



SAPIENZA
UNIVERSITÀ DI ROMA

Rete neurale U-Net per la segmentazione semantica di immagini satellitari

Facoltà di Ingegneria dell'informazione, informatica e statistica
Corso di Laurea in Ingegneria Informatica ed Automatica

Emiliano Pedica
Matricola 1849824

Relatore
Prof. Domenico Lembo

Anno Accademico 2024/2025

Tesi non ancora discussa

Rete neurale U-Net per la segmentazione semantica di immagini satellitari
Tesi di laurea. Sapienza Università di Roma

© 2025 Emiliano Pedica. Tutti i diritti riservati

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

Email dell'autore: emiliano4458@gmail.com

Sommario

L'analisi delle immagini satellitari è una disciplina fondamentale per l'osservazione della Terra, che consente di estrarre informazioni dettagliate sulla superficie terrestre attraverso dati acquisiti da sensori remoti. Queste immagini, spesso multispettrali o iperspettrali, permettono di monitorare fenomeni naturali, cambiamenti ambientali e uso del suolo. Grazie all'evoluzione delle tecniche di intelligenza artificiale e deep learning, è possibile automatizzare l'interpretazione di grandi volumi di dati satellitari, migliorando l'efficienza e la precisione delle analisi. Applicazioni tipiche includono la mappatura delle foreste, il monitoraggio delle colture, la gestione delle risorse idriche e la risposta a disastri naturali. L'integrazione di modelli di segmentazione semantica consente di classificare ogni pixel dell'immagine, fornendo una visione dettagliata e strutturata del territorio.

Lo scopo di questo progetto è quello di risolvere il problema della segmentazione semantica applicata alle immagini satellitari multispettrali del dataset DynamicEarthNet. In particolare, verrà sviluppato da zero un modello in grado di classificare ogni pixel dell'immagine in una delle categorie di copertura del suolo presenti nel dataset. L'obiettivo è ottenere una segmentazione accurata e coerente. Il lavoro prevede l'analisi del dataset, la progettazione del modello, l'addestramento e la valutazione delle prestazioni tramite metriche quantitative e visualizzazioni qualitative.

La relazione è strutturata come segue: nel Capitolo 1 verranno introdotti i concetti teorici alla base della segmentazione semantica e delle reti neurali convoluzionali. Il Capitolo 2 descriverà il dataset utilizzato. Il Capitolo 3 illustrerà la rete neurale U-Net e la sua implementazione, mentre il Capitolo 4 discuterà delle tecniche di addestramento utilizzate. Il Capitolo 5 analizzerà i risultati ottenuti e infine il Capitolo 6 concluderà con una riflessione sul lavoro svolto.

Indice

| | |
|---|-----------|
| 1 Introduzione | 1 |
| 1.1 La segmentazione semantica | 1 |
| 1.2 Obiettivi del progetto | 2 |
| 1.3 Strumenti e tecnologie | 2 |
| 1.3.1 Reti neurali convoluzionali | 2 |
| 1.3.2 Matrice di convoluzione | 3 |
| 1.3.3 Ambiente e librerie | 3 |
| 2 Il dataset DynamicEarthNet | 5 |
| 2.1 Descrizione del dataset | 5 |
| 2.2 Implementazione della classe DynamicEarthNetDataset | 7 |
| 2.3 Visualizzazione delle immagini multispettrali | 8 |
| 2.4 Data augmentation | 8 |
| 3 Rete neurale U-Net | 10 |
| 3.1 Introduzione | 10 |
| 3.2 Fasi di contrazione ed espansione | 12 |
| 3.2.1 I blocchi di contrazione | 12 |
| 3.2.2 Bottleneck | 14 |
| 3.2.3 Blocchi di espansione | 14 |
| 3.3 Skip connections | 15 |
| 4 Addestramento | 16 |
| 4.1 Ambiente di addestramento | 16 |
| 4.2 Salvataggio delle epoche | 16 |
| 4.3 Algoritmo di ottimizzazione | 17 |
| 4.4 Funzione di loss e bilanciamento delle classi | 17 |
| 4.5 Metriche utilizzate | 18 |
| 4.6 Tecniche per migliorare l'apprendimento | 19 |
| 5 Risultati | 21 |
| 5.1 Risultati visivi | 21 |
| 5.2 Metriche addestramento | 25 |
| 6 Conclusioni | 27 |
| Bibliografia | 28 |

Capitolo 1

Introduzione

1.1 La segmentazione semantica

La segmentazione semantica è un processo di computer vision che associa ogni pixel di un'immagine ad una categoria utilizzando un algoritmo di deep learning. I modelli per la segmentazione delle immagini permettono alle macchine di analizzare e comprendere i contenuti visivi replicando processi cognitivi simili a quelli della percezione umana. Le applicazioni di questa tecnologia spaziano in una varietà di settori come la guida autonoma, l'imaging medicale, la visione robotica, l'agricoltura e in generale in tutti quei campi dove è essenziale conoscere con precisione la disposizione spaziale degli oggetti. La segmentazione semantica si distingue da altri approcci di analisi delle immagini per la sua capacità di fornire una comprensione dettagliata del contenuto visivo. A differenza della classificazione di immagini, che assegna una singola etichetta all'intera immagine, o della object detection, che identifica oggetti specifici attraverso dei riquadri, la segmentazione semantica offre una granularità a livello di pixel che consente di identificare con precisione i confini tra le diverse aree. Nel contesto delle immagini satellitari, questa tecnologia assume particolare rilevanza considerando l'eterogeneità dei paesaggi.



Figura 1.1 Esempio di segmentazione semantica⁽¹⁾

1.2 Obiettivi del progetto

L’obiettivo del progetto è applicare la segmentazione semantica alle immagini satellitari con il fine di associare ogni pixel a una specifica categoria di copertura del suolo. Questa tecnologia consentirebbe di identificare e mappare automaticamente diverse tipologie di superficie terrestre, come aree urbane, foreste, terreni agricoli e acqua, fornendo informazioni cruciali per la pianificazione territoriale, il monitoraggio ambientale e la gestione sostenibile delle risorse.

Negli ultimi anni, la crescente disponibilità di immagini satellitari ad alta risoluzione, insieme ai progressi nell’intelligenza artificiale e nel deep learning, ha aperto nuove opportunità per l’analisi automatica del territorio. In questo campo, la segmentazione semantica si è affermata come uno strumento fondamentale per estrarre informazioni significative da grandi quantità di dati.

1.3 Strumenti e tecnologie

1.3.1 Reti neurali convoluzionali

Per affrontare il problema della segmentazione semantica, il lavoro si basa sull’utilizzo delle reti neurali convoluzionali (CNN), una classe di algoritmi di deep learning particolarmente efficace nell’elaborazione di dati strutturati come le immagini. Le CNN si sono dimostrate fondamentali nel campo della computer vision grazie alla loro capacità di apprendere automaticamente features gerarchiche attraverso operazioni di convoluzione, pooling e attivazione non lineare. Tuttavia, mentre le CNN tradizionali eccellono in compiti di classificazione di immagini, la semantic segmentation richiede un approccio che permetta di mantenere la risoluzione spaziale dell’immagine di input e di produrre una mappa di segmentazione pixel-wise. Per questo motivo, è stata adottata l’architettura U-Net, una rete neurale convoluzionale specificamente progettata per compiti di segmentazione semantica di immagini mediche. Nei capitoli successivi verranno approfonditi il suo funzionamento e la sua struttura.

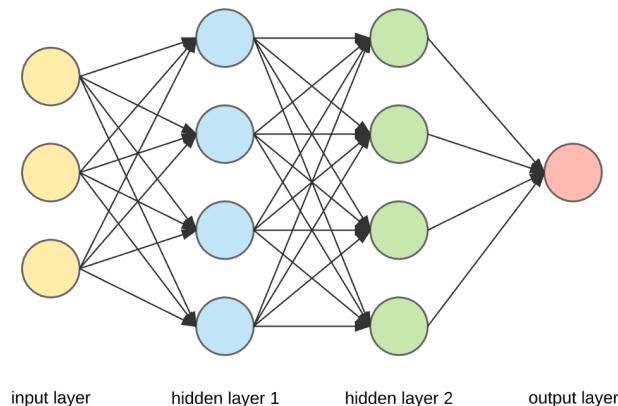


Figura 1.2 Layer di una rete neurale⁽²⁾

1.3.2 Matrice di convoluzione

Nell'elaborazione delle immagini una matrice di convoluzione (kernel) è una matrice di valori utilizzata per applicare dei filtri alle immagini, con il fine di evidenziare delle caratteristiche utili. Queste matrici sono generalmente di piccole dimensioni come 3x3 o 5x5 e contengono coefficienti numerici che determinano il tipo di trasformazione da applicare all'immagine. Durante l'operazione di convoluzione, il kernel viene fatto scorrere su ogni pixel dell'immagine di input calcolando per ciascuna posizione la somma pesata dei prodotti di ciascun elemento della matrice kernel con il corrispondente pixel della matrice sottostante. I kernel possono essere progettati per svolgere diverse funzioni specifiche: i filtri passa-basso permettono di ridurre il rumore e sfuocare l'immagine, mentre i filtri passa-alto sono utilizzati per l'individuazione dei contorni e l'accentuazione dei dettagli. Kernel specifici come l'operatore di Sobel o il filtro di Prewitt sono ottimizzati per il rilevamento di bordi in direzioni particolari, mentre il filtro gaussiano è impiegato per operazioni di smoothing preservando le caratteristiche principali dell'immagine.

