15 |
| 2.4 Creazione dei Set e Visualizzazione | 18 |
| Capitolo 3 – Struttura della Rete U-Net | 21 |
| 3.1 Introduzione | 21 |
| 3.2 Doppia Convoluzione | 22 |
| 3.3 Codice e Struttura Generale | 23 |
| 3.4 Encoder e Decoder | 24 |
| 3.5 Skip Connections | 25 |
| Capitolo 4 – Approcci Utilizzati | 27 |
| 4.1 Riproducibilità | 27 |
| 4.2 Funzione di Ottimizzazione | 27 |
| 4.3 Metriche di Valutazione | 29 |
| 4.4 Funzioni di Loss | 29 |
| 4.5 Sbilanciamento Dataset | 30 |
| 4.6 Learning Rate e Data Augmentation | 31 |
| 4.7 Salvataggi | 33 |
| Capitolo 5 – Prove e Risultati | 34 |
| 5.1 Le Prime Prove | 34 |
| 5.2 Addestramento 30 Epoche | 35 |
| 5.3 10 e 15 Epoche | 36 |
| 5.4 Valutazione del Modello Finale nella Prima Versione del Dataset | 38 |
| 5.5 Modello Finale nella Seconda Versione del Dataset | 40 |
| 5.6 Confronto dei due Modelli con Valutazione dei Punti Deboli e di Forza | 43 |
| Capitolo 6 – Conclusioni | 45 |
| Bibliografia e Sitografia | 48 |

Capitolo 1 — Introduzione

1.1 Task



Il satellite Landsat 8 è stato lanciato nello spazio l'11 febbraio 2013. Questo satellite fa parte del progetto Landsat, una costellazione di satelliti iniziata nel 1972 con il primo lancio. La serie di missioni Landsat è fondamentale per lo studio dell'ambiente, delle risorse naturali e dei cambiamenti che avvengono sulla superficie terrestre. Landsat 8 è dotato di due strumenti principali: l'Operational Land Imager (OLI) e il Thermal Infrared Sensor (TIRS). L'OLI raccoglie dati nelle bande spettrali del visibile (VNIR), del vicino infrarosso (NIR) e dell'infrarosso a onde corte (SWIR).

| Landsat-8 OLI and TIRS Bands (μm) | | | |
|--|-----------------|---------------|---------|
| 30m | Coastal/Aerosol | 0.435 – 0.451 | Band 1 |
| 30m | Blue | 0.452 – 0.512 | Band 2 |
| 30m | Green | 0.533 – 0.590 | Band 3 |
| 30m | Red | 0.636 – 0.673 | Band 4 |
| 30m | NIR | 0.851 – 0.879 | Band 5 |
| 30m | SWIR-1 | 1.566 – 1.651 | Band 6 |
| 100m | TIR-1 | 10.60 – 11.19 | Band 10 |
| 100m | TIR-2 | 11.50 – 12.51 | Band 11 |
| 30m | SWIR-2 | 2.107 – 2.294 | Band 7 |
| 15m | Pan | 0.503 – 0.676 | Band 8 |
| 30m | Cirrus | 1.363 – 1.384 | Band 9 |

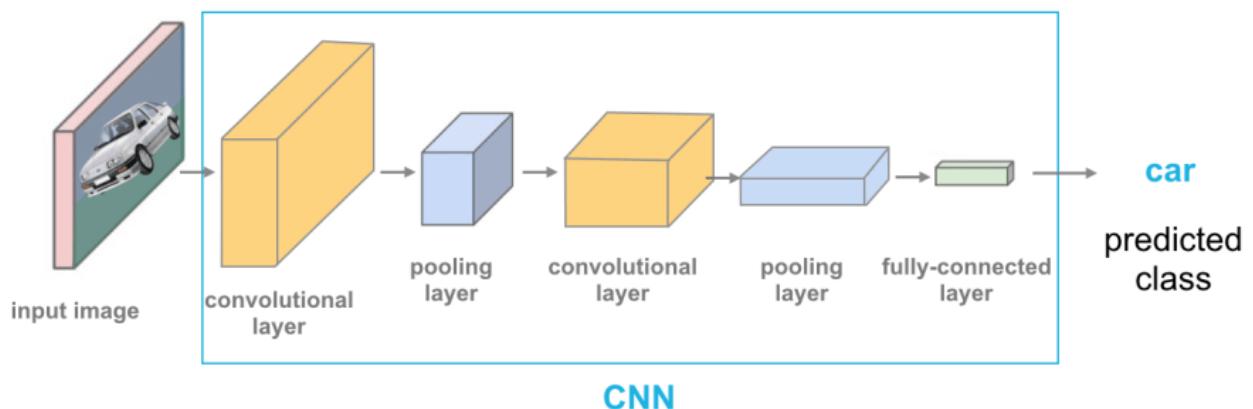
Le nuvole rappresentano una fonte significativa di rumore nei dati Landsat, poiché possono oscurare gli oggetti target e influenzare negativamente l'analisi delle immagini telerilevate. Il rilevamento delle nuvole è un campo di ricerca in continuo sviluppo, essenziale per migliorare la qualità dei dati satellitari. In media, la copertura nuvolosa nelle missioni Landsat è di circa il 40% per alcune regioni e il 35% per altre. Poiché le nuvole possono essere suddivise in due categorie principali – quelle spesse e opache, che sono più facili da rilevare, e quelle semitransparenti, che risultano molto più difficili da individuare – la sfida nel rilevamento consiste nel trattare efficacemente entrambe le tipologie.

I metodi di rilevamento delle nuvole si dividono principalmente in due categorie: quelli che utilizzano le bande dell'infrarosso termico e quelli che non le utilizzano. Nel caso specifico del rilevamento basato su Landsat 8, la banda NIR (Near Infrared) è particolarmente utile grazie alle sue proprietà spettrali. Le nuvole hanno una riflettanza elevata in questa banda, in contrasto con la maggior parte delle superfici terrestri che mostrano una riflettanza più bassa, facilitando così il loro rilevamento.

I modelli di Machine Learning (ML) hanno notevolmente contribuito al rilevamento delle nuvole, dimostrandosi tra i metodi più efficaci grazie alla loro capacità di gestire grandi quantità di dati. Uno degli approcci più utilizzati è rappresentato dalle Convolutional Neural Networks (CNN), che sono particolarmente efficaci in questo contesto poiché ideali per l'estrazione automatica delle caratteristiche delle immagini tramite operazioni sui pixel.

1.2 Reti Neurali Convoluzionali

Le reti neurali convoluzionali (CNN) hanno rivoluzionato il campo della Computer Vision, offrendo un metodo altamente efficace per l'analisi e il riconoscimento delle immagini. In passato, l'identificazione di oggetti nelle immagini richiedeva metodi manuali di estrazione delle caratteristiche, che erano laboriosi e dispendiosi in termini di tempo. Le CNN, invece, sfruttano i principi dell'algebra lineare, in particolare la moltiplicazione di matrici, per estrarre automaticamente caratteristiche salienti dai dati delle immagini, ottenendo risultati di gran lunga superiori.



1.2.1 Architettura di una CNN

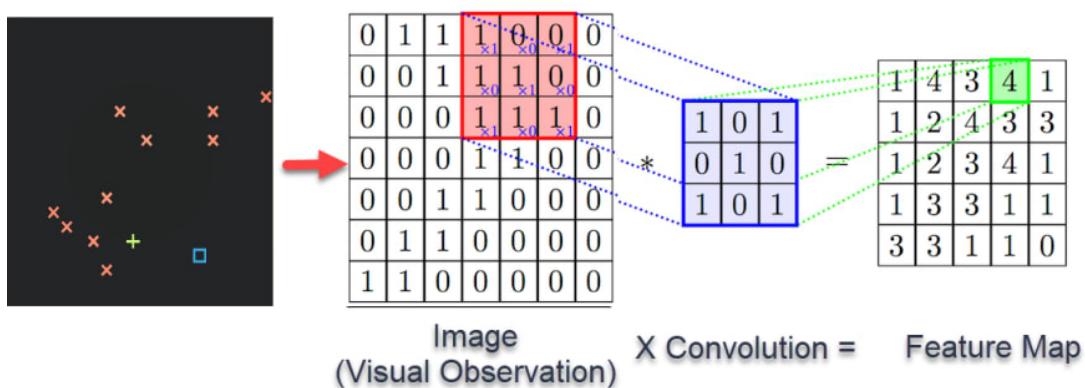
L'architettura di una CNN è composta da quattro tipi principali di strati:

1. Strato Convoluzionale

Lo strato convoluzionale rappresenta il cuore di una CNN, dove avviene la maggior parte dell'estrazione delle caratteristiche. In questo processo, un rilevatore di caratteristiche, noto anche come kernel o filtro, si muove attraverso i campi ricettivi dell'immagine, analizzando piccole porzioni alla ricerca di pattern specifici. Questo rilevatore moltiplica i valori del kernel con quelli dell'area di input, generando un valore che viene mappato in una "mappa di feature" o "mappa di attivazione". Ripetendo l'operazione di convoluzione per l'intera immagine, si ottiene un volume di output tridimensionale in cui la profondità corrisponde al numero di filtri utilizzati.

L'architettura a strati sovrapposti di una CNN permette di estrarre caratteristiche sempre più complesse man mano che l'informazione fluisce attraverso la rete. Ogni strato convoluzionale aggiunge un nuovo livello di astrazione, catturando dettagli sempre più sofisticati. Gli strati di attivazione, come il ReLU (Rectified Linear Unit), introducono una non linearità nel modello, permettendogli di apprendere relazioni complesse tra i dati e migliorare le capacità di rappresentazione.

Inoltre, l'uso di kernel di diverse dimensioni consente alla rete di rilevare caratteristiche su varie scale, migliorando la capacità di identificare sia piccoli dettagli che elementi più grandi. Questa flessibilità è cruciale per l'efficacia delle CNN, poiché consente di adattarsi a diverse risoluzioni e complessità delle immagini. Grazie a questa struttura, le CNN sono in grado di comprendere e interpretare le immagini con un'accuratezza senza precedenti, rendendole uno strumento potente in molte applicazioni di Machine Learning.



2. Strato di Pooling

I livelli di pooling svolgono un ruolo cruciale nella riduzione della dimensionalità dell'input, diminuendo il numero di parametri e la complessità del modello. I due tipi più comuni di livelli di pooling sono il max pooling e l'average pooling. Questa operazione di downsampling aiuta a contrastare l'overfitting e migliora l'efficienza computazionale del modello. Il max pooling seleziona il valore massimo da una regione di feature map, mantenendo così le caratteristiche più rilevanti e significative, mentre l'average pooling calcola la media dei valori presenti nella regione, risultando in una rappresentazione più morbida e meno sensibile ai valori estremi. Entrambe le tecniche contribuiscono a sintetizzare l'informazione in modo efficace, riducendo la dimensionalità dei dati e permettendo al modello di focalizzarsi sulle caratteristiche essenziali delle immagini.

3. Strato Completamente Connesso

A differenza degli strati convoluzionali e quelli di pooling, nello strato completamente connesso (Fully Connected Layer) ogni neurone è collegato a tutti i neuroni del livello precedente. Questo strato ha il compito di classificare i dati in base alle caratteristiche estratte dai livelli precedenti e dai loro diversi filtri. La funzione di attivazione Softmax è comunemente utilizzata in questo contesto per la classificazione, assegnando a ciascun input una probabilità di appartenenza a ogni possibile classe. L'uso degli strati completamente connessi alla fine della rete permette di

combinare e interpretare le caratteristiche rilevate nei livelli precedenti, facilitando la decisione finale di classificazione. Questo processo consente al modello di sintetizzare tutte le informazioni apprese, integrandole per fornire un output preciso e accurato.

4. Appiattimento

Dopo i livelli di convoluzione e pooling, le mappe di feature vengono trasformate in un unico vettore unidimensionale. Questo passaggio, noto come appiattimento, è necessario per poter utilizzare i dati in uno strato completamente connesso. L'appiattimento consente di convertire le rappresentazioni spaziali, che catturano le caratteristiche locali dell'immagine, in una forma vettoriale, rendendo i dati pronti per la classificazione finale. Questa trasformazione permette di integrare tutte le caratteristiche estratte in un formato che il modello può utilizzare per prendere decisioni di classificazione.

1.2.2 Applicazioni delle CNN nel Rilevamento delle Nuvole

Le Convolutional Neural Networks (CNN) sono strumenti fondamentali nel campo del rilevamento delle nuvole nelle immagini satellitari per diversi motivi. Queste reti neurali sono particolarmente adatte perché eliminano la necessità di estrazione manuale delle caratteristiche, automatizzando il processo di individuazione dei pattern significativi nelle immagini. Grazie alla loro capacità di apprendimento profondo su vasti insiemi di dati, le CNN sono robuste rispetto alle variazioni nelle condizioni atmosferiche e nelle caratteristiche delle nuvole. Questo permette loro di distinguere con precisione tra nuvole spesse, che possono oscurare aree sottostanti, e nuvole sottili, che possono essere più difficili da identificare.

Le ottimizzazioni nelle operazioni di convoluzione e pooling rendono le CNN efficienti nell'elaborazione delle immagini, consentendo una rapida analisi delle scene satellitari. Questo aspetto è cruciale per applicazioni che richiedono il monitoraggio continuo delle nuvole, come nel caso della gestione delle risorse naturali, le previsione meteorologiche e lo studio dei cambiamenti climatici. L'uso della CNN non solo migliora l'accuratezza del rilevamento delle nuvole, ma ottimizza anche l'elaborazione dei dati, rendendo più efficaci le decisioni basate sull'analisi delle immagini satellitari.

1.3 Strumenti

Per lo sviluppo di modelli complessi di Deep Learning, è essenziale disporre di potenti risorse computazionali. A tal fine, ho scelto di utilizzare Google Colab in versione premium, che offre accesso a RAM elevata e GPU performanti. Queste risorse sono cruciali per l'addestramento di reti neurali profonde su grandi dataset, come quello utilizzato per il rilevamento delle nuvole.

