UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Elettronica

IMPLEMENTAZIONE DI UNA RETE NEURALE PER IL RILEVAMENTO DI INONDAZIONI BASATA SU SEGMENTAZIONE SEMANTICA

IMPLEMENTATION OF A NEURAL NETWORK FOR FLOOD DETECTION BASED ON SEMANTIC SEGMENTATION

Relatore: Prof.ssa

Laura Falaschetti

Tesi di Laurea di:

Alessandro Polverari

A.A. **2021/2022**

INDICE

1. Introduzione

Il problema delle inondazioni è un tema sempre più attuale a causa del cambiamento climatico che si sta osservando negli ultimi decenni. Come afferma l'ONU, le temperature in continuo aumento stanno modificando i modelli metereologici e sconvolgendo gli equilibri naturali. L'innalzamento delle temperature ha provocato una maggiore umidità che accentua le tempeste e le precipitazioni causando temporali sempre più devastanti e, di conseguenza, si registra un numero sempre maggiore di inondazioni. Ciò che è accaduto a Senigallia, in provincia di Ancona, a settembre 2022 è una prova del fatto che la problematica delle inondazioni non vada sottovalutata. Un primo passo per il controllo di questi fenomeni estremi è quello di utilizzare sistemi di sorveglianza sui livelli dei bacini idrici che permettano di intervenire tempestivamente per prevenire le esondazioni. Le reti neurali sono un'ottima soluzione per realizzare sistemi in grado di controllare l'altezza dell'acqua e di segnalare in anticipo la possibilità che questa possa oltrepassare gli argini.

La presente tesi tratterà dell'implementazione di una rete neurale per la rilevazione delle inondazioni basata sulla segmentazione semantica. Per prima cosa si definiranno i concetti di rete neurale convoluzionale e di segmentazione semantica. Nei capitoli successivi verrà illustrato il codice della rete e la sua implementazione. Infine,

saranno mostrati e spiegati i risultati ottenuti, analizzando possibili utilizzi e progetti futuri.

2. Reti neurali

In generale, le reti neurali sono un modello di calcolo la cui struttura stratificata assomiglia alla rete di neuroni nel cervello. Esse sono note anche come reti neurali artificiali o ANN (artificial neural network) o reti neurali simulate o SNN (simulated neural network); inoltre sono un sottoinsieme del machine learning e sono l'elemento centrale degli algoritmi di deep learning.

Le reti neurali artificiali (ANN) sono composte da livelli di nodi che contengono un livello di input, uno o più livelli nascosti e un livello di output. Ciascun neurone artificiale, si connette ad un altro e ha un peso e una soglia associati. Se l'output di qualsiasi singolo nodo è al di sopra del valore di soglia specificato, tale nodo viene attivato, inviando i dati al successivo livello della rete. In caso contrario, non viene passato alcun dato al livello successivo della rete.

Le reti neurali fanno affidamento sui dati di addestramento per imparare e migliorare la loro accuratezza nel tempo. Di conseguenza, una volta ottimizzati per l'accuratezza, questi algoritmi di apprendimento sono dei potenti strumenti nella computer science e nell'intelligenza artificiale, consentendoci di classificare e organizzare in cluster i dati ad alta velocità.

Le reti neurali si possono classificare in diverse tipologie, in base all'utilizzo che ne viene fatto. Le più comuni sono le seguenti:

- Le reti neurali feedforward sono formate da un livello di input, uno o più livelli nascosti e un livello di output. I dati vengono solitamente passati in questi modelli per addestrarli e sono la base per la visione artificiale, l'NLP (natural language processing) e altre reti neurali.
- Le reti neurali convoluzionali o CNN (convolutional neural network) sono simili alle reti feedforward, ma sono di solito utilizzate per il riconoscimento delle immagini, il riconoscimento dei modelli e/o la visione artificiale. Queste reti sfruttano i principi dall'algebra lineare, in particolar modo la moltiplicazione della matrice, per identificare i modelli all'interno di un'immagine.
- Le reti neurali ricorrenti o RNN (recurrent neural network) sono identificate dai loro loop di feedback. Ci si avvale di questi algoritmi di apprendimento soprattutto quando si utilizzano dati di serie temporali per fare delle previsioni su risultati futuri, come ad esempio delle previsioni relative al mercato azionario o delle previsioni di vendita.

2.1 Reti neurali convoluzionali

Nella presente tesi verranno trattate le reti neurali convoluzionali, o CNN. Una rete neurale convoluzionale è un algoritmo di Deep Learning che elabora un'immagine di input durante la fase di

convoluzione e in seguito le attribuisce un'etichetta. Inoltre, in una CNN è richiesta una pre-elaborazione molto più bassa rispetto ad altri algoritmi di classificazione.

I componenti principali di una CNN sono:

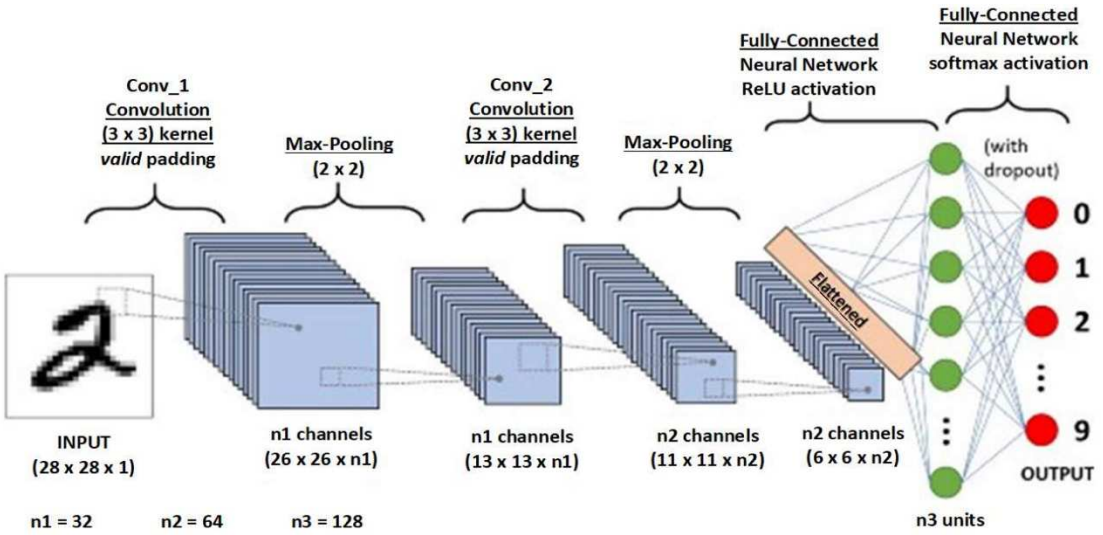
- *Convolutional layer (Kernel)*: come detto in precedenza, lo scopo della convoluzione è quello di estrarre localmente le caratteristiche dell'oggetto sull'immagine; ciò significa che la rete è in grado di imparare modelli specifici all'interno di un'immagine e sarà poi capace di riconoscerli ovunque nell'immagine. L'operazione di convoluzione consiste in una moltiplicazione elementare: si esegue una scansione di una parte dell'immagine di dimensione 3x3 e la si moltiplica per un filtro e il risultato dell'operazione viene chiamato "Feature map". Tale operazione viene ripetuta finché non è stata scansionata tutta l'immagine e al termine si ottiene un'immagine di dimensioni ridotte. Le caratteristiche del livello convoluzionale della rete sono controllate da tre parametri:

- Depth: definisce il numero di filtri da applicare durante la convoluzione;
- Passo(stride): definisce il numero di salto di pixel tra due sezioni;
- Zero-Padding: aggiunge un numero corrispondente di righe e colonne su ciascun lato dei feature map di input.

- *Funzione di attivazione* (ReLU): dopo la convoluzione l'output è soggetto a una funzione di attivazione per consentire la non linearità; la tipica funzione di attivazione per una CNN è la ReLU, che sostituisce i pixel con valori negativi con il valore zero.
- *Pooling layer*: lo scopo del pooling è quello di ridurre la dimensionalità dell'immagine, in questo modo la rete ha meno

pesi da calcolare e si evita il problema del sovradimensionamento.

- *Fully connected layer*: l'ultimo livello consiste nel costruire una tradizionale rete neurale artificiale; si collegano tutti i neuroni dal livello precedente al livello successivo e si utilizza una funzione di attivazione softmax per classificare il numero sull'immagine di input.