In questa implementazione, le matrici di convoluzione saranno fondamentali per l'estrazione di features.

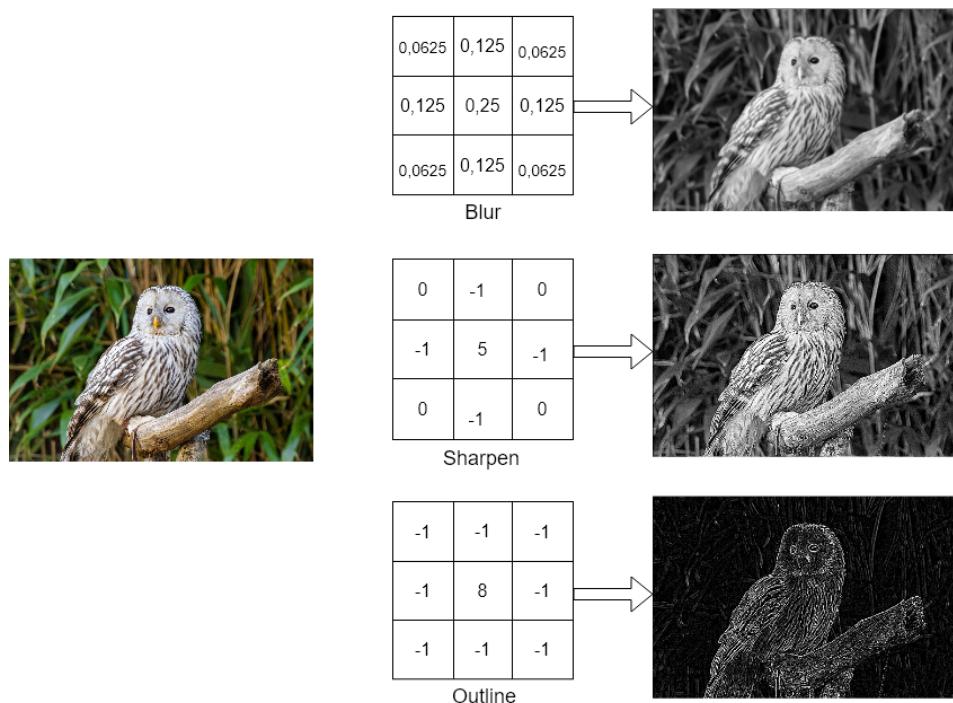


Figura 1.3 Esempi di kernel applicati ad un'immagine

1.3.3 Ambiente e librerie

Soltanamente, per l'addestramento di reti neurali convoluzionali è necessaria una potenza di calcolo elevata. Non avendo a disposizione una macchina abbastanza potente è stata utilizzata la piattaforma Google Colab che mette a disposizione delle GPU (Graphic Processing Unit) adatte all'addestramento. All'interno di Colab è stato usato il linguaggio di programmazione Python, essendo diventato lo standard

de facto per l'implementazione delle reti neurali grazie alla sua sintassi semplice e leggibile che facilita la prototipazione rapida. Python è particolarmente adatto anche grazie alla ricchezza del suo ecosistema di librerie specializzate come TensorFlow, PyTorch , Keras e Scikit-learn che offrono strumenti potenti e ottimizzati per il machine learning. Qui, nello specifico, verrà utilizzato il framework PyTorch per l'implementazione.

Per la gestione delle immagini verrà usata Rasterio, una libreria Python open source che permette di aprire le immagini satellitari multispettrali e le rispettive label. Altre librerie utilizzate sono Albumentations per il preprocessing delle immagini, Scikit-learn per le metriche, NumPy e Matplotlib per la manipolazione dei dati e la visualizzazione. Tutto il codice del progetto è disponibile sulla mia repository github: <https://github.com/emi4458/unet-satellite-segmentation/tree/main>⁽³⁾

Capitolo 2

Il dataset DynamicEarthNet

2.1 Descrizione del dataset

DynamicEarthNet⁽⁴⁾ è un dataset per l'analisi multitemporale di dati satellitari ad alta risoluzione. Contiene sequenze temporali di immagini satellitari multispettrali che catturano i cambiamenti dinamici della superficie terrestre in 75 regioni geografiche, con il fine di fornire dati utili per lo sviluppo di algoritmi di machine learning capaci di comprendere e predire l'evoluzione spazio-temporale dei fenomeni terrestri. Nella versione originale è composto da 54750 immagini con una risoluzione di 1024x1024 in 4 bande: RGB e near-infrared. DynamicEarthNet è corredata da annotazioni ground truth complete che lo rendono ideale per approcci di supervised learning. Ogni area di interesse dispone di etichette semantiche che identificano ogni pixel nelle 7 diverse categorie del suolo.

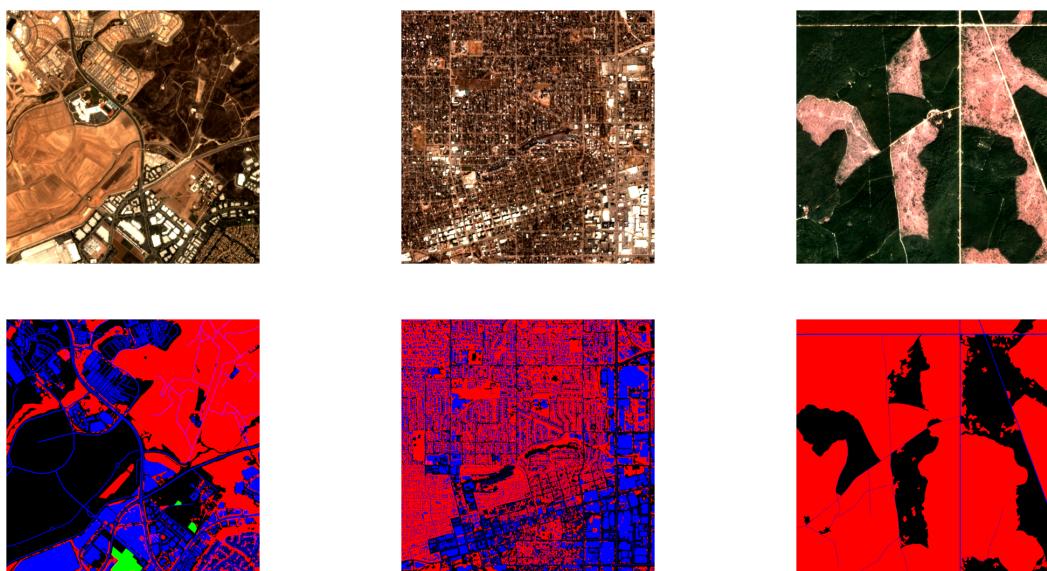


Figura 2.1 Campioni di immagini dal dataset con le rispettive Ground Truth

Il dataset presenta una risoluzione spaziale di 10 metri per pixel, derivata dalle immagini di Sentinel-2 dell’Agenzia Spaziale Europea (ESA). La copertura temporale si estende dal 2018 al 2019, con acquisizioni mensili che permettono di osservare l’evoluzione stagionale dei paesaggi terrestri. Le 75 regioni geografiche sono distribuite su diversi continenti e includono una varietà di ecosistemi: foreste, aree agricole, zone urbane, deserti e regioni costiere. Questa diversità geografica garantisce la robustezza dei modelli addestrati, rendendoli applicabili a contesti ambientali differenti. Il processo di annotazione delle etichette, come riportato nel paper originale, è stato rigoroso con un’emfasi sulla coerenza temporale delle etichette. La prima immagine del mese per ogni area di interesse è stata annotata manualmente e utilizzata come base per i mesi seguenti. Le mappe successive sono state aggiornate solo nel caso di un cambiamento percettibile in una certa regione che è evidente all’annotatore umano, tre controlli di qualità garantiscono l’accuratezza delle annotazioni.