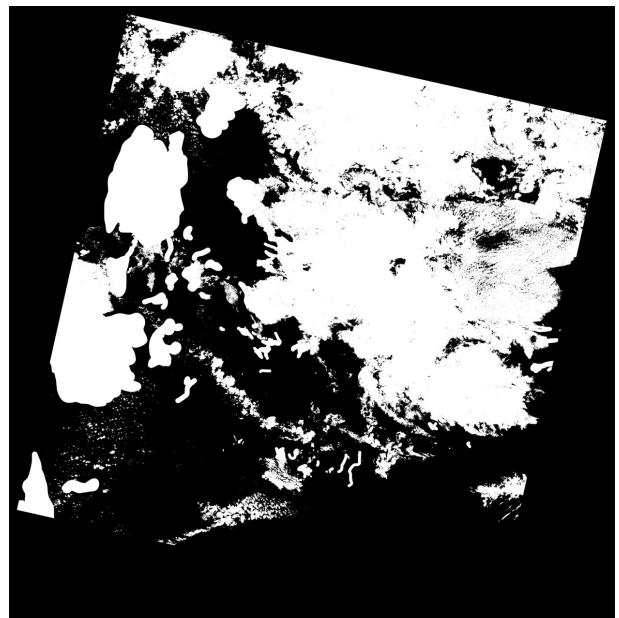
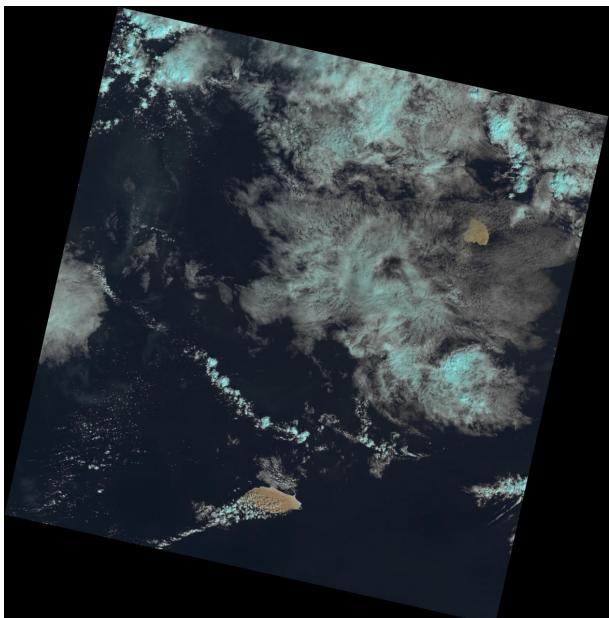
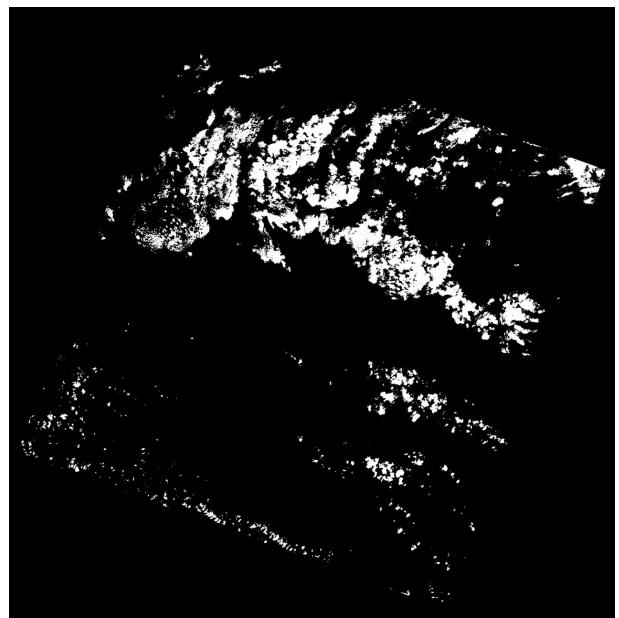
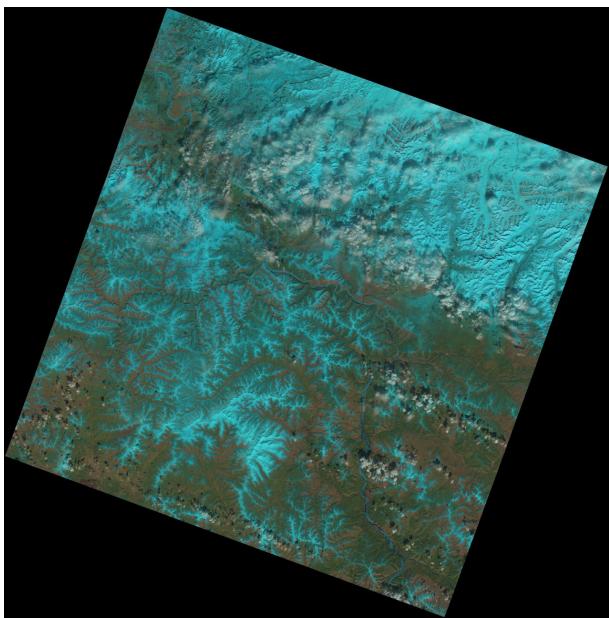
Python è il linguaggio di programmazione ideale per lo sviluppo nel campo dell'intelligenza artificiale, grazie alla sua flessibilità e alle numerose librerie dedicate. PyTorch, una delle librerie più

utilizzate e potenti per il Deep Learning, è stata la mia scelta principale per implementare la rete neurale U-Net. Quest'architettura è particolarmente efficace per la segmentazione semantica delle immagini, un processo fondamentale nel rilevamento preciso delle nuvole nelle immagini satellitari ad alta risoluzione.

L'implementazione di PyTorch mi ha fornito non solo strumenti avanzati per la gestione dei dati e l'ottimizzazione del processo di addestramento, ma anche un ambiente flessibile per la sperimentazione e l'analisi dei risultati. PyTorch ha migliorato significativamente l'efficienza del mio lavoro di ricerca, consentendomi di affrontare con successo sfide complesse nel campo del rilevamento delle nuvole tramite dati satellitari.

Capitolo 2 — Dataset 38-Cloud

2.1 Struttura del Dataset



Le immagini sopra mostrano un esempio di cattura satellitare multispettrale e la corrispondente ground truth del dataset. La prima immagine è una rappresentazione delle bande multispettrali acquisite dal satellite Landsat 8, mentre la seconda immagine mostra la classificazione della copertura nuvolosa, utilizzata per l'addestramento e la valutazione degli algoritmi.

Il dataset utilizzato per questa tesi è il 38-Cloud, sviluppato presso il Laboratory for Robotics Vision (LRV) della School of Engineering Science presso la Simon Fraser University, a Burnaby, nella Columbia Britannica, Canada. Questo dataset è una risorsa fondamentale per lo studio e lo sviluppo di algoritmi di rilevamento delle nuvole nelle immagini satellitari. Le immagini satellitari utilizzate provengono dal satellite Landsat 8, noto per la sua alta risoluzione spettrale e spaziale.

Il dataset è composto da un totale di 38 scene satellitari, ciascuna delle quali contiene immagini multispettrali catturate da Landsat 8. Queste immagini coprono diverse bande spettrali, tra cui il vicino infrarosso (NIR), fornendo informazioni dettagliate sulle condizioni atmosferiche e sulla copertura nuvolosa. Il dataset 38-Cloud è diviso in due parti: un insieme di 18 immagini Landsat 8, utilizzate per l'addestramento dei modelli, e un insieme di 20 immagini utilizzate per il testing. Questa suddivisione permette di valutare l'efficacia dei modelli su dati non visti durante l'addestramento, garantendo una valutazione obiettiva delle capacità di generalizzazione degli algoritmi.

2.2 Suddivisione e Organizzazione File

Le immagini satellitari del dataset 38-Cloud sono state ritagliate in pezzetti più piccoli chiamati "patch", ciascuno delle dimensioni di 384x384 pixel, per consentire un'elaborazione più efficiente delle immagini. Il dataset è stato suddiviso in due insiemi principali: training e testing. L'insieme di training contiene 8.400 patches, mentre l'insieme di testing contiene 9.201 patches. Questi patches vengono utilizzati per addestrare e valutare le prestazioni degli algoritmi di Deep Learning. Ogni patch ha 4 canali spettrali corrispondenti a diverse bande:

- » Rosso (banda 4)
- » Verde (banda 3)
- » Blu (banda 2)
- » Vicino infrarosso (banda 5)

A differenza di altri dataset di immagini per Computer Vision, questi canali non sono combinati in un'unica immagine RGB ma sono salvati in cartelle separate, corrispondenti a ciascuna banda spettrale. Questa separazione consente agli algoritmi di Deep Learning di apprendere relazioni più complesse tra le diverse bande spettrali, migliorando così la precisione nella segmentazione delle nuvole rispetto all'utilizzo di immagini RGB combinate.

I file del dataset sono disponibili al seguente link: <https://github.com/SorourMo/38-Cloud-A-Cloud-Segmentation-Dataset>. I file sono in formato TIFF (Tagged Image File Format), che ha il vantaggio di preservare la qualità originale delle immagini, fondamentale per l'analisi delle nuvole. Inoltre, il formato TIFF è flessibile in termini di archiviazione delle immagini ed è ampiamente diffuso e supportato. La struttura del file zip contenente il dataset è la seguente:

| | | |
|---|--|--------------------------------|
| 38-Cloud_95-...Metadata_Files > | Entire_scene_gts > | blue_patch_1_1...324_01_T1.TIF |
| 38-Cloud_test > | Natural_False_Color > | blue_patch_1_1...324_01_T1.TIF |
| 38-Cloud_training > | train_blue > | blue_patch_1_1...323_01_T1.TIF |
| 38-Cloud_Trai...Metadata_Files > | train_green > | blue_patch_1_1...323_01_T1.TIF |
| bibtext.txt | train_gt > | blue_patch_1_1...222_01_T1.TIF |
| training_patch..._nonempty.csv | train_nir > | blue_patch_1_1...223_01_T1.TIF |
| | train_red > | blue_patch_1_1...223_01_T1.TIF |
| | training_patches_38-Cloud.csv | blue_patch_1_1...224_01_T1.TIF |
| | training_scene..._38-Cloud.csv | blue_patch_1_1...224_01_T1.TIF |

Ho scaricato il dataset tramite Kaggle. Esiste anche un'estensione del dataset 38-Cloud, denominata 95-Cloud, che però non verrà utilizzata in questa relazione. La struttura del dataset prevede una cartella principale contenente le sottocartelle per il training e il testing. La cartella di training include cinque sottocartelle di nostro interesse: quattro per i canali spettrali e una contenente la Ground Truth.

2.3 Cloud Dataset

Per gestire il dataset 38-Cloud, ho utilizzato PyTorch per definire una sottoclasse della classe principale Dataset, che permette di manipolare i dati in modo efficiente. Prima di fare ciò, ho realizzato una funzione chiamata **combine** che crea un dizionario denominato **files**. Questa funzione unisce le immagini riferite alla stessa immagine principale dei quattro canali spettrali e della Ground Truth (GT).

```
def combine (r_file: Path, g, b, nir, gt):
    files = {'red': r_file,
              'green':g/r_file.name.replace('red','green'),
              'blue': b/r_file.name.replace('red', 'blue'),
              'nir': nir/r_file.name.replace('red', 'nir'),
              'gt': gt/r_file.name.replace('red', 'gt')}
    return files
```

Una volta creato il dizionario **files**, ho sviluppato la classe **CloudDataset**, basata su una classe preesistente fornita per il dataset 38-Cloud, che ho poi modificato per adattarla meglio alle esigenze del mio compito.

```
class CloudDataset1(torch.utils.data.Dataset):
    def __init__(self, files, transform=False, trasformN=None,
                 pytorch=True):
        super().__init__()
        self.files = files
        self.b=transform

        if(self.b==True):
            self.transformazione = get_train_augmentations()

        self.trasformN=trasformN
        self.pytorch=pytorch
```

La classe **CloudDataset** è progettata per gestire il dataset 38-Cloud. Gli input principali della classe sono il dizionario **files** creato con la funzione **combine**, un parametro **transform** che, se settato a **True**, applica le trasformazioni mentre il paramento **transformN** contiene la trasformazione per la normalizzazione. All'interno della classe **CloudDataset** ci sono alcuni metodi di particolare utilità, tra cui **open_mask** e **open_as_array**.

```
def open_as_array(self, idx, invert=False, include=False):
    rgb = np.stack([np.array(Image.open(self.files[idx]['red'])),
                    np.array(Image.open(self.files[idx]['green'])),
                    np.array(Image.open(self.files[idx]['blue'])),
                    ], axis=2)
    if include:
        nir = np.expand_dims(np.array(Image.open(self.files[idx]
                                                ['nir'])), 2)
        rgb = np.concatenate([rgb, nir], axis=2)
    if invert:
        rgb = rgb.transpose((2, 0, 1))
    return rgb / np.iinfo(rgb.dtype).max
```

Il metodo **open_as_array** apre le immagini in posizione **idx** e le converte in un array NumPy utile per l'elaborazione con PyTorch. Per prima cosa carica i tre canali per le immagini RGB e li impila lungo la terza dimensione dell'array. C'è anche la possibilità di includere la GT se richiesto e, tramite il valore **invert**, posso invertire gli assi dell'array in modo da renderlo subito utilizzabile da PyTorch. Prima del **return** viene applicata una normalizzazione dividendo i valori dell'array per il valore massimo, portando quindi ad avere valori compresi tra 0 e 1. Questo passaggio è essenziale e comune in tutte le documentazioni relative al dataset 38-Cloud. La normalizzazione non solo facilita l'addestramento dei modelli, ma aiuta anche a stabilizzare il processo di ottimizzazione, riducendo il rischio di gradienti esplosivi o scomparsi e migliorando la velocità di convergenza del modello.

```
def open_mask(self, idx, add_dims=False):
    mask = np.array(Image.open(self.files[idx]['gt']))
    mask = np.where(mask==255, 1, 0)
    return np.expand_dims(mask, 0) if add_dims else mask
```

Il metodo **open_mask** apre le maschere di segmentazione e converte i pixel in valori binari (0 e 1), rappresentando rispettivamente le aree di nuvola e non nuvola. Questo metodo è cruciale per preparare le etichette utilizzate durante l'addestramento del modello di segmentazione. Il metodo **calc_mean_std** calcola la media (mean) e la deviazione standard (std) dei quattro canali spettrali dell'intero dataset. Questo è cruciale per normalizzare i dati prima dell'addestramento. Il codice del metodo è il seguente:

```

def calc_mean_std(self, dataset):
    mean = torch.zeros(4)
    std = torch.zeros(4)
    for i in range(len(dataset)):
        images, _ = self[i]
        with warnings.catch_warnings():
            warnings.simplefilter("ignore")
            images_tensor = torch.tensor(images)
        batch_mean = torch.mean(images_tensor, dim=(0, 1, 2))
        batch_std = torch.std(images_tensor, dim=(0, 1, 2))
        mean += batch_mean
        std += batch_std
    mean /= len(self)
    std /= len(self)
    return mean, std

```

Questo metodo, dato un dataset, calcola la media e la deviazione standard dei quattro canali utilizzando le funzioni `torch.mean()` e `torch.std()`, con il parametro `dim=(0, 1, 2)` per calcolare la media e la deviazione standard per ciascun canale su tutte le immagini e su tutti i pixel di ogni immagine nel batch.

L'ultimo passo per permettere alla classe `DataLoader` di PyTorch di gestire il dataset è definire il metodo `__getitem__`, che specifica come accedere agli elementi del dataset. Questo metodo deve restituire le coppie (immagine, maschera) e può includere delle trasformazioni se specificate.

```

def __getitem__(self, idx):
    x = self.open_as_array(idx, invert=self.pytorch, include=True)
    y = self.open_mask(idx, add_dims=False)
    if(self.b==True):
        im=x.transpose(1,2,0)
        augm=self.transformazione(image =im, mask =y)
        x=augm[“image”].transpose(2,0,1)
        y=augm[“mask”]
    x = torch.tensor(x, dtype=torch.float32)
    y = torch.tensor(y, dtype=torch.int64)
    if self.transformN is not None:
        x=self.transformN(x)
    return x, y

```

2.4 Creazione dei Set e Visualizzazione

Il dataset è fornito in modo da essere già diviso in training set e testing set. Successivamente, ho suddiviso ulteriormente il training set in due sottoinsiemi: il training set effettivo e un validation set, utilizzando una proporzione dell'80% per il primo e del 20% per il secondo. Questo approccio consente di valutare l'efficacia del modello su dati non visti durante l'addestramento, fornendo una misura obiettiva delle sue capacità di generalizzazione. Per effettuare questa suddivisione, ho utilizzato la funzione:

`torch.utils.data.random_split(files, [0.8, 0.2]).`

Una funzione fornita da PyTorch che divide casualmente un dataset in base alle proporzioni specificate.