3. Image processing

L'elaborazione digitale delle immagini è un processo che consiste nell'utilizzo di algoritmi che, a partire dai valori dei pixel dell'immagine di partenza, restituiscono un'immagine modificata o un dato numerico che rappresenta una particolare caratteristica dell'immagine di input. Strettamente legate all'image processing sono la *computer graphics*, dove le immagini sono realizzate manualmente tramite modelli fisici e non acquisite da scene naturali e la *computer vision*, considerata una tecnica di elaborazione digitale ad alto livello in quanto decifra il contenuto di un'immagine o di una sequenza di immagini tramite una macchina o un software. Gli operatori di base che sono utilizzati nell'elaborazione delle immagini digitali sono:

- *Operatori punto*
  Effettuano una trasformazione del valore di un pixel a seconda del valore che esso ha nell'immagine originale. Definendo u il pixel dell'immagine originale e v il pixel nella stessa posizione in quella di destinazione, si può scrivere v=f(u), il tipo di trasformazione dipende solo da f(u). Un esempio di operatore punto è l'*operatore di thresholding.*

- *Operatori spaziali*
  Per determinare il valore di un pixel dell'immagine finale, questi operatori sfruttano il valore di tale pixel nell'immagine

di partenza ma anche il valore di alcuni pixel prossimi ad esso. Il numero di pixel vicini considerati dipende dall'utente che deve impostare una "finestra" all'interno della quale l'operatore deve lavorare. Due esempi di operatori spaziali sono il *filtro media* e il *filtro mediana*, il primo calcola la media aritmetica dei valori dei pixel all'interno della finestra considerata, il secondo invece calcola la mediana statistica. Il filtro media è particolarmente utile per eliminare disturbi distribuiti in modo gaussiano, mentre il filtro mediana elimina i disturbi casuali, detti *shot noise*.

Esistono diverse operazioni che possono essere effettuate su immagini digitali e tra queste vi sono:

- *Classificazione*
  Tecnica che utilizza un algoritmo statistico per individuare una rappresentazione di alcune caratteristiche di un'entità da classificare, associandole un'etichetta.

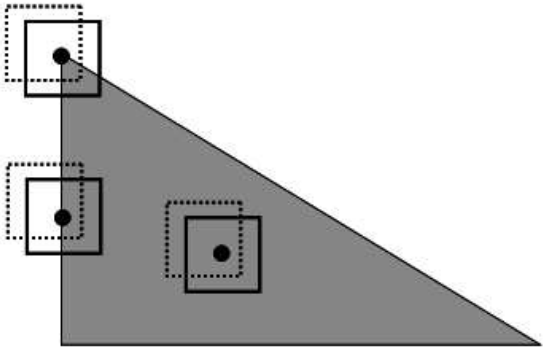
- *Segmentazione*
  La segmentazione è un processo di partizionamento dell'immagine in regioni significative. In particolare, è un'operazione che classifica i pixel con caratteristiche comuni, in questo modo i pixel di una stessa regione risultano simili tra loro per una qualche proprietà. Mentre regioni vicine sono significativamente differenti per almeno una caratteristica. Lo scopo di questa tecnica è quello di semplificare o modificare la rappresentazione delle immagini in modo che risultino più facili da analizzare.

- *Video tracking*
  Il video tracking è un processo che utilizza un algoritmo per analizzare i frame di un video e individuare la posizione di un oggetto che si muove nel tempo. Questa tecnica è utilizzata nei

sistemi di sorveglianza, nel controllo del traffico e per le video comunicazioni.

Per elaborare un'immagine digitale ci sono poi diverse tecniche che si applicano a parti diverse di un'immagine e che svolgono compiti differenti tra loro. Di seguito ne vengono descritte alcune:

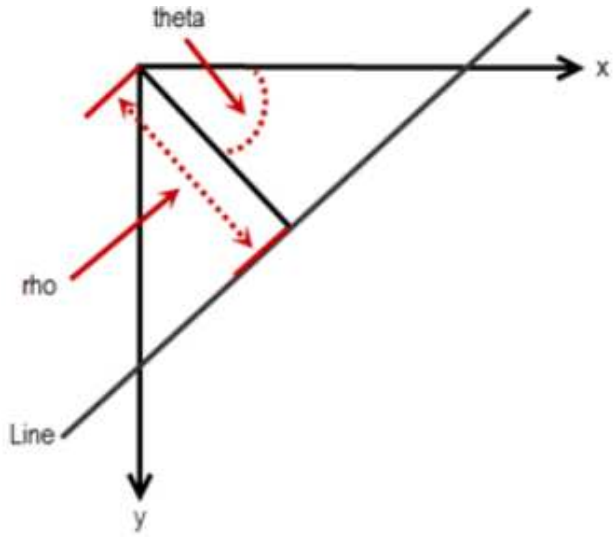
- *Edge Detection*
  Innanzitutto, in un'immagine un *edge* è una curva che segue un percorso di rapida variazione di intensità dell'immagine. Poiché i bordi sono spesso associati con i confini degli oggetti di una scena, l'edge detection viene utilizzato per individuare i bordi dell'immagine. L'operazione di riconoscimento dei contorni genera immagini contenenti molte meno informazioni rispetto a quelle originali, poiché conserva solo quelle necessarie per descrivere la forma e le caratteristiche strutturali e geometriche degli oggetti rappresentati. La funzione edge cerca le zone dell'immagine dove l'intensità cambia rapidamente e restituisce un'immagine binaria in cui gli 1 corrispondono agli edges trovati e 0 altrimenti. Esistono vari metodi di rilevamento dei bordi forniti dalla funzione edge e il migliore è il metodo Canny, che utilizza due soglie differenti per rilevare bordi ad alto e basso contrasto e per questo risulta più robusto al rumore.
- *Boundary Tracing*
  La funzione B = bwboundaries(BW) traccia i confini esterni degli oggetti e i confini di eventuali fori all'interno di questi oggetti, nell'immagine binaria BW, in cui i pixel non nulli appartengono a un oggetto e i pixel 0 costituiscono lo sfondo.
- *Corner Detection*
  I corners sono la caratteristica più affidabile per trovare una corrispondenza tra le immagini. La figura seguente mostra tre pixel, il primo all'interno, il secondo sul bordo e il terzo nell'angolo.



La linea continua indica l'area che circonda i pixel mentre la linea tratteggiata indica il contenuto energetico. Per quanto riguarda il primo pixel (quello al centro), il suo contenuto energetico è uguale a qualsiasi pixel nelle sue vicinanze preso in ogni direzione. Nel caso del secondo pixel (quello sul lato) la sua regione di appartenenza è differente da quella dei pixel vicini, in una direzione, ma è uguale nella direzione opposta; perciò, non possiamo definire univocamente la posizione del pixel. Il pixel nell'angolo invece avrà un'area che lo definisce diversa dai pixel vicini in tutte le direzioni. La funzione *corner* identifica gli angoli di un'immagine e utilizza metodi basati su algoritmi che dipendono dagli autovalori della sommatoria della squared difference matrix (SSD). Tali autovalori rappresentano la differenza tra la regione nell'intorno del pixel considerato con le regioni dei pixel vicini. L'autovalore è direttamente proporzionale al valore di questa differenza e più grande è l'autovalore maggiore è la probabilità che il pixel si trovi in un angolo.

- *Detect Lines*

  Per individuare le linee nelle immagini esistono delle funzioni che sfruttano la *trasformata di Hough*, che utilizza la rappresentazione parametrica di una linea: rho = x*cos(theta) + y*sin(theta), dove rho è la distanza dall'origine della linea e theta è l'angolo compreso tra l'asse x e questo vettore.



La funzione Hough implementa la Standard Hough Transform (SHT) e genera una matrice le cui righe e colonne indicano i valori di rho e theta. Dopo la realizzazione della trasformata di Hough si utilizza la funzione *houghpeaks* per trovare i valori di picco nello spazio parametrico, che rappresentano le potenziali linee nell'immagine in ingresso. Successivamente, è possibile utilizzare la funzione *houghlines* per trovare i punti finali dei segmenti corrispondenti ai picchi della trasformata di Hough.

- *Noise Removal*
  Le immagini digitali sono soggette a diversi tipi di rumore. Il rumore è il risultato di errori nel processo di acquisizione delle immagini che si traducono in valori di pixel che non riflettono la vera intensità della scena reale.

- *Deblurring*
  Il *blurring* è una degradazione dell'immagine che può essere dovuta a diversi fattori, come i movimenti durante l'acquisizione della fotografia, l'out-of-focus optics e la distorsione causata dalla luce diffusa. Un'immagine sfocata può essere descritta dal seguente modello: g = Hf + n, dove g è l'immagine sfocata, H è l'operatore di distorsione, detto *point*

spread function (PSF), f è l'immagine in perfette condizioni e n è il rumore additivo. Lo scopo fondamentale del deblurring è deconvoluire l'immagine sfocata con la distorsione PSF che descrive perfettamente la distorsione introdotta.