Le immagini del dataset sono nel formato TIFF, essendo uno standard ottimale per i dati satellitari che supporta profondità di colore elevate (16-32 bit per canale) che preservano l’intera gamma dinamica dei sensori spaziali. Il formato TIFF consente inoltre di incorporare metadati geospaziali come coordinate e sistemi di riferimento. La sua compressione senza perdite garantisce che non si perdano informazioni preziose durante l’archiviazione, mantenendo l’accuratezza scientifica necessaria per analisi approfondate.

In questa implementazione, essendo un dataset di grandi dimensioni, verrà utilizzata una versione ridotta composta da 1080 immagini per il train set e 120 per il validation set. Proprio per questo sono state escluse le classi WetLands e Ice&Snow perché quasi assenti.



Figura 2.2 Il satellite Sentinel-2A⁽⁵⁾

2.2 Implementazione della classe DynamicEarthNetDataset

La suddivisione del dataset nei train, validation e test set è stata già effettuata in precedenza nei file txt che contengono la posizione nel file system delle rispettive immagini.

```

1  class DynamicEarthNetDataset(Dataset):
2      def __init__(self, file_name, transform=None):
3          self.transform = transform
4          self.images, self.masks = get_paths(file_name)
5          self.num_classes = 7

```

Listing 2.1 Inizializzazione della classe DynamicEarthNetDataset

La funzione init prende in input il file txt con la lista dei percorsi delle immagini già suddivise nei vari dataset e le eventuali trasformazioni.

Listing 2.2 Metodo getitem

```

6  def __getitem__(self, index):
7      img_path = self.images[index]
8      mask_path = self.masks[index]
9
10     image = rasterio.open(img_path).read()
11     image = image.astype(np.float32)
12
13     label = rasterio.open(mask_path).read()
14     mask = np.zeros((label.shape[1], label.shape[2]), dtype=np.int64)
15
16     for i in range(self.num_classes):
17         if i == 6:
18             mask[label[i, :, :] == 255] = -1
19         elif i == 3:
20             mask[label[i, :, :] == 255] = -1
21         elif i > 3:
22             mask[label[i, :, :] == 255] = i - 1
23         else:
24             mask[label[i, :, :] == 255] = i
25
26     return image, mask

```

Successivamente è stato definito il metodo getitem che restituisce l'immagine e la relativa label. La funzione sfrutta la libreria Rasterio per aprire i file in formato .tif e successivamente prepara le maschere creando una matrice di zeri che verrà utilizzata per convertire le immagini da multi-channel a single-channel con gli indici delle classi. Avendo ignorato le due classi di indice 6 (icesnow) e 3 (wetlands) il codice imposta i relativi pixel a -1 così che possano essere ignorati durante l'addestramento e il calcolo delle metriche.

2.3 Visualizzazione delle immagini multispettrali

Le immagini del dataset essendo in 4 bande non possono essere visualizzate a schermo come normali foto RGB. È quindi necessario trasformarle per effettuare controlli visivi:

Listing 2.3 Metodo multispectral_to_rgb_visualization

```

1 def multispectral_to_rgb_visualization(img, lower_percentile=5,
2                                         upper_percentile=95):
3     if img.ndim == 2:
4         img = img[:, :, np.newaxis]
5         img = img.transpose(1, 2, 0)
6         img = img[:, :, [2, 1, 0]]
7         img = np.clip(img, np.percentile(img, lower_percentile), np.
8             percentile(img, upper_percentile))
9         img = (img - np.min(img)) / (np.max(img) - np.min(img))
10        img = (img * 255).astype(np.uint8)
11
12    return img

```

La funzione multispectral_to_rgb_visualization trasforma un'immagine multispettrale in una standard RGB che può essere mostrata a schermo. Per farlo è necessario togliere la banda NIR ed effettuare alcune trasformazioni. Il metodo prende in input un'immagine e riorganizza le dimensioni dell'array per mappare le varie bande nel formato standard. Poi, per migliorare il contrasto, taglia i valori estremi mantenendo solo quelli tra il 5° e il 95° percentile, eliminando così valori elevati che potrebbero compromettere la visualizzazione. Successivamente normalizza l'immagine in un range 0-1 sottraendo il minimo e dividendo per l'escursione totale, quindi scala tutto a valori compresi tra 0 e 255 convertendo in formato uint8. Il risultato è un'immagine RGB standard.

2.4 Data augmentation

Per aumentare la varietà dei dati viene usata la libreria Albumentations di Python che ha permesso di applicare delle trasformazioni alle immagini del dataset. L'obiettivo è quello di migliorare le prestazioni del modello introducendo rotazioni e specchiature delle immagini, riducendo l'overfitting e aumentando la capacità di generalizzazione su nuovi dati non visti durante l'addestramento.

Listing 2.4 Trasformazioni delle immagini al set di training

```

1 train_transform = Albumentations.Compose(
2     [
3         Albumentations.Rotate(limit=35, p=1.0),
4         Albumentations.HorizontalFlip(p=0.5),
5         Albumentations.VerticalFlip(p=0.1),
6         ToTensorV2(),
7     ],
8 )

```

La variabile p rappresenta la probabilità che una specifica trasformazione venga applicata. Di seguito un paio di esempi:

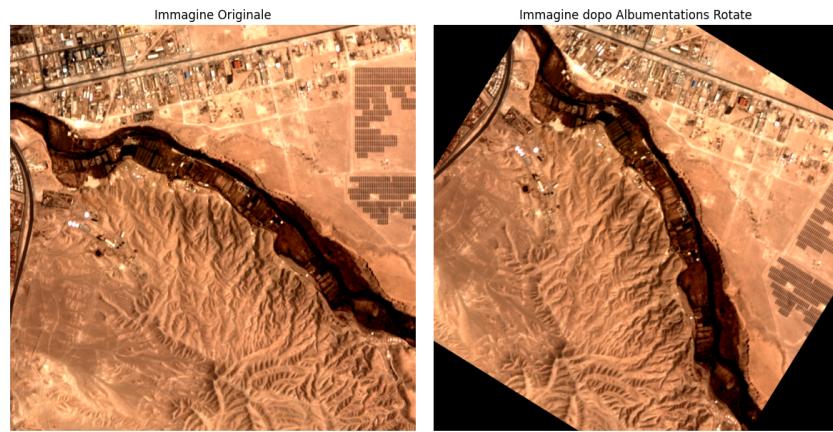


Figura 2.3 Esempio di rotazione

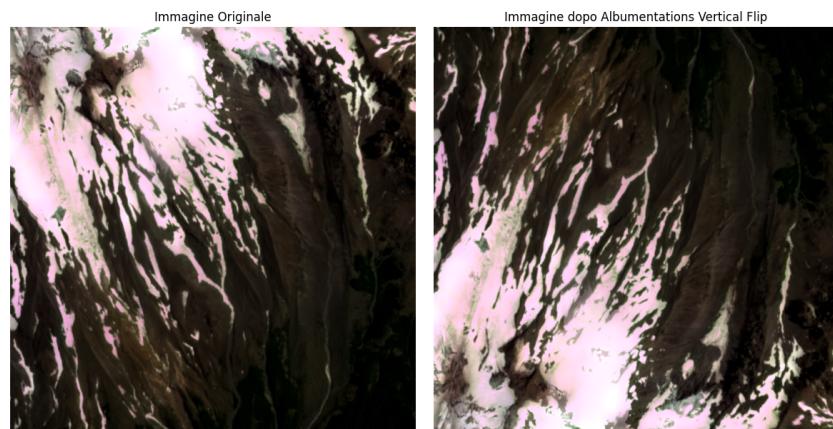


Figura 2.4 Esempio di specchiatura verticale

Capitolo 3

Rete neurale U-Net

3.1 Introduzione

Una U-Net è una rete neurale convoluzionale progettata appositamente per la segmentazione semantica. Nel suo paper originale *U-Net: Convolutional Networks for Biomedical Image Segmentation*⁽⁶⁾ viene applicata a delle immagini mediche per la segmentazione delle cellule, dei nuclei e delle loro membrane. La U-Net permetteva di mantenere un'accuratezza comparabile a quella umana, anche con dataset di training relativamente piccoli (aspetto cruciale in ambito medico). L'architettura si è rivelata così efficace che è poi diventata lo standard de facto per compiti di segmentazione in molti altri domini oltre quello medico. La struttura della rete si basa sulle fasi di contrazione ed espansione che le conferiscono la caratteristica forma ad U da cui deriva il nome.

