UNIVERSITÀ
DI PARMA

DIPARTIMENTO DI SCIENZE
MATEMATICHE, FISICHE
E INFORMATICHE
Corso di Laurea in Informatica

Predizione di zone asciutte e bagnate in una alluvione con reti neurali

Neural networks for the prediction of Wet/Dry areas in floodings

Relatore:
Chiar.mo Prof.
Alessandro Dal Palù

Correlatore:
Prof. Ing. Renato Vacondio

Laureando:
Paterniti Barbino Niko
287296

Anno Accademico 2019/2020

Ai miei genitori,
a mio fratello,
ai miei nonni,
e ai miei amici.

Indice

Introduzione

Alla base di questo studio vi è la predizione di zone asciutte e bagnate in una alluvione con reti neurali.
L'obiettivo di questa tesi di laurea è quello di utilizzare una rete neurale feed forward per l'analisi delle mappe asciutto/bagnato relative al fiume considerato.
Il presente lavoro si articola in 5 sezioni:

- Nella prima si pone l'enfasi sul concetto di rete neurale al fine di chiarire lo strumento principale che andremo ad utilizzare descrivendone l'importanza e l'utlità nell'ambito dei disastri naturali. Viene inoltre descritto l'utilizzo del cluster di ateneo.
- Nella seconda ci si concentra sull'obiettivo della tesi
- Nella terza si analizza la struttura della rete neurale
- Nella quarta vengono mostrati e analizzati i risultati ottenuti. con relative immagini e grafici.
- Nella quinta e ultima sezione verranno infine descritte possibili ipotesi migliorative.

Capitolo 1

Background

In questo capitolo verrà approfondito il concetto di rete neurale al fine di fornire un background al lettore relativamente al tipo di tecnologia che rappresenta e all'importante utilizzo che rappresenta entro il concetto di disastri naturali.
Verrà inoltre fornito un background relativamente all'utilizzo del cluster di ateneo.

1.1 Reti neurali

Una **rete neurale artificiale** (in inglese *artificial neural network*) è un modello matematico composto di "neuroni" artificiali, che si isipira a una *rete neurale biologica*.
Le reti neurali artificiali(ANN) sono una metodologia utilizzata per risolvere problemi di natura complessa non facilmente codificabili, e sono una colonna portante del machine learning come viene inteso oggi.
Possiamo considerare una rete neurale come una scatola nera, con degli input, degli strati intermedi in cui "succedono le cose" e degli output che costituiscono il risultato finale.
La rete neurale è composta da "unità" chiamate **neuroni**, arrangiati in strati successivi, ciascun neurone è tipicamente collegato a tutti i neuroni dello

strato successivo tramite *connsessioni pesate*, una connessione non è altro che un valore numerico(il peso) che viene moltiplicato per il valore del neurone collegato.
Ciascun neurone somma i valori pesati di tutti i neuroni ad esso collegati e aggiunge un valore di **bias**, a questo risultato viene applicata una **funzione di attivazione**, che non fa altro che trasformare matematicamente il valore prima di passarlo allo strato successivo.
In questo modo i valori di input vengono propagati attraverso la rete fino ai neuroni di output.

1.1.1 Il cervello umano

Le reti neurali artificiali(ANN) sono chiamate "reti neurali" in quanto il comportamento dei nodi che le compongono ricorda vagamente quello dei neuroni biologici.
Il **neurone** è la piu' piccola unità funzionale del tessuto nervoso, che insieme al tessuto della **neuroglia** e al tessuto vascolare, formano il sistema nervoso. Grazie alle sue caratteristiche fisiologiche e chimiche, ogni neurone permette di ricevere,integrare e trasmettere impulsi nervosi, nonché di produrre sostanze denominate neuro-trasmettitori.
Unitamente ad altri neuroni facenti parte della stessa regione cerebrale, esso consente la messa in atto di una serie di **funzioni cognitive e comportamentali**, come pensare,camminare,parlare,ecc.
I neuroni si trovano all'interno del sistema nervoso e comunicano tra loro tramite collegamenti intercellulari chiamati **sinapsi**. La comunicazione sinaptica avviene attraverso sostanze chimiche dette **neurotrasmettitori**, che stimolano, tramite il passaggio dell'impulso nervoso, la cellula successiva.
I neuroni sono polarizzati in quanto all'esterno della membrana cellulare presentano una carica elettrica differente da quella all'interno.Questa differenza di carica è determinata dalla percentuale diversa di ioni sodio e potassio tra interno ed esterno:con l'ausilio della pompa sodio-potassio è possibile tra-