Successivamente, ho effettuato la visualizzazione delle immagini e delle relative maschere. Questo passaggio è cruciale per comprendere meglio il tipo di dati su cui si sta lavorando e per valutare l'output atteso dal modello. Inoltre, visualizzare le immagini consente di individuare eventuali sbilanciamenti nelle classi e di identificare visivamente eventuali problemi o peculiarità dei dati.

Il seguente codice è stato utilizzato per visualizzare 5 immagini casuali del training set:

```
indici = random.sample(range(100), 5)
fig, axes = plt.subplots(5, 2, figsize=(15, 25))
for i, idx in enumerate(indici):

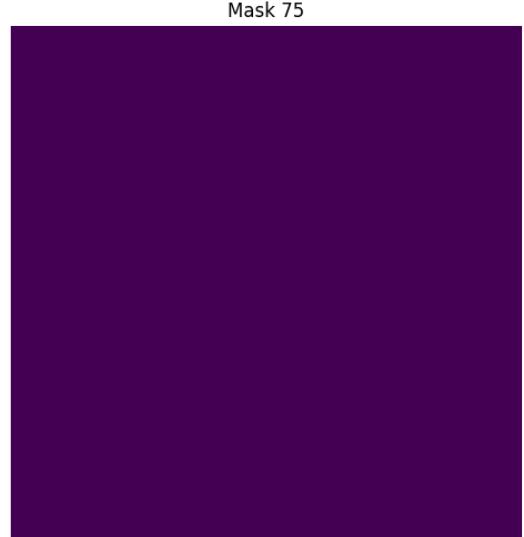
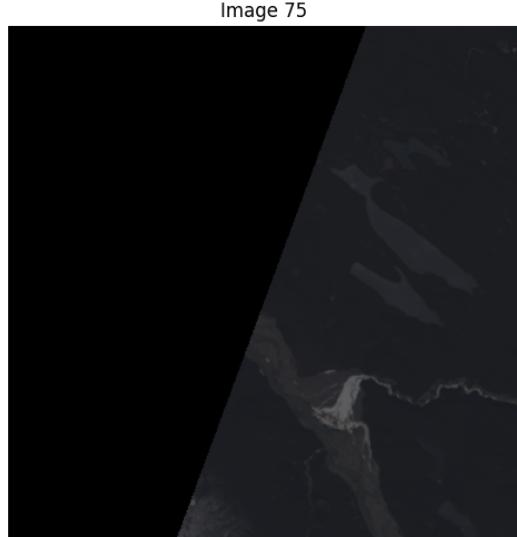
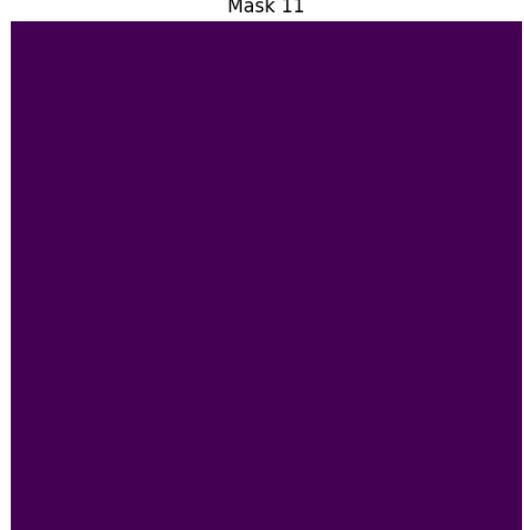
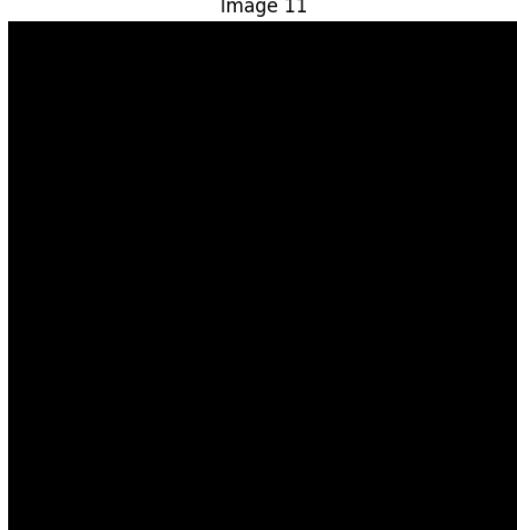
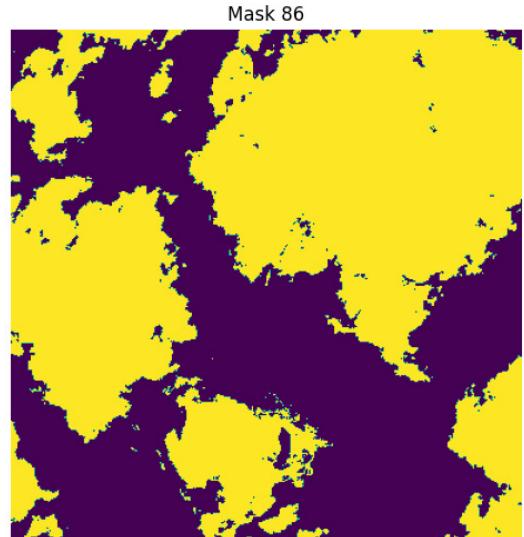
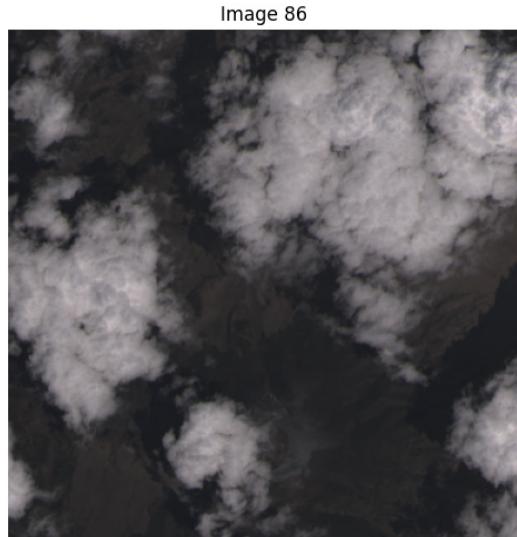
    img = train_set.open_as_array(idx)
    mask = train_set.open_mask(idx)

    axes[i, 0].imshow(img)
    axes[i, 0].set_title(f'Image {idx}')
    axes[i, 0].axis('off')

    axes[i, 1].imshow(mask)
    axes[i, 1].set_title(f'Mask {idx}')
    axes[i, 1].axis('off')

plt.tight_layout()
plt.show()
```

Ottenendo questo tipo di immagini:



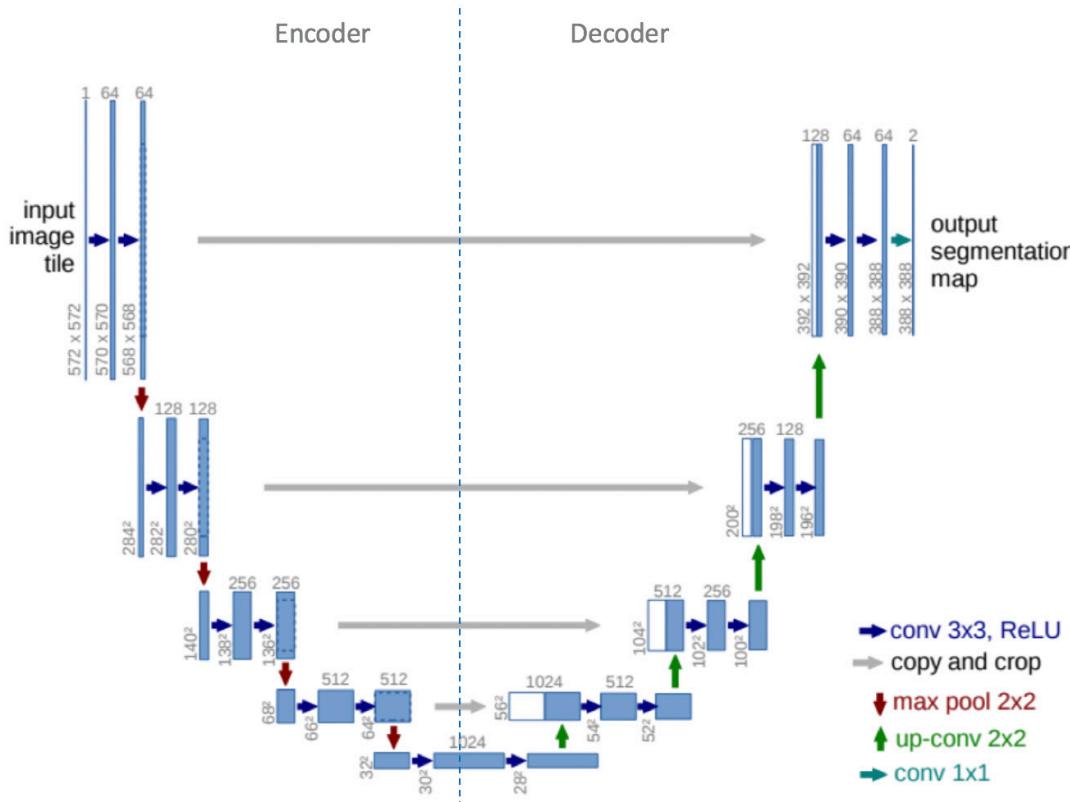
Le immagini sono di diverso tipo, abbiamo immagini con paesaggi innevati, con presenza d'acqua, con nuvole o senza. Durante l'analisi visiva, ho notato che la classe "non nuvola" è nettamente più rappresentata rispetto alla classe "nuvola". Questo sbilanciamento è dovuto principalmente alla presenza di numerose immagini completamente nere nel dataset, che possono influenzare

negativamente l'addestramento del modello. Per affrontare questa sfida e garantire un addestramento equilibrato, inizialmente ho provato ad utilizzare dei pesi per le classi ma alla fine ho deciso di eliminare tutte le immagini completamente nere. Questa scelta è stata fatta per alleggerire il quantitativo di dati da elaborare, evitando così di rallentare il programma e migliorando la qualità complessiva del lavoro. Le immagini completamente nere non fornivano informazioni utili per l'addestramento del modello e la loro eliminazione ha contribuito a rendere il processo di addestramento più efficiente, questi approcci verranno discussi in dettaglio nei capitoli successivi.

Capitolo 3 — Struttura della rete U-Net

3.1 Introduzione

La rete U-Net è una delle architetture più influenti e ampiamente utilizzate per la segmentazione semantica di immagini. È stata sviluppata presso l'Università di Friburgo nel 2015, inizialmente per la segmentazione di immagini biomediche. La sua capacità di segmentare con precisione oggetti a livello di pixel ha trovato applicazione in molti campi, tra cui la medicina, l'agricoltura, e la climatologia. La struttura della rete è a forma di "U", da cui il suo nome, come facilmente visibile dallo schema, ed è costituita principalmente da due parti: un encoder e un decoder.



L'encoder, noto anche come percorso di contrazione, utilizza una serie di convoluzioni 2D per estrarre le feature dall'immagine di input. Ogni strato dell'encoder applica convoluzioni seguite da funzioni di attivazione (tipicamente ReLU) e operazioni di pooling, riducendo la risoluzione spaziale ma aumentando la profondità delle feature map. Questo processo permette di estrarre caratteristiche più generali e astratte a livelli più profondi della rete.

Il decoder, noto come il percorso di espansione, ha il compito di ricostruire l'immagine segmentata tornando alla risoluzione originale tramite operazioni di upsampling e convoluzioni trasposte. Riassumendo, possiamo immaginare che nel percorso di contrazione impariamo a riassumere ciò che c'è nell'immagine, mentre nel percorso espansivo (decoder) si impara dove si trovano tali caratteristiche.

Una delle caratteristiche principali delle reti U-Net è la presenza delle "skip connection" (connessioni di salto), che aiutano a ottenere segmentazioni più precise prevenendo la perdita di informazioni

durante il processo di downsampling e upsampling. Queste connessioni sono collegamenti diretti tra le feature map di specifici layer dell'encoder e le feature map del corrispondente decoder con la stessa risoluzione spaziale. In questo modo, il decoder ha accesso diretto alle informazioni sui dettagli locali dell'immagine estratti dall'encoder, migliorando l'apprendimento delle informazioni e, di conseguenza, la precisione nella segmentazione. Grazie a queste caratteristiche, la U-Net si è dimostrata estremamente efficace non solo nel campo della segmentazione di immagini biomediche, ma anche in altre applicazioni, come la segmentazione di immagini satellitari, la mappatura delle colture agricole e la segmentazione di scene urbane.

3.2 Doppia Convoluzione

Una delle caratteristiche principali della rete U-Net è la presenza della Doppia convoluzione. La convoluzione è un'operazione fondamentale per le reti CNN, ed è usata per estrarre le caratteristiche delle immagini in input.

```
class DoppiaConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoppiaConv, self).__init__()
        self.conv=nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )
    def forward(self,x):
        return self.conv(x)
```

Nella classe **DoppiaConv** abbiamo due stati convoluzionali 3x3 con normalizzazione e una funzione di attivazione ReLU. Le convoluzioni sono eseguite utilizzando la funzione **Conv2d** di PyTorch, che preso in ingresso un input applica un kernel 3x3, un pixel alla volta senza modificare le dimensioni dell'immagine in uscita grazie al parametro **padding=1** che gestisce i bordi aggiungendo un pixel di padding attorno all'immagine di input. Dopo ciascuna convoluzione, viene applicata la funzione di attivazione ReLU, che introduce non linearità nella rete aiutando l'apprendimento. La funzione ReLU (Rectified Linear Unit) è una delle funzioni di attivazione più utilizzate, in quanto semplice da implementare e computazionalmente efficiente, che introduce la non linearità nella rete, permettendo di modellare relazioni complesse tra input e output. Non è altro che una funzione che vale 0 per input negativi e corrispondente all'identità per i valori positivi, la sua espressione matematica è:

$$f(x) = \max(0, x)$$

Infine, è presente la funzione ***BatchNorm2d*** di PyTorch, che esegue la normalizzazione di batch sui canali dell'output della convoluzione. La normalizzazione di batch aiuta a stabilizzare l'addestramento della rete e migliorare le prestazioni.

3.3 Codice e Struttura Generale

```
class UNET(nn.Module):
    def __init__(self, in_channels, out_channels, features=[64, 128, 256, 512]):
        super(UNET, self).__init__()
        self.ups=nn.ModuleList()
        self.downs=nn.ModuleList()
        self.pool=nn.MaxPool2d(kernel_size=2, stride=2)

        for f in features:
            self.downs.append(DoppiaConv(in_channels, f))
            in_channels=f

        for f in reversed(features):
            self.ups.append(nn.ConvTranspose2d(f*2, f,
                kernel_size=2, stride=2))
            self.ups.append(DoppiaConv(f*2,f))

        self.bneck=DoppiaConv(features[-1], features[-1]*2)
        self.f_conv=nn.Conv2d(features[0],out_channels,
            kernel_size=1)

    def forward(self, x):
        skip_connections=[]
        for d in self.downs:
            x=d(x)
            skip_connections.append(x)
            x=self.pool(x)

        x=self.bneck(x)
        skip_connections=skip_connections[: : -1]
        for idx in range(0, len(self.ups),2):
            x=self.ups[idx](x)
            skip_connection=skip_connections[idx//2]
            if x.shape != skip_connection.shape:
```

```

        x=TF.resize(x, size=skip_connection.shape[2:])
concat_skip=torch.cat((skip_connection, x), dim=1)
x=self.ups[idx+1](concat_skip)

return self.f_conv(x)

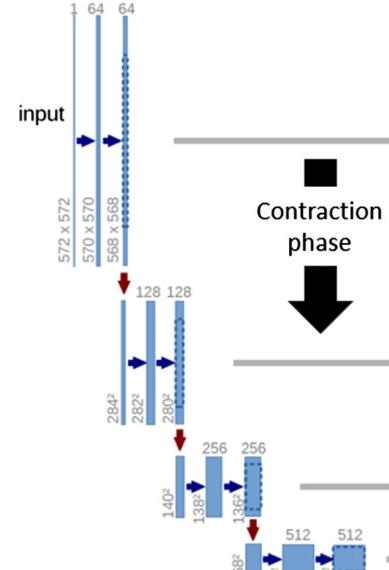
```

La rete U-Net è implementata come una sottoclasse di ***nn.Module*** di PyTorch, dove vengono specificati i canali di input (nel caso del dataset 38-Cloud sono 4, corrispondenti a RGB + NIR), i canali di output (nel nostro caso 2, indicando la probabilità che un pixel sia nuvola o non nuvola) e le dimensioni delle feature che definiscono la profondità degli strati di convoluzione. La scelta del numero di canali di input e output è cruciale poiché determina la capacità della rete di catturare le informazioni pertinenti nell'immagine e di produrre un output significativo.

