Segmentazione di immagini aeree basata sull'architettura U-Net per la segmentazione multiclasse.

SEGURA, Elka Matricola:241327
BESSIOUD, Ghofran Matricola: 247562

August 30, 2024

1 Introduction

La crescente disponibilità di grandi volumi di immagini satellitari provenienti da varie missioni e modalità costituisce attualmente un catalizzatore per la realizzazione di studi su larga scala geografica e temporale. In questo senso, la segmentazione delle immagini è un processo fondamentale nel campo della visione artificiale con la disponibilità di tante immagini, la cui è responsabile della divisione di un'immagine in diverse regioni o segmenti con caratteristiche simili. Questi segmenti possono rappresentare oggetti, trame o qualsiasi altra parte significativa dell'immagine. Eseguendo questa divisione, si cerca di comprendere meglio la struttura e il contenuto dell'immagine, il che facilita compiti come il riconoscimento di modelli, il rilevamento di oggetti e il tracciamento del movimento. In questo studio, diversi modelli di reti neurali convoluzionali vengono valutati e confrontati per questo compito specifico. L'accuratezza e la generalizzabilità di questi modelli vengono analizzate in dettaglio per identificare punti di forza e aree di miglioramento. Abbiamo sviluppato 3 modelli, Unet-pretrained, Unet-pretrained1 e Resnet50-Unet. Questi modelli presentavano una precisione rispettivamente del 78,71%, 81,03% e 84,57%.

1.1 Problem

L'eterogeneità dei soggetti presenti come edifici, strade, alberi e automobili nei dati ad altissima risoluzione porta ad un'elevata variazione delle classi da identificare in un'immagine. L'attenzione si concentra sulla segmentazione semantica 2D dettagliata che assegna etichette a più categorie di oggetti.

A tal fine se forniscono due set di dati di immagini aeree di altissima risoluzione. Entrambe le aree coprono scene urbane. Mentre Vaihingen è una città relativamente piccola con molti edifici indipendenti e piccoli edifici a più piani, Potsdam mostra una tipica città storica con grandi blocchi di edifici, strade strette e una fitta struttura insediativa.

1.2 Objective

L'obiettivo di questo lavoro è l'estrazione automatizzata di oggetti urbani da dati acquisiti da sensori aerei sulla base del modello UNET. Anche, verrà effettuata una ricerca per modelli implementati simili alla ricerca, che saranno discussi in questo lavoro. In questo caso, non è disponibile un codice originale per l'analisi, quindi verrà implementato un codice per risolvere il problema. Utilizzeremo un algoritmo di segmentazione semantica basato sul deep learning che ci permette

di categorizzare ed etichettare ogni pixel all'interno di un'immagine in modo da distinguere le diverse categorie. In questo modo possiamo classificare il contenuto di un'immagine

2 Background

2.1 CNN e caratteristiche

Le reti convoluzionali CNN sono le reti di deep learning che presenta i migliori risultati in problemi di riconoscimento, classificazione e rilevamento di oggetti in immagini e video, sulla base di operazioni di convoluzione. Lo scopo delle CNN è estrarre tutte le caratteristiche da un'immagine e quindi utilizzare queste funzionalità per rilevare o classificare gli oggetti nell'immagine.

Lo strato convoluzionale (Convolutional Layer) è lo strato principale delle reti neurali convoluzionali e l'uso di uno o più strati di questo tipo in dette reti è essenziale. I parametri in uno strato convoluzionale in pratica si riferiscono ad un insieme di filtri addestrabili. Ciascun filtro occupa un piccolo spazio lungo le dimensioni di larghezza e altezza, ma si estende per l'intera profondità del volume di input a cui viene applicato. Durante la propagazione in avanti, ciascun filtro lungo la larghezza e l'altezza del volume di input produce una mappa delle caratteristiche per quel filtro. Man mano che il filtro si sposta attraverso l'area di input, viene eseguito un prodotto scalare tra i valori del filtro e quelli della regione di input a cui è applicato.

Tra strati convoluzionali successivi, è usuale posizionare al centro di essi quello che viene chiamato strato di pooling (Max-Pooling). Lo strato di pooling prende le mappe delle caratteristiche prodotte nel livello di convoluzione e le raggruppa in un'immagine. In questo strato avviene la riduzione della dimensionalità, riducendo così la complessità del modello, contribuendo a evitarne l'overfitting. Cioè, ciò che fanno i livelli di pooling è semplificare le informazioni nell'output di lo strato convoluzionale. In genarale consiste nell'eseguire una riduzione della matrice ottenuta con la convoluzione. Ad esempio, mantenendo per un gruppo di pixel solo il pixel con il valore più alto.

Le reti neurali convoluzionali (CNN) sono una forma ampiamente utilizzata di deep learning per la segmentazione delle immagini. Queste reti sono progettate specificamente per elaborare dati di immagine e sono in grado di apprendere caratteristiche e modelli visivi complessi.

Nel nostro caso, l'utilizzo della CNN per il nostro obiettivo, ovvero segmentare l'immagine in base agli oggetti presenti al suo interno, è la base per l'implementazione del modello.

2.2 Segmentazione

Allo scopo del nostro lavoro la segmentazione delle immagini è un processo fondamentale nel campo della visione artificiale, che è responsabile della divisione di un'immagine in diverse regioni o segmenti con caratteristiche simili. Questi segmenti possono rappresentare oggetti, trame o qualsiasi altra parte significativa dell'immagine. Eseguendo questa divisione, cerchiamo di comprendere meglio la struttura e il contenuto dell'immagine, il che facilita compiti come il riconoscimento di modelli, il rilevamento di oggetti e il tracciamento del movimento.

Esistono diversi concetti di base che è importante comprendere in relazione alla segmentazione delle immagini. Uno di questi è la soglia, che consiste nel dividere l'immagine in regioni in base a una soglia predefinita. Ad esempio, se vogliamo segmentare un'immagine in bianco e nero, possiamo impostare una soglia per separare i pixel scuri da quelli chiari. Un altro concetto è il rilevamento dei bordi, che prevede l'identificazione dei punti dell'immagine in cui si verificano improvvisi cambiamenti di intensità o colore. Rilevando questi bordi, possiamo separare gli oggetti e definire confini chiari tra le diverse parti dell'immagine.

Nel campo del riconoscimento delle immagini, la segmentazione delle immagini permette di separare e distinguere varie parti di un'immagine, facilitandone l'identificazione e la classificazione. È particolarmente utile nelle applicazioni di sicurezza, dove è necessario raccogliere e tracciare persone o oggetti di interesse in tempo reale.

2.3 Segmentazione semantiche

La segmentazione semantica è una tecnica avanzata nella visione artificiale che viene utilizzata per assegnare un'etichetta semantica a ciascun pixel in un'immagine. A differenza di altri metodi di segmentazione, come la segmentazione basata sui bordi o la segmentazione basata su regioni, la segmentazione semantica si concentra sulla comprensione del contenuto visivo di un'immagine a un livello superiore, identificando gli oggetti e le classi di oggetti presenti.

Un'architettura comunemente utilizzata per la segmentazione semantica con la CNN è la rete neurale convoluzionale profondamente

connessa (FCN). Questa architettura utilizza livelli convoluzionali per estrarre le caratteristiche dell'immagine e livelli di convoluzione trasposti per generare mappe di segmentazione ad alta risoluzione.

L'addestramento di una rete neurale convoluzionale per la segmentazione delle immagini richiede un set di addestramento etichettato, in cui ogni pixel nell'immagine è etichettato con la classe corrispondente. Durante l'addestramento, la rete regola i pesi delle sue connessioni per ridurre al minimo la differenza tra la segmentazione prevista e la segmentazione effettiva. Le reti neurali convoluzionali hanno dimostrato prestazioni eccezionali nella segmentazione semantica e in molti casi hanno sovraperformato gli approcci basati su funzionalità e quelli basati su modelli. Ciò è dovuto alla sua capacità di apprendere automaticamente caratteristiche e modelli rilevanti e alla sua capacità di gestire immagini ad alta dimensione.

L'architettura delle reti neurali per la segmentazione delle immagini ha un encoder-decoder. Il endoder è la parte iniziale del modello che prende un input, ad esempio un'immagine completa, e lo trasforma in una rappresentazione più compatta o codificata, preservando le caratteristiche più importanti dell'input. Questa rappresentazione codificata può essere vista come una versione compressa dell'input originale. Nel contesto della segmentazione dell'immagine, il encoder è responsabile dell'elaborazione dell'immagine di input e dell'estrazione di caratteristiche rilevanti, come bordi, forme e trame. Queste caratteristiche sono fondamentali per comprendere la struttura e il contenuto dell'immagine, il che facilita il compito di segmentazione, che consiste nel dividere l'immagine in diverse regioni o segmenti in base a determinate caratteristiche. Dopo che il encoder ha trasformato l'immagine in ingresso in una rappresentazione codificata, questa rappresentazione viene passata al decoder.