smettere l'informazione all'interno e all'esterno.
Nel momento in cui si ha il passaggio dell'informazione da una cellula all'altra, si ha **l'impulso nervoso**:una volta raggiunta la sinapsi, le vescicole presenti sulla superficie liberano un neurotrasmettitore che si diffonde rapidamente tramite la fibra post-sinaptica e si lega ad alcune molecole della membrana della cellula recettrire.La reazione con il neurotrasmettitore cambia la permeabilità della membrana della fibra post-sinaptica, dando origine ad un potenziale d'azione che permetterà a un ulteriore impulso nervoso di continuare a propagarsi.
Quindi unn neurone riceve in ingresso segnali da vari altri neuroni tramite connessioni sinaptiche e li integra.Se l'attivazione che ne risulta supera una certa soglia genera un **Potenziale d'Azione** che si propaga attraverso il suo assone a uno o piu' neuroni.

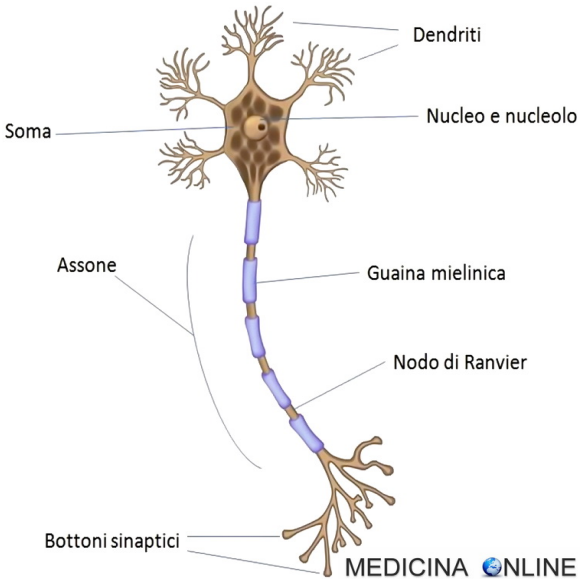

Figura 1.1: La struttura del neurone

1.1.2 Il neurone artificiale

Una rete neurale può essere immaginata come composta di diversi "layer"(strati) di nodi, ciascuno dei quali è collegato ai nodi del layer successivo.

Tutte le reti neurali sono composte da almeno tre strati:

- Un input layer, ovvero i dati di ingresso
- Uno o piu' hidden layer, dove avviene l'elaborazione vera e propria
- Un output layer,contenente il risultato finale

I nodi sono connessi a tutti i nodi del layer successivo, e nell'algoritmo queste connessioni vengono "pesate" tramite fattori moltiplicativi, che rappresentano la "forza" della connessione stessa.
Come per i neuroni biologici, i nodi delle ANN integrano i segnali in ingresso secondo una apposita funzione. Se il valore di questa operazione supera la soglia prevista, il nodo sollecitato trasmette questo valore al layer successivo, altrimenti il valore trasmesso sarà zero.
Ad ogni input x_i è associato un peso w_i, con valore positivo o negativo per eccitare o inibire il neurone.
Il bias varia secondo la propensione del neurono ad attivarsi, per variare la soglia di attivazione. Caricati quindi i valori degli input x_i e dei pesi w_i, viene eseguita la somma dei valori in input pesati con i relativi pesi.
Successivamente viene calcolato il valore della funzione di attivazione f utilizzando come argomento il risultato della somma pesata.
L'output del neurone y è quindi il risultato della seguente funzione:

$$y(x) = f(\sum_{i=1}^{d} w_i x_i + w_0)$$

Dove d è numericamente uguale al numero di input totali, e x_0 è il valore del bias.
Risulta quindi essere la funzione di attivazione a determinare la risposta del neurone.
Una rete neurale senza funzione di attivazione equivale semplicemente a un modello di regressione, ovvero cerca di approssimare la distribuzione dei dati con una retta.Con questo modello praticamente ogni layer si comporterebbe

allo stesso modo del precedente, e 100 layer sarebbero di fatto equivalenti ad averne uno soltanto: il risultato sarebbe sempre lineare.