Nella prima parte della classe U-Net, vengono create due liste (***self.ups*** e ***self.downs***) per memorizzare gli strati di upsampling e downsampling della rete. Viene anche istanziato uno strato di pooling massimo con un kernel 2x2 con stride 2, utilizzato per la fase di downsampling al fine di ridurre la risoluzione dell'immagine e mantenere le feature più rilevanti. La scelta delle dimensioni del kernel e dello stride nel pooling può influenzare le prestazioni complessive della rete, poiché determina il grado di riduzione della risoluzione spaziale e dell'informazione conservata durante il processo di downsampling. Nel mio caso, ho deciso di utilizzare un kernel 2x2, ma poteva essere utile testare anche un kernel 3x3 per valutare se avrebbe portato a miglioramenti sostanziali nelle prestazioni della rete.

3.4 Encoder e Decoder

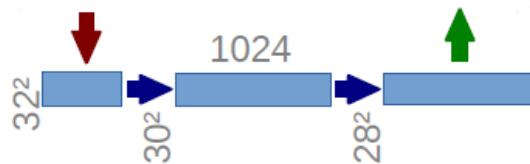
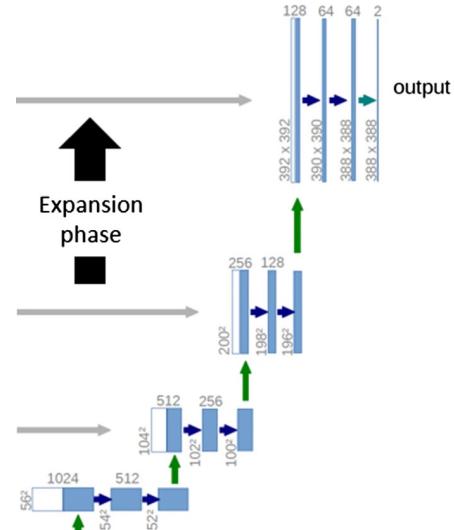
Nel primo ciclo for della classe U-Net, si itera sulle feature, che rappresentano le dimensioni dei canali per gli strati di convoluzione nel percorso di downsampling. Ogni elemento della lista feature indica il numero di canali di output per uno strato di convoluzione. Durante l'iterazione, viene creato uno stato ***DoppiaConv*** con la funzione ***append***. Ogni stato di ***DoppiaConv*** estrae caratteristiche dall'immagine di input e aumenta il numero di canali per estrarre caratteristiche sempre più complesse. Alla fine di ogni iterazione, il numero di ***in_channels*** viene aggiornato al numero di canali di output del layer ***DoppiaConv*** appena creato. Questo significa che il prossimo strato nel ciclo avrà come input il numero di canali in uscita del precedente strato, garantendo una transizione fluida e coerente tra gli strati della rete.



La funzione del Decoder, come anticipato, è quella di ricostruire l'immagine segmentata. Viene quindi creato un secondo ciclo for che itera sulla lista feature in ordine invertito. In questo modo, gli strati di upsampling e convoluzione vengono creati in maniera speculare a quelli creati nell'encoder, mantenendo simmetrica l'architettura della rete U-Net.

Nel decoder, prima di applicare la doppia convoluzione, viene utilizzata la funzione `nn.ConvTranspose2d`. Questo strato è essenziale nella fase di risalita poiché permette di aumentare la risoluzione spaziale dell'immagine, tipicamente utilizzando un kernel di dimensione 2x2. Successivamente, si applica la doppia convoluzione, simile a quella presente nell'encoder, per estrarre caratteristiche e raffinare la segmentazione. Come ultimo strato del decoder, troviamo una doppia

convoluzione che traduce le feature map estratte dalla rete in una mappa di probabilità tra classi. In questo caso, avendo due classi ("nuvola" e "non nuvola"), la rete produce una mappa di probabilità che indica la probabilità che ciascun pixel appartenga a una delle due classi. Infine, viene creato un "collo di bottiglia" tra l'upsampling e il downsampling, composto anch'esso da una doppia convoluzione. Questo strato raddoppia i canali di output, aumentando la capacità di rappresentazione dello strato e facilitando l'apprendimento delle caratteristiche più complesse.



3.5 Skip Connections

Una delle caratteristiche fondamentali della rete U-Net è la presenza delle skip connections. Queste connessioni permettono di combinare le informazioni semantiche estratte a diverse risoluzioni spaziali, migliorando significativamente la precisione e la robustezza della segmentazione.

Le skip connections collegano direttamente le feature map, estratte nei livelli di downsampling, ai livelli di upsampling corrispondenti. Questo meccanismo è essenziale per due motivi principali:

1. Salvaguardia delle informazioni: le reti neurali profonde possono "dimenticare" alcune funzionalità durante il passaggio delle informazioni attraverso i livelli successivi. Questo può portare a una perdita di informazioni sulla posizione ("dove") degli oggetti nell'immagine, mentre l'apprendimento della parte "cosa" (forma e caratteristiche) viene preservato.

Le skip connections risolvono questo problema introducendo nuovamente le informazioni sulla posizione (“dove”) dall’encoder al decoder corrispondente in ogni livello di upsampling. Questo permette alla rete di mantenere una rappresentazione accurata sia della forma che della posizione degli oggetti nell’immagine segmentata.

2. Problema della scomparsa del gradiente: nelle reti profonde, è comune che i gradienti diventino molto piccoli man mano che si propagano all’indietro attraverso la rete, un fenomeno noto come “vanishing gradient problem”. Le skip connections aiutano a mitigare questo problema fornendo percorsi di retropropagazione alternativi per i gradienti, che possono migliorare l’aggiornamento dei pesi nei livelli precedenti della rete.

Le skip connections sono particolarmente utili per segmentazioni più precise perché permettono al decoder di accedere direttamente ai dettagli locali catturati dall’encoder. In altre parole, durante il processo di upsampling, il decoder non solo basa le sue decisioni sulle feature map a bassa risoluzione create dalla fase di downsampling, ma anche sui dettagli ad alta risoluzione mantenuti attraverso le skip connections. Un altro vantaggio delle skip connections è la robustezza ai rumori e alle distorsioni. Poiché le informazioni dettagliate dell’immagine originale sono continuamente reintrodotte nel processo di decodifica, la rete diventa meno sensibile ai rumori presenti nelle immagini di input, migliorando così la qualità complessiva della segmentazione.

Capitolo 4 — Approcci Utilizzati

4.1 Riproducibilità

Prima di iniziare ad addestrare la rete, come già anticipato nel capitolo 2, ho diviso il dataset, contenente 8400 patches nel training set, in 6720 per il training e 1680 per il validation set. Nell'ambito dell'apprendimento automatico, la suddivisione dei dati in set di training e validation è un passaggio fondamentale per ottenere modelli affidabili e generalizzabili. La scelta di come dividere i dati è cruciale e può influenzare significativamente i risultati del training.

Essendo necessario riprendere il codice in momenti diversi, è essenziale garantire la riproducibilità. Riproducibilità significa che la stessa suddivisione dei dati dovrebbe essere ottenuta ogni volta che si esegue il codice di training, indipendentemente dall'ambiente o dall'hardware utilizzato. Questo garantisce che i risultati del training siano coerenti e confrontabili, facilitando la valutazione delle diverse configurazioni del modello e la condivisione dei risultati con altri ricercatori. Infatti, essendo necessario rieseguire il codice in momenti diversi, se la divisione cambiasse, potrebbe succedere che si addestri su dati che prima facevano parte del validation set e viceversa. Inoltre, tale riproducibilità permette di confrontare le prestazioni di diversi modelli addestrati sugli stessi dati, garantendo che la differenza di performance sia dovuta al modello stesso e non a fattori casuali nella divisione dei dati. Nel mio codice, ho quindi impostato un seed casuale fisso con **set_seed(42)** prima di utilizzare la funzione **random_split** per dividere i dati in training e validation set. Questo garantisce che la suddivisione dei dati sarà sempre la stessa ogni volta che si esegue il codice favorendo la riproducibilità:

```
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    set_seed(42)
```

L'impostazione di un seed fisso è una pratica comune e raccomandata per garantire la riproducibilità in esperimenti di Machine Learning. Essa consente di avere un controllo più rigoroso sulle esperienze di addestramento e validazione, assicurando che i risultati siano attribuibili alle modifiche nel modello o nei dati piuttosto che a variazioni casuali nella suddivisione dei dati o nei processi di inizializzazione.

4.2 Funzione di Ottimizzazione

L'ottimizzatore è responsabile dell'aggiornamento dei parametri della rete in base ai gradienti. Gli ottimizzatori sono strumenti cruciali nell'addestramento delle reti neurali, aiutano le reti ad apprendere dai dati in modo efficace attraverso la regolazione iterativa dei parametri. Il loro

obiettivo è trovare la combinazione ottimale di parametri che minimizza l'errore. Esistono diversi tipi di ottimizzatori, ognuno con i suoi punti di forza e debolezza, i più usati sono:

- » SGD (Stochastic Gradient Descent)
- » SGD con Momentum
- » Adam (Adaptive Momentum Estimation)

Per il 38-Cloud ho usato Adam per via di alcuni vantaggi che presenta:

- » Facile implementazione
- » Efficienza computazionale: Adam è computazionalmente efficiente, adatto anche a hardware con limitate capacità di calcolo
- » Adatto a problemi di grandi dimensioni

Adam è un algoritmo efficiente e adattivo che converge rapidamente. La sua capacità di fare stime adattive dei momenti del gradiente è dovuta all'utilizzo dei momenti di primo ordine (media dei gradienti) e di secondo ordine (media dei gradienti al quadrato), consentendogli di regolare il tasso di apprendimento per ciascun parametro.

Adam combina i vantaggi di due popolari algoritmi:

- » AdaGrad: che funziona bene con gradienti sparsi, adattando il tasso di apprendimento per ogni parametro in base alla somma cumulativa dei gradienti passati
- » RMSprop: che funziona bene in ambienti online e non stazionari, ovvero con dati di flusso continuo con funzione di perdita e distribuzione di dati che possono variare nel tempo.

Adam si è quindi rilevata la soluzione migliore, anche dopo aver analizzato diverse fonti che per il

```
optimizer=optim.Adam(model.parameters(), lr=lr)
```

dataset 38-Cloud consigliavano il suo utilizzo.

È possibile modificare anche altri parametri, ma nel mio caso mi sono limitato nella scelta del Learning Rate che verrà affrontata sempre in questo capitolo. Adam è particolarmente efficace grazie alla sua capacità di adattarsi dinamicamente ai cambiamenti nei gradienti e al suo utilizzo dei momenti del gradiente per migliorare la stabilità e la velocità di convergenza. Questi vantaggi lo rendono una scelta eccellente per una vasta gamma di applicazioni di Deep Learning, inclusa la segmentazione delle immagini come nel caso del dataset 38-Cloud.

La formula di aggiornamento dei parametri θ_t in Adam è la seguente:

1. Aggiornamento dei momenti

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t \quad v_t = \beta_2 m_{t-1} + (1-\beta_2)g_t^2$$

Dove g_t è il gradiente al tempo t , m_t è la stima della media mobile dei primi momenti, v_t è la stima della media mobile dei secondi momenti, β_1 e β_2 sono i tassi di decadimento esponenziale.

2. Correzione del bias

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad \hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

3. Aggiornamento dei parametri

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

Dove n è il learning rate e ϵ è un piccolo valore per prevenire la divisione per zero.

4.3 Metriche di Valutazione

Per valutare i vari modelli analizzati ho utilizzato diverse metriche, tra cui l'Accuracy e l'F1-score, pur calcolando anche Precision e Recall.

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP} \quad F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

L'Accuracy misura il rapporto tra le predizioni corrette e il totale degli elementi analizzati. Tuttavia, questo valore può essere poco significativo nel caso di dataset non bilanciati. In tali situazioni, una metrica come l'Accuracy può risultare fuorviante poiché potrebbe indicare elevate prestazioni anche quando il modello sta semplicemente predicendo la classe maggioritaria la maggior parte delle volte.

L'F1-score invece viene calcolato partendo dalla Precision e dalla Recall:

$$\text{Recall} = \frac{TP}{TP + FN} \quad \text{Precision} = \frac{TP}{TP + FP}$$

La Recall si concentra sulla capacità del modello di identificare gli esempi effettivamente positivi, indicando la percentuale di **True Positives** identificati rispetto al numero totale di esempi "effettivamente positivi". La Precision misura la capacità del modello di evitare di classificare erroneamente dei negativi come positivi, indicando la percentuale di **True Positives** identificati rispetto al numero totale di esempi positivi classificati dal modello. L'F1-score non è influenzato dalla classe "negativa" (nel nostro caso, i pixel non nuvola). Questo permette una valutazione più equilibrata del modello, evitando che la maggioranza dei pixel appartenenti alla classe negativa influenzi eccessivamente la valutazione. Questo è particolarmente utile nel nostro contesto, dove la distribuzione delle classi è fortemente sbilanciata. Utilizzando l'Accuracy, infatti, si ottenevano sempre valori molto alti, dando un'impressione fuorviante delle reali prestazioni del modello.

4.4 Funzioni di Loss

Le funzioni di perdita, o funzioni di costo, sono cruciali nell'apprendimento delle reti neurali. Insieme alle funzioni di ottimizzazione, sono direttamente responsabili dell'adattamento del modello ai

dati del training set. La funzione di perdita confronta i valori di output predetti e quelli di target, misurando quanto efficacemente la rete neurale sta apprendendo. L'obiettivo è minimizzare questa perdita, riducendo così l'errore tra le predizioni del modello e i valori attesi.

Esistono diversi tipi di funzioni di perdita, ciascuna adatta a specifici problemi e tipi di dati. Nonostante la Binary Cross Entropy Loss (BCE) sia una scelta comune per problemi di classificazione binaria, ho deciso di usare la classe **CrossEntropyLoss** disponibile nella libreria PyTorch, che può gestire sia la classificazione multiclasse che quella binaria. La formula della BCE è:

$$BCE = -\frac{1}{N} \sum_{i=1}^N [y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

Dove N è il numero di esempi nel batch, y_i è il valore target per l'esempio i , mentre p_i è la probabilità predetta dal modello per l' i -esimo esempio.