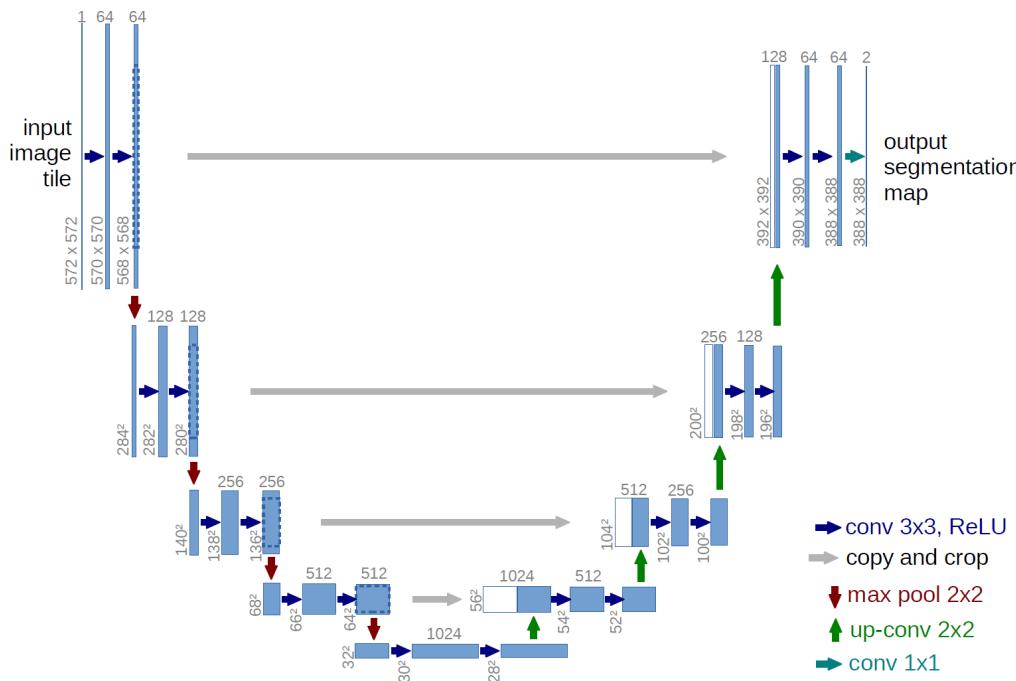


Figura 3.1 Architettura della U-Net⁽⁶⁾

La rete è composta da 4 blocchi di contrazione seguiti dalla parte più profonda della rete chiamata bottleneck dove viene raggiunto il massimo numero di canali con la minor risoluzione spaziale. Essa rappresenta il collegamento con la fase di espansione formata da altri 4 blocchi che riportano le dimensioni allo stato iniziale. Ad ogni fase di contrazione ed espansione vengono effettuate due convoluzioni consecutive e vengono aggiunte le skip connections. Le skip connections sono collegamenti diretti che trasportano informazioni dai blocchi di contrazione ai corrispondenti blocchi di espansione dello stesso livello. Queste connessioni sono fondamentali per il funzionamento della U-Net e verranno approfondite nei paragrafi successivi.

Lo scopo della riduzione spaziale e l'aumento dei canali è quello di creare una rappresentazione gerarchica dell'immagine che catturi progressivamente features sempre più complesse e astratte. Mentre le dimensioni spaziali si riducono, la rete può concentrare la sua capacità computazionale su un maggior numero di filtri specializzati, ridistribuendo le risorse di elaborazione.

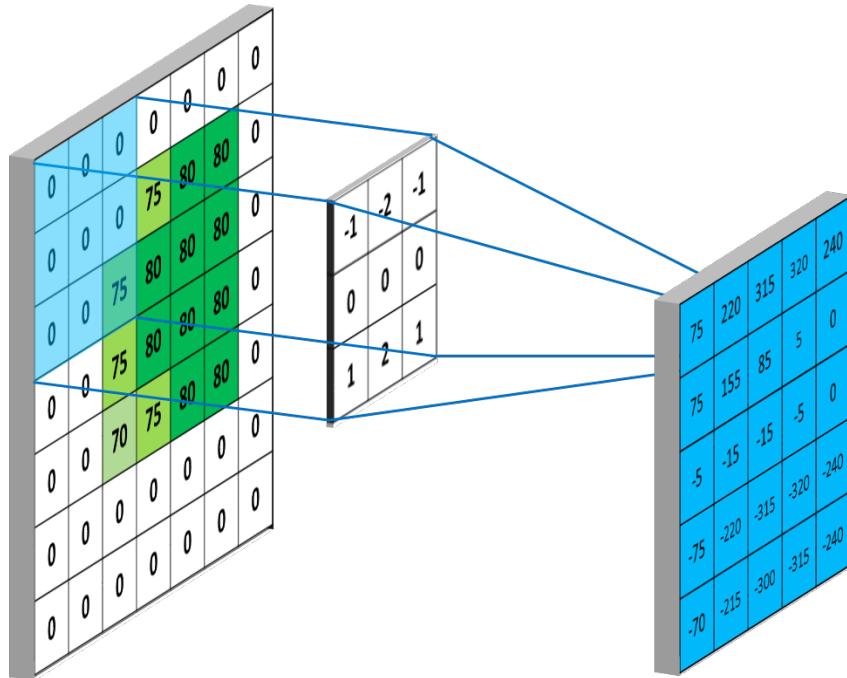


Figura 3.2 Operazione di convoluzione

Listing 3.1 Classe UNET

```

1 class UNET(nn.Module):
2     def __init__(self, in_channels=3, out_channels=1, features=[64, 128, 256,
3         512],
4     ):
5         super(UNET, self).__init__()
6         self.ups = nn.ModuleList()
7         self.downs = nn.ModuleList()
8         self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
9
10        for feature in features:
11            self.downs.append(DoubleConv(in_channels, feature))
12            in_channels = feature
13
14        for feature in reversed(features):
15            self.ups.append(
16                nn.ConvTranspose2d(
17                    feature*2, feature, kernel_size=2, stride=2,
18                )
19            )
20            self.ups.append(DoubleConv(feature*2, feature))
21
22        self.bottleneck = DoubleConv(features[-1], features[-1]*2)
23        self.final_conv = nn.Conv2d(features[0], out_channels, kernel_size
=1)

```

3.2 Fasi di contrazione ed espansione

Listing 3.2 Classe DoubleConv

```

1 class DoubleConv(nn.Module):
2     def __init__(self, in_channels, out_channels):
3         super(DoubleConv, self).__init__()
4         self.conv = nn.Sequential(
5             nn.Conv2d(in_channels, out_channels, 3, 1, 1, bias=False),
6             nn.BatchNorm2d(out_channels),
7             nn.ReLU(inplace=True),
8             nn.Conv2d(out_channels, out_channels, 3, 1, 1, bias=False),
9             nn.BatchNorm2d(out_channels),
10            nn.ReLU(inplace=True),
11        )

```

3.2.1 I blocchi di contrazione

La fase di contrazione avviene in 4 blocchi all'interno dei quali sono presenti 2 layer convoluzionali. Tra questi due layer vengono effettuate due operazioni fondamentali: la batch normalization con `BatchNorm2d(7)` e l'applicazione della funzione di attivazione `ReLU(8)`. La batch normalization è una tecnica che normalizza le

attivazioni di ogni strato di una rete neurale per migliorare la stabilità e velocizzare la convergenza. Il processo funziona calcolando la media e la varianza delle attivazioni all'interno di ciascun batch di dati, sottraendo la media e dividendo per la deviazione standard per ottenere attivazioni con media zero e varianza unitaria. Aiuta a risolvere il problema del covariate shift interno, dove la distribuzione degli input a ciascun strato cambia continuamente durante l'addestramento, rendendo più difficile l'ottimizzazione.

La funzione di attivazione ReLU ha lo scopo di introdurre una non linearità nella rete neurale, consentendole di apprendere relazioni complesse tra le caratteristiche estratte. È particolarmente apprezzata grazie alla sua semplicità di calcolo ed efficacia nel mitigare il problema della scomparsa del gradiente che limita l'apprendimento. Essa è definita come:

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

Nella pratica va a spegnere i neuroni con valori negativi impostandoli a zero e lascia tutti gli altri invariati.