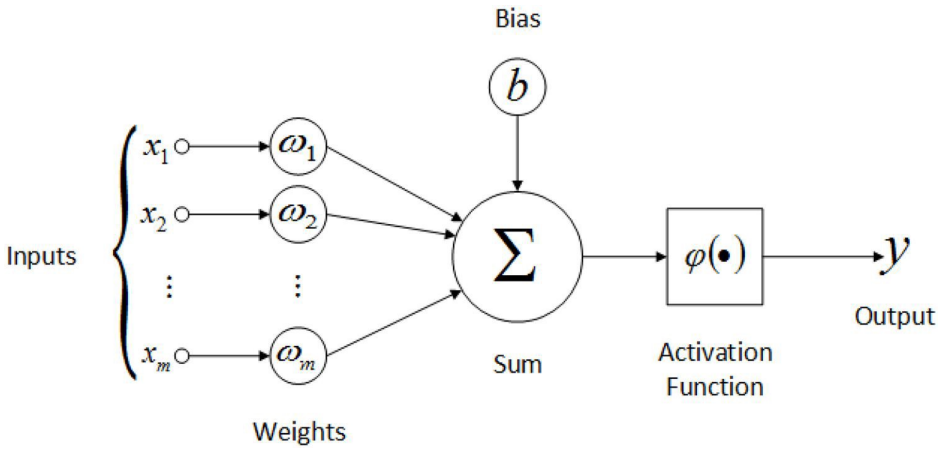

Figura 1.2: Neurone artificiale

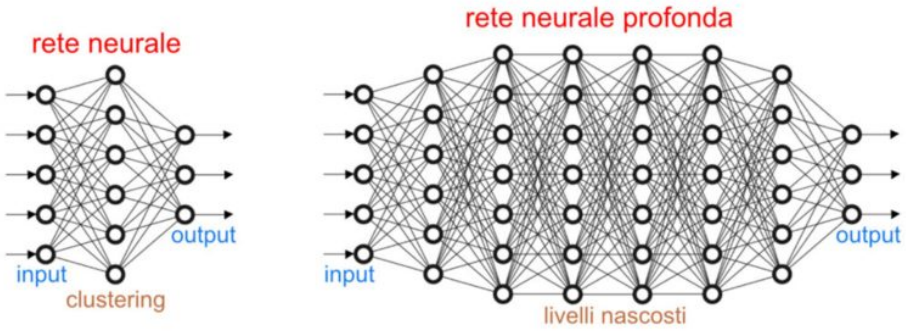

Figura 1.3: Omicron Academy: Reti neurali e deep learning

Lo scopo delle reti neurali è di essere un **Universal Function Approximator**, ovvero essere in grado di approssimare qualsiasi funzione, e per fare questo è necessario introdurre un fattore di non linearità.
Le tre funzioni di attivazione piu' note sono:

1. **Funzione a gradino (Step Function):**
   La funzione piu' intuitiva è quella a gradino, in un certo senso piu' simile al funzionamento biologico. Per tutti i valori negativi la risposta

rimane 0, mentre salta a +1 appena il valore raggiunge o supera anche di poco lo zero.
Il vantaggio è che è semplice da calcolare, e "normalizza" i valori di output comprimendoli tutti in un range compreso tra 0 e +1.

2. **Funzione sigmoide:**
   Ha delle similitudini con la funzione a gradino, ma il passaggio da 0 a +1 è piu' graduale.
   Il vantaggio di questa funzione, oltre ad essere differenziabie, è di comprimere i valori in un range tra 0 e 1 e di essere quindi molto stabile anche per grosse variazioni nei valori.

3. **Funzione ReLU**
   La funzione ReLu(rectifier linear unit) è una funzione divenuta ultimamente molto utilizzata, specialmente nei layer intermedi.
   La ragione è che si tratta di una funzione molto semplice da calcolare; appiattisce a zero la risposta a tutti i valori negativi, mentre lascia tutto invariato per valori uguali o superiori a zero.
   Questa semplicità, unita al fatto di ridurre drasticamente il problema del vanishing gradient, la rende una funzione particolarmente utile nei layer intermedi, dove la quantità di passaggi e di calcoli è importante.