3.1 Segmentazione semantica

La segmentazione semantica è la tecnica di elaborazione delle immagini utilizzata per la realizzazione della nostra rete neurale. Questa tecnica permette di comprendere completamente un'immagine e ciò è fondamentale nell'ambito dell'intelligenza artificiale; infatti, sempre più applicazioni si basano sulla totale conoscenza di immagini, come ad esempio la guida autonoma o la realtà aumentata. La segmentazione è un processo di partizionamento dell'immagine in regioni significative e il suo utilizzo può essere esteso anche a video e dati volumetrici. È un'operazione che classifica i pixel con caratteristiche comuni, in questo modo i pixel di una stessa regione risultano simili tra loro per una qualche proprietà mentre regioni vicine sono significativamente differenti per almeno una caratteristica. Lo scopo di questa tecnica è quello di semplificare o modificare la rappresentazione delle immagini per facilitarne l'analisi. Il risultato finale è un insieme di segmenti che, collettivamente, ricoprono l'intera immagine.

Modelli di deep learning come UNet, PSPNet e DeepLabV3+ hanno fornito risultati eccezionali per questo compito. L'architettura di UNet è divisa in due parti: percorso di contrazione e percorso espansivo. Il percorso di contrazione segue la struttura generica di una rete convoluzionale mentre il percorso espansivo subisce la

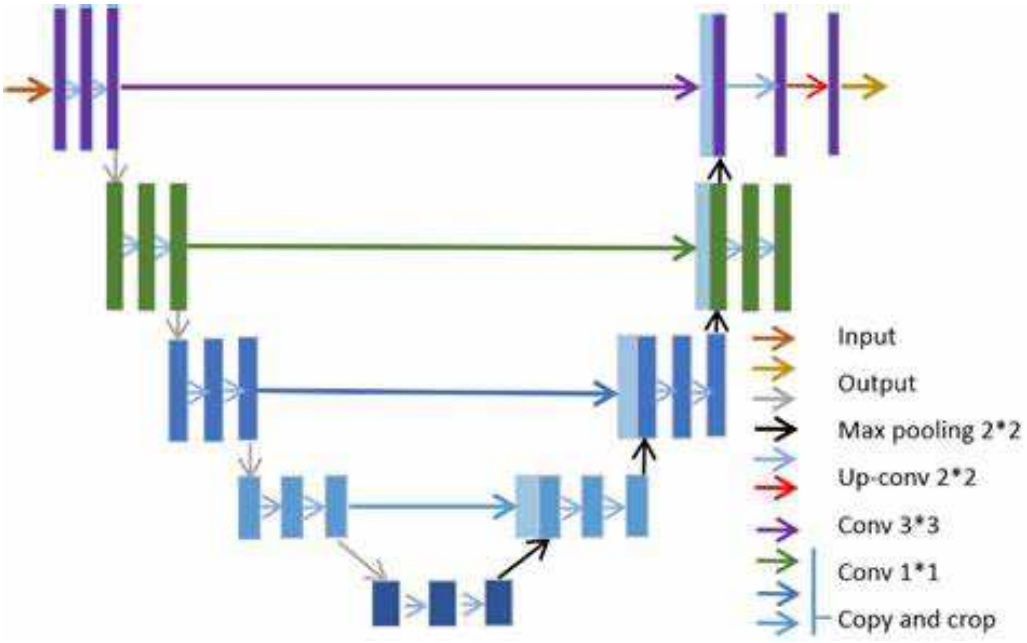
deconvoluzione per ricostruire l'immagine segmentata. PSPNet sfrutta la capacità delle informazioni contestuali globali utilizzando diverse aggregazioni di contesti basate su regioni introducendo un modulo di pooling piramidale insieme all'analisi della scena piramidale proposta. DeepLabV3+ è un perfezionamento di DeepLabV3 che utilizza la convoluzione atrosa. Essa è un potente strumento per regolare in modo esplicito il campo visivo del filtro e controllare la risoluzione delle risposte delle caratteristiche calcolate dalla rete neurale a convoluzione profonda.

4. Implementazione della rete neurale

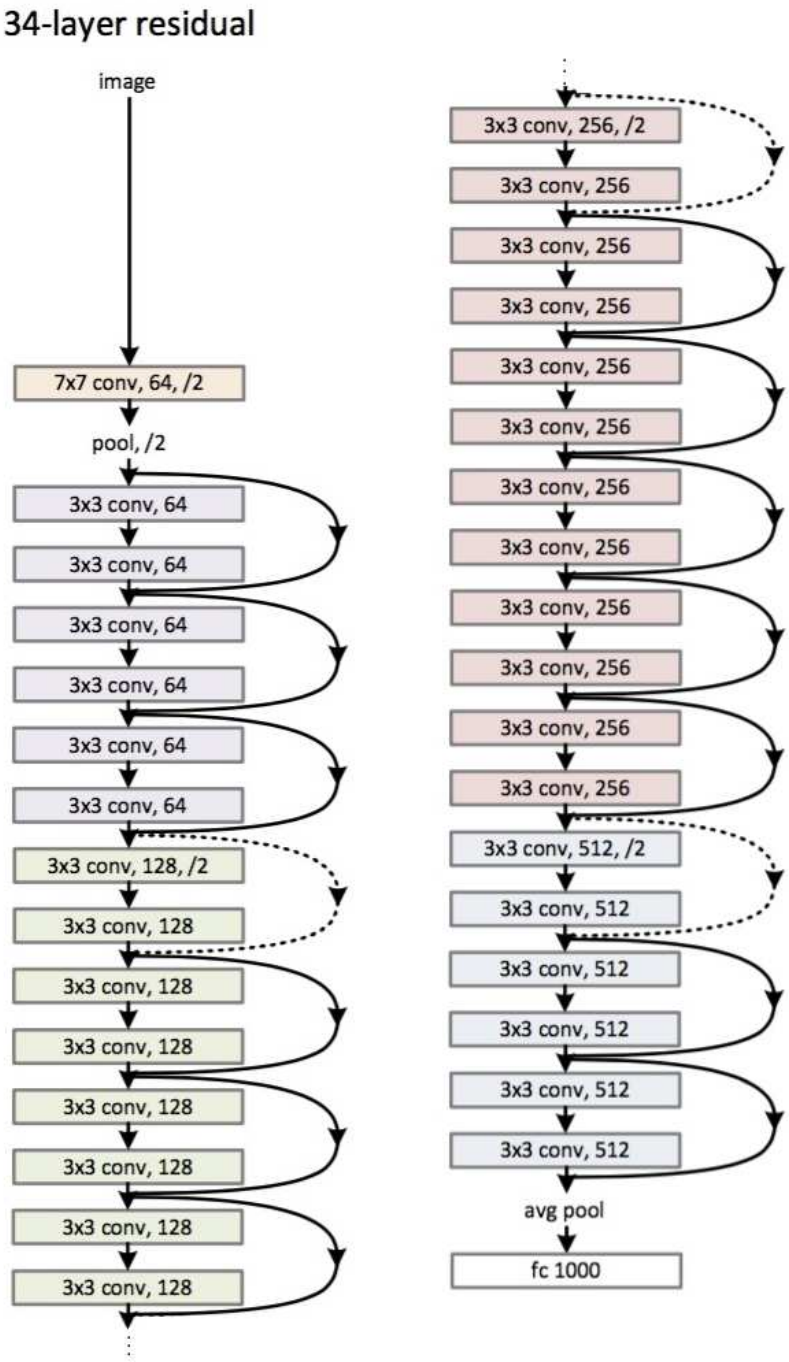
Il set di dati utilizzato per l'addestramento e per testare la rete neurale è FloodNet, il quale fornisce immagini di piccoli sistemi aerei senza equipaggio (UAV) ad alta risoluzione con annotazioni semantiche dettagliate relative ai danni. In particolare, i dati sono stati raccolti dopo l'uragano Harvey ed è costituito da video e immagini tratti da diversi voli effettuati tra il 30 agosto e il 4 settembre 2017 nella contea di Ford Bend in Texas e in altre aree direttamente interessate. Il set di dati è unico per due motivi: il primo è la fedeltà, in quanto contiene immagini di UAV scattate durante la fase di risposta dai soccorritori. Il secondo è che questo è l'unico database noto di immagini di UAV per i disastri. Tali immagini hanno una risoluzione molto elevata, il che le rende uniche rispetto ad altri set di dati per i disastri naturali.