Listing 3.3 Funzione forward

```

1 def forward(self, x):
2     skip_connections = []
3
4     for down in self.downs:
5         x = down(x)
6         skip_connections.append(x)
7         x = self.pool(x)
8
9         x = self.bottleneck(x)
10    skip_connections = skip_connections[::-1]
11
12    for idx in range(0, len(self.ups), 2):
13        x = self.ups[idx](x)
14        skip_connection = skip_connections[idx//2]
15
16        if x.shape != skip_connection.shape:
17            x = TF.resize(x, size=skip_connection.shape[2:])
18
19        concat_skip = torch.cat((skip_connection, x), dim=1)
20        x = self.ups[idx+1](concat_skip)
21
22    return self.final_conv(x)

```

Tra un blocco e l'altro viene effettuata l'operazione di pooling tramite la funzione MaxPool2d⁽⁹⁾ che, con un kernel 2x2, dimezza le dimensioni spaziali delle immagini preservando allo stesso tempo le feature più prominenti, scartando quelle meno rilevanti.

3.2.2 Bottleneck

La fase bottleneck rappresenta il punto più profondo della U-Net, dove le feature map raggiungono la risoluzione spaziale minima ma il numero massimo di canali. In questo stadio, dopo aver attraversato tutti i blocchi di contrazione, l'immagine ha subito il massimo downsampling possibile con una risoluzione di 62x62, mentre i canali sono aumentati fino al valore massimo, 512 in questo caso.

3.2.3 Blocchi di espansione

Un blocco di espansione è molto simile a quello di contrazione. La differenza è nell'ultima operazione dove anziché effettuare l'operazione di pooling viene utilizzata la convoluzione trasposta ConvTranspose2d⁽¹⁰⁾ che raddoppia le dimensioni spaziali per ricostruire gradualmente la risoluzione originale dell'immagine.

Nel dettaglio, la convoluzione trasposta con kernel 2x2 e stride 2 ha l'effetto opposto del max pooling: mentre il pooling riduce le dimensioni da $H \times W$ a $H/2 \times W/2$, la convoluzione trasposta le aumenta da $H/2 \times W/2$ a $H \times W$.

Di seguito la rappresentazione delle operazioni nei blocchi di contrazione ed espansione :

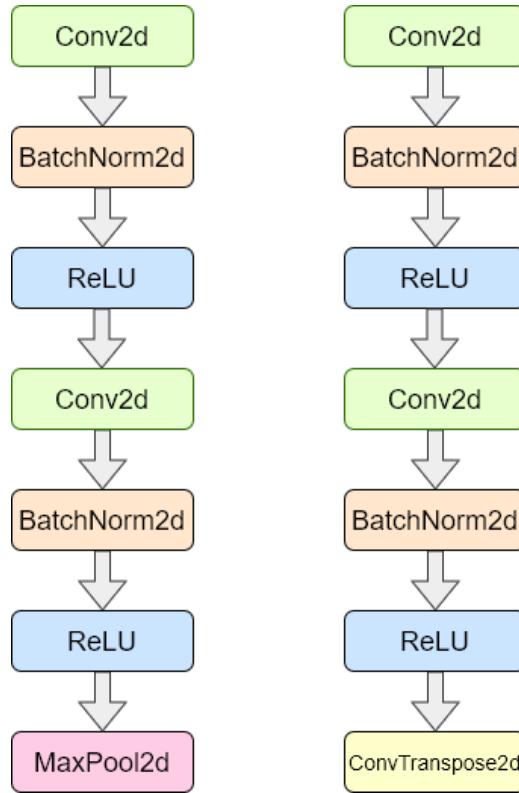


Figura 3.3 A sinistra i passaggi di un blocco di contrazione, a destra quelli di espansione

3.3 Skip connections

Le skip connections rappresentano uno degli elementi architetturali più innovativi e fondamentali della U-Net, costituendo il collegamento diretto tra i layer del percorso di contrazione e quelli corrispondenti del percorso di espansione. Queste connessioni permettono di trasferire informazioni ad alta risoluzione dai primi stadi della rete direttamente ai layer di ricostruzione. Durante la fase di contrazione, le feature map vengono salvate prima di ogni operazione di max pooling e successivamente concatenate con le corrispondenti feature map del percorso di espansione, che si trovano alla stessa risoluzione spaziale grazie alla simmetria dell'architettura.

Questo meccanismo risolve un problema critico nella segmentazione semantica: la perdita di dettagli spaziali durante la fase di contrazione. Mentre il percorso di contrazione cattura informazioni semantiche sempre più astratte riducendo la risoluzione, le skip connections preservano i dettagli locali e i bordi precisi degli oggetti che andrebbero altrimenti persi. In questa implementazione, le operazioni verranno effettuate all'interno della funzione forward.

Capitolo 4

Addestramento

4.1 Ambiente di addestramento

Come anticipato, l’addestramento della rete verrà effettuato nell’ambiente di Google Colab⁽¹¹⁾ che mette a disposizione GPU con elevata potenza di calcolo. Inizialmente sono state effettuate delle prove per utilizzare la versione gratuita che offre la possibilità di utilizzare la GPU NVIDIA Tesla T4 con 16 GB di memoria, ma sono subito nati problemi con la saturazione della RAM a causa del peso dei batch che superavano lo spazio disponibile. Si è reso necessario quindi l’upgrade al piano pro che mette a disposizione la GPU NVIDIA A100 con 40 GB di memoria, sufficienti per addestrare la rete con una dimensione dei batch di 8. In questo modo i GB occupati si aggirano intorno a 36.

4.2 Salvataggio delle epochhe

Durante l’addestramento, può capitare che si venga disconnessi dal runtime di Colab a causa di inattività o anche perché è stata chiusa la connessione. Questo rappresenta un problema poiché quando un runtime viene disconnesso vengono eliminati anche tutti i dati presenti in memoria. Per gestire questo inconveniente è stata utilizzata la funzione di PyTorch `torch.save` che permette di salvare un checkpoint alla fine di un’epoca e di ripristinarlo in seguito. Il salvataggio avviene in un file con estensione .pth sul drive che può essere ripristinato tramite la funzione `torch.load_state_dict`. Nello specifico quest’ultima restituisce un dizionario ordinato contenente tutti i parametri e buffer del modello, dove le chiavi sono i nomi dei layer e i valori sono i tensori corrispondenti. In questo modo può essere ripresa l’esecuzione dell’addestramento o possono essere effettuati ulteriori test sui risultati raggiunti.

Listing 4.1 Codice per il salvataggio delle epochhe

```

1  checkpoint = {
2      'epoch': epoch,
3      "state_dict": model.state_dict(),
4      "optimizer":optimizer.state_dict(),
5  }
6  torch.save(checkpoint, f'checkpoint_epoch_{epoch+1}.pth')
```

4.3 Algoritmo di ottimizzazione

La scelta dell'algoritmo di ottimizzazione è di cruciale importanza nell'implementazione di una rete neurale. Per il problema della segmentazione semantica le scelte più comuni sono Stochastic Gradient Descent (SGD) con momentum e Adaptive Momentum Estimation (Adam).

Adam è popolare perché utilizza momenti adattivi che permettono learning rate diversi per ogni parametro. Nella segmentazione semantica, dove sono presenti sia feature di basso livello (bordi, texture) che di alto livello (oggetti complessi) questa adattabilità è cruciale. Adam mantiene anche una memoria dei gradienti passati, utile quando alcuni pixel sono più difficili da segmentare.

SGD con momentum può raggiungere minimi più profondi perché non ha la tendenza di Adam a oscillare vicino ai minimi. Tuttavia, richiede più affinamento perché un singolo learning rate deve funzionare per tutti i parametri del modello.

In questa implementazione è stato scelto di usare l'ottimizzatore Adam per i vantaggi che offre. A differenza di SGD con momentum, Adam calcola le stime dei primi e dei secondi momenti dei gradienti per adattare il tasso di apprendimento dei pesi. I momenti del gradiente vengono utilizzati per regolare la dimensione degli aggiornamenti dei pesi, consentendo un controllo più fine del processo di apprendimento.