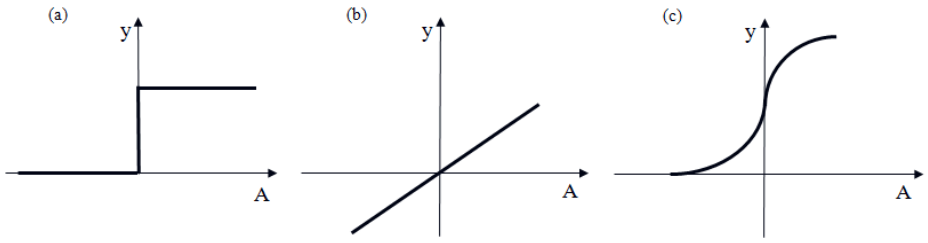

Figura 1.4: Funzioni di attivazione

1.1.3 Background sulla rete neurale

Vi sono due grandi paradigmi di aprrendimento, ciascuno corrispondente ad un particolare compito astratto di apprendimento.

1. **Apprendimento supervisionato (supervised learning):**
   Qualora si disponga di un insieme di dati per l'addestramento **Training set** comprendente esempi tipici di ingressi con le relative uscite loro corrispondenti la rete può imparare ad inferire la relazione che li lega.
   Successivamente la rete viene addestrata mediante un opportuno algoritmo noto come **Backpropagation** che è appunto un algoritmo di apprendimento supervisionato.
   Questo algoritmo, una volta calcolata la Cost Function, ci permette di avere un'idea abbastanza precisa di quanto ciascun neurone di output sia lontano dal proprio valore atteso, e in che direzione (positiva o negativa).
   Se l'addestramento ha successo, la rete impara a riconoscere la relazione incognita che lega le variabili di ingresso e quelle di uscita, ed è quindi in grado di fare previsioni anche laddove l'uscita non è nota a priori.
   In altri termini, l'obiettivo finale dell'apprendimento supervisionato è la previsione del valore di uscita per ogni valore valido di ingresso, basandosi soltanto su un numero limitato di esempi di corrispondenza.

2. **Apprendimento non supervisionato (unsupervised learning):**
   È basato su algoritmi d'addestramento che modificano i pesi della rete facendo esclusivamente riferimento ad un insieme di dati che include le sole variabili d'ingresso.
   Tali algoritmi tentano di raggruppare i dati d'ingresso e di individuare pertanto degli opportuni *c*luster rappresentativi dei dati stessi, facendo uso tipicamente di metodi topologici o probabilistici.

La differenza tra questi due tipi di apprendimento sta nel fatto che nell'apprendimento supervisionato sono note le "categorie",le "classi" o le "etichette".
Nell'apprendimento non supervisionato, non lo sono e il processo di apprendimento tenta di trovare "categorie" appropriate.

In entrambi i casi, vengono considerati tutti i parametri per determinare quali sono i piú appropriati per eseguire la classificazione.