Abbiamo utilizzato due modelli, UNet, una rete basata su codificatore-decodificatore, e DeepLabV3+, che impiega sia un modulo basato su decoder-encoder che un pooling piramidale. Entrambi sono stati addestrati in modo supervisionato, con la differenza che per UNet è stato scelto ResNet34 come backbone mentre per DeepLabV3+ si è preferito EfficientNet-B3. Per quanto riguarda l'architettura di ResNet34, per prima cosa si ha uno strato convoluzionale con 64 filtri e una dimensione del kernel di 7×7 e questa è la prima convoluzione, seguita da uno strato di pooling

massimo. Dopo il livello di pooling vi sono vari livelli di convoluzione, i quali sono normalmente raggruppati a coppie a causa del modo in cui i residui sono collegati (le frecce mostrano che saltano ogni due strati). Quindi sono presenti i due strati con una dimensione del kernel di 3×3 e 64 filtri e tutti questi sono ripetuti tre volte, sei strati in totale. Poi vi sono due livelli con una dimensione del kernel sempre di 3×3 ma con 128 filtri e anche questi vengono ripetuti ma per quattro volte, otto in totale. Successivamente vi sono sei coppie di strati con kernel 3×3 e 256 filtri e tre coppie di strati con kernel 3×3 e 512 filtri. Infine, sono presenti la average pooling e la funzione softmax. Di seguito viene riportate le rappresentazioni delle due reti; la prima è quella composta da UNet combinata con Resnet34.

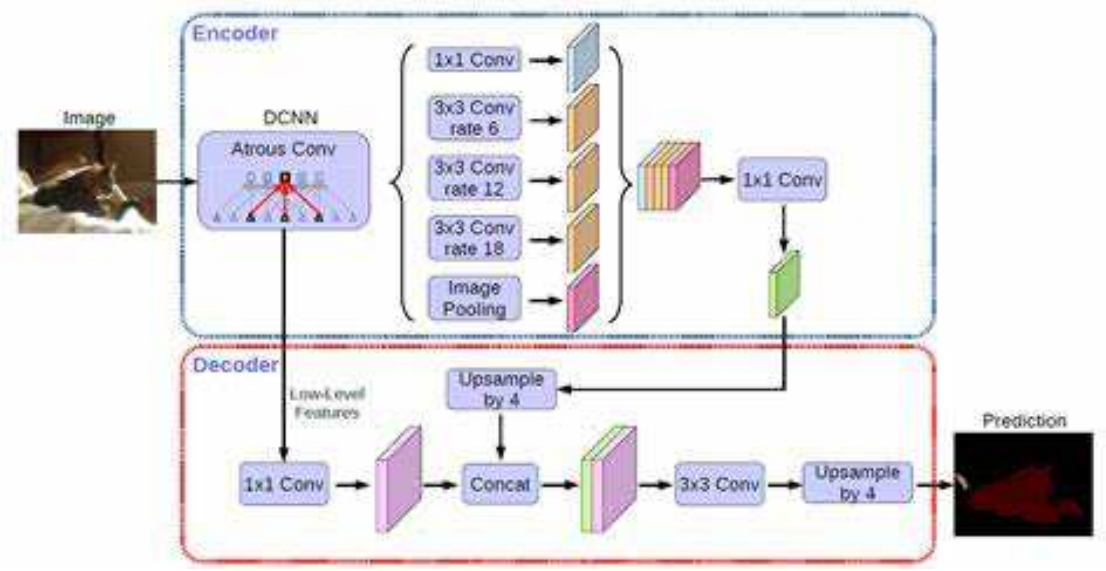


architettura UNet

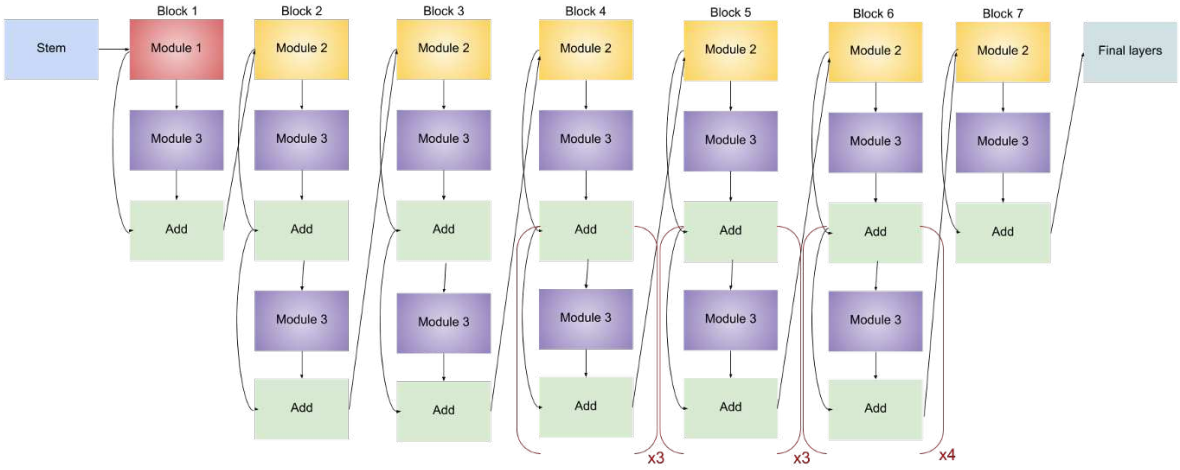


architettura Resnet34

La seconda rete è quella costituita da DeepLabV3+ e EfficientNet-B3, le cui architetture sono rappresentata di seguito:

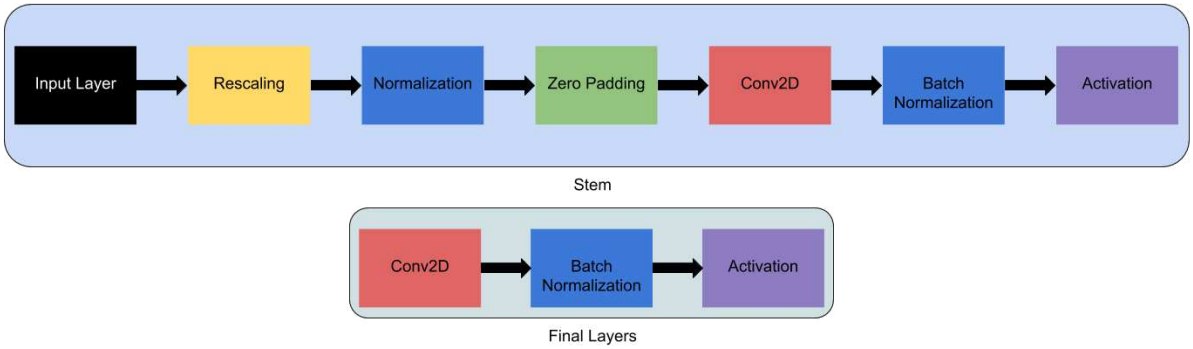


architettura DeepLabV3+

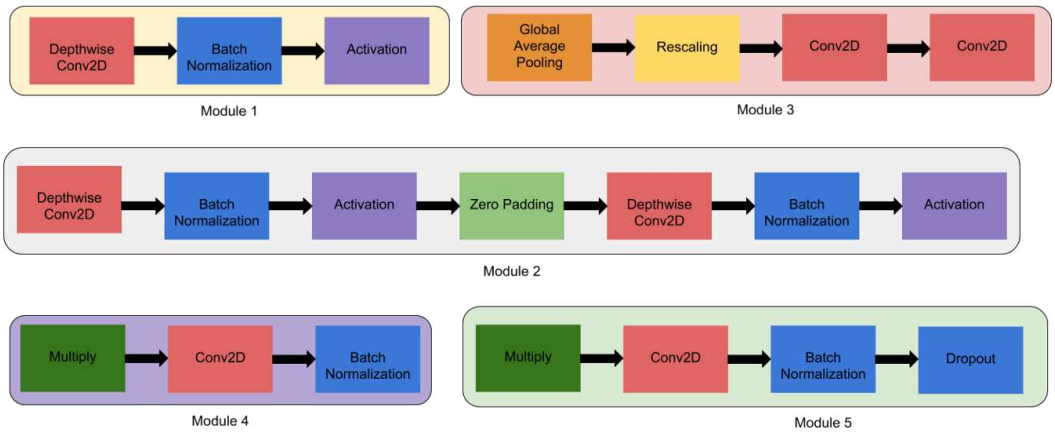


architettura Efficientnet-B3

Il primo blocco che costituisce la sua rete prende il nome di Stem ed è caratterizzato da sette strati uniti a tre stati terminali, i quali sono riportati qui sotto.