Il decoder prende le mape di caratteristiche di alto livello prodotte dal encoder e le utilizza per generare una versione segmentata o etichettata dell'immagine originale. Converte la rappresentazione delle caratteristiche di alto livello (che potrebbe essere a bassa risoluzione a causa del sottocampionamento da parte del encoder) in una rappresentazione a risoluzione più elevata, corrispondente alla risoluzione dell'immagine di input. Aggiungi dettagli spaziali alla segmentazione finale combinando informazioni provenienti da diverse scale e livelli di astrazione.

3 Related Work

In questo contesto di cui parliamo esistono oggi molti modelli che rispondono con buoni risultati alla segmentazione dell'immagine e al riconoscimento degli oggetti all'interno dell'immagine. In questa sezione parleremo di lavori precedenti che affrontano l'argomento.

Abbiamo la prova che esistono attualmente molti studi che confrontano le reti neuronali preallenate. Facciamo riferimento all'articolo "A Comparative Analysis of Image Segmentation Using Classical and Deep Learning Approach" in merito al obiettivo di valutazione dei metodi di apprendimento profondo nelle aree di segmentazione delle immagini RGB. Gli autori hanno proposto due ipotesi: (1) La qualità della segmentazione utilizzando metodi di apprendimento profondo è soprattutto l'utilizzo di metodi classici per immagini RGB, e (2) L'aumento della risoluzione dell'immagine RGB ha un impatto positivo su la qualità della segmentazione. Confronta gli algoritmi di segmentazione tradizionali (Umbral e K-media) con l'obiettivo di apprendimento profondo (U-Net, SegNet e FCN 8) per verificare la qualità della segmentazione RGB. Due risoluzioni Vengono visualizzati i tipi di immagine: 160x240 e 320x480 pixel. La qualità della segmentazione per ciascun algoritmo è stimata in funzione di quattro parametri: : Accuracy, Precision, Recall and Sorensen-Dice ratio (Dice score). Nello studio è stato applicato il insieme di dati Carvana, che contiene 5.088 immagini di automobili ad alta risoluzione. El conjunto inicial è stato diviso in sottocongiunti di addestramento, convalida e prova come 60%, 20%, 20%, rispettivamente. Come risultato, si è ottenuta per U-Net la migliore Accuracy, Dice score e Recall con risoluzione 160x240, raggiungendo un 99,37%. 98,56% e 98,93%, rispettivamente. Alla stessa risoluzione, la accuratezza più alta del 98,19% è ottenuta per FCN-8 architettura. Per una risoluzione più alta, 320x480, si ottengono i migliori media di Accuracy, Dice score e Recall. para la red FCN-8, alcanzando el 99,55%, 99,95% e 98,85%, rispettivamente. I risultati più alti dei metodi classici sono stati ottenuti dall'algoritmo umbral, ottenendo l'80,41% di Accuracy, il 58,49% di punteggiatura dei dati, il 67,32% di Dice score e 52,62% Recall. I risultati confermano questa ipotesi. In conclusione hanno affermato che gli approcci di deep learning proposti rappresentano un'opzione più appropriata per la segmentazione delle immagini rispetto agli algoritmi classici.

Secondo gli autori di "Satellite Image Segmentation Using Neural Networks: A Comprehensive Review", l'uso delle reti neurali per la segmentazione delle immagini satellitari ha guadagnato un'attenzione significativa grazie alla loro capacità di apprendimento caratteristiche

e modelli complessi direttamente dai dati. Varie architetture, come le Reti Neurali Convoluzionali (CNN), U-Net, Fully Convolutional Networks (FCN) e DeepLab sono stati proposti e hanno mostrato risultati promettenti nella segmentazione delle immagini satellitari. Queste reti sfruttano la struttura gerarchica dei dati e utilizzano la convoluzione livelli per estrarre caratteristiche su scale diverse, consentendo una segmentazione accurata e dettagliata.

Le tecniche di preelaborazione svolgono un ruolo fondamentale nel migliorare la qualità dei dati di input per le reti neurali. Aumento dei dati metodi, tecniche di miglioramento delle immagini e algoritmi di estrazione delle caratteristiche possono aiutare a ridurre l'overfitting, migliorare le caratteristiche rilevanti e migliorare la robustezza dei modelli di segmentazione. Inoltre, appropriate tecniche di normalizzazione e standardizzazione garantiscono che i dati di input siano adeguatamente dimensionati per prestazioni di rete ottimali. Le metriche di valutazione sono essenziali per quantificare le prestazioni degli algoritmi di segmentazione. Union (IoU), Dice Coefficient, pixel accuracy, e F1-score sono parametri comunemente utilizzati per valutare l'accuratezza e la coerenza dei risultati della segmentazione. Queste metriche consentono ai ricercatori di confrontare diversi algoritmi e selezionare quello più adatto per applicazioni specifiche.

Di qualche modo anche come applicazione gli autori dicono che la segmentazione delle immagini satellitari aiuta nella classificazione della copertura del territorio urbano, identificando gli edifici. aree, strade e infrastrutture urbane.

Un altro utilizzo di quanto spiegato di seguito lo troviamo nell'articolo "Segmentazione semantica di immagini aeree con un ensemble di CNNS". Questo articolo descrive un approccio di deep learning alla segmentazione semantica di immagini (aeree) ad altissima risoluzione. La segmentazione semantica nelle aree urbane pone l'ulteriore sfida che molte categorie di oggetti creati dall'uomo sono composte da un gran numero di materiali diversi e che gli oggetti in le città (come edifici o alberi) sono piccole e interagiscono tra loro altro attraverso occlusioni, ombre proiettate, interriflessi, ecc.

Gli autori presentano un metodo di segmentazione semantica che offre una resa di segmentazione semantica di ultima generazione per le immagini aeree del insieme di dati dell'etichetta semantica ISPRS. L'FCN, come altri metodi di apprendimento approfondito, include l'estrazione di caratteristiche come parte della capacità, il che significa che puoi leggere dati di immagini senza elaborarli e liberarti dell'utente del design delle funzioni mediante test ed errori. FCN

e CNN in generale ora sono una tecnologia matura che nessun esperto può utilizzare immediatamente. Nel processo della lingua e nella visione del computer in generale, è stato convertito nel metodo standard per una varietà di tabelle di previsione.

4 Method

Come accennato in precedenza per la segmentazione, come metodo generale da applicare, l'architettura di rete neurale utilizzata è una rete di codificatori seguita da una rete di decodificatori. Il codificatore avrà il compito di estrarre le caratteristiche dell'immagine, mentre il compito del decoder sarà quello di recuperare i dettagli degli oggetti proiettando le caratteristiche estratte nello spazio dei pixel, per ottenerne una classificazione. Per fare questo, solitamente si parte da architetture già realizzate, di cui parleremo più avanti, e si sostituiscono i layer completamente connessi (quelli che effettuano la classificazione) con layer che effettuano l'interpolazione o l'upsampling con l'obiettivo di generare un output della stessa dimensione. rispetto all'ingresso. Questo tipo di reti sono quindi completamente convoluzionali (Fully Convolutional Networks, FCN).

4.1 Description

La maggior parte delle architetture proposte per la segmentazione si basano sulle già note classificazioni CNN, che hanno dato contributi significativi al campo della visione artificiale. Attualmente esistono reti neurali in grado di risolvere il problema della segmentazione delle immagini con buoni risultati.

4.2 Models

U-Net è un'architettura di rete neurale convoluzionale progettata specificamente per attività di segmentazione di immagini biomediche a livello di pixel. Creata nel 2015 da Olaf Ronneberger, Philipp Fischer e Thomas Brox, U-Net ha rivoluzionato non solo il campo biomedico ma si è dimostrato efficace anche in altri settori che richiedono un'accurata segmentazione delle immagini. La sua struttura di codifica-decodifica simmetrica facilita la localizzazione e la classificazione di caratteristiche su scale diverse ed è nota per la sua efficienza nell' apprendimento da una quantità limitata di dati.