4.4 Funzione di loss e bilanciamento delle classi

La funzione di loss scelta è la CrossEntropyLoss, essendo particolarmente adatta per problemi di classificazione multi-classe come la segmentazione semantica. Questa funzione combina intrinsecamente la softmax activation con la negative log-likelihood loss, semplificando l'implementazione e migliorando la stabilità numerica durante l'addestramento. La CrossEntropyLoss è formulata matematicamente come la somma pesata delle log-probabilità delle classi corrette, penalizzando maggiormente le predizioni con alta confidenza ma errate. Nel contesto della segmentazione semantica, la funzione viene applicata pixel per pixel, confrontando l'etichetta predetta con quella ground truth per ogni elemento dell'immagine. Questa è la sua formula matematica:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^C y_i \cdot \log(\hat{y}_i)$$

Dove:

- C è il numero di classi nel problema di classificazione
- y_i è il valore vero per la classe i -esima: è 1 se è la classe corretta, altrimenti 0
- \hat{y}_i è la probabilità predetta dal modello che l'esempio appartenga alla classe i -esima
- $\log(\hat{y}_i)$ Il logaritmo della probabilità predetta. Penalizza fortemente le probabilità basse

Come anticipato, i pixel delle immagini satellitari sono classificati in 7 classi, di cui 2 sono state escluse perché quasi assenti nel dataset ridotto utilizzato. Le restanti 5

sono presenti in modo sbilanciato. Un altro vantaggio della CrossEntropyLoss è la possibilità di passargli il parametro weight che permette alla funzione di compensare lo sbilanciamento delle classi. Nello specifico il dataset ridotto di training è composto in questo modo:

- 7,5% per la classe Impervious surface
- 9,3% per la classe Agriculture
- 49,2% per la classe Forest & other vegetation
- 27,3% per la classe Soil
- 6,5% per la classe Water

Ciò ha portato a miglioramenti nei risultati finali senza però influire in modo sostanziale.

4.5 Metriche utilizzate

Per valutare l'apprendimento della rete inizialmente sono state usate l'Accuracy e la Mean Average Precision. L'accuracy misura semplicemente il rapporto tra il totale delle prediction corrette e il totale degli elementi esaminati:

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

Dove:

- TP = true positive
- TN = true negative
- FP = false positive
- FN = false negative

È una metrica generale e indicativa che in alcune situazioni può non essere precisa su quanto il modello stia apprendendo realmente. Per questo verrà usata la Mean Average Precision come metrica principale.

$$mAP = \frac{1}{C} \sum_{c=1}^C AP_c$$

La Mean Average Precision (mAP) è una metrica ampiamente utilizzata per valutare le prestazioni di modelli di classificazione e segmentazione, in particolare nel contesto della computer vision. Essa rappresenta la media delle precision calcolate a diversi livelli di recall per ciascuna classe, offrendo una misura complessiva dell'accuratezza del modello nel distinguere correttamente le categorie. La recall è definita come:

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

Nel caso della segmentazione semantica, la mAP viene calcolata confrontando le maschere di segmentazione predette con quelle reali (ground truth), valutando quanto bene il modello riesce a identificare correttamente i pixel appartenenti a ciascuna classe. Per ogni classe viene calcolata la Average Precision, che è l'area sotto la curva Precision-Recall. La mAP è poi la media delle AP su tutte le classi.

4.6 Tecniche per migliorare l'apprendimento

Per migliorare l'apprendimento della rete neurale vengono adottate diverse strategie. Come prima cosa vengono usate delle tecniche di augmentation per aumentare la varietà del dataset e ridurre l'overfitting. La data augmentation consiste nell'applicare delle trasformazioni alle immagini del dataset; in questo caso, vengono ruotate e specchiate verticalmente ed orizzontalmente in modo casuale.

Inoltre, come già spiegato nel paragrafo 4.4, per migliorare le prestazioni della rete viene calcolato quanto sono presenti le varie classi nel dataset per poi passare questa informazione alla funzione di loss CrossEntropyLoss. Per calcolare i pesi viene usata la funzione `calculate_class_weights`:

Listing 4.2 Funzione per il calcolo dei pesi

```

1 def calculate_class_weights(loader):
2     pixel_counts = torch.zeros(5, dtype=torch.float32)
3
4     for _, (_, targets) in enumerate(tqdm(loader)):
5         targets = targets.view(-1)
6
7         mask = targets >= 0
8         valid_targets = targets[mask]
9
10        for class_idx in range(5):
11            pixel_counts[class_idx] += (valid_targets == class_idx).sum().item()
12
13        total_pixels = pixel_counts.sum()
14        frequencies = pixel_counts / total_pixels
15        print(f"frequenze delle classi: {frequencies}")
16
17        weights = 1.0 / torch.sqrt(frequencies + 1e-8)
18
19        weights = weights / weights.sum() * len(weights)
20        return weights.to(DEVICE)

```

Quando si calcolano i pesi finali è possibile scegliere tra tre metodi di calcolo:

- $weights = \frac{1}{(frequencies+10^{-8})}$
- $weights = \frac{1}{\sqrt{frequencies+10^{-8}}}$
- $weights = \log\left(\frac{1}{frequencies+10^{-8}}\right)$

Dopo vari tentativi i risultati migliori sono stati ottenuti con il calcolo della radice, che ha permesso di correggere lo sbilanciamento senza creare pesi estremi che potrebbero destabilizzare l'addestramento.

L'ultimo passaggio effettuato per migliorare le prestazioni della rete è l'ottimizzazione dei parametri di learning rate ed epoche. Inizialmente il valore del learning rate era di 10^{-3} che si è rivelato troppo grande, in quanto le metriche della rete risultavano altalenanti durante le varie epoche. Dopo vari tentativi, il risultato migliore è stato ottenuto con un learning rate di 10^{-5} , un valore conservativo che permette alla rete di oscillare meno e migliorare in modo costante l'apprendimento. Per quanto riguarda il numero di epoche, dopo circa 30 sono stati raggiunti i risultati migliori. Nel capitolo successivo verranno approfonditi questi dati.

Capitolo 5

Risultati

In questo capitolo, come anticipato, vengono mostrati i risultati ottenuti con l’addestramento descritto nella sezione precedente. Verranno presentati risultati grafici e numerici su quanto ottenuto.

Riassumendo, i risultati migliori sono stati ottenuti con questi parametri:

- Numero di epoche: 30
- Dimensione del batch: 8
- Ottimizzatore: Adam
- Funzione di loss: CrossEntropyLoss
- Learning rate: 10^{-5}

5.1 Risultati visivi

Per visualizzare a schermo i progressi della rete durante l’addestramento, è stata implementata la funzione `predict_and_visualize(model, image_index)` che prende un’immagine dal dataset e predice la maschera. La predizione viene convertita in classi tramite argmax e poi trasformata in formato RGB per la visualizzazione. Questa funzione si rivela particolarmente utile per monitorare l’evoluzione delle capacità predittive del modello in tempo reale. Il processo di conversione da predizione probabilistica a maschera colorata viene gestito attraverso una mappa di colori predefinita che associa ogni categoria a un colore specifico, rendendo immediatamente riconoscibili le diverse categorie di oggetti. Infine stampa a schermo tre immagini affiancate: l’immagine originale, la maschera predetta dal modello e la maschera ground truth, permettendo di confrontare visivamente le performance del modello e di valutare la qualità della segmentazione in modo immediato.

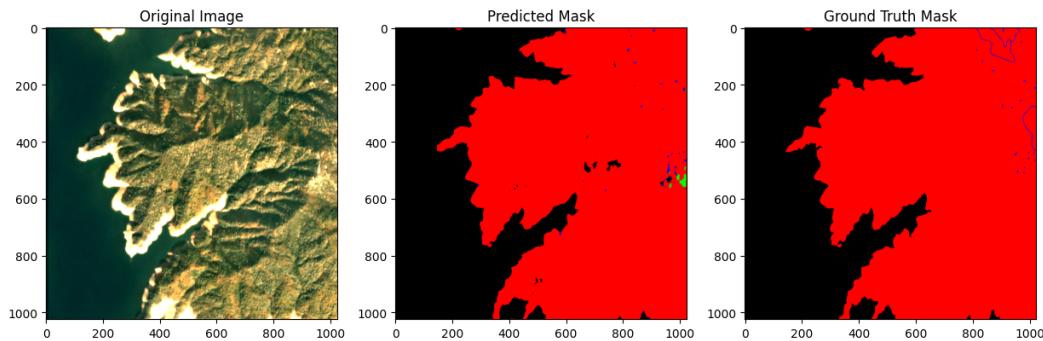


Figura 5.1 Immagine 1 con la relativa maschera predetta e ground truth

In quest’immagine, abbastanza semplice, sono presenti solo due categorie di suolo: *Water* e *Forest&other vegetation*. Possiamo notare che il modello ha classificato correttamente la maggior parte dei pixel e, visivamente, il risultato è buono. Qualche pixel (in questo caso di colore verde) viene classificato in modo errato.