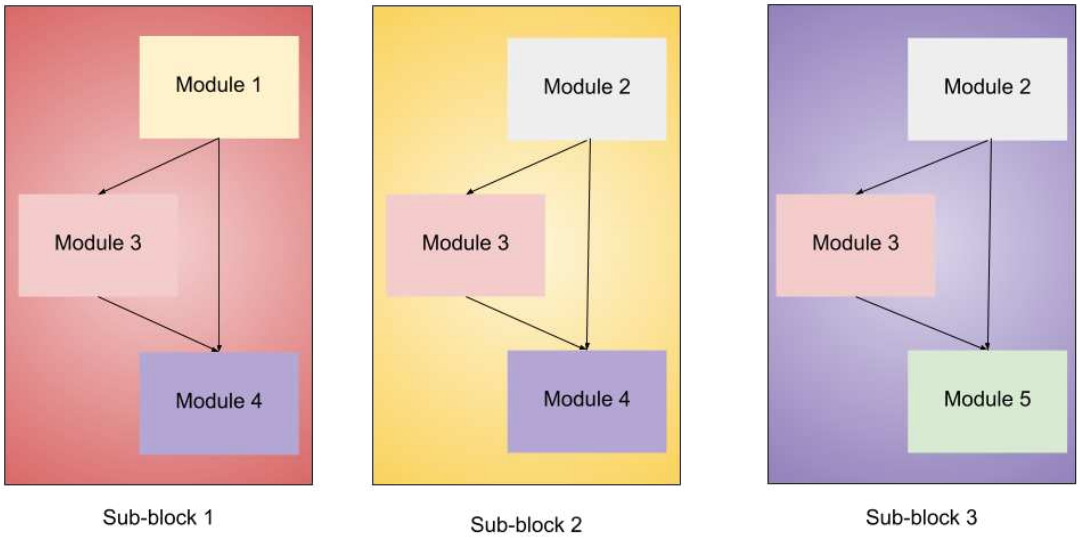


La parte centrale dell'architettura della rete è costituita da sette blocchi suddivisi in cinque moduli, ciascuno dei quali è caratterizzato da strati specifici.



- Il modulo 1 è usato come punto di partenza per gli altri sotto-blocchi.
- Il modulo 2 è usato come punto di partenza per il primo sotto-blocco di tutti i sette blocchi principali tranne il primo.
- Il modulo 3 è collegato come connessione skip a tutti i sotto-blocchi.
- Il modulo 4 viene utilizzato per combinare la connessione skip nei primi sotto-blocchi.
- Il modulo 5 serve per combinare ogni sotto-blocco con quello precedente, i quali sono connessi tramite una connessione skip.

Questi moduli sono ulteriormente combinati per formare sottoblocchi che verranno utilizzati in un certo modo nei blocchi.



- Il sotto-blocco 1 viene utilizzato solo come primo sotto-blocco nel primo blocco.
- Il sotto-blocco 2 viene utilizzato come primo sotto-blocco in tutti gli altri blocchi.
- Il sotto-blocco 3 viene utilizzato per qualsiasi sotto-blocco tranne il primo in tutti i blocchi.

Per UNet il tasso di apprendimento era 0,01 con lo scheduler LR a passi impostato a intervalli e il fattore di decadimento γ impostato a 0,1. Per DeepLabV3+ il tasso di apprendimento era 0,001 con lo scheduler LR della fase impostato a intervalli e γ impostato su 0,1. L'ottimizzatore Adam e la dimensione del batch di 8 sono stati utilizzati per tutti i modelli con Mean Intersection over Union (MIoU) come metrica di valutazione. Una volta riconosciuto il modello con le migliori prestazioni per l'attività, abbiamo addestrato un modello DeepLabV3+ con EfficientNet-B3 come spina dorsale e abbiamo utilizzato l'ottimizzatore SGD invece dell'ottimizzatore Adam in modo semi-supervisionato.

Di seguito viene riportato il codice della rete:

```
[4] !pip install segmentation-models-pytorch
```

```
################ IMPORT PACKAGES #############
%load_ext tensorboard
import os
import sys
import cv2
import random
import numpy as np
from pathlib import Path
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
from google.colab.patches import cv2_imshow
import sklearn
from sklearn import model_selection
from sklearn import metrics

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.utils.data import DataLoader
import torch.nn as nn
from collections import defaultdict
import torchvision
from torchvision import transforms
import torch.nn.functional as F
from tqdm.auto import tqdm
from torch.utils.tensorboard import SummaryWriter
import copy
import time

import segmentation_models_pytorch

import albumentations as A
from albumentations.pytorch import ToTensorV2
```

```
[6] ################ MOUNTING DRIVE #################
    from google.colab import drive
    drive.mount('/content/drive')
```

Data stats

```
################ CHECKING DATA ##################
dir_struct = """
FloodNet Challenge @ EARTHVISION 2021 - Track 1
|
|   class_mapping.csv
|
└─>Train
|   |
|   └─>Labeled
|   |   |
|   |   └─>Flooded
|   |   |   |
|   |   |   └─>image
|   |   |   |
|   |   |   └─>mask
|   |   |
|   |   └─>Non-Flooded
|   |       |
|   |       └─>image
|   |       |
|   |       └─>mask
|   |
|   └─>Unlabeled
|       |
|       └─>image
|
└─>Validation
    |
    └─>image
"""
```

```
[8] !ls
```

```
[9] %%capture
    !unzip -n /content/drive/MyDrive/512_Images/512_Images.zip
```

```
[10] os.makedirs("/content/512_Images/Train/Labeled/Flooded/image", exist_ok=True)
     os.makedirs("/content/512_Images/Train/Labeled/Non-Flooded/image", exist_ok=True)
     os.makedirs("/content/512_Images/Train/Labeled/Flooded/mask", exist_ok=True)
     os.makedirs("/content/512_Images/Train/Labeled/Non-Flooded/mask", exist_ok=True)
     os.makedirs("/content/512_Images/Train/Unlabeled/image", exist_ok=True)
     os.makedirs("/content/512_Images/Test/image", exist_ok=True)
```

```
[11] #!ls /content/drive/MyDrive/FloodNet\ Challenge\ @\ EARTHVISION\ 2021\ -\ Track\ 1/Train/Labeled
     #!ls /content/drive/MyDrive/Colab\ Notebooks/flood_detection/flood_segmentation/FloodNet\ Challenge\ @\ EARTHVISION\ 2021\ -\ Track\ 1/Train/Labeled
     !ls /content/drive/MyDrive/Train/Labeled
```

```
[12] !ls /content/512_Images/Train/Labeled/Flooded
```

```
#RESIZE=(512,512)
RESIZE=(256,256)
#temp_root = "/content/drive/MyDrive/FloodNet Challenge @ EARTHVISION 2021 - Track 1"
#temp_root = "/content/drive/MyDrive/Colab Notebooks/flood_detection/flood_segmentation/FloodNet Challenge @ EARTHVISION 2021 - Track 1"
temp_root = "/content/drive/MyDrive"
local_root = "/content/512_Images"
def resize_and_save(path, resize=RESIZE, samples='all'):
  if len(os.listdir(os.path.join(local_root, path))) == 0:
    print(f"{path} --> Saving...\n")
    if samples == 'all':
      samples = len(os.listdir(os.path.join(temp_root, path)))
    for img_name in tqdm(os.listdir(os.path.join(temp_root, path))[:samples]):
      img = cv2.imread(os.path.join(temp_root, path, img_name))
      img = cv2.resize(img, RESIZE)
      cv2.imwrite(os.path.join(local_root, path, img_name), img)
  else:
    print(f"{path} --> images are already saved")
```

```
[14] !ls
```

```
[16] resize_and_save("Train/Labeled/Flooded/image")
     resize_and_save("Train/Labeled/Non-Flooded/image")
     resize_and_save("Train/Labeled/Flooded/mask")
     resize_and_save("Train/Labeled/Non-Flooded/mask")
     resize_and_save("Train/Unlabeled/image")
     resize_and_save("Test/image")
     resize_and_save("Train/Unlabeled/image", samples=100)
```

```
flood_dir = "Train/Labeled/Flooded"
non_flood_dir = "Train/Labeled/Non-Flooded"

f_img_dir = os.path.join(local_root, flood_dir, "image")
f_mask_dir = os.path.join(local_root, flood_dir, "mask")
n_img_dir = os.path.join(local_root, non_flood_dir, "image")
n_mask_dir = os.path.join(local_root, non_flood_dir, "mask")

f_img = len(os.listdir(f_img_dir))
f_mask = len(os.listdir(f_mask_dir))
n_img = len(os.listdir(n_img_dir))
n_mask = len(os.listdir(n_mask_dir))
print(f"Flooded images:      {f_img} Flooded masks:     {f_mask}")
print(f"Non-Flooded images: {n_img} Non-Flooded masks: {n_mask}")
```