Nei vari modelli che ho testato, ho utilizzato la funzione **CrossEntropyLoss** con pesi, in modo da penalizzare in modo differenziato i “falsi positivi” (i pixel non nuvola classificati come nuvola) rispetto ai “falsi negativi” (i pixel nuvola classificati come non nuvola). Questo approccio è particolarmente utile nel nostro caso, dove il dataset presenta uno sbilanciamento tra le classi. Assegnando un peso maggiore ai “falsi negativi”, posso incentivare il modello a migliorare la rilevazione delle nuvole, che sono meno frequenti rispetto ai non nuvola.

4.5 Sbilanciamento Dataset

Durante l'osservazione del dataset e nei primi addestramenti, ho notato uno sbilanciamento evidente tra la classe “nuvola” e la classe “non nuvola”. Un terzo di tutte le immagini, sia di allenamento che di test, sono costituite solo da 0. Questo squilibrio può portare a problemi di apprendimento, in cui il modello diventa molto bravo a predire la classe dominante (“non nuvola”), ma fallisce nel predire correttamente la classe minoritaria (“nuvola”).

```
loss_fn = torch.nn.CrossEntropyLoss(weight =
torch.tensor([0.43,1]).to(device))
```

Per quantificare questo sbilanciamento, ho stimato il rapporto tra i pixel “nuvola” e quelli “non nuvola” utilizzando il seguente codice:

```
def conta_pix(data):
    clouds = 0
    no_clouds = 0
    for i in range(len(data)):
        array = data[i][1].flatten()
```

```

nc = np.count_nonzero(array)
nn = len(array) - nc
clouds += nc
no_clouds += nn
return (no_clouds, clouds)

```

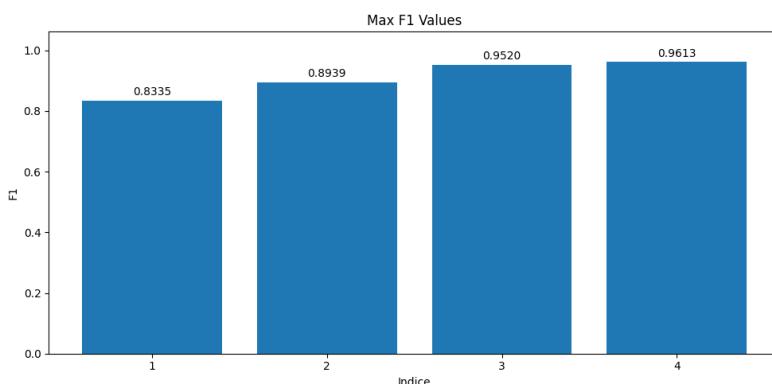
In questo modo ho contato i pixel “nuvola” e quelli “non nuvola”, ottenendo che il rapporto nuvola/non nuvola è di circa 0.43. Ho quindi utilizzato questo valore come peso nella Funzione di Loss per bilanciare il contributo delle due classi, mitigando l’effetto dello sbilanciamento sul processo di addestramento.

Come riportato nella pagina ufficiale del dataset, lo sbilanciamento nel dataset 38-Cloud è dovuto alla forma delle immagini, che sono parallelogrammi con quattro triangoli vuoti. Quando le patches vengono estratte da queste immagini, alcune di esse risultano completamente nere, contribuendo ulteriormente allo sbilanciamento delle classi. In una seconda fase, dopo aver eliminato le immagini completamente nere, ho deciso comunque di usare i pesi. In questo caso il rapporto nuvola/non nuvola è di circa 0.83.

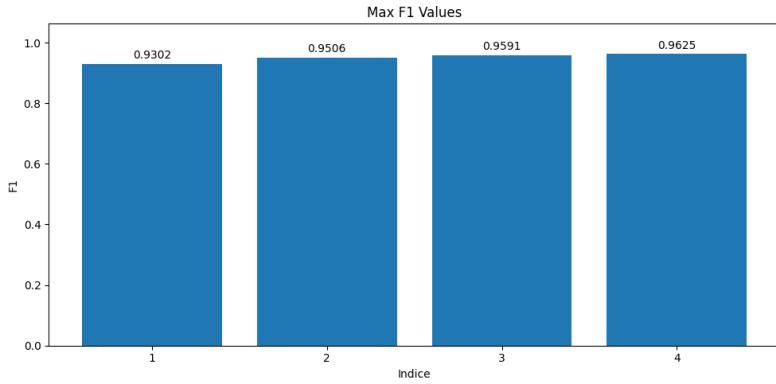
4.6 Learning Rate e Data Augmentation

La scelta del Learning Rate è uno degli ultimi passaggi cruciali per addestrare il nostro modello. Il Learning Rate determina la velocità con cui i parametri del modello vengono aggiornati durante il training e ha un impatto significativo sulla convergenza e sulle prestazioni finali del modello. Un Learning Rate troppo alto può causare instabilità e mancata convergenza, mentre uno troppo basso può rendere l’addestramento troppo lento e far convergere il modello a un minimo locale subottimale.

Per determinare il Learning Rate ottimale ho testato diversi valori su scala logaritmica: $1e^{-2}$, $1e^{-3}$, $1e^{-4}$, $1e^{-5}$. Ho fatto lo stesso tipo di test sia per il Dataset originale che per il Dataset privo delle immagini nere. Per quanto riguarda la prima parte, dopo aver testato tutti e quattro i valori, ho deciso di confrontare i vari risultati ottenuti basandomi sull’F1-score e di selezionare il Learning Rate che offre le migliori prestazioni per la fase di addestramento definitiva. Come visibile dal grafico sottostante risulta che con un **$lr=1e-5$** si ottiene il miglior risultato di F1-score.



Anche per il “secondo” DataSet ho ottenuto risultati simili:



Dopo aver individuato il miglior Learning Rate, ho eseguito la fase di Data Augmentation per evitare che la presenza limitata di patches potesse in qualche modo limitare l'apprendimento e portare a rischi di overfitting. La Data Augmentation è una tecnica utilizzata nell'apprendimento automatico per aumentare artificialmente la dimensione e la diversità di un dataset, permettendo di addestrare modelli più robusti e generalizzabili. Nel mio caso ho deciso di effettuare le seguenti trasformazioni:

```
def get_train_augmentations():
    transforms=[]
    transforms.extend([
        A.HorizontalFlip(p=0.5),
        A.VerticalFlip(p=0.5),
        A.RandomRotate90(p=1.0)
    ])
    return A.Compose(transforms)

Nor=transforms.Compose([
    transforms.Normalize(data_mean, data_std)
])
```

Le trasformazioni scelte includono ribaltamenti casuali lungo l'asse orizzontale e verticale, e una rotazione casuale delle immagini. Queste operazioni aiutano il modello a riconoscere le nuvole indipendentemente dal loro orientamento e aumentano la variabilità del dataset, rendendo il modello più robusto e meno incline all'overfitting.

Infine, viene applicata una normalizzazione utilizzando la media (mean) e la deviazione standard (std) calcolate in precedenza sui dati del training set. La **get_train_augmentations** viene applicata al training set, mentre per il validation set e il test set ho applicato la seconda trasformazione, **Nor**, che include solo la normalizzazione.

```
train_set=CloudDataset1(train_file, transform=True, transformN=Nor)
val_set=CloudDataset1(val_file, transform=False, transformN=Nor)
```

4.7 Salvataggi

Durante i vari addestramenti di modelli diversi, per poterli confrontare in un secondo momento, ho deciso di salvare il modello di ogni addestramento che ha ottenuto il valore di F1-score migliore sul validation set. Questo approccio mi ha permesso di conservare i migliori risultati e di confrontarli successivamente per selezionare il modello ottimale.

Per salvare i modelli, ho utilizzato la libreria PyTorch, che fornisce la funzione **`torch.save`** per memorizzare le informazioni di stato del modello e dell'ottimizzatore, e **`torch.load`** per caricarle in un secondo momento.

Il processo di salvataggio si basa sul confronto dell'F1-score ottenuto in ogni epoca. Se il nuovo F1-score è migliore di quello precedentemente salvato, il modello e il suo stato vengono aggiornati e memorizzati. Il codice utilizzato è il seguente:

```
if best_f1 < f1_score:  
    best_f1 = f1_score  
    print("cambio best f1->",best_f1 )  
    best_model_state_dict = model.state_dict()  
  
    torch.save({'model_state_dict':best_model_state_dict,  
               'optimizer_state_dict': optimizer.state_dict(),  
               'best_f1': f1_score  
             }, f'{nome}.pt'
```

In questo modo, salvo lo stato del modello con **`model.state_dict()`**, le informazioni sull'ottimizzatore con **`optimizer.state_dict()`**, e il valore del miglior F1-score ottenuto. Questo approccio è utile non solo per testare il modello in un secondo momento, ma anche per riprendere l'addestramento da dove è stato interrotto, aumentando eventualmente il numero di epoche.

Per effettuare un confronto dettagliato tra i risultati dei diversi modelli (in termini di loss, accuracy, F1-score, recall e precision), ho salvato i dati di performance per ogni modello durante le diverse epoche in file CSV, separati per il training e il validation set. Questo metodo di salvataggio dei dati mi ha permesso di confrontare i risultati ottenuti dai diversi modelli nelle varie fasi di addestramento e validation, facilitando l'identificazione del modello migliore. Inoltre, l'utilizzo di file CSV rende i dati facilmente accessibili e analizzabili anche con strumenti esterni a PyTorch. È stata anche utilizzata una seconda versione per il salvataggio del tutto simile alla prima ma che salva i modelli e i file direttamente in una cartella di Google Drive in modo da rendere i salvataggi sempre aggiornati in tempo reale.

Capitolo 5 — Risultati e Prove

5.1 Le Prime Prove

Dopo aver stabilito il Learning Rate su cui lavorare, ho avviato i primi addestramenti della rete esplorando diverse configurazioni dei parametri al fine di individuare il modello più performante.

In generale i modelli sono stati addestrati con le seguenti specifiche:

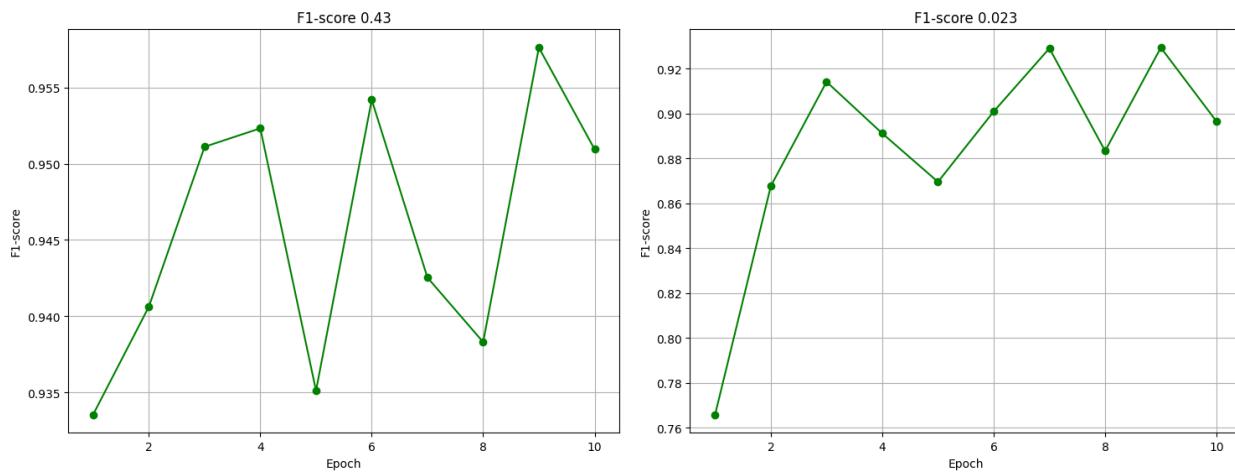
- » Training set: 6720
- » Validation set: 1680
- » Dimensione batch: 8
- » Funzione di Loss: Binary Cross Entropy Loss
- » Ottimizzatore: Adam
- » Learning Rate: 1e-5

Come discusso nei capitoli precedenti, valutare i modelli in base all'Accuracy portava a valori elevati (quasi sempre intorno al 98%), ma poco veritieri a causa dell'enorme differenza tra i pixel "nuvola" e "non nuvola". Ho quindi scelto di valutare i modelli in base al valore dell'F1-score.

Nei primi tentativi, ho esplorato l'utilizzo di pesi diversi per la Funzione di Loss. In particolare, ho provato con due impostazioni:

```
loss_fn = torch.nn.CrossEntropyLoss(weight = torch.tensor([0.43,1]).to(device))
```

Sia con 0.43 che con un valore di 0.23 per verificare se vi fossero miglioramenti significativi. I risultati ottenuti sono stati visualizzati attraverso un grafico degli F1-score che ha evidenziato un netto vantaggio nell'utilizzo del peso 0.43.



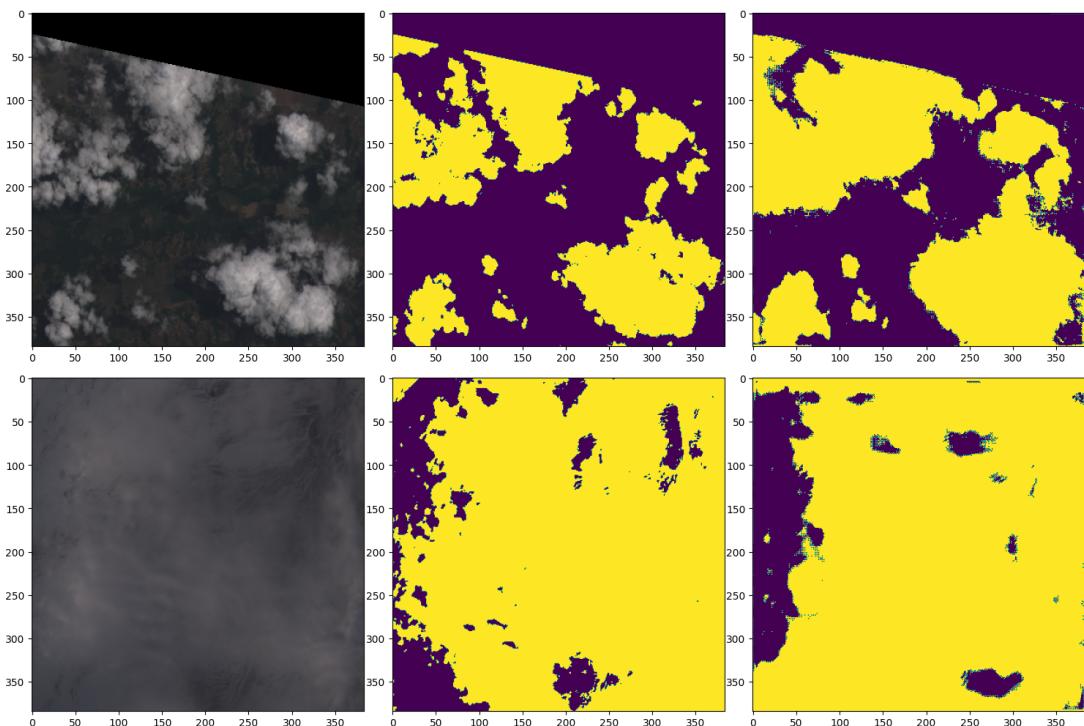
Tuttavia, a causa della quantità considerevole di dati nel dataset e delle risorse computazionali limitate, soprattutto negli ambienti di sviluppo come Colab, ho riscontrato difficoltà nell'addestramento di modelli complessi con un numero elevato di epoche. Ad esempio, anche nell'ambiente Colab Pro, l'addestramento con 15 epoche poteva richiedere oltre 15 ore, e vi era il rischio che il sistema si potesse bloccare all'improvviso. Ho dovuto inizialmente limitare