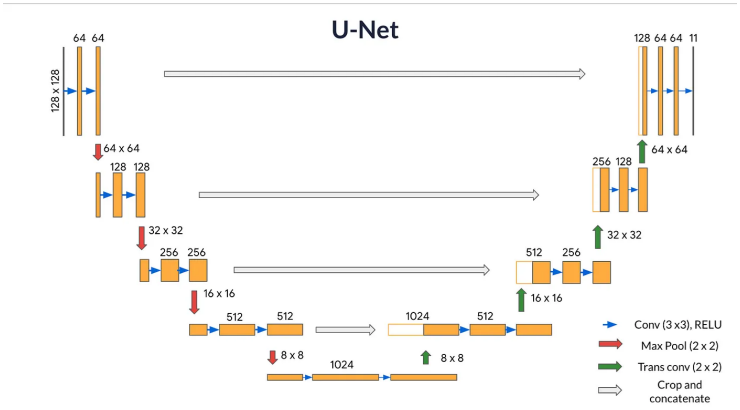

Figure 1: Architettura Modello U-NET

L'architettura della rete U-Net (riferimento) Consiste in un codificatore e un decodificatore. L'encoder segue l'architettura tipica di una rete convoluzionale. Consiste nell'applicazione ripetuta di due convoluzioni 3x3 (unpaded convolutions), ciascuna seguita da un'unità lineare rettificata (ReLU) e un'operazione di max pooling 2x2 con stride 2 per downsampling. In ogni fase di downsampling raddoppiamo il numero di canali caratteristici. Ogni passaggio nel decoder consiste in un upsampling della mappa delle caratteristiche seguito da una convoluzione 2x2 ("up-convolution") che dimezza il numero di canali delle caratteristiche, una concatenazione con la mappa delle caratteristiche corrispondentemente eliminata dal percorso di contrazione e due 3x3. convoluzioni, ciascuna seguita da una ReLU. Il ritaglio è necessario a causa della perdita di pixel sui bordi ciascuna convoluzione. Nello strato finale, viene utilizzata una convoluzione 1x1 per mappare ciascun vettore di caratteristiche a 64 componenti sul numero desiderato di classi. In totale la rete ha 23 strati convoluzionali.

Nell'aprile 2023, il dipartimento di ricerca di Meta ha presentato un nuovo modello di intelligenza artificiale (AI) che hanno chiamato SAM (Segment Anything Model). Con SAM si puo segmentare un'immagine in tre modi: Selezionando un punto dell'immagine verrà cercato e distinto l'oggetto che si interseca con quel punto e verranno ricercati tutti gli oggetti identici presenti nell'immagine. Delimitando la finestra o il riquadro di delimitazione, viene disegnato un rettangolo sull'immagine e vengono identificati tutti gli oggetti presenti in quell'area. Per le parole, una parola viene digitata tramite una console e SAM può identificare gli oggetti che corrispondono a quella parola o all'ordine esplicito in immagini o video, anche se tali dati non sono stati inclusi nel suo addestramento.

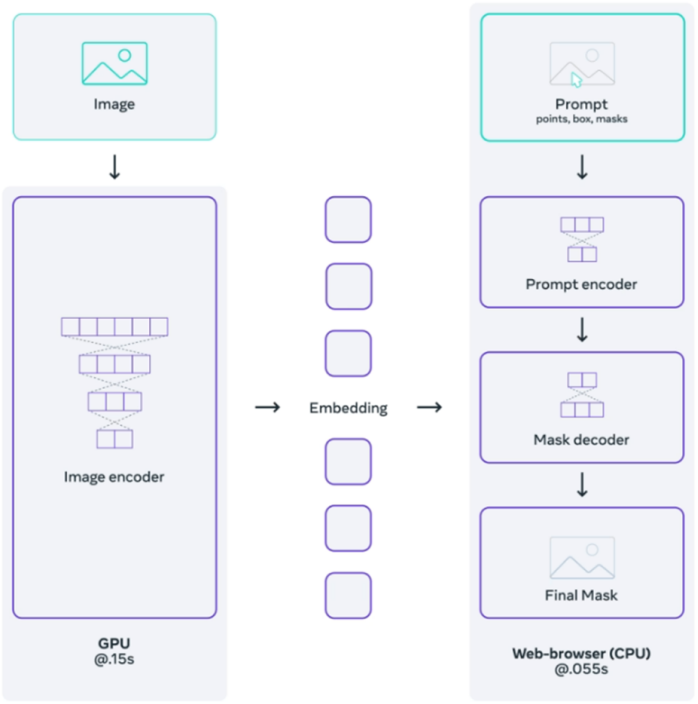

Figure 2: Architettura Modello SAM

SAM è stato progettato per essere sufficientemente efficiente per alimentare il motore dei dati. Il modello è composto da 1) un codificatore di immagini da una sola volta e 2) un decodificatore di maschera liviana che può essere eseguito su un navigatore web solo pochi milioni di messaggi.

Il modello trae la sua forza da un enorme insieme di dati che lo addestrano. 11 milioni di immagini e 1,1 miliardi di maschere, che è considerato il più grande set di dati di segmentazione fino ad oggi. In questo database è coperta un'ampia gamma di oggetti e categorie, come animali, piante, veicoli, mobili, cibo e altro ancora.

4.3 Formal definition

Il problema della segmentazione dell'immagine satellitare in questo caso comporta la partizione di un'immagine satellitare in regioni distinte corrispondenti a diverse categorie come sfondo, sfondo disordinato, edifici, vegetazione bassa, alberi e automobili. A ciascuna regione dell'immagine verrà assegnata un'etichetta da un set predefinito e l'output sarà un'immagine maschera in cui il valore di ciascun pixel

rappresenta la categoria della regione corrispondente nell'immagine satellitare. Categorie ed etichette corrispondenti:

1. Background: (0,0,0) Nero
2. Roads: (255,255,255) Bianco
3. Building: (255,0,0) Rosso
4. Low vegetation: (255,255,0) Giallo
5. Tree: (0,255,0) Verde
6. Car: (0,255,255) Blue Cian

5 Experimental evaluation

5.1 Datasets

Questo lavoro sarà svolto sulla base di una precedente ricerca che riguarda la segmentazione delle immagini aeree, reperibile al seguente link https://www.kaggle.com/datasets/aletbm/urban-segmentation - isprs/data

Il set di dati contiene 72 immagini ad alta risoluzione con misure di 6000 x 6000 in formato RGB. Il set di dati proviene da un concorso ISPRS lanciato nel 2014 "ISPRS Test Project on Urban Classification and 3D Building Reconstructionen" che presentava due problemi da risolvere, la ricostruzione 3D e l'altro è la costruzione di un modello per la segmentazione dell'immagine in cui gli oggetti vengono segmentati all'interno dell'immagine.

Le immagini rivelano le città di Toronto, Vaihingen e Potsdam dallo spazio aereo. Queste immagini contengono i rispettivi masks, che sarebbero la nostra Grand True (eccetto la città di Torono). Il set di dati si trova sulla piattaforma Kaggle quindi viene scaricato direttamente.

In ordine del secondo problema, affinché le immagini possano essere manipolate da un modello di segmentazione devono essere preelaborate. In questo caso ci sono solo 72 immagini, che sono considerati pochi dati per l'apprendimento di una rete neurale, quindi generalmente vengono utilizzate tecniche di data-aumentation per generare più immagini per l'apprendimento o nel caso in cui sia consentito dividere l'immagine in parches.

```python
import os
import cv2
import numpy as np

# Ruta al directorio que contiene las imágenes y máscaras
data_directory = "/content/ImagenR/Imagenes"
data_directory1 = "/content/ImagenR/Mask"

# Listar todos los archivos de imágenes y máscaras en el directorio
image_files = sorted([file for file in os.listdir(data_directory) if file.endswith('.jpg')])
mask_files = sorted([file for file in os.listdir(data_directory1) if file.endswith('.png')])

# (Este codigo se implementa para poder seleccionar tambien menores cantidades de imagenes por ejemplo 5000) Seleccionar aleatoriamente 22356 índices únicos
random_indices = np.random.choice(len(image_files), 22356, replace=False)

# Inicializar listas para almacenar imágenes y máscaras como arrays numpy
images = []
masks = []

# Cargar imágenes y máscaras en arrays numpy
for idx in random_indices:
    image_path = os.path.join(data_directory, image_files[idx])
    mask_path = os.path.join(data_directory1, mask_files[idx])

    image = cv2.imread(image_path)
    mask = cv2.imread(mask_path)

     # Redimensionar las imágenes y máscaras
#    image = cv2.resize(image, (256, 256))
#    mask = cv2.resize(mask, (256, 256))

    images.append(image)
    masks.append(mask)

# Convertir listas de imágenes y máscaras a arrays numpy
X_images = np.array(images)
y_masks = np.array(masks)

```

Come disponiamo di un'immagine ad alta risoluzione, possiamo applicare la patch all'immagine. Realizzare le patch significa aumentare la numerosità del dataset senza intaccare la qualità dell'immagine; più piccole sono le patch, più immagini risulteranno; Per questo lavoro, si è pensato di creare patch 256x256, che sarebbe la dimensione dell'input della rete neurale. Viene definita la funzione precedente che realizza le patch e poi le salva in modo organizzato in una cartella immagini e in una cartella maschere

Figure 3: X-masks[0]

Figure 4: y-masks[0]

Verifichiamo che le patch dell'immagine siano salvate in modalità organizzata e che l'immagine corrisponda alla sua maschera.