We have a imbalance with Flooded and Non-Flooded images

```
sample_f_img = cv2.imread(os.path.join(f_img_dir, sorted(os.listdir(f_img_dir))[0]))
sample_f_mask = cv2.imread(os.path.join(f_mask_dir, sorted(os.listdir(f_mask_dir))[0]))
sample_n_img = cv2.imread(os.path.join(n_img_dir, sorted(os.listdir(n_img_dir))[0]))
sample_n_mask = cv2.imread(os.path.join(n_mask_dir, sorted(os.listdir(n_mask_dir))[0]))

print(np.min(sample_f_mask),
np.max(sample_f_mask),
np.min(sample_n_mask),
np.max(sample_n_mask))

sample_f_mask = sample_f_mask * (255/9)
sample_n_mask = sample_n_mask * (255/9)

print(np.min(sample_f_mask),
np.max(sample_f_mask),
np.min(sample_n_mask),
np.max(sample_n_mask))

cv2_imshow(cv2.resize(sample_f_img, dsize=(300, 300)))
cv2_imshow(cv2.resize(sample_f_mask, dsize=(300, 300)))
cv2_imshow(cv2.resize(sample_n_img, dsize=(300, 300)))
cv2_imshow(cv2.resize(sample_n_mask, dsize=(300, 300)))
```

Why are the masks black?
Because the values are mapped 0 --> 9 for 10 classes

```
[19] kernel = np.ones((2,2),np.uint8)
     cv2_imshow(cv2.erode(cv2.dilate(cv2.bilateralFilter(sample_f_img, 5, 75, 75), kernel, iterations=2), kernel, iterations=1))
```

Configuration

```
[20] CLASSES={'Background':0,'Building-flooded':1,'Building-non-flooded':2,'Road-flooded':3,'Road-non-flooded':4,
             'Water':5,'Tree':6,'Vehicle':7,'Pool':8,'Grass':9}
     IMG_DIM= 256
```

```
# CONFIG
#TENSORBOARD_DIR = "/content/drive/MyDrive/Colab Notebooks/flood_detection/flood_segmentation/runs/"
TENSORBOARD_DIR = "/content/drive/MyDrive/runs/"
#MODEL_DIR = "/content/drive/MyDrive/Colab Notebooks/flood_detection/flood_segmentation/models/"
MODEL_DIR = "/content/drive/MyDrive/models/"
UNLABELLED_SPLIT = 100 # first 100 unlabelled examples will be used
## RAM storage error
os.makedirs(TENSORBOARD_DIR, exist_ok=True)
os.makedirs(MODEL_DIR, exist_ok=True)

#### UNET CONFIG #########
ENCODER_DEPTH = 5
DECODER_CHANNELS = (256, 128, 64, 32, 16)
BATCH_SIZE = [8, 16, 32, 64, 128]
LR = [1, 1e-2, 1e-4, 1e-6]
BATCH_SIZE = [8]
LR = [1e-2]
EPOCHS = 50
ENCODER_NAME = 'resnet34'

####### DEEPLAB V3+ CONFIG #######
#ENCODER_DEPTH = 5
#DECODER_CHANNELS = 256
#BATCH_SIZE = [8]
#EPOCHS = 50
#LR = [1e-3]
#ENCODER_NAME = 'efficientnet-b3'
```

Dataloader

```
dirs = """
flood_dir
non_flood_dir

f_img_dir
f_mask_dir
n_img_dir
n_mask_dir
"""
x_f = [os.path.join(f_img_dir, file) for file in sorted(os.listdir(f_img_dir))]
x_n = [os.path.join(n_img_dir, file) for file in sorted(os.listdir(n_img_dir))]

x = x_f + x_n

x_train, x_test = model_selection.train_test_split(x, test_size=0.3, shuffle=True)

train_transform1 = A.Compose([
        A.Resize(IMG_DIM, IMG_DIM),
        A.ShiftScaleRotate(shift_limit=0.2, scale_limit=0.2, rotate_limit=30, p=0.5),
        # A.RGBShift(r_shift_limit=25, g_shift_limit=25, b_shift_limit=25, p=0.5),
        A.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, p=0.5),
        A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
    ])

train_transform2=A.Compose([ToTensorV2()])

val_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

```
class SegDataset:
    def __init__(self, x_paths, trans1,trans2, img_dim, unlabelled=False):
        self.x_paths = x_paths
        self.unlabelled = unlabelled
        if not self.unlabelled:
          self.y_paths = [x.replace("image", "mask").replace(".jpg", "_lab.png") for x in self.x_paths]
        else:
          self.y_paths = None
        self.img_dim = img_dim
        self.trans1=trans1
        self.trans2=trans2

    def __len__(self):
      return len(self.x_paths)

    def get_newMask(self, mask, classes, dim=IMG_DIM):
      mask = torch.as_tensor(mask[:,:,0], dtype=torch.int64)
      return torch.moveaxis(torch.nn.functional.one_hot(mask, num_classes=len(classes)), -1, 0)

    def __getitem__(self, index):
      image = cv2.imread(self.x_paths[index])

      kernel = np.ones((2,2),np.uint8)
      image = cv2.bilateralFilter(image, 5, 75, 75)
      image = cv2.erode(cv2.dilate(image, kernel, iterations=2), kernel, iterations=1)
      if not self.unlabelled:
        mask = cv2.imread(self.y_paths[index])
      else:
        mask = np.random.rand(512, 512, 3)
      if self.trans1:
        transformed1 = self.trans1(image=image, mask=mask)
        image = transformed1["image"]
        mask = self.get_newMask(transformed1["mask"],CLASSES)
      if self.trans2:
        transformed2 = self.trans2(image=image, mask=mask)
        image = transformed2["image"]
      return image, mask
```

Functions

```
[25] def dice_loss(pred, target, smooth = 1e-5):
        pred = pred.contiguous()
        target = target.contiguous()

        intersection = (pred * target).sum(dim=2).sum(dim=2)

        loss = (1 - ((2. * intersection + smooth) / (pred.sum(dim=2).sum(dim=2) + target.sum(dim=2).sum(dim=2) + smooth)))

        return loss.mean()
```

```
def calc_loss(pred, target, metrics, bce_weight=0.5, unlabelled=False):
    bce = F.binary_cross_entropy_with_logits(pred, target.to(torch.float32))
    pred = F.sigmoid(pred)
    dice = dice_loss(pred, target)

    target_np=target.data.cpu().numpy()
    pred_np=pred.data.cpu().numpy()
    MIoU= np.mean(sklearn.metrics.jaccard_score(np.argmax(target_np,axis=1).flatten(), np.argmax(pred_np,axis=1).flatten(), average=None))
    loss = bce * bce_weight + dice * (1 - bce_weight)
    if not unlabelled:
      metrics['bce'] += bce.data.cpu().numpy() * target.size(0)
      metrics['dice'] += dice.data.cpu().numpy() * target.size(0)
      metrics['loss'] += loss.data.cpu().numpy() * target.size(0)
      metrics['MIoU'] += MIoU * target.size(0)

    return loss
```

```
[27] def print_metrics(metrics, epoch_samples, phase):
        outputs = []
        for k in metrics.keys():
            outputs.append("{}: {:4f}".format(k, metrics[k] / epoch_samples))
        print("{}: {}".format(phase, ", ".join(outputs)))
```

Train function

```
[28] training_set = SegDataset(x_train,  train_transform1, train_transform2, img_dim=IMG_DIM)
     testing_set = SegDataset(x_test, train_transform1,train_transform2, img_dim=IMG_DIM)
     image_datasets = {'train': training_set, 'valid': testing_set}
```

```
[29] u_dir = "/content/512_Images/Train/Unlabeled/image"
     unlabelled_paths = [os.path.join(u_dir, file) for file in os.listdir(u_dir)]
     unlabelled_set = SegDataset(unlabelled_paths[0:-500], trans1=train_transform1,trans2=train_transform2, img_dim=IMG_DIM, unlabelled=True)
     image_datasets["unlabelled"] = unlabelled_set
```

```
def train_model(unique_name, num_epochs=EPOCHS, start_alpha_from=15, reach_max_alpha_in=655, max_alpha=0.5, retrain=False):
  for lr in LR:
    for bs in BATCH_SIZE:
       #if(lr == 1 and (bs == 8 or bs == 16)):
        # continue
      print("__"*80)
      print("__"*80)
      print(f"name: {unique_name} LR: {lr} BS: {bs}")
      print("__"*80)
      print("__"*80)

      alphas = np.linspace(0, max_alpha, reach_max_alpha_in-start_alpha_from)

      os.makedirs(os.path.join(TENSORBOARD_DIR, f'{unique_name}-{lr}-{bs}'), exist_ok=True)
      writer = SummaryWriter(log_dir=os.path.join(TENSORBOARD_DIR, f'{unique_name}-{lr}-{bs}'))

      best_loss = 1e10
      best_epoch = 0
      best_miou = 0

      train_data_loader = torch.utils.data.DataLoader(training_set, batch_size=bs, num_workers=0)
      test_data_loader = torch.utils.data.DataLoader(testing_set, batch_size=bs, num_workers=0)
      ulbl_data_loader = torch.utils.data.DataLoader(unlabelled_set, batch_size=bs, num_workers=0)
      dataloaders = {'train': train_data_loader,'valid': test_data_loader, "unlabelled" : ulbl_data_loader}
```