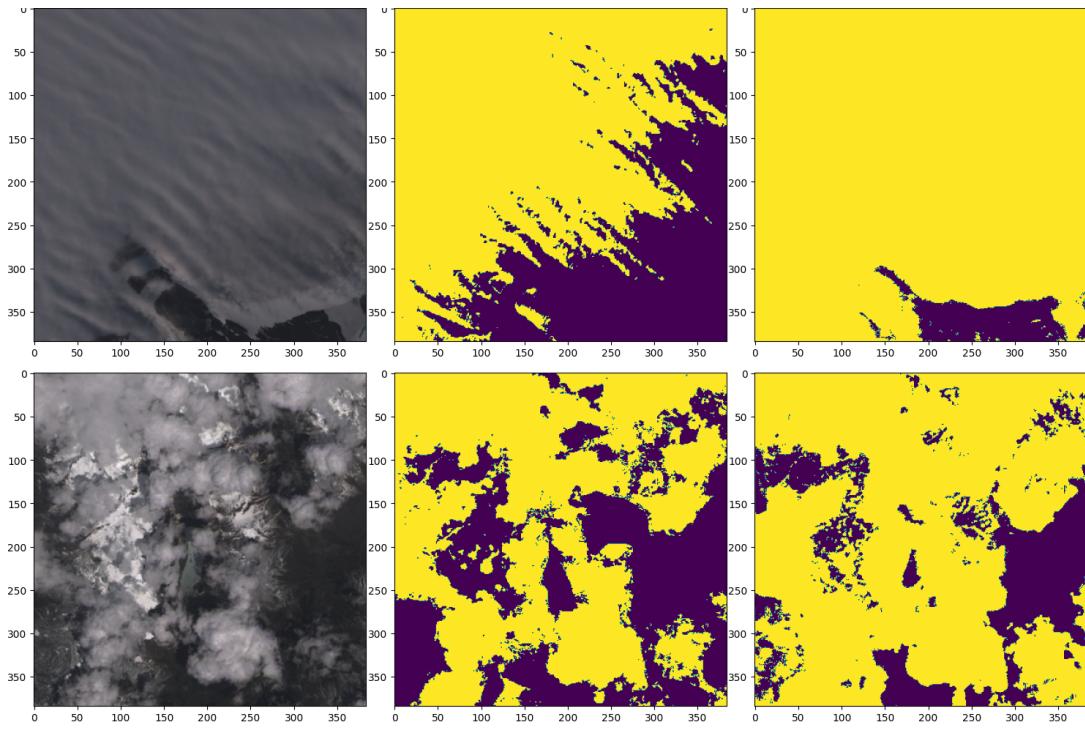
il numero di epochi negli addestramenti e valutare i modelli basandomi su un minor numero di iterazioni. Inoltre, ho eseguito addestramenti più lunghi, fino a 30 epochi, solo per i modelli più promettenti, senza riscontrare miglioramenti sostanziali. Anzi, con l'aumentare delle epochi, ho notato un aumento del rischio di overfitting.

Uno dei principali problemi che ho identificato è stato lo sbilanciamento del dataset, evidenziato anche dalla disponibilità di una versione aggiornata del dataset, il 95-Cloud. Questa estensione del 38-Cloud presenta un numero maggiore di patches nel training set, risolvendo parzialmente lo sbilanciamento tra pixel nuvola e non nuvola. Nel 95-Cloud sono presenti 34.701 patches per l'addestramento, con un set di test che include 9201 patches di 20 scene. Sebbene i set di test di 95-Cloud e 38-Cloud siano identici, i loro set di allenamento differiscono. Il 95-Cloud presenta 57 scene in più rispetto al 38-Cloud per l'allenamento, il che contribuisce a un notevole miglioramento nell'apprendimento.

5.2 Addestramento 30 epochhe

Sebbene 10 epochhe possano sembrare insufficienti per addestrare modelli complessi, come già discusso, riscontravo problemi nell'aumentare il numero di epochhe di addestramento. Oltre a problematiche legate alla memoria utilizzata durante l'addestramento, il problema principale era rappresentato da un calo delle prestazioni. Questo non era sempre evidente dai valori di F1-score, ma era chiaramente visibile nelle predizioni fatte sul validation set. Come visibile dalle immagini, con un modello di 30 epochhe, sebbene ottenessse un F1-score migliore, circa il 97%, tendeva nella maggior parte dei casi a sovrastimare la presenza di nuvole. Questo fenomeno è probabilmente dovuto al fatto che il modello ha appreso eccessivamente dai dati di training, manifestando segni di overfitting intorno alla ventesima epoca.





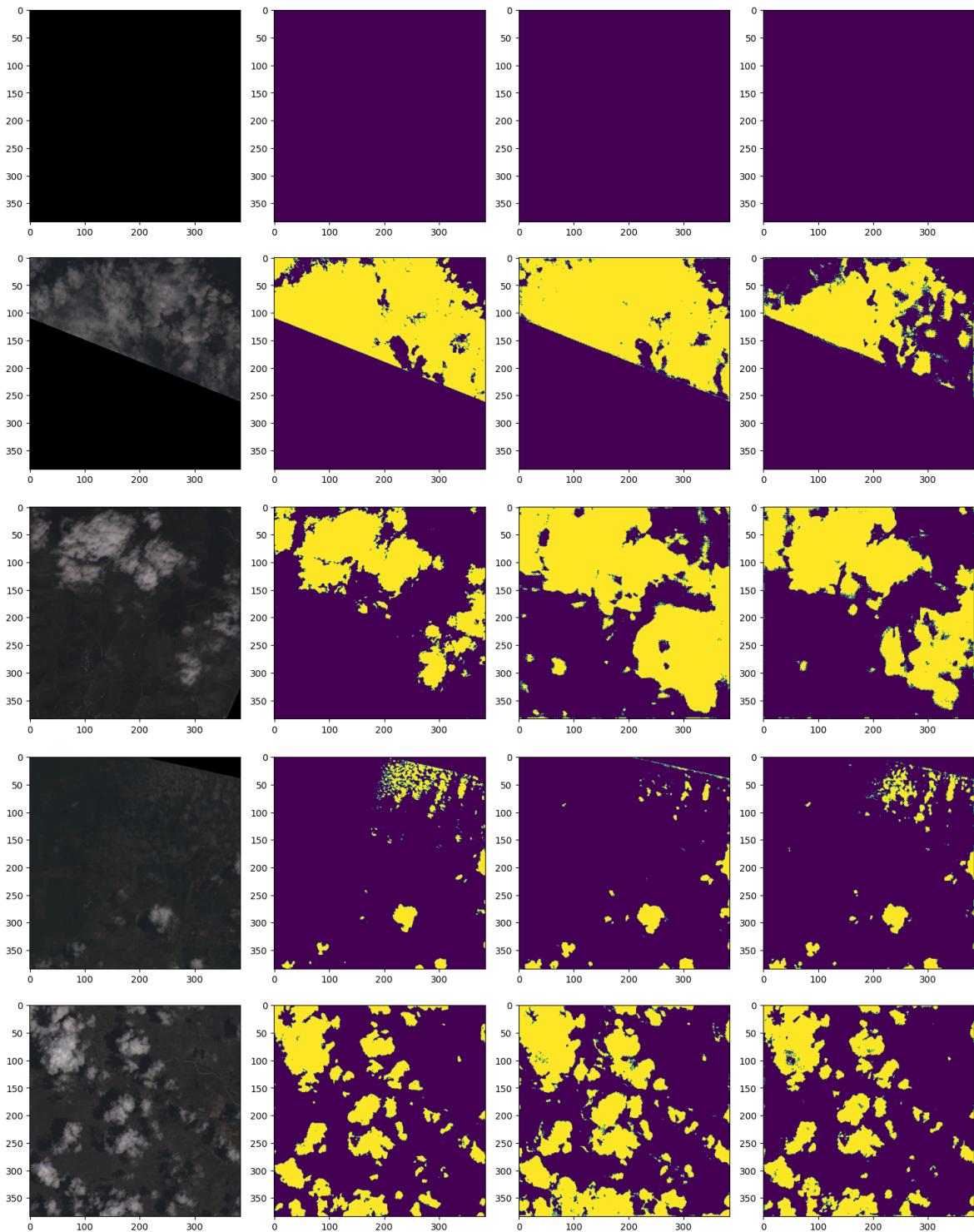
L'overfitting è un problema comune nelle reti neurali, dove il modello diventa troppo specifico rispetto ai dati di training e perde la capacità di generalizzare su nuovi dati. Questo comporta un miglioramento apparente nelle metriche di training ma un peggioramento delle performance sui dati di validazione e test.

La scelta migliore, quindi, si è rivelata essere l'addestramento con 10 o 15 epoche. Questi modelli, pur avendo un F1-score leggermente inferiore rispetto a quelli addestrati con 30 epoche, producevano predizioni più accurate e realistiche sul validation set. Una possibile spiegazione per questo comportamento risiede nel fatto che con meno epoche il modello non ha il tempo di sovra-adattarsi ai dettagli specifici del training set, mantenendo una capacità di generalizzazione migliore. Inoltre, modelli con meno epoche tendono a essere meno complessi e quindi meno inclini a memorizzare rumori o anomalie presenti nei dati di training.

5.3 10 e 15 Epoche

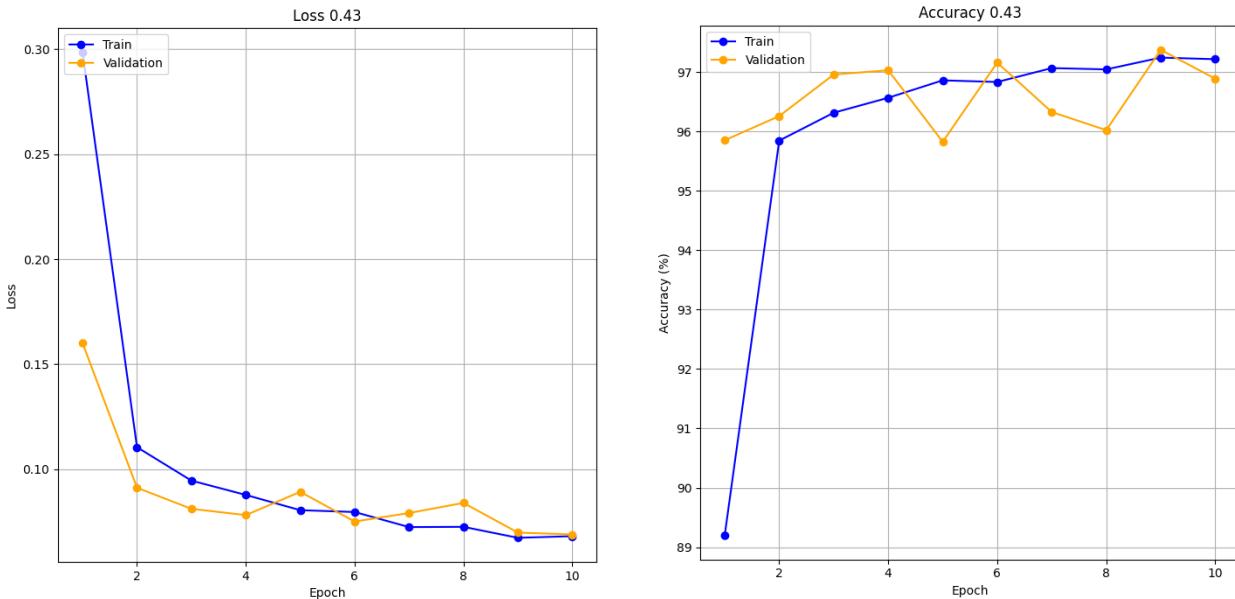
La scelta finale è quindi ricaduta sui modelli addestrati con 10 e 15 epoche. I loro valori di performance sono abbastanza simili; ho quindi deciso di valutare i modelli principalmente dal punto di vista visivo delle predizioni sul validation set, piuttosto che basarmi esclusivamente su metriche come Accuracy o F1-score, poiché la differenza risultava minima.

Per una valutazione visiva, mostrerò delle predizioni effettuate sul validation dei due modelli. L'ordine delle immagini sarà: immagine di input, Ground Truth, predizione del modello addestrato per 15 epoche e predizione del modello addestrato per 10 epoche.



Dall'analisi visiva, i risultati dei due modelli sono molto simili. Tuttavia, il modello addestrato con 10 epocha tende a non sovrastimare i pixel nuvola rispetto a quello addestrato per 15 epocha, che invece tende ad aumentare leggermente le dimensioni delle nuvole. Entrambi i modelli riescono a classificare correttamente le immagini completamente prive di pixel nuvola. Sebbene vi siano delle differenze, il modello addestrato con 10 epocha risulta, a mio parere, più preciso, soprattutto dopo aver osservato molte predizioni. Tuttavia, entrambi i modelli sono abbastanza validi e presentano buone prestazioni complessive. La scelta del modello da utilizzare dipenderà quindi dal contesto specifico e dalle esigenze pratiche del progetto.

5.4 Valutazione del Modello Finale nella Prima Versione del Dataset

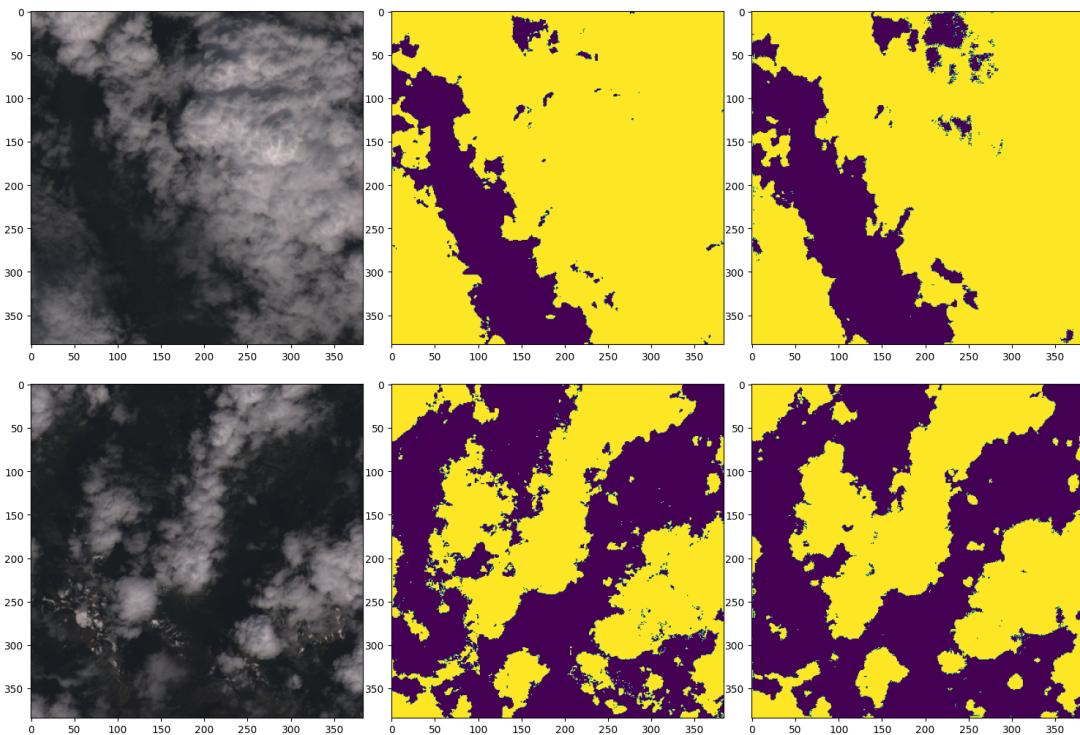


Dopo aver confrontato i due modelli, è utile fare una valutazione visiva dei risultati per identificare i punti di forza e di debolezza del modello. Prima di procedere con l'analisi visiva, esaminiamo gli andamenti di Accuracy e Loss del modello addestrato con 10 epoche con bilanciamento dei pesi tramite parametri della funzione di Loss. Osservando il grafico, si nota che la Loss, sia per il Training che per il Validation set, diminuisce con poche oscillazioni, iniziando a stabilizzarsi intorno alla decima epoca. Questo è il punto in cui ho deciso di fermare l'addestramento, poiché la limitata quantità di dati disponibili avrebbe potuto portare a risultati poco significativi e aumentare il rischio di overfitting.