5.1.1 Features

Dopo aver realizzato le patch, vengono riprodotte 22356 immagini 256x256. Prima di dividere il nostro set di dati in set di dati di training, test e convalida. Dobbiamo trasformare le nostre maschere in modo tale che il modello riconosca le categorie di oggetti contenute nell'immagine. Vengono definite le tuple RGB corrispondenti a ciascuna classe di oggetti nelle immagini. [H]

```
import numpy as np
from sklearn.model_selection import train_test_split

# Definir colores
Background = (0, 0, 0) # Negro
Roads = (255, 255,255) #Blanco
Building = (0, 0, 255) #Azul
Low_vegetation = (0, 255, 255) #Blue Cean
Tree = (0, 255, 0) #Verde
Car = (255, 255, 0) #Amarillo
```

La funzione rgb-to-2D-label prende un'immagine di etichetta in formato RGB e la converte in un array 2D in cui ogni pixel viene sostituito da un indice di classe. Questo viene fatto controllando se i valori RGB di un pixel corrispondono a una qualsiasi delle tuple definite e assegnando l'indice corrispondente. [H]

```

def rgb_to_2D_label(label):
    """
    Reemplace los píxeles con valores RGB específicos según su
    ↪  índice de clase.
    """
    label_seg = np.zeros(label.shape[:2], dtype=np.uint8)
    label_seg[np.all(label == Background, axis=-1)] = 0
    label_seg[np.all(label == Roads, axis=-1)] = 1
    label_seg[np.all(label == Building, axis=-1)] = 2
    label_seg[np.all(label == Low_vegetation, axis=-1)] = 3
    label_seg[np.all(label == Tree, axis=-1)] = 4
    label_seg[np.all(label == Car, axis=-1)] = 5
    return label_seg

```

Per ogni maschera nell'elenco y-masks, la maschera RGB viene convertita in una matrice 2D di etichette utilizzando la funzione rgb-to-2D-label e memorizzata nell'elenco delle etichette. Quindi, l'elenco delle etichette viene convertito in un array NumPy. [H]

```

labels = []
for mask in y_masks:
    label = rgb_to_2D_label(mask)
    labels.append(label)

labels = np.array(labels)
labels = np.expand_dims(labels, axis=-1)

Unique labels in label dataset are:  [0 1 2 3 4 5]
```

5.1.2 Statistics

Train-test-split viene utilizzato per dividere il set di dati X-images ed etichette in set di training, test e convalida. Innanzitutto, il set di dati è suddiviso in formazione (80%) e test (20%). Il training set viene quindi preso e suddiviso nuovamente per ottenere un validation set (10%) del training set originale. Il random-state garantisce che la divisione sia riproducibile. [H]

```
#Divisione del dataset
X_train, X_test, y_train, y_test =
↪  train_test_split(X_images[:10000], labels[:10000],
↪  test_size=0.20, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train,
↪  y_train, test_size=0.1, random_state=42)

#Divisione extra
X_trainA, X_testA, y_trainA, y_testA =
↪  train_test_split(X_images[-12356:], labels[-12356:],
↪  test_size=0.20, random_state=42)
X_trainA, X_valA, y_trainA, y_valA = train_test_split(X_train,
↪  y_train, test_size=0.1, random_state=42)

print("Dimesioni del train, test and validation:",X_train.shape,
↪  X_val.shape, X_test.shape )
print("Dimesioni del train, test and
↪  validation:",X_trainA.shape, X_valA.shape, X_testA.shape )
Dimesioni del train, test and validation: (7200, 256, 256, 3)
↪  (800, 256, 256, 3) (2000, 256, 256, 3)
Dimesioni del train, test and validation: (6480, 256, 256, 3)
↪  (720, 256, 256, 3) (2472, 256, 256, 3)
```

In questo caso faremo due set, che a loro volta saranno divisi train-val-test. Il primo set conterrà le prime 10.000 immagini e il secondo le restanti 12.356 immagini. Il primo set verrà utilizzato per addestrare il modello e il secondo per fine-tuning, congelare livelli o blocchi specifici del modello.

5.2 Experimental setting

Come vediamo in questo caso, viene definita un'architettura basata sull'architettura U-Net. Innanzitutto, vengono importate le librerie necessarie per creare una funzione che definisce il modello. Librerie Os e cv2 per la manipolazione di file e l'elaborazione di immagini, numpy per operazioni numeriche e tensorflow insieme a keras per costruire e addestrare la rete neurale.

La funzione implementata crea e restituisce un modello basato nell' architettura U-Net per la segmentazione delle immagini multiclasse. Prende come input la forma dell'immagine (input-shape) e il numero di classi da segmentare (num-classes). Il livello di input viene definito

e l'immagine di input viene ridimensionata per normalizzare i valori dei pixel in un intervallo compreso tra 0 e 1.

La parte del codificatore è composta da cinque blocchi di livelli convoluzionali e livelli di pooling. Ogni blocco ha due strati convoluzionali seguiti da uno strato di pooling che riduce la dimensione spaziale delle caratteristiche. Invece la parte del decodificatore è costituita da blocchi di strati di trasposizione convoluzionale (che aumentano la dimensione spaziale) e strati convoluzionali. Ogni blocco nel decodificatore esegue anche la concatenazione con le corrispondenti funzionalità del codificatore (skip connections), consentendo alla rete di recuperare dettagli a risoluzione fine.

```
import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
↪  Dropout, concatenate, Conv2DTranspose, Rescaling
from tensorflow.keras.models import Model

def unet_multiclass(input_shape, num_classes):
    inputs = Input(input_shape)
    scaled_inputs = Rescaling(1./255)(inputs)

    # Encoder
    conv1 = Conv2D(32, 3, activation='relu',
    ↪  padding='same')(scaled_inputs)
    conv1 = Conv2D(32, 3, activation='relu',
    ↪  padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(64, 3, activation='relu',
    ↪  padding='same')(pool1)
    conv2 = Conv2D(64, 3, activation='relu',
    ↪  padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(128, 3, activation='relu',
    ↪  padding='same')(pool2)
    conv3 = Conv2D(128, 3, activation='relu',
    ↪  padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(256, 3, activation='relu',
    ↪  padding='same')(pool3)
    conv4 = Conv2D(256, 3, activation='relu',
    ↪  padding='same')(conv4)
    drop4 = Dropout(0.5)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

    conv5 = Conv2D(512, 3, activation='relu',
    ↪  padding='same')(pool4)
    conv5 = Conv2D(512, 3, activation='relu',
    ↪  padding='same')(conv5)
    drop5 = Dropout(0.5)(conv5)


```

```
# Decoder
    up6 = Conv2DTranspose(256, 2, strides=(2, 2),
    ↪  padding='same')(drop5)
    merge6 = concatenate([drop4, up6], axis=3)
    conv6 = Conv2D(256, 3, activation='relu',
    ↪  padding='same')(merge6)
    conv6 = Conv2D(256, 3, activation='relu',
    ↪  padding='same')(conv6)

    up7 = Conv2DTranspose(128, 2, strides=(2, 2),
    ↪  padding='same')(conv6)
    merge7 = concatenate([conv3, up7], axis=3)
    conv7 = Conv2D(128, 3, activation='relu',
    ↪  padding='same')(merge7)
    conv7 = Conv2D(128, 3, activation='relu',
    ↪  padding='same')(conv7)

    up8 = Conv2DTranspose(64, 2, strides=(2, 2),
    ↪  padding='same')(conv7)
    merge8 = concatenate([conv2, up8], axis=3)
    conv8 = Conv2D(64, 3, activation='relu',
    ↪  padding='same')(merge8)
    conv8 = Conv2D(64, 3, activation='relu',
    ↪  padding='same')(conv8)

    up9 = Conv2DTranspose(32, 2, strides=(2, 2),
    ↪  padding='same')(conv8)
    merge9 = concatenate([conv1, up9], axis=3)
    conv9 = Conv2D(32, 3, activation='relu',
    ↪  padding='same')(merge9)
    conv9 = Conv2D(32, 3, activation='relu',
    ↪  padding='same')(conv9)

    outputs = Conv2D(num_classes, 1,
    ↪  activation='softmax')(conv9)

    model = Model(inputs=inputs, outputs=outputs)
    return model

```

Lo strato di output è uno strato convoluzionale con un filtro per ciascuna classe, che utilizza la funzione di attivazione softmax per ottenere probabilità di appartenenza a ciascuna classe per pixel.