```
      model = segmentation_models_pytorch.Unet(encoder_name=ENCODER_NAME, encoder_depth=ENCODER_DEPTH,
                                               decoder_channels=DECODER_CHANNELS, classes=len(CLASSES))
      #model = segmentation_models_pytorch.DeepLabV3Plus(encoder_name=ENCODER_NAME, encoder_depth=ENCODER_DEPTH,
       #                                        decoder_channels=DECODER_CHANNELS, classes=len(CLASSES))
      model = model.to(device)
      if retrain == True: #MOD
         model.load_state_dict(torch.load(os.path.join(MODEL_DIR, "abhi_sudo_full_2-pretrained_preproc_1-DeepLabv3+-ep_20-0.001-8.pt"))) #MOD
      best_model_wts = copy.deepcopy(model.state_dict())

      optimizer = optim.Adam(model.parameters(), lr=lr) ## IF YOU CHANGE THIS, CHANGE THE UNIQUE ABOVE
      #optimizer = optim.SGD(model.parameters(), lr=0.01)
      scheduler = lr_scheduler.MultiStepLR(optimizer, milestones=[7, 20], gamma=0.1)
      #scheduler = None

      for epoch in range(num_epochs):
          print('__' * 80)
          print('Epoch {}/{}'.format(epoch, num_epochs - 1))
          print('__' * 80)

          if epoch < start_alpha_from:
            alpha = 0
          elif epoch-start_alpha_from >= len(alphas):
            alpha = alphas[-1]
          else:
            alpha = alphas[max(0,epoch-start_alpha_from)]

          since = time.time()

          for phase in ['train', 'unlabelled', 'valid']:
              print(phase)
              print("__"*20)
              if alpha == 0 and phase == 'unlabelled':
                continue
              if phase in ['train', 'unlabelled']:
                model.train()
              else:
                model.eval()
```

```
              metrics = defaultdict(float)
              epoch_samples = 0

              for batch_no, (inputs, labels) in enumerate(tqdm(dataloaders[phase])):
                  inputs = inputs.to(device)
                  if phase in ['train', 'valid']:
                    labels = labels.to(device)

                  # zero the parameter gradients
                  optimizer.zero_grad()

                  # forward
                  with torch.set_grad_enabled(phase in ['train', 'unlabelled']):
                      outputs = model(inputs)
                      # loss = calc_loss(outputs, labels, metrics)

                      if phase in ["train", "valid"]:
                        loss = calc_loss(outputs, labels, metrics)
                      else:
                        loss = alpha * calc_loss(outputs, pseudo_labels[batch_no].to(device), metrics, unlabelled=True)
                      # backward + optimize only if in training phase
                      if phase in ['train', 'unlabelled']:
                          loss.backward()
                          optimizer.step()

                  # statistics
                  epoch_samples += inputs.size(0)

              if scheduler is not None and phase == "train":
                scheduler.step()

              print_metrics(metrics, epoch_samples, phase)
              epoch_loss = metrics['loss'] / epoch_samples
              epoch_miou = metrics['MIoU'] / epoch_samples
```

```
            ## tensorboard writer
            if phase == "train":
              writer.add_scalar(f'Loss/{phase}', epoch_loss, epoch)
              writer.add_scalar(f'MIoU/{phase}', epoch_miou, epoch)
              writer.add_scalar(f'Alpha/{phase}', alpha, epoch)
              # writer.add_scalars(f'Loss/{phase}', {'loss':epoch_loss, 'alpha':alpha}, epoch)
              # writer.add_scalars(f'MIoU/{phase}', {'miou':epoch_miou, 'alpha':alpha}, epoch)

            if phase == "valid": # older implementation had phase == 'val'
              writer.add_scalar(f'Loss/val', epoch_loss, epoch)
              writer.add_scalar(f'MIoU/val', epoch_miou, epoch)
              writer.add_scalar(f'Alpha/val', alpha, epoch)
              # writer.add_scalars(f'Loss/val', {'loss':epoch_loss, 'alpha':alpha}, epoch)
              # writer.add_scalars(f'MIoU/val', {'miou':epoch_miou, 'alpha':alpha}, epoch)

            ## generate pseudo labels
            if phase == 'train' and epoch >= start_alpha_from-1:
              pseudo_labels = []
              model.eval()
              for inputs, _ in tqdm(dataloaders['unlabelled'], desc="Predicting pseudo labels"):
                inputs = inputs.to(device)
                with torch.no_grad():
                  outputs = model(inputs)
                  pseudo_labels.append(outputs.detach().cpu())

            # deep copy the model
            if phase == 'valid' and epoch_miou > best_miou:
                best_miou = epoch_miou
                best_epoch = epoch
                best_model_wts = copy.deepcopy(model.state_dict())
                print(f'Best miou: {best_miou:.4f} Epoch: {epoch}')

            if epoch % 5 == 0:
                PATH = os.path.join(MODEL_DIR, f'{unique_name}-ep_{best_epoch}-{lr}-{bs}.pt')
                torch.save(best_model_wts, PATH)
                PATH = os.path.join(MODEL_DIR, f'{unique_name}-ep_{epoch}-{lr}-{bs}.pt')
                torch.save(model.state_dict(), PATH)
```

```
[ ]
            time_elapsed = time.time() - since
            print('{:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))

        print('Best val loss: {:4f}'.format(best_loss))
        writer.close()
        # load best model weights
        model.load_state_dict(best_model_wts)
        PATH = os.path.join(MODEL_DIR, f'{unique_name}-ep_{best_epoch}-{lr}-{bs}.pt')
        torch.save(best_model_wts, PATH)

    return model
```

Training

```
[ ] device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    print(device)
    os.chdir("/content/")
```

```
[ ] torch.cuda.empty_cache()
```

```
[ ] unique_name = "abhi_sudo_full_2-pretrained_preproc_2-Unet"
```

```
[ ] model = train_model(unique_name, num_epochs=EPOCHS, retrain=False)
```

```
%ls /content/drive/MyDrive/models/
```

```
[ ] print(model)
```

```
[ ] # %rm -r /content/drive/MyDrive/runs
```

```
[ ] !ls /content/drive/MyDrive/runs
    #!ls /content/drive/MyDrive/Colab\ Notebooks/flood_detection/flood_segmentation/runs
```

```
!ls /content/drive/MyDrive/models/abhi_sudo_full*
#ls /content/drive/MyDrive/Colab\ Notebooks/flood_detection/flood_segmentation/models
```

```
[ ] # %rm -r /content/drive/MyDrive/runs/abhi_sudo-pretrained_preproc_1-deeplabv3+-0.001-8/
```

```
[ ] %tensorboard --logdir=/content/drive/MyDrive/runs # TENSORBOARD_DIR
```

Saving Masks

Config

```
[ ] MODEL_LOAD_PATH = f'/content/drive/MyDrive/models/abhi_sudo_full_2-pretrained_preproc_1-Deeplabv3+-ep_10-0.001-8.pt'
    #DIR_PATH = f'/content/drive/MyDrive/FloodNet Challenge @ EARTHVISION 2021 - Track 1/Test/image'
    DIR_PATH = f'/content/drive/MyDrive/Test/image'
    SAVE_PATH = f'/content/predictions'

    os.makedirs(SAVE_PATH, exist_ok=True)

    LOAD_SIZE = 8 # pred generating load size
```

Code

```
[ ] # model = segmentation_models_pytorch.Unet(encoder_name=ENCODER_NAME, encoder_depth=ENCODER_DEPTH,
    #                                            decoder_channels=DECODER_CHANNELS, classes=len(CLASSES))

    model = segmentation_models_pytorch.DeepLabV3Plus(encoder_name=ENCODER_NAME, encoder_depth=ENCODER_DEPTH,
                                                      decoder_channels=DECODER_CHANNELS, classes=len(CLASSES))
    model = model.to(device)
    model.load_state_dict(torch.load(MODEL_LOAD_PATH))
    # model.eval()
```

```
    all_images=sorted(os.listdir(DIR_PATH))
    all_masks=[x.replace(".jpg", ".png") for x in all_images]

    image_paths=[os.path.join(DIR_PATH, file) for file in sorted(os.listdir(DIR_PATH))]
    mask_paths=[os.path.join(SAVE_PATH, file) for file in all_masks]
    print(len(image_paths))
    print(mask_paths)

    val_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ])
```

```
[ ]  def reverse_transform_mask(inp):
        inp=inp.transpose((1, 2, 0))
        t_mask=np.argmax(inp,axis=2).astype('float32')
        t_mask=cv2.resize(t_mask, dsize=(4000, 3000))
        kernel = np.ones((3,3),np.uint8)
        t_mask = cv2.erode(t_mask, kernel, iterations=1)
        return t_mask
```