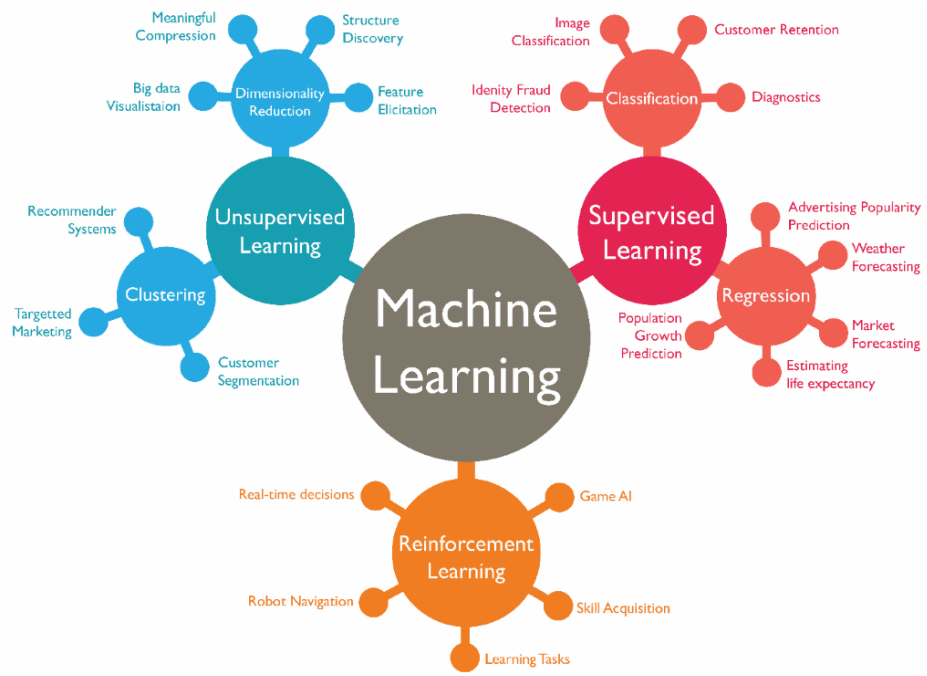

Figura 1.5: Omicron Academy: Apprendimento Supervisionato e non Supervisionato

La rete neurale deve avere capacità di comprensione del modello statistico dei dati e non memorizzare i soli pattern del training set.
È molto importante trovare un rapporto adeguato tra le dimensioni del training set e le dimensioni della rete in quanto un numero troppo elevato di parametri in ingresso e una troppo potente capacità di elaborazione rendono difficile alla rete neurale imparare a generalizzare i dati che riceve. Questo perché input esterni vengono valutati come troppo dissimili dai complessi e dettagliati modelli che conosce.
Per questo motivo si utilizza un validation set, ovvero coppie (input,output corretto) su cui calcolare l'errore E.
Si noti che questi pattern non saranno presenti nel training set, e quindi non saranno utilizzati per allenare la rete.

Una volta calcolato l'output, qualunque valore assuma, non causerà la modifica dei pesi e la rete rimarrà invariata.
L'errore calcolato su validation set risulta quindi essere una buona approssimazione della generalizzazione della rete.
Inizialmente l'errore sarà simile a quello del training set diminuendo di pari passo durante le epoche di apprendimento.
Una volta raggiunto il valore ottimale, se si continuerà ad addestrare la rete, quest'ultima inizierà ad imparare il training set, il cui errore continuerà a scendere, ma non il suo modello statistico, causando l'incremento dell'errore del validation set, questo fenomeno è noto come **Overfitting**.
Viceversa nel caso in cui il numero di variabili nel training set risulti troppo scarso, la rete non avrà abbastanza parametri per imparare a generalizzare causando il fenomeno noto come **Underfitting**.

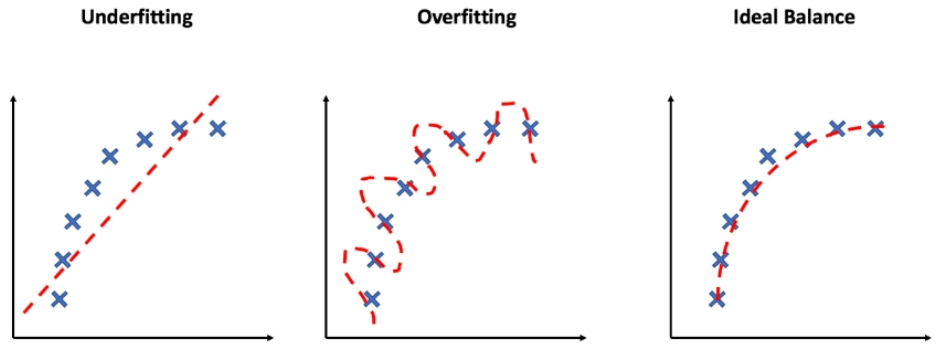

Figura 1.6: Overfitting e Underfitting

1.2 Predizioni di alluvioni

Le alluvioni sono tra le calamità naturali in grado di generare i maggiori danni all'interno di territori abitati. Diversi studi si sono occupati di fornire stime dei danni che saranno provocati da questo tipo di fenomeni nel futuro. Ad esempio, secondo il Progetto Europeo "ClimateCost", entro il 2050, senza migliorie dei sistemi di Protezione Idraulica, vi saranno un numero di

esondazioni tali da provocare danni per 46 miliardi/anno con piu' di 170 mila persone coinvolte in tutto il territorio europeo.

1.2.1 Importanza delle predizioni

Con l'aumento della densità di popolazione e l'utilizzo intensivo del suolo, infatti, sempre piu' persone saranno esposte a rischi naturali come le alluvioni e di conseguenza le perdite umane ed economiche saranno maggiori.
La maggior parte delle catastrofi si ripetono spesso negli stessi luoghi causando danni talvolta molto gravi.
Oggi tuttavia grazie all'utilizzo di vari strumenti informatici come le simulazioni numeriche, siamo in grado di conoscere sia dove potrebbero verificarsi queste calamità,con quale probabilità e con quale intensità, e quali contesti colpiranno(patrimonio culturale, caratteristiche naturali e costruttive).
Risulta quindi necessario valutare il rischio di disastri e mettere in campo delle misure di prevenzione e protezione.