In sintesi, questo codice definisce un modello basado nell' architetura U-Net per la segmentazione delle immagini, progettata per classificare ciascun pixel in una delle diverse classi possibili, utilizzando una combinazione di strati convoluzionali e trasposti convoluzionali con connessioni skip tra la parte codificatrice e il decodificatore.

Le principali differenze nell'architettura del modello di questo progetto con il modello originale, U-Net Multiclass utilizza un livello di ridimensionamento per normalizzare le immagini di input dividendo i valori dei pixel per 255, mentre l'U-net originale non include questa normalizzazione livello esplicitamente nella sua implementazione. Il numero di filtri nelle convoluzioni nella multiclasse U-Net inizia con 32 filtri nel primo livello convoluzionale e raddoppia il numero di filtri in ogni livello successivo (32, 64, 128, 256, 512). Mentre nell'U-Net originale inizia con 64 filtri nel primo strato convoluzionale e raddoppia anche il numero di filtri in ogni livello successivo (64, 128, 256, 512, 1024).

La multiclasse U-Net introduce livelli di dropout dopo determinati livelli convoluzionali ai livelli più profondi (conv4 e conv5) per evitare un adattamento eccessivo e U-Net originale non include livelli di dropout nell'architettura. U-Net Multiclass utilizza un livello di output Conv2D con attivazione softmax per gestire la segmentazione multiclasse, dove il numero di filtri nell'ultimo livello convoluzionale è uguale al numero di classi (num-classes). Mentre nell'U-Net originale utilizza un livello di output Conv2D con attivazione softmax e due filtri, per un'attività di segmentazione binaria (sfondo vs oggetto).

```
# Definir el tamaño de entrada de las imágenes y el número de clases
input_shape = (256, 256, 3)
num_classes = 6  # Número de clases

# Construir el modelo U-Net para segmentación multiclase
model = unet_multiclass(input_shape, num_classes)

# Compilar el modelo
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Resumen del modelo
#model.summary()

# Ajuste del tamaño del lote
history = model.fit(X_train, y_train, epochs=30, batch_size=64, validation_data=(X_val, y_val))
```

La funzione model.compile configura il modello per l'addestramento. Specifica l'ottimizzatore, la funzione di perdita e le metriche da utilizzare durante l'addestramento e la valutazione. Viene utilizzato l'ottimizzatore Adam, che è un algoritmo di ottimizzazione comunemente utilizzato nell'addestramento delle reti neurali. Si tratta di un'estensione della discesa stocastica del gradiente che utilizza le stime del primo e del secondo momento dei gradienti per adattare i tassi di apprendimento di diversi parametri. La funzione di perdita sparse-categorical-crossentropy viene utilizzata per problemi di classificazione multiclasse come nel caso. Questa funzione di perdita prevede etichette intere invece di vettori one-hot. In questo caso, la metrica utilizza l'accuratezza, che misura la percentuale di previsioni corrette.

Il modello è stato addestrato con le 10000 immagini risultanti, ottenendo una accuratezza approssimativa dell'80%, con epoche = 30 e bach-size = 64. È bene precisare che i parametri precedenti sono stati modificati rispettivamente con 50 e 32/64, senza ottenere risultati migliori. Con più di 50 epoche il modello cade in overfitting quindi la funzione di perdita inizia a crescere e l'accuratezza a diminuire. Il tempo di addestramento è durato 23 minuti e abbiamo ottenuto all'incirca le stesse prestazioni che avevamo ottenuto con meno immagini (l'accuratezza dell'80%). Inoltre è stato addestrato con sole

5057 immagini 512x512 ridimensionate a 256x256, ottenendo risultati simili.

Il modello addestrato con le 10.000 immagini viene salvato in "unet-pretrained.h5". Questo file salva l'architettura del modello, consentendo di ricostruire il modello, i pesi del modello, la configurazione dell'ottimizzatore e lo stato dell'addestramento (numero di epoche completate, stato dell'ottimizzatore, ecc.). In questo caso, come avevamo detto all'inizio, realizziamo due set train-test-val-split, uno con le prime 10.000 immagini e l'altro con 12.356 immagini con cui faremo il fine tuning.

5.3 Fine-tuning

```
model.save('unet_pretrained.h5')
# Fine-tuning del modelo

# Cargar el modelo preentrenado
model1 = unet_multiclass(input_shape, num_classes)
model1.load_weights('unet_pretrained.h5')

# Congelar capas específicas
for i, layer in enumerate(model1.layers):
    if i in range(2, 19):  # Cambiado a `range` para incluir las capas entre 2 y 18
        layer.trainable = False

# Compilar el modelo
model1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo con las capas congeladas
history1 = model1.fit(X_trainA, y_trainA, epochs=20, batch_size=32, validation_data=(X_valA, y_valA))

# Descongelar algunas capas y continuar entrenando
for i, layer in enumerate(model1.layers):
    if i in range(2, 19):  # Cambiado a `range` para incluir las capas entre 2 y 18
        layer.trainable = True

# Compilar el modelo nuevamente con una tasa de aprendizaje más baja
model1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Continuar entrenando
history_finetune = model1.fit(X_trainA, y_trainA, epochs=20, batch_size=32, validation_data=(X_valA, y_valA))
```

I pesi preaddestrati vengono caricati da un file (unet-pretrained.h5). Nella prima parte congeliamo l'encoder per evitare che i loro pesi si aggiornino durante l'addestramento. Ciò è utile quando si desidera

ottimizzare solo gli ultimi livelli del modello mantenendo la conoscenza appresa dai primi livelli. "layer.trainable = False" garantisce che i pesi di questi strati specifici non vengano modificati durante l'addestramento. Alcuni parametri vengono impostati allo stesso modo dell'addestramento iniziale con 10.000 immagini, ad eccezione di quelli che aumentiamo per vedere dove il modello inizia ad adattarsi eccessivamente. Per l'addestramento del modello viene utilizzato il secondo set, composto da 12.356 immagini.

In un secondo training del modello, gli strati precedentemente congelati vengono sbloccati in modo che i loro pesi possano essere aggiornati durante il nuovo processo di training. Ciò consente al modello di apprendere nuove funzionalità specifiche del nuovo set di dati. Viene utilizzata una velocità di apprendimento inferiore per consentire regolazioni più precise ai pesi degli strati non congelati senza disturbare troppo ciò che hai già appreso.

Di conseguenza, per ottenere un modello migliore, in questo lavoro implementiamo un terzo modello in cui realizziamo un'architettura ibrida. Il modello resnet50-unet (model2) combina un'architettura U-Net con ResNet50 come base per l'encoder. ResNet50 è un'architettura di rete neurale profonda ampiamente utilizzata in attività di visione artificiale come la classificazione e la segmentazione delle immagini. Il suo design innovativo include il concetto di blocchi residui, che facilitano l'addestramento di reti neurali molto profonde.

ResNet50 viene utilizzato come encoder del modello U-Net. Si tratta di una rete pre-addestrata sul set di dati ImageNet, nota per i suoi blocchi di apprendimento residui che aiutano ad addestrare reti profonde senza problemi di degrado. Tutti i layer ResNet50 sono congelati, il che significa che durante l'addestramento i pesi di questi layer non verranno aggiornati. Questo viene fatto per sfruttare le funzionalità apprese da ResNet50 senza disturbarne l'addestramento. [H]

```python
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, UpSampling2D, concatenate
from tensorflow.keras.models import Model

def resnet50_unet(input_shape, num_classes):
    base_model = tf.keras.applications.ResNet50(input_shape=input_shape, include_top=False, weights='imagenet')

    # Freeze the ResNet50 layers
    for layer in base_model.layers:
        layer.trainable = False

    # Encoder: Extract features from the ResNet50 layers
    conv1 = base_model.get_layer('conv1_relu').output
    conv2 = base_model.get_layer('conv2_block3_out').output
    conv3 = base_model.get_layer('conv3_block4_out').output
    conv4 = base_model.get_layer('conv4_block6_out').output
    conv5 = base_model.get_layer('conv5_block3_out').output

    # Decoder: Up-sampling and concatenation
    up6 = UpSampling2D(size=(2, 2))(conv5)
    up6 = concatenate([up6, conv4], axis=3)
    conv6 = Conv2D(512, 3, activation='relu', padding='same')(up6)
    conv6 = Conv2D(512, 3, activation='relu', padding='same')(conv6)

    up7 = UpSampling2D(size=(2, 2))(conv6)
    up7 = concatenate([up7, conv3], axis=3)
    conv7 = Conv2D(256, 3, activation='relu', padding='same')(up7)
    conv7 = Conv2D(256, 3, activation='relu', padding='same')(conv7)

    up8 = UpSampling2D(size=(2, 2))(conv7)
    up8 = concatenate([up8, conv2], axis=3)
    conv8 = Conv2D(128, 3, activation='relu', padding='same')(up8)
    conv8 = Conv2D(128, 3, activation='relu', padding='same')(conv8)

    up9 = UpSampling2D(size=(2, 2))(conv8)
    up9 = concatenate([up9, conv1], axis=3)
    conv9 = Conv2D(64, 3, activation='relu', padding='same')(up9)
    conv9 = Conv2D(64, 3, activation='relu', padding='same')(conv9)