```
[ ] class ValDataset:
        def __init__(self, img_paths, val_trans, img_dim):
            self.img_paths = img_paths
            self.img_dim = img_dim
            self.val_trans=val_trans

        def __len__(self):
          return len(self.img_paths)

        def __getitem__(self, index):
          image = cv2.resize(cv2.imread(self.img_paths[index]), dsize=(self.img_dim, self.img_dim))

          kernel = np.ones((2,2),np.uint8)
          image = cv2.bilateralFilter(image, 5, 75, 75)
          image = cv2.erode(cv2.dilate(image, kernel, iterations=2), kernel, iterations=1)
```

```
[ ]       if self.val_trans:
            image=self.val_trans(image)
          return image, index
```

```
[ ] val_set = ValDataset(image_paths, val_transform , img_dim=IMG_DIM)
    val_dataloader = torch.utils.data.DataLoader(val_set, batch_size=LOAD_SIZE, shuffle=False)
```

```
[ ] torch.cuda.empty_cache()
```

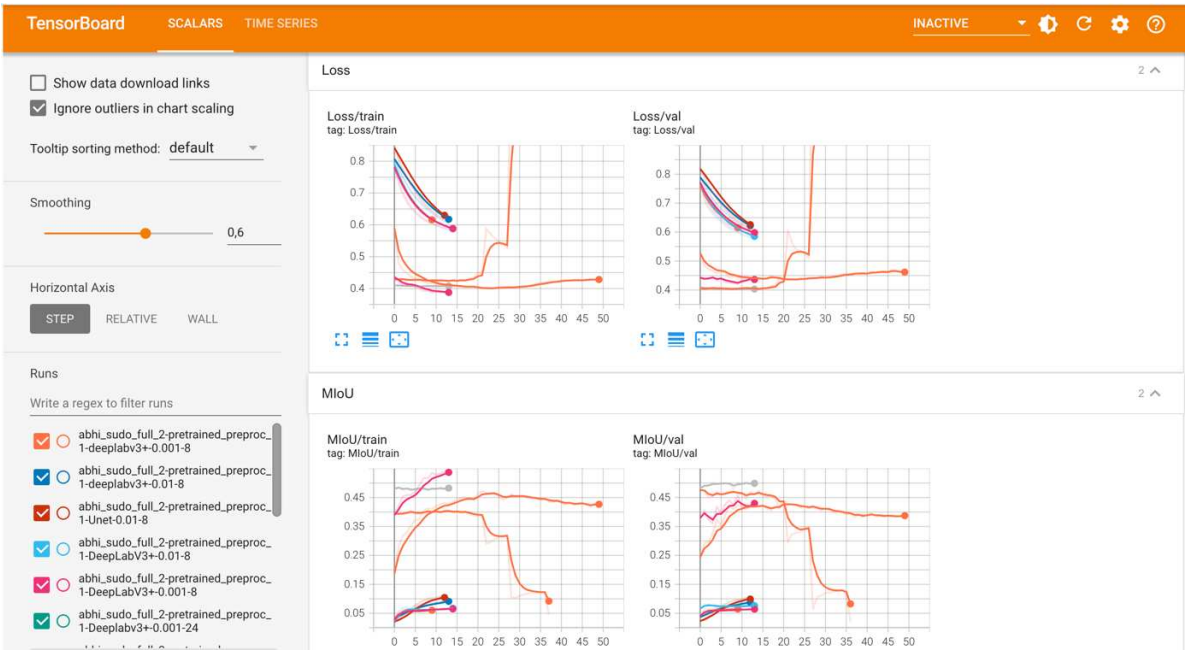
```
    model.eval()

    for inputs, index in tqdm(val_dataloader):
      inputs = inputs.to("cuda")
      with torch.no_grad():
        preds = model(inputs)
        preds = F.sigmoid(preds)
        preds = preds.data.cpu().numpy()
      for i in range(LOAD_SIZE):
        f_mask=reverse_transform_mask(preds[i])
        cv2.imwrite(mask_paths[index[i]],f_mask)
```

```
[ ] !zip -r sudo_full-pretrained_preproc_2_8.zip predictions
    !cp sudo_full-pretrained_preproc_2_8.zip /content/drive/MyDrive/512_Images
```

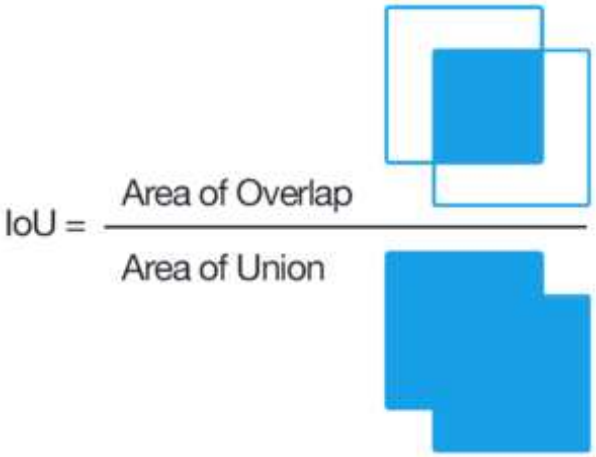
```
[ ] id = 6488
    plt.figure()
    plt.imshow(cv2.imread(f'predictions/{id}.png'))
    print(cv2.imread(f'predictions/{id}.png').shape)
    plt.figure()
    #plt.imshow(cv2.imread(f'/content/drive/MyDrive/FloodNet Challenge @ EARTHVISION 2021 - Track 1/Test/image/{id}.jpg'))
    plt.imshow(cv2.imread(f'/content/drive/MyDrive/Test/image/{id}.jpg'))
```

5. Conclusioni e sviluppi futuri

Sono stati effettuati diversi training e test della rete sia per il modello UNet che per DeepLabV3+ e i risultati sono stati confrontati grazie all'utilizzo di una TensorBoard.



Come affermato nel capitolo precedente, la metrica di valutazione per la qualità della rete è la MIoU, ovvero Mean Intersection over Union. La Intersection over Union (IoU), nota anche come indice di Jaccard, è una delle metriche più comunemente utilizzate nella segmentazione semantica, in quanto è semplice e molto efficace. La IoU è il rapporto tra l'area di overlap fra la segmentazione prevista e l'immagine reale e l'area di unione fra la segmentazione prevista e l'immagine reale.



Per la segmentazione semantica con più classi la MIoU si calcola facendo la media dell'IoU di ogni classe.

Basandoci su questo parametro i risultati ottenuti sono i seguenti:

Method	Backbone	MIoU
UNet	Resnet34	0.21
DeepLabV3+	EfficientNet-B3	0.50

La rete realizzata con il modello UNet ha portato a un risultato abbastanza deludente in quanto si è ottenuto un MIoU di appena 0.21, ovvero il 21%. Al contrario, il modello ha dato un esito positivo con un MIoU di 0.50, ovvero del 50%.

I risultati ottenuti sono accettabili per una prima sperimentazione ma non sufficientemente solidi per un'immediata applicazione pratica. In futuro saranno necessarie nuove sperimentazioni al fine di ottenere dei dati che consentano l'applicazione della rete su dei microcontrollori.

6. Bibliografia e sitografia

- Effetti del cambiamento climatico, https://unric.org/it/effetti-del-cambiamento-climatico/
- Reti neurali, https://www.ibm.com/it-it/topics/neural-networks
- Siddharth M. Building Resnet-34 model using Pytorch – A Guide for Beginners, https://www.analyticsvidhya.com/blog/2021/09/building-resnet-34-model-using-pytorch-a-guide-for-beginners/
- Vardan Agarwal. Complete architectural details of all EfficientNet models. 2020 https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142
- Ekin Tiu. Metrics to evaluate your semantic segmentation model. 2019 https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2
- Garcia-Garcia, A.; Orts-Escolano, S.; Oprea, S.; Villena-Martinez, V.; Martinez-Gonzalez, P.; Garcia-Rodriguez, J. A survey on deep learning techniques for image and video semantic segmentation. Appl. Soft Comput. 2018.
- Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff and Hartwig Adam. Encoder-decoder with atrous

separable convolution for semantic image segmentation. In Proceedings of the European Conference on Computer Vision (ECCV), September 2018.

- Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, pages 234–241, Cham, 2015. Springer International Publishing.
- Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.
- Sahil Khose, Abhiraj Tiwari, Ankita Ghosh. Semi-Supervised Classification and Segmentation on High Resolution Aerial Images. https://arxiv.org/abs/2105.08655 https://github.com/sahilkhose/FloodNet/blob/main/FloodNet_T2.ipynb Manipal Institute of Technology, Manipal, 2021.
- Dispense di lezione del corso “Sistemi Embedded”, Prof. Claudio Turchetti, Corso di Laurea Magistrale in Ingegneria Elettronica, Università Politecnica delle Marche.