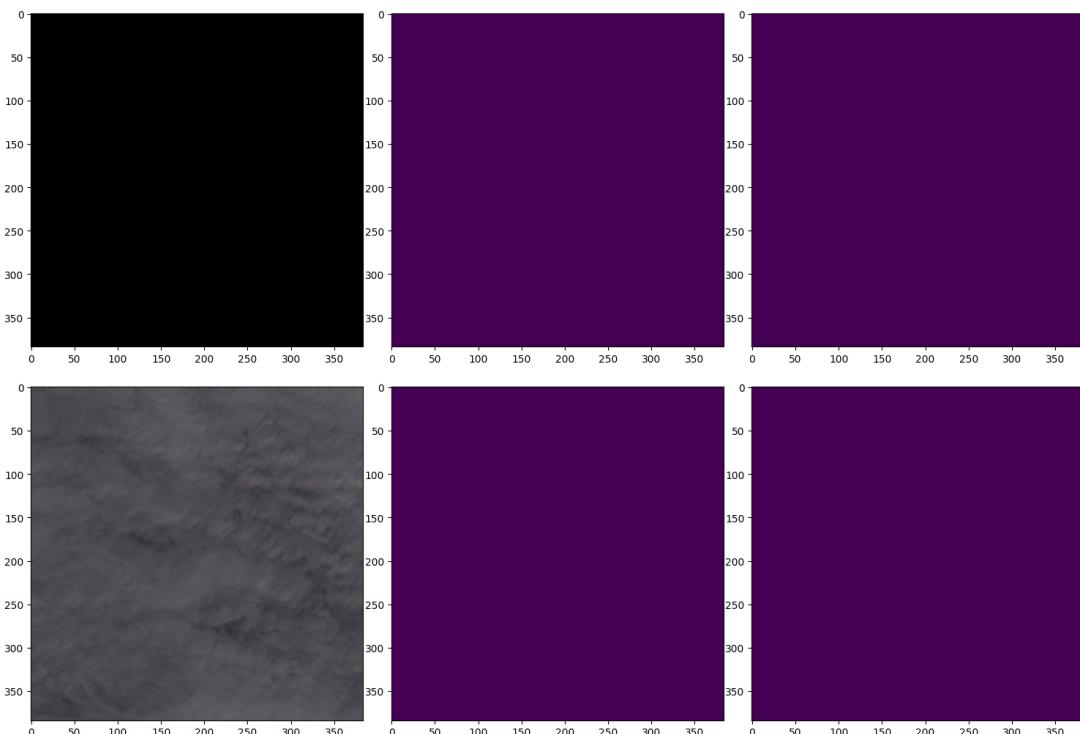
Nei grafici soprastanti è presente anche il grafico dell'Accuracy. Sebbene, come discusso nei capitoli precedenti, abbia utilizzato principalmente il valore dell'F1-score come metrica di valutazione a causa del forte sbilanciamento dei dati, riporto comunque anche il grafico dell'Accuracy per completezza. Ricordiamo che, sebbene questi valori superiori al 96% possano sembrare ottimi, possono essere fuorviatori. A causa del forte sbilanciamento del dataset, con una preponderanza di pixel "non nuvola" rispetto a quelli "nuvola", il modello tende a sovrapprendere i pixel "non nuvola". Come discusso nel capitolo 4.3, il modello potrebbe ottenere elevati valori di Accuracy classificando semplicemente tutti i pixel come "non nuvola". Questo potrebbe mascherare una scarsa capacità del modello di identificare le nuvole. Inoltre, un modello che non riesce a rilevare correttamente le nuvole potrebbe risultare inutile per applicazioni pratiche di telerilevamento, dove la precisione nella classificazione delle nuvole è essenziale.

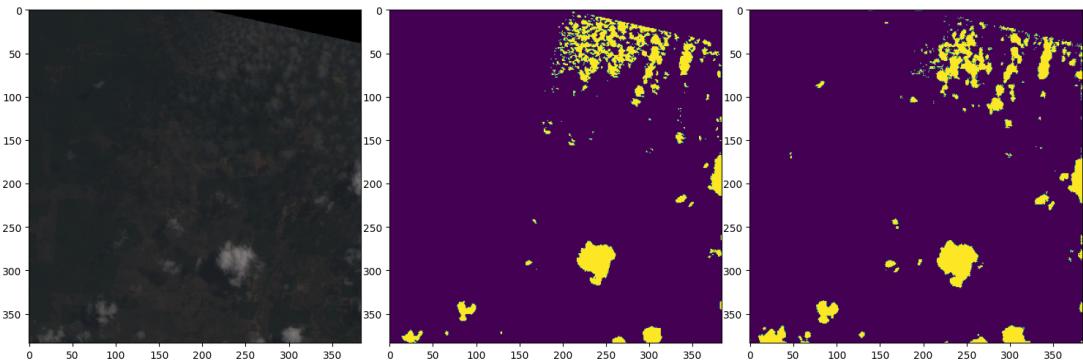
Considerando queste osservazioni, possiamo ora valutare la qualità della classificazione da un punto di vista visivo. Analizzeremo alcune predizioni usando il validation set, poiché non è presente una Ground Truth per il test set, per valutare i punti di forza e di debolezza del modello. Questa analisi visiva ci permette di vedere come il modello si comporta in situazioni reali e di

identificare eventuali aree di miglioramento. Inoltre, ci consente di comprendere meglio come il modello gestisce le diverse condizioni ambientali presenti nel dataset, come la presenza di superfici riflettenti, aree urbane e regioni innevate.



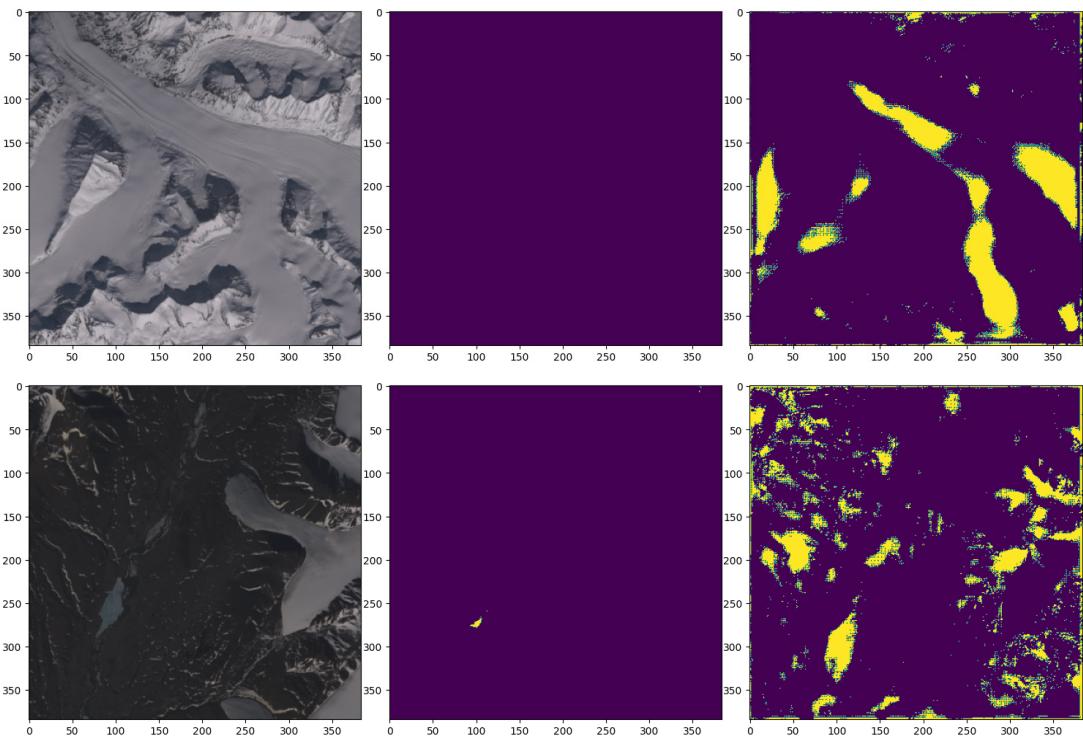
Come si può osservare nelle immagini che seguono, il modello riesce a riconoscere abbastanza bene le forme delle nuvole quando lo sfondo è scuro. Continua comunque a commettere dei piccoli errori che però sono potenzialmente trascurabili. Tali risultati si ottengono in genere su sfondi rurali oppure su sfondi in cui è presente un netto contrasto tra “nuvola” e “non nuvola”.





Un altro caso in cui il modello si comporta bene è con immagini prive di nuvola, sia completamente scure, sia in cui non sono presenti nuvole. Anche in situazioni più complesse, con poche nuvole o con nuvole molto sottili, il modello riesce a individuarle abbastanza bene, delimitandone correttamente le forme.

Il modello presenta alcune lacune significative nella distinzione tra nuvole e neve, soprattutto quando entrambe sono presenti nella stessa immagine. Come visibile nelle immagini sottostanti, se le zone sono innevate o presentano fiumi o laghi, il modello tende a classificare queste aree come "nuvola".



5.5 Modello Finale nella Seconda Versione del Dataset

Durante l'analisi dei risultati dei modelli, è emerso che la presenza di numerose immagini completamente nere nel dataset influiva negativamente sul processo di addestramento, rallentando il programma e peggiorando le prestazioni complessive del modello. Pertanto, ho deciso di implementare una fase di Data Preprocessing per eliminare queste immagini completamente

nere. Il Data Preprocessing è un passaggio cruciale nell'addestramento dei modelli di Machine Learning, poiché permette di pulire e preparare i dati in modo che il modello possa apprendere più efficacemente. In questo caso, eliminando le immagini completamente nere, ho ridotto il quantitativo complessivo di dati da processare, migliorando così l'efficienza computazionale e la qualità dell'addestramento.

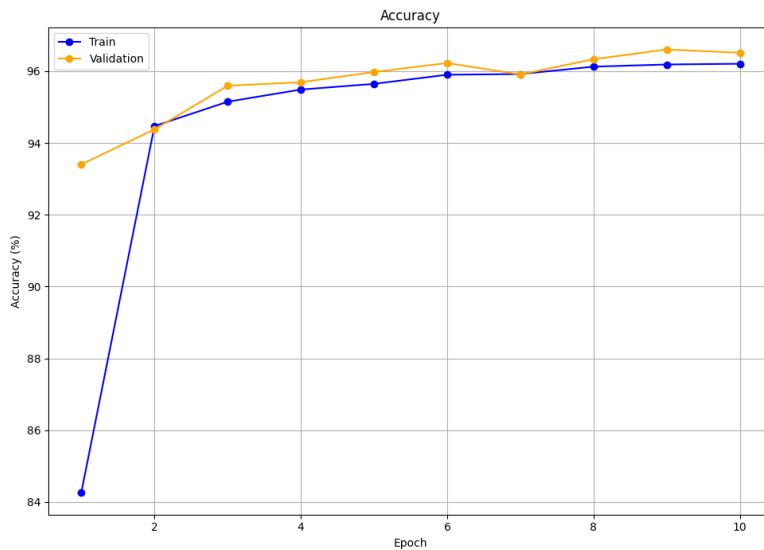
```
def elimina_black(self, num_to_remove=None):
    non_black_files = []
    black_files = []
    for i in range(len(self.files)):
        image = self.open_as_array(i, invert=True, include=True)
        if np.all(image == 0):
            black_files.append(self.files[i])
        else:
            non_black_files.append(self.files[i])

    if num_to_remove is not None and num_to_remove <
    len(black_files):
        black_files = black_files[num_to_remove:]
    else:
        black_files = []
    self.files = non_black_files + black_files

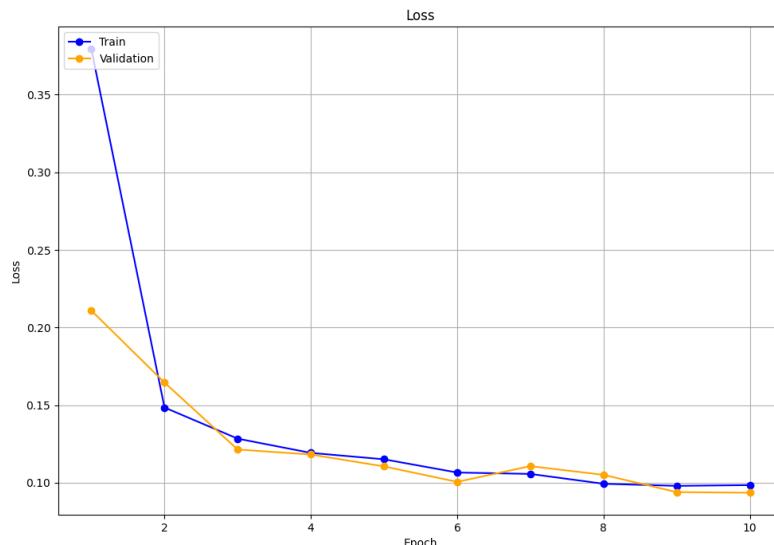
def count_black_images(self):
    black_count = 0
    for i in range(len(self)):
        image = self.open_as_array(i, invert=True, include=True)
        if np.all(image == 0):
            black_count += 1
    return black_count
```

Ho quindi introdotto nella classe **CloudDataset1** due funzioni **count_black_images** e **elimina_black**. La prima funzione, **count_black_images**, conta quante immagini nel dataset sono completamente nere. La seconda funzione, **elimina_black**, identifica queste immagini nere e le rimuove dal dataset. Se viene specificato un numero di immagini da rimuovere, la funzione elimina solo quel numero di immagini nere, altrimenti le rimuove tutte. Ho quindi identificato ed eliminato le 2813 patches completamente nere, riducendo così il dataset a 5587 patches suddivise in 4469 per il training e 1118 per il validation. Dopo aver rimosso le immagini completamente nere, ho calcolato di nuovo il rapporto “nuvola”/“non nuvola”, che è risultato essere di circa 0.83. Ho continuato ad utilizzare questo valore come peso nella funzione di loss, per mantenere il bilanciamento tra le due classi e migliorare ulteriormente le prestazioni del modello.