```

```
1
2       up10 = UpSampling2D(size=(2, 2))(conv9)
3       conv10 = Conv2D(32, 3, activation='relu',
        ↪  padding='same')(up10)
4       conv10 = Conv2D(32, 3, activation='relu',
        ↪  padding='same')(conv10)
5
6       outputs = Conv2D(num_classes, 1,
        ↪  activation='softmax')(conv10)
7
8       model = Model(inputs=base_model.input, outputs=outputs)
9       return model
10      # Compilar el modelo
11      model2.compile(optimizer='rmsprop',
        ↪  loss='sparse_categorical_crossentropy',
        ↪  metrics=['accuracy'])
12
13       # Verificar la estructura del modelo
14       #model.summary()
15
16       # Entrenar el modelo
17       history2 = model2.fit(X_train, y_train,
         ↪  validation_data=(X_val, y_val), epochs=30,
         ↪  batch_size=128)
18
```

5.4 Metrics

Per valutare le prestazioni di un modello di segmentazione semantica, vengono utilizzate diverse metriche chiave. Uno dei più importanti è la Global Accuracy, che misura la percentuale di pixel correttamente classificati nell'intera immagine. Un'altra metrica cruciale è l'Intersection over Union (IoU), noto anche come indice Jaccard, che valuta la sovrapposizione tra l'area prevista e l'area effettiva per ciascuna classe, fornendo una misura di precisione più rigorosa. Il coefficiente di DICE, o punteggio F1, è simile a IoU ma tende ad essere più sensibile alle classi con meno pixel. Inoltre, puoi considerare l'accuratezza per classe e l'IoU per classe, che forniscono una visualizzazione dettagliata delle prestazioni del modello in ogni classe specifica. Altre metriche includono Recall e Precision, che si concentrano sulla capacità del modello di identificare correttamente tutti i pixel in una classe (recall) e sull'accuratezza dei pixel previsti per quella classe (precision). Queste metriche combinate offrono una va-

lutazione completa delle prestazioni del modello nella segmentazione semantica, aiutando a identificare sia i suoi punti di forza che le aree di miglioramento.

5.4.1 Modello Unet-pretrained

L'immagine mostra la funzione di perdita e accuratezza dell'addestramento e della convalida per un modello UNet-pretrained su 30 epoche. [H]

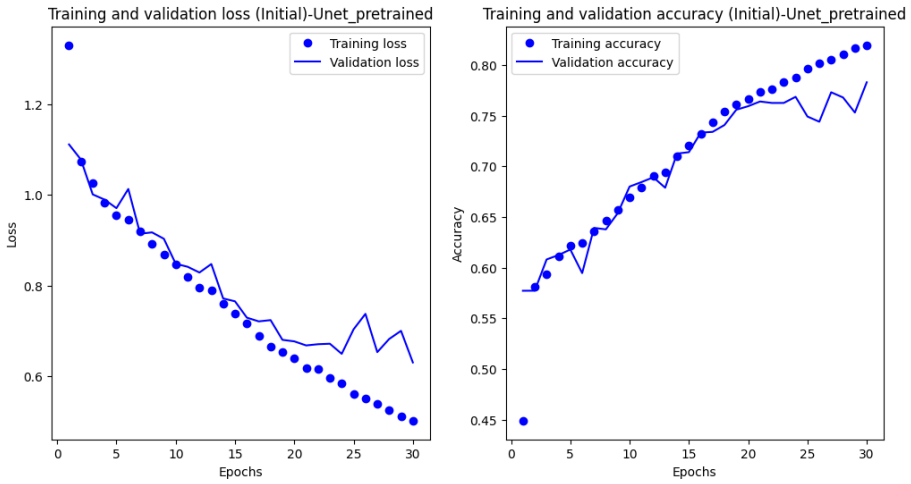

Figure 5: Model 1: Unet-pretraning

Le prestazioni del modello migliorano con più epoche, come si vede nella diminuzione della perdita e nell'aumento dell'accuratezza sia per i set di training che per quelli di convalida. Le fluttuazioni nei parametri di validazione sono normali e suggeriscono che il modello si generalizza abbastanza bene, anche se è presente una certa variabilità. Non si osserva alcun overfitting significativo entro le prime 30 epoche.

Per valutare il modello Unet-pretrained analizzeremo le metriche calcolate sul set di test. È stata calcolata la matrice di confusione per poi calcolare le metriche di valutazione Accuracy, Precision, Recall, F1-Score e l'indice Jaccard. La matrice di confusione viene visualizzata in percentuale, indicando la distribuzione delle previsioni tra le diverse classi. Gli elementi diagonali rappresentano istanze classificate correttamente per ciascuna classe, mentre gli elementi fuori diagonale indicano classificazioni errate. Classe 1: Background è classificato correttamente nell'84.57% dei casi, Classe 2: Roads-84.51%,

Classe 3: Building-44.97%, Classe 4: Low vegetation-70.08%, Classe 5: Tree -74.77%, Classe 6: Car - 73.53%. [H]

```

# Calcular la exactitud, precisión, recall y F1-score
accuracy = accuracy_score(y_test_flat, y_pred_flat)
precision = precision_score(y_test_flat, y_pred_flat,
↪  average='macro')
recall = recall_score(y_test_flat, y_pred_flat, average='macro')
f1 = f1_score(y_test_flat, y_pred_flat, average='macro')
jaccard = jaccard_score(y_test_flat, y_pred_flat,
↪  average='macro')

print("Confusion Matrix (in %):")
print(conf_matrix_df)
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
print(f"Jaccard Index: {jaccard}")
63/63 [==============================] - 5s 86ms/step
Confusion Matrix (in %):
                BG  Roads  Building   LowV   Tree    Car
BG           84.57   8.77      1.73   0.22   0.97   3.75
Roads         5.33  84.51      2.16   0.17   3.46   4.37
Building     17.41  18.41     44.97   0.41   5.61  13.19
LowV          5.86  16.46      4.99  70.08   2.12   0.48
Tree          1.08   4.73      1.59   0.21  74.77  17.63
Car           3.54   5.11      2.91   0.01  14.91  73.53
Accuracy: 0.7871505661010743
Precision: 0.7489655132403841
Recall: 0.7206829994740334
F1-Score: 0.7328970210614908
Jaccard Index: 0.5940351817817756
```

Il modello mostra prestazioni decenti con una acuratezza di circa il 78,71%. La precisione, la sensibilità e il punteggio F1 sono relativamente bilanciati, indicando che il modello identifica bene i veri positivi mentre gestisce i falsi positivi e i negativi. La matrice di confusione suggerisce che la Classe 3 è la più impegnativa per il modello, con un numero significativo di istanze erroneamente classificate in altre classi. Il modello ha imparando in modo efficace, pero con il potenziale per un'ulteriore aggiustamento per migliorare la accuracy

complessiva.

5.4.2 Modello Unet-pretrained2

Come avevamo detto in precedenza, il secondo Modello U-net_pre trained2 avrà la base del primo modello salvata in unet-pretained.h5. Che conterrà i pesi corrispondenti del primo modello, addestrato con le prime 10.000 immagini. Questo secondo modello verrà addestrato con le restanti 12.356 immagini. Nella prima parte dell'addestramento gli strati dell'Encoder vengono congelati in modo che i loro pesi non vengano aggiornati e in una seconda parte l'Encoder viene sbloccato, quindi i pesi vengono aggiornati nell'addestramento con il secondo set di dati.