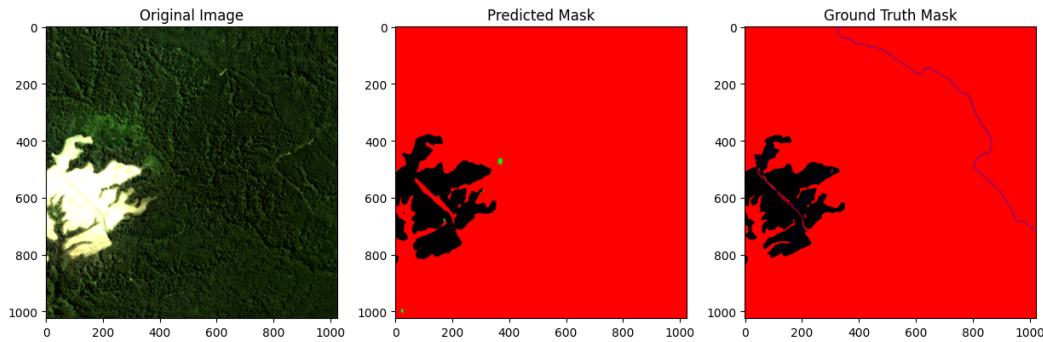


Figura 5.2 Immagine 2 con la relativa maschera predetta e ground truth

In questo caso la rete fa un buon lavoro, probabilmente grazie alla chiara differenza tra le diverse aree e ai forti contrasti di colore presenti nell’immagine. I confini ben definiti tra le zone aiutano il modello a riconoscere correttamente le varie categorie. Tuttavia, la rete non riesce a riconoscere la strada nella categoria *Impervious surfaces*, ma bisogna dire che anche per una persona sarebbe difficile individuarla a prima vista senza sapere che c’è. La strada si mescola con il paesaggio circostante e non è facile distinguere visivamente. In questa immagine sono presenti solo due categorie principali di suolo.

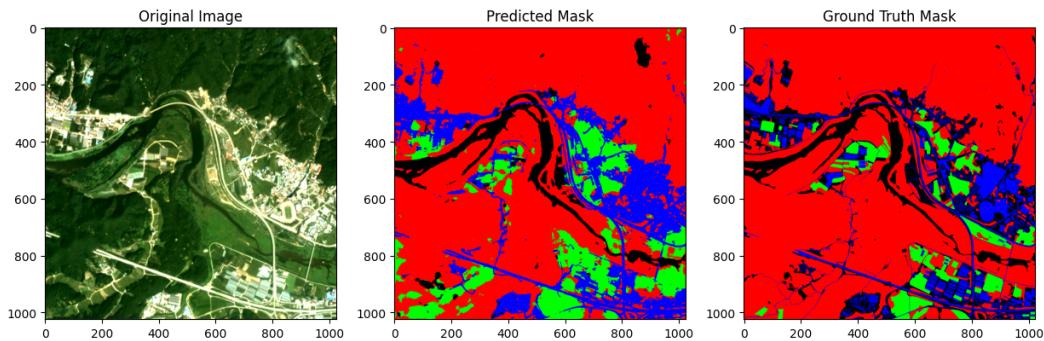


Figura 5.3 Immagine 3 con la relativa maschera predetta e ground truth

Passando ad un’immagine più complessa, possiamo notare come la rete perda precisione nello scontornare i confini delle aree. In questo caso i pixel comprendono più di due categorie e nonostante i difetti di scontornatura il risultato è complessivamente buono. Dove fatica la rete è nel delimitare con precisione i confini tra aree simili e dove gli elementi sono molto piccoli.

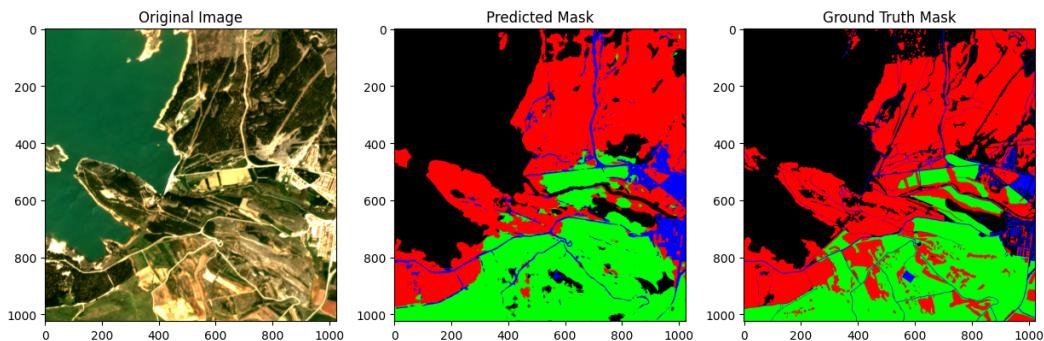


Figura 5.4 Immagine 4 con la relativa maschera predetta e ground truth

Anche qui i risultati sono simili al caso precedente, con un buon risultato finale e imprecisioni nelle aree più piccole e nei confini. Di seguito altri esempi visivi:

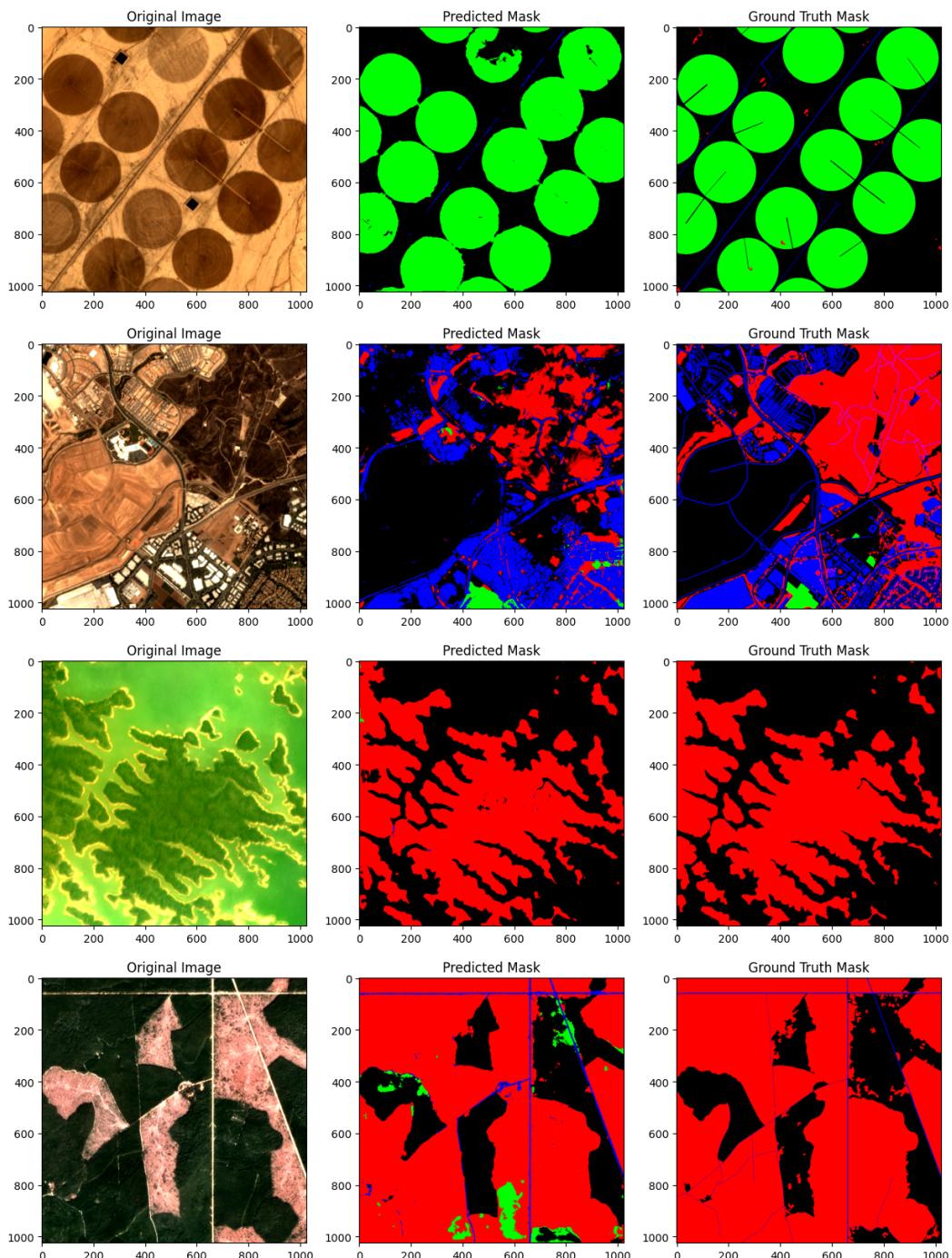


Figura 5.5 Esempi di immagini con la relativa maschera predetta e ground truth

5.2 Metriche addestramento

Durante l'addestramento viene calcolata la Mean Average Precision ad ogni epoca sul validation set per monitorare l'apprendimento della rete, poi ogni 5 epoche la stessa metrica sul set di training. Questo perché il calcolo sul set di training richiede molto più tempo essendo molto più grande. È importante confrontare le metriche sui due set per verificare che il modello non sia in overfitting o underfitting. Inoltre, viene tenuta traccia della loss di ogni epoca tracciandone i miglioramenti.