L'eliminazione delle immagini completamente nere ha permesso di ridurre significativamente i tempi di addestramento, consentendo di eseguire più esperimenti in un arco di tempo ridotto. Ho effettuato diversi esperimenti per trovare il modello migliore, testando sia con un learning rate di 10e-5 che di 10e-4, ottenendo in entrambi i casi risultati abbastanza simili. Per fare un confronto con il modello ottenuto nella prima versione del dataset, ho deciso di prendere in considerazione il modello addestrato con il learning rate di 10e-5. Con questo modello ho ottenuto i seguenti risultati:



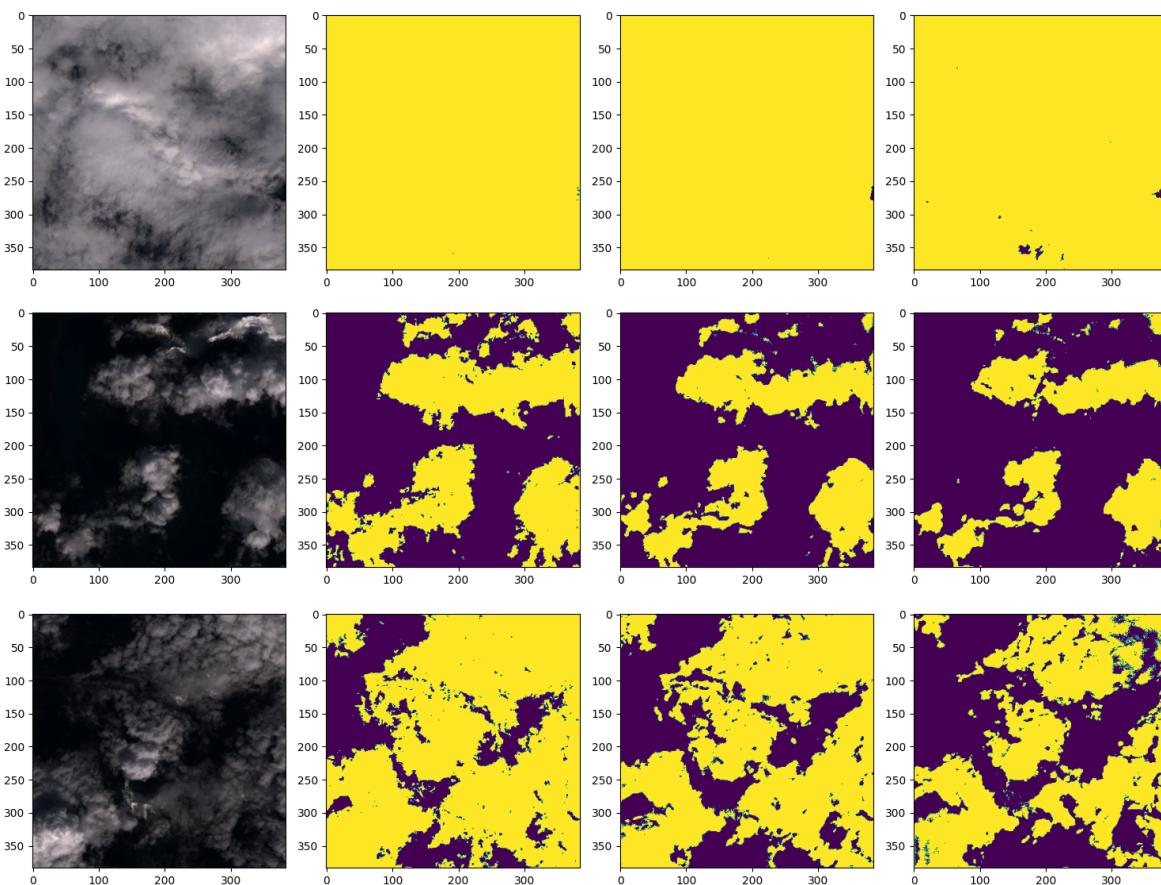
Come per i modelli precedenti, anche in questo caso ho usato come metrica di valutazione la F1-score, ma ho utilizzato l'Accuracy e la Loss principalmente per valutare se il modello è in overfitting. In questo caso, restando sulle 10 epoche, il modello continua ad apprendere senza mostrare segni di overfitting.



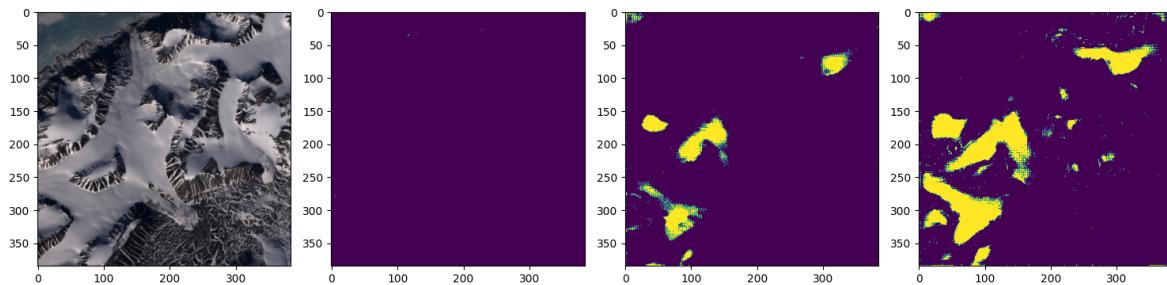
5.6 Confronto dei Due Modelli con Valutazione dei Punti Deboli e di Forza

I due modelli che analizzerò in questo paragrafo sono i migliori ottenuti con e senza l'utilizzo di immagini completamente nere. Il primo modello è stato addestrato utilizzando l'intero dataset originale, che includeva 2813 immagini completamente nere. Come discusso nei capitoli precedenti, la presenza di queste immagini ha influenzato negativamente il processo di addestramento, rallentando il programma e peggiorando le prestazioni del modello. Nonostante ciò, questo modello ha comunque mostrato buone capacità di rilevamento delle nuvole, soprattutto in situazioni con un netto contrasto tra nuvole e sfondo.

Il secondo modello, invece, è stato addestrato con il dataset pre-elaborato, dal quale sono state eliminate le immagini completamente nere. L'eliminazione di queste immagini ha migliorato l'efficienza computazionale e la qualità dell'addestramento. I valori di F1-score tra i due modelli sono molto simili, ma per valutare al meglio le differenze ho preferito fare una valutazione visiva. Questo approccio permette di verificare le differenze tra il modello addestrato sul dataset originale (ultima immagine a destra) e il modello addestrato sul dataset modificato (terza immagine da sinistra).



In generale, entrambi i modelli si comportano correttamente nell'individuare le nuvole e nel delimitarne le forme. Tuttavia, il modello addestrato sul dataset pre-elaborato riesce spesso a non sovrastimare le nuvole. Come visibile nelle immagini sottostanti, in caso di immagini innevate, il modello continua a fare errori, ma in maniera meno accentuata rispetto al modello addestrato sul dataset originale.



I punti di forza del precedente modello vengono mantenuti, e iniziano a esserci piccoli miglioramenti riguardanti i punti di debolezza che erano presenti precedentemente. In conclusione, il modello addestrato con il dataset pre-elaborato si dimostra più efficiente e preciso nel rilevamento delle nuvole rispetto al modello originale. Tuttavia, entrambi i modelli presentano aree di miglioramento che potrebbero essere ulteriormente esplorate per ottimizzare le loro prestazioni. Il principale problema è proprio quello di non confondere neve e acqua con le nuvole. Differenziare con precisione nuvole e neve nelle immagini satellitari rappresenta una sfida significativa nelle applicazioni di telerilevamento. Sia le nuvole che la neve presentano caratteristiche spettrali simili, tra cui un'elevata riflettività. Questa somiglianza può portare a classificazioni errate, dove le aree coperte di neve vengono identificate erroneamente come nuvole e viceversa.

Nel dataset 38-Cloud vengono utilizzate solo le bande RGB e NIR. Queste bande catturano le informazioni sulla luce visibile e vicino infrarosso, utili per distinguere le caratteristiche generali delle nuvole e della neve, ma non sempre forniscono dettagli sufficienti per distinguere con precisione le due. Tuttavia, vari studi dimostrano che l'utilizzo di bande aggiuntive, come quelle termiche e SWIR (Short-Wave Infrared), possono fornire informazioni supplementari che migliorano la classificazione. Un ulteriore problema è dovuto anche alla scarsità di immagini contenenti neve nel dataset. Una quantità più variegata e con maggiore presenza di neve potrebbe sicuramente migliorare l'apprendimento del modello. La mancanza di esempi sufficienti di neve durante il training limita la capacità del modello di generalizzare bene su queste caratteristiche, portando a predizioni meno accurate in queste situazioni. L'acquisizione di un dataset più bilanciato che contenga una varietà di scenari con neve potrebbe contribuire a ridurre queste lacune nel modello.

Capitolo 6 — Conclusioni

In questa relazione ho cercato di addentrarmi in un ambito molto attivo, ovvero quello del Cloud Detection. Nonostante oggigiorno si abbiano modelli sempre più performanti e precisi, si è ancora alla ricerca di miglioramenti, specialmente per immagini contenenti sia neve che nuvole. Durante la stesura della relazione e soprattutto durante la realizzazione dei modelli, mi sono soffermato molto sul dataset, poiché un punto cruciale per realizzare un modello efficace parte dallo studio approfondito del dataset stesso.

Ho osservato che il dataset 38-Cloud era fortemente sbilanciato, con una predominanza di immagini senza nuvole. Un terzo di tutte le immagini, sia di allenamento che di test, sono costituite solo da 0. Questo sbilanciamento ha portato a modelli che, sebbene mostrassero valori di Accuracy elevati, producevano predizioni imprecise. Solo dopo un attento studio del dataset e dopo aver ricercato diverse informazioni sul 38-Cloud, sono giunto alla conclusione che il dataset, essendo molto sbilanciato, non permette un adeguato addestramento dei modelli. La differenza enorme tra i pixel “nuvola” e “non nuvola” porta a risultati poco significativi e aumenta il rischio di overfitting, specialmente in contesti con molti dettagli, per questo ho cercato di ribilanciare il dataset eliminando le immagini completamente nere. Un altro problema significativo è rappresentato dalle Ground Truth imprecise, che spesso contengono errori. Questi errori sono dovuti alla classificazione manuale dei pixel per il rilevamento delle nuvole, che potrebbe essere poco precisa o approssimativa. Tali imprecisioni influenzano negativamente l’addestramento dei modelli, portando a predizioni errate. Questo è stato confermato osservando le predizioni dei modelli sul Validation Set, dove le differenze tra i pixel “nuvola” e “non nuvola” risultavano spesso sovrastimate o sottostimate.

Questi fattori possono risultare decisivi per un corretto addestramento dei modelli, soprattutto lo sbilanciamento eccessivo. Non a caso questo dataset è stato aggiornato con un’evoluzione che contiene più immagini, in modo da permettere un miglior apprendimento e evitare un’eccessiva differenza tra le classi “nuvole” e “non nuvola” oltre ad aver aggiunto ulteriori scenari per permettere un apprendimento più preciso. Questi fattori sono decisivi per un corretto addestramento dei modelli. Il forte sbilanciamento e le Ground Truth imprecise rappresentano limitazioni significative. La nuova versione, il dataset 95-Cloud, contiene molte più immagini rispetto al 38-Cloud. In particolare, il 95-Cloud include 34.701 patches per l’addestramento estratte da 75 scene, con 57 scene in più rispetto al 38-Cloud, un incremento del 317%. Questo aumento del numero di immagini e la maggiore varietà di scene potrebbero permettere un miglioramento nell’apprendimento, risolvendo in parte il problema dello sbilanciamento tra pixel “nuvola” e “non nuvola”.

D’altra parte, ritengo che al netto delle mie possibilità computazionali, la rete U-Net si sia rivelata molto performante e accurata. La U-Net ha dimostrato di essere una scelta eccellente per la segmentazione delle immagini, e i risultati ottenuti sono stati soddisfacenti considerando le limitazioni del dataset. Tuttavia, per migliorare ulteriormente il modello, sarebbe interessante lavorare con la versione aggiornata del dataset 95-Cloud, che consente di addestrare la rete U-Net

su molti più feature. L'incremento del numero di patches per l'addestramento offre maggiori opportunità per migliorare l'accuratezza e la robustezza dei modelli di rilevamento delle nuvole.

In conclusione, la realizzazione di un modello di Cloud Detection efficace richiede non solo un'architettura di rete ben progettata, ma anche un dataset di alta qualità e bilanciato. Il dataset 38-Cloud, nonostante le sue limitazioni, ha permesso di comprendere l'importanza di un dataset bilanciato e di Ground Truth accurate. L'aggiornamento al dataset 95-Cloud rappresenta un passo avanti significativo, offrendo maggiori opportunità per migliorare l'accuratezza e la robustezza dei modelli di rilevamento delle nuvole. Con più risorse computazionali e un dataset più ricco e variegato, sarà possibile ottenere modelli ancora più precisi e affidabili, capaci di distinguere con maggiore accuratezza tra nuvole e altri elementi presenti nelle immagini satellitari, come la neve.

Bibliografia

- » David P. Roy et al. “**Web-enabled Landsat Data (WELD): Landsat ETM+ composited mosaics of the conterminous United States**”, in Remote Sensing of Environment, Volume 114, 2010.
- » Zhe Zhu, Curtis E. Woodcock, “**Object-based cloud and cloud shadow detection in Landsat imagery**”, in Remote Sensing of Environment, Volume 118, 2012.
- » Sergey Ioffe, Christian Szegedy, “**Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**”, in Pmlr, 2015.
- » J. Guo et al. “**CDnetV2: CNN-Based Cloud Detection for Remote Sensing Imagery With Cloud-Snow Coexistence**”, in IEEE Transactions on Geoscience and Remote Sensing, vol. 59, 2021.
- » P. Kingma e Jimmy Ba, “**Adam: A Method for Stochastic Optimization**”, 2014.
- » T. Bai et al. “**Cloud Detection for High-Resolution Satellite Imagery Using Machine Learning and Multi-Feature Fusion**”, Remote Sens ,2016.
- » Olaf Ronneberger et al. “**U-Net: Convolutional Networks for Biomedical Image Segmentation**”, in Medical Image Computing and Computer-Assisted Intervention , 2015.

Sitografia

- » **What are Convolutional Neural Networks?**. IBM.
<https://www.ibm.com/topics/convolutional-neural-networks>
- » **Introduction to Convolution Neural Network**. Geeksforgeeks.
<https://www.geeksforgeeks.org/introduction-convolution-neural-network>
- » **Overview of Colaboratory Features**. Google Colab.
https://colab.research.google.com/notebooks/basic_features_overview.ipynb
- » **PyTorch documentation**. Pytorch.
<https://pytorch.org/>
- » **38-Cloud: Cloud Segmentation in Satellite Images**. Kaggle.
<https://www.kaggle.com/datasets/sorour/38cloud-cloud-segmentation-in-satellite-images>
- » **38-Cloud: A Cloud Segmentation Dataset**. Github.
<https://github.com/SorourMo/38-Cloud-A-Cloud-Segmentation-Dataset>
- » **How U-net Works?**. Esri Developer.
<https://developers.arcgis.com/python/guide/how-unet-works/>

- » **38-Cloud: a Cloud Segmentation Dataset**, Issues n.6, ¼ of all images in the data set are blank, Github.
<https://github.com/SorourMo/38-Cloud-A-Cloud-Segmentation-Dataset/issues/6>
- » **95 Cloud: an Extension to 38-Cloud Dataset**. Github.
<https://github.com/SorourMo/95-Cloud-An-Extension-to-38-Cloud-Database>
- » **Landsat 8**. USGS Science for a Changing World.
<https://www.usgs.gov/landsat-missions/landsat-8>
- » **Landsat 8**. NASA. <https://landsat.gsfc.nasa.gov/satellites/landsat-8/>
- » **Understanding U-net**. Towards Data Science.
<https://towardsdatascience.com/understanding-u-net-61276b10f360>
- » **Repository del progetto**. Tesi: 38-Cloud. Github.
https://github.com/franfab/Tesi_38-cloud