Il modello di segmentazione ha mostrato buone prestazioni complessive con una accuratezza dell'81,03%. La precisione media è del 77,97%, il che indica che, in media, quando il modello fa una previsione, è corretta il 77,97% delle volte. Il reacall medio è del 76.28%, suggerendo che il modello riesce a catturare il 76,28% dei veri positivi in ciascuna classe. Il punteggio F1, che bilancia precisione e recall, è del 77,01%, e l'indice medio di Jaccard è del 63,75%, che misura la somiglianza tra le previsioni e le etichette reali. [H]

```
print(f"Confusion Matrix (in %):\n{conf_matrix_percentA}")
print(f"Accuracy: {accuracyA}")
print(f"Precision: {precisionA}")
print(f"Recall: {recallA}")
print(f"F1-Score: {f1A}")
print(f"Jaccard Index: {jaccardA}")

78/78 [==============================] - 9s 115ms/step
Confusion Matrix (in %):
                BG  Roads  Building   LowV   Tree    Car
BG           87.65   6.28      1.13   0.21   1.08   3.65
Roads         5.21  84.48      2.28   0.40   2.01   5.61
Building     12.86  14.92     52.06   0.79   3.65  15.72
LowV          2.33  10.51      4.16  80.68   1.52   0.80
Tree          1.40   5.56      0.94   0.25  71.14  20.70
Car           2.84   4.39      2.42   0.03   8.64  81.68
Accuracy: 0.8102957407633463
Precision: 0.7797728902198716
Recall: 0.7628283269569379
F1-Score: 0.7701548011042797
Jaccard Index: 0.6375737355559844
```

Nel dettaglio, le classi 1 e 2 mostrano prestazioni elevate con una accuratezza rispettivamente dell'87.65% e dell'84.48% e con poche confusioni. Tuttavia, la classe 3 ha prestazioni notevolmente inferiori, con una accurateza del 52.06% e molte confusioni con la classe 3 e altre. La classe 5, con una accuratezza del 71.14%, è significativamente confusa con la classe 6 (20.70%). La classe 6, invece, ha una accuratezza dell'81.68%.

Confrontando i risultati dei due modelli, si osserva che Unet_pretrained2 (Modello 2) supera Unet_pretrained (Modello 1) in tutte le metriche valutate. Il modello 2 mostra una accuracy dell'81,02%, mentre il modello 1 ha una accuracy del 78,71%. Ciò indica che il Modello 2 effettua previsioni corrette più frequentemente. In termini di precisione, anche il Modello 2 è superiore, al 77.97% contro il 74,89% del Modello 1, suggerendo che le previsioni positive del Modello 2 sono più affidabili.

Il reacall del Modello 2 è del 76,28%, rispetto al 72,06% del Modello 1, indicando che il Modello 2 è più efficace nel catturare i veri positivi. Inoltre, il punteggio F1, che bilancia precisione e recall, è

più alto nel Modello 2 (77,01%) rispetto al Modello 1 (73,28%), indicando prestazioni complessive migliori. L'indice Jaccard, che misura la somiglianza tra le previsioni e le etichette effettive, è del 63,75% per il Modello 2 e del 59,40% per il Modello 1, riaffermando il vantaggio del Modello 2.

In particolare, il Modello 2 è più efficace nella classe 1, con una accuracy dell'87.65% rispetto al 84.57% del Modello 1. Tuttavia, entrambi i modelli faticano con la classe 3, dove l'accuratezza è significativamente inferiore (52.06% per il Modello 2 e 44.97% per il Modello 1), suggerendo potenziali problemi di distinguibilità nei dati di questa classe. In sintesi, il Modello 2 dimostra prestazioni complessive migliori ed è una scelta migliore per la segmentazione semantica.

5.4.3 Resnet50-unet

Il terzo e ultimo modello sviluppato è un ibrido tra il modello Resnet50 e U-net. Il codificatore del modello sarà costituito da determinati livelli del modello Resnet con i relativi pesi. Il decoder sarà formato dal decoder UNET (512, 256, 128, 64, 32). Si tratta di un modello U-Net che utilizza l'architettura ResNet50 pre-addestrata come base per l'attività di segmentazione delle immagini. ResNet50 è un modello di rete neurale profonda che è stato pre-addestrato sul set di dati ImageNet e ha appreso funzionalità visive generali utili per l'attività di visione artificiale. All'interno di questa funzione, l'addestramento degli strati del modello ResNet50 viene congelato impostando layer.trainable = False per ogni strato, il che significa che durante il processo di addestramento, i pesi di questi strati preaddestrati non verranno aggiornati. Questo approccio consente al modello di sfruttare le caratteristiche visive generali già apprese da ResNet50 mettendo a punto solo i nuovi livelli dell'architettura U-Net progettati specificamente per l'attività di segmentazione, accelerando l'addestramento ed evitando l'adattamento eccessivo su dati piccoli o specializzati. [H]

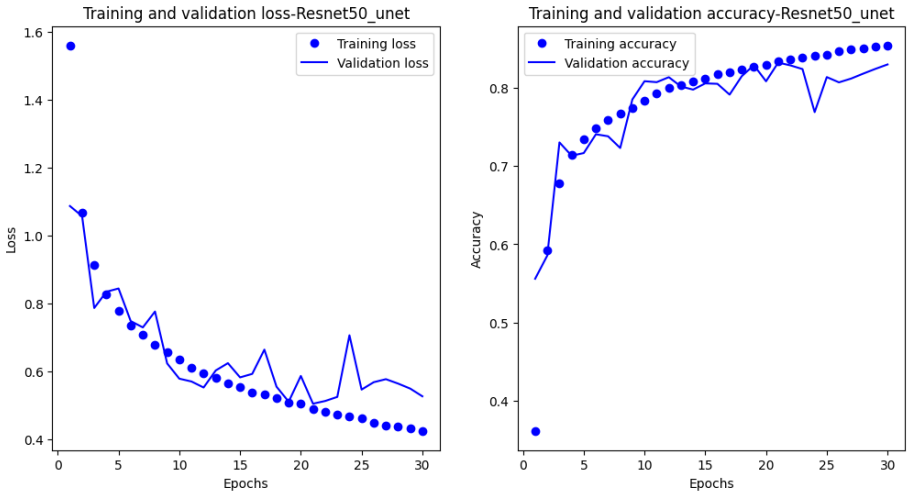

Figure 6: Modello Resnet50_UNET

L'immagine mostra la funzione di perdita e accuratezza dell'addestramento e della convalida per un modello Resnet50_UNET su 30 epoche.

L'analisi comparativa tra i modelli U-Net_pretrained2 e ResNet50-U-Net rivela differenze significative nelle loro prestazioni per l'attività di segmentazione semantica, evidenziate attraverso diverse metriche chiave. Il modello U-Net_pretrained2, addestrato con le 12.356 immagini, ha raggiunto una accuracy dell'81,02%, un recall del 76,28%, un punteggio F1 di 0,77 e un indice Jaccard di 0,637. Questi parametri indicano che il modello ha avuto prestazioni relativamente forti, ma con aree di miglioramento nel recupero per le classi più impegnative.

```
1
2  print(f"Confusion Matrix (in %):\n{conf_matrix_percent}")
3  print(f"Accuracy: {accuracy}")
4  print(f"Precision: {precision}")
5  print(f"Recall: {recall}")
6  print(f"F1-Score: {f1}")
7  print(f"Jaccard Index: {jaccard}")
8
9  125/125 [==============================] - 27s 153ms/step
10 Confusion Matrix (in %):
11                BG  Roads  Building   LowV   Tree    Car
12 BG          88.86   5.90      0.29   0.20   0.71   4.03
13 Roads        3.37  87.61      0.70   0.27   1.42   6.62
14 Building    11.68  14.83     57.77   0.39   1.55  13.78
15 LowV         3.44  12.64      0.59  81.12   1.27   0.94
16 Tree         0.62   4.09      0.37   0.16  74.12  20.64
17 Car          1.80   3.97      0.85   0.02   5.13  88.24
18 Accuracy: 0.8457621192932129
19 Precision: 0.8472074370839437
20 Recall: 0.7962116113919736
21 F1-Score: 0.8160565458791428
22 Jaccard Index: 0.6955008937631887
```

Al contrario, il modello ResNet50-U-Net, che incorporava come base un'architettura di rete neurale pre-addestrata ResNet50 ha mostrato prestazioni superiori rispetto al tradizionale modello U-Net. Il modello è stato addestrato con le prime 20.000 immagini. Con una accuracy dell'84,57%, un recall dell'79,62%, un punteggio F1 di 0,816 e un indice Jaccard di 0,70, ResNet50-U-Net non solo ha sovraperformato U-Net in tutti i parametri chiave, ma ha anche dimostrato una migliore capacità di identificare correttamente le diverse classi all'interno del set di dati.

5.5 Results

Di conseguenza, ResNet50_U-Net non solo ha appreso meglio le caratteristiche dell'immagine, ma ha anche mostrato una maggiore capacità di generalizzare e gestire la variabilità nei dati di input. L'aumento dell'accuratezza del 3,54% e del recall del 3,34% tra ResNet50_U-Net e U-Net_pretrained2 suggerisce che il modello ResNet50-U-Net è più efficace nel generalizzare caratteristiche visive complesse e gestire le variazioni nei dati. Questo progresso si riflette in un punteggio F1 più elevato e in un indice Jaccard più elevato, indicando

non solo una migliore capacità di identificare correttamente i pixel rilevanti, ma anche un miglioramento della qualità complessiva delle segmentazioni prodotte. [H]

Metrica	**U-Net pretrained2**	**ResNet50-U-Net**	**Differenza**
Accuracy	81.03%	84.57%	+3.54%
Recall	76.28%	79.62%	+3.34%
F1-Score	0.77	0.82	+0.05
Jaccard Index	0.63	0.70	+0.07

Table 1: Confronto dei parametri prestazionali tra il modello U-Net_pretrained2 e il modello ResNet50_U-Net.