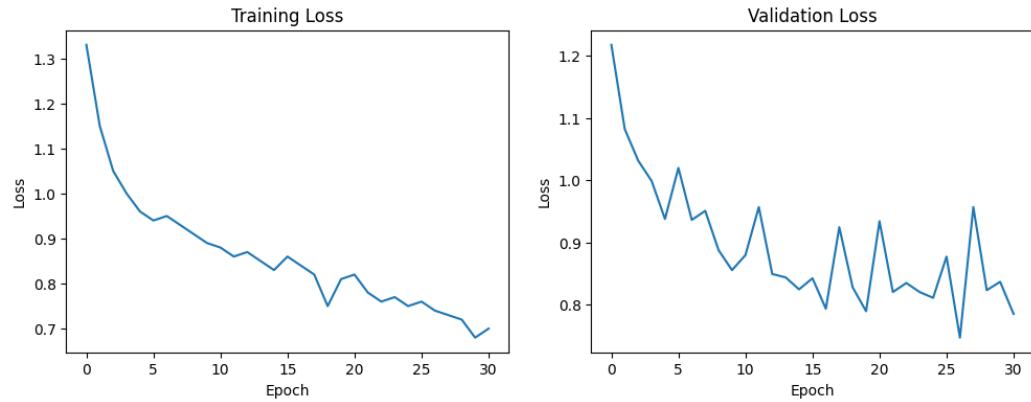


Figura 5.6 Risultati dell'addestramento

Le curve di loss evidenziano una convergenza efficace, con la loss di training che decresce costantemente da 1.35 a circa 0.7, mentre la validation loss, seppur più oscillante, segue un trend decrescente generale attestandosi intorno a 0.8. L'assenza di un divario eccessivo tra training e validation loss suggerisce che il modello non presenta significativi fenomeni di overfitting.

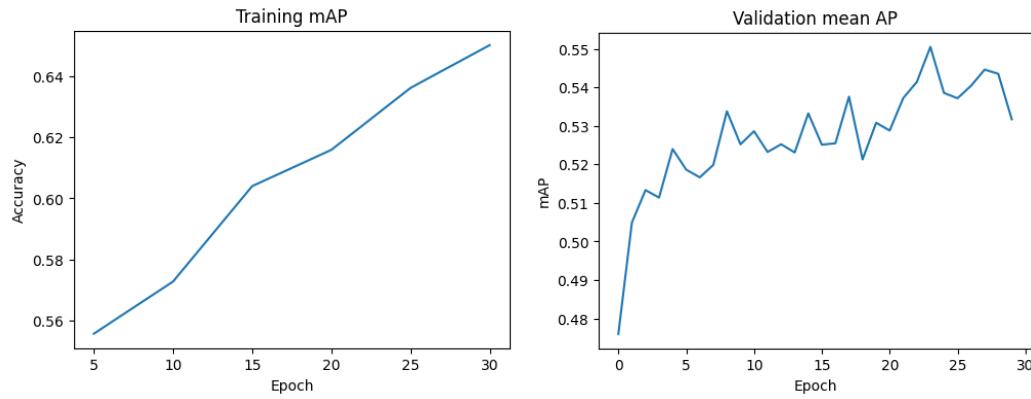


Figura 5.7 Risultati dell'addestramento

Le metriche di mAP confermano l'efficacia dell'addestramento: il training mAP raggiunge progressivamente 0.64, mentre il validation mAP si stabilizza intorno a 0.54 nelle ultime epoche. La differenza contenuta tra questi valori indica che il modello ha raggiunto una buona capacità di generalizzazione sui dati di validazione,

confermando la solidità dell'approccio adottato.

Come anticipato nel capitolo 4, con learning rate maggiore di 10^{-5} la rete non apprende in modo ottimale e ha dei risultati altalenanti. Di seguito i grafici delle metriche durante l'addestramento con un valore di 10^{-4} :

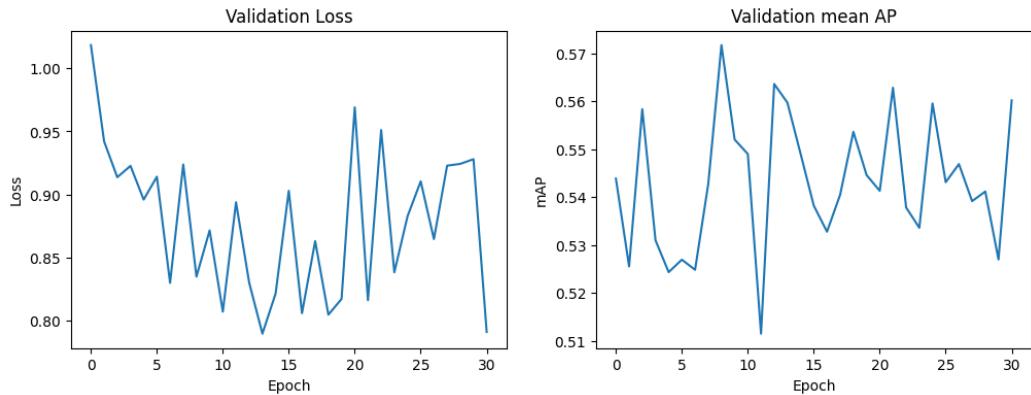


Figura 5.8 Risultati dell'addestramento con learning rate di 10^{-4}

Le curve della loss e della mAP evidenziano un andamento impreciso e incostante. La Validation loss passa da dei minimi di 0.8 circa a poco sotto 1. Anche il valore della validation mAP non ha una vera e propria tendenza a diminuire o aumentare, ma oscilla tra 0.57 e 0.51.

Capitolo 6

Conclusioni

Con i risultati ottenuti è stata confermata l'efficacia della rete neurale U-Net nel risolvere il problema della segmentazione semantica. L'implementazione sviluppata da zero ha permesso di ottenere buoni risultati nella classificazione dei pixel delle immagini nelle diverse categorie di copertura del suolo, confermando la validità dell'approccio scelto.

La segmentazione semantica delle immagini satellitari rappresenta uno strumento fondamentale per l'osservazione e il monitoraggio automatico della superficie terrestre. Nonostante la classificazione dei pixel non sia perfetta, la rete si è dimostrata adatta a processare automaticamente grandi volumi di dati satellitari, fornendo mappe dettagliate e aggiornate in tempi ridotti rispetto ai metodi tradizionali. Le applicazioni potrebbero essere molteplici e strategiche: dal monitoraggio dei cambiamenti ambientali e della deforestazione, alla pianificazione urbana e alla gestione delle risorse agricole.

In conclusione, la rete ha ottenuto buoni risultati, pur presentando margini di miglioramento significativi raggiungibili attraverso l'implementazione di tecniche più avanzate, sessioni di addestramento più intensive che sfruttino maggiori risorse computazionali, l'utilizzo del dataset originale nella sua completezza e una rete più grande.

Bibliografia

- [1] KDnuggets. Misconceptions in semantic segmentation annotation. <https://www.kdnuggets.com/2022/01/misconceptions-semantic-segmentation-annotation.html>, 2022.
- [2] Webmaster, elementi di machine learning: gli algoritmi di base. <https://apolis.it/2019/01/machine-learning-gli-algoritmi-di-base-guida/>.
- [3] Repository github del progetto. <https://github.com/emi4458/unet-satellite-segmentation/tree/main>.
- [4] Mark Weber Marvin Eisenberger Andrés Camero Jingliang Hu Ariadna Pregel Hoderlein Çağlar Şenaras Timothy Davis Daniel Cremers Giovanni Marchisio Xiao Xiang Zhu Laura Leal-Taixé Aysim Toker, Lukas Kondmann. Dynamicearthnet: Daily multi-spectral satellite dataset for semantic change segmentation: <https://arxiv.org/abs/2203.12560>.
- [5] European space agency, sentinel-2 brings land into focus https://www.esa.int/ESA_Multimedia/Images/2014/07/Sentinel-2_brings_land_into_focus, July 2014.
- [6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. <https://arxiv.org/abs/1505.04597>, 2015.
- [7] Documentazione pytorch. <https://docs.pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html>.
- [8] Documentazione pytorch. <https://docs.pytorch.org/docs/stable/generated/torch.nn.ReLU.html>.
- [9] Documentazione pytorch. <https://docs.pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html>.
- [10] Documentazione pytorch. <https://docs.pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html>.
- [11] Google colab. <https://colab.google/>.