Nelle attività di segmentazione semantica, l'output di un modello è una maschera che classifica ogni pixel di un'immagine in una delle numerose classi predefinite. Questa maschera prevista viene confrontata con una maschera di verità per valutare la qualità e l'accuratezza del modello nell'identificare regioni specifiche all'interno dell'immagine. Di seguito sono riportate le immagini comparative della maschera di verità e della maschera prevista generata dai modelli Unet_pretrained2 e ResNet50-U-Net, insieme ad esempi dell'immagine originale.

Le immagini illustrano tre elementi chiave nella valutazione della segmentazione: l'immagine originale, la maschera di verità e la maschera prevista generata dal modello. Attraverso queste immagini, è evidente che ResNet50-U-Net raggiunge una segmentazione delle classi più accurata, catturando dettagli più fini e riducendo gli errori rispetto al modello precedente. Questo miglioramento nella qualità della maschera prevista è un indicatore dell'efficacia del modello in compiti complessi di segmentazione semantica. [H]

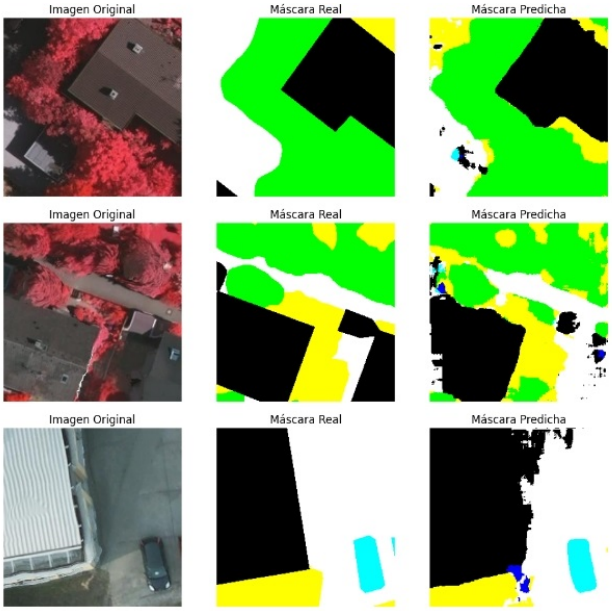

Figure 7: Maschera prevista Modello Unet_pretrained2

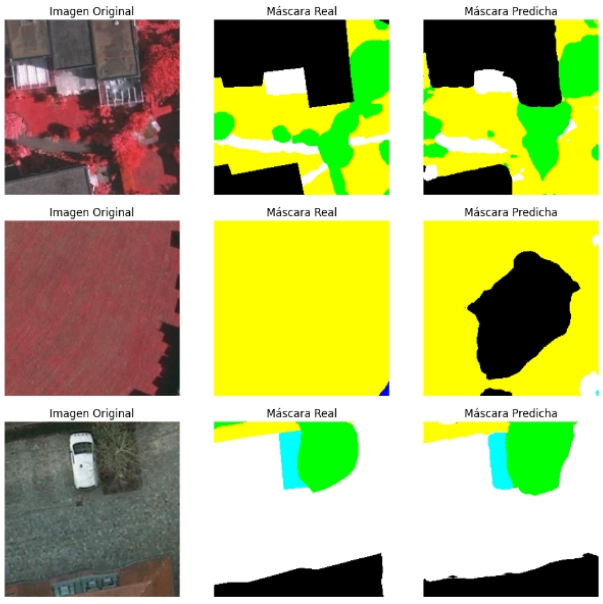

Figure 8: Maschera prevista Modello Resnet50_Unet

6 Discussion

I risultati ottenuti mostrano chiaramente che il modello ResNet50_U-Net non solo ha appreso meglio le caratteristiche dell'immagine, ma ha anche dimostrato una maggiore capacità di generalizzare e gestire la variabilità dei dati di input. L'aumento del 3,54% nell'accuratezza e l'aumento del 3,34% nel recall tra ResNet50_UNet e U-Net_pre-trained2 suggerisce che il modello ResNet50_UNet è più efficace nella generalizzazione di caratteristiche visive complesse e nella gestione delle variazioni in dati. La superiorità di ResNet50_UNet in questi parametri chiave sottolinea la sua efficacia e robustezza nel compito di segmentazione semantica, rendendo questo modello una scelta preferita per applicazioni in cui l'accuratezza e la capacità di generalizzazione sono cruciali.

7 Conclusion

7.1 Obtained results

Il confronto tra il modello U-Net_pre-trained2 e ResNet50_U-Net per la segmentazione semantica rivela che ResNet50_U-Net supera costantemente il modello U-Net_pretrained2 su tutti i parametri valutati. Con una accuratezza dell'84,57%, un recall dell'79,62%, un punteggio F1 di 0,816 e un indice Jaccard di 0,70, ResNet50 U-Net ha dimostrato una capacità superiore di generalizzare e gestire la variabilità nei dati di input rispetto all'U-Net preaddestrato. Net, che ha ottenuto una accuratezza dell'81,03%, un reacll del 76,28%, un punteggio F1 di 0,77 e un indice Jaccard di 0,63.

7.2 Limits

Nonostante i significativi progressi osservati, sia U-Net_pretrained2 che ResNet50_U-Net affrontano sfide nel segmentare accuratamente determinate classi di oggetti, soprattutto in classificare la classe 3. Inoltre, la scalabilità e i tempi di formazione possono rappresentare limitazioni significative per le distribuzioni di dati in tempo reale o di grandi dimensioni. Migliorare la robustezza rispetto a queste variazioni estreme rimane un compito in sospeso nello sviluppo di modelli di segmentazione semantica.

7.3 Future work

Per gli sviluppi futuri, è fondamentale affrontare le limitazioni identificate ricercando e implementando strategie che migliorino la capacità dei modelli di gestire condizioni avverse e dati di alta qualità. Ciò potrebbe includere l'esplorazione di tecniche di aumento dei dati più

avanzate (data augmentation), che migliora la capacità di generalizzazione dei modelli di machine learning generando versioni modificate delle immagini originali. Ciò si ottiene attraverso trasformazioni come rotazioni, traslazioni, ridimensionamento, inversioni, ritaglio, regolazione di luminosità, contrasto e saturazione, aggiunta di rumore, sfocatura e filtri colorati. Queste modifiche aiutano a creare un set di dati più diversificato e solido, riducendo il rischio di overfitting e migliorando le prestazioni del modello. Inoltre, la ricerca continua sui modelli ibridi che sfruttano sia le caratteristiche generali apprese che le strutture specializzate per compiti specifici rimarrà fondamentale per migliorare l'accuratezza e l'applicabilità dei modelli di segmentazione semantica in vari campi di applicazione.

8 Riferimenti Bibliografici

1. ISPRS Test Project on Urban Classification and 3D Building Reconstruction, allegato a questo lavoro.

2. Dataset: Aerial Image Segmentation

3. Plaksyvyi, A., Skublewska-Paszkowska, M., Powroznik, P. (2023). A Comparative Analysis of Image Segmentation Using Classical and Deep Learning Approach. Advances in Science and Technology Research Journal, 17(6), 127-139.

4. Malik, Pankaj & Chourasiya, Ms & Pandit, Rakesh & Bharaskar, Mr & Medi, Asst. (2023). Satellite Image Segmentation Using Neural Networks: A Comprehensive Review. International Journal of Enhanced Research in Educational Development. Vol. 11. 2320-8708.

5. Marmanis, Dimitris & Wegner, Jermaine & Galliani, Silvano & Schindler, Konrad & Datcu, Mihai & Stilla, Uwe. (2016). SEMANTIC SEGMENTATION OF AERIAL IMAGES WITH AN ENSEMBLE OF CNNS. ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences. III-3. 473-480. 10.5194/isprs-annals-III-3-473-2016.

6. Olaf Ronneberger, Philipp Fischer, Thomas Brox: 0U-Net: Convolutional Networks for Biomedical Image Segmentation. MICCAI (3) 2015: 234-241

7. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A.C., Lo, W., Dollár, P., & Girshick, R.B. (2023). Segment Anything. 2023 IEEE/CVF International Conference on Computer Vision (ICCV), 3992-4003.