1.2.2 Simulazioni

Il Modello Numerico utilizzato nella presente attività di ricerca**(PARFLOOD)** è stato sviluppato presso il DIA dell'università degli Studi di Parma dal Gruppo di Ricerca Hylab e permette la risoluzione delle equazioni alle acque basse bidimesionali **(Shallow Water Equations-SWE)** attraverso una discretizzazione ai Volumi Finiti.
La presente tesi si focalizza in particolare sull'analisi del tratto di fiume Toce, fiume principale della Val d'Ossola che percorre da nord a sud il vasto bacino idrografico e sul torrente Baganza della provincia di Parma che scorre nell'omonima Val Baganza con un'area di bacino di 228 km^2
Il modello **PARFLOOD** prevede quattro principali fasi di lavoro:

- Fase di inserimento dei dati o fase di Input;
- Fase di calcolo e risoluzione SWE;

- Fase di restituzione dei risultati o fase di Output
- Fase di decode dei risultati

I programmi utilizzati per poter preparare i dati di input necessari o per poter visualizzare i valori restituiti dal modello sono principalmente i seguenti:

- Surfer;
- Qgis;
- Matlab;
- Notepad++
- WinSCP;
- Cygwin64("openssh")

Dati File Input

Per risolvere le equazioni alle acque basse il modello necessita di informazioni riguardanti la batimetria e la scabrezza della zona che si intende studiare, nonché le condizioni iniziali e al contorno, richiedendo complessivamente nove file di input:

- Batrimetria(file.BTM);
- Scabrezza(file.MAN);
- Condizione iniziale(file.INH);
- Condizione al contorno(file.BLN);
- Velocità iniziale lungo x(file.VHX);
- Velocità iniziale lungo y(file.VHY);
- Specifica sulle condizioni al contorno(file.BCC);

- Eventuali brecce(file.BRE);
- Multi risoluzione(file.PTS);

Se i primi quattro file sopra citati sono indispensabili per qualsiasi simulazione, gli ultimi, in alcuni casi potrebbero anche non comparire: ipotizzando ad esempio di avere una condizione di acqua inizialmente ferma, appare evidente come non esistano condizioni iniziali di velocità e quindi i file .VHX e .VHY non sono richiesti.

Batimetria(file.BTM)

Questo file contiene informazioni relative all'andamento della quota del terreno z per i diversi punti x,y appartenenti ad una determinata griglia. Generalmente questo file è ottenuto dalle elaborazioni di modelli digitali del terreno. È possibile visualizzare graficamente tali file attraverso progammi come Surfer.

Scabrezza(file.MAN)

In questo file è definito un valore di scabrezza per ciascuna delle celle della grigia. rappresentando cosí la resistenza al moto. Tale valore è funzione delle caratteristiche litologiche e dell'uso del suolo.
Le formulazioni piú note riguardanti la scabrezza sono quelle di Strickler:

$$\chi = k_s R^{1/6}$$

oppure quella di Manning:

$$\chi = 1/nR^{1/6}$$

dove $k_s(m^{1/3}/s)$ e $n = 1/k_s(m^{-1/3}s)$ sono rispettivamente i coefficienti di Strickler e Manning mentre R è il raggio idraulico.

Condizione iniziale(file.INH)

All'interno di questo file si definisce la quota idrica iniziale in ciascuna cella della griglia. Tale condizione può essere ricavata anche da una simulazione preliminare.
Anche se si vuole effettuare una simulazione partendo da una condizione iniziale di alveo asciutto, per necessità del modello, è necessario bagnare un numero limitato di celle in prossimità della condizione al contorno di monte.
Se invece si vuole effettuare una simulazione partendo da una simulazione preliminare si dovrà semplicemente convertire il file di outpute WSE della simulazione preliminare (opportunamente decodificato) in formato INH.
In entrambi i casi bisogna sempre prestare attenzione che la griglia della batimetria e la griglia delle condizioni iniziali abbiano le stesse dimensioni.

Portate specifiche iniziali(file.VHX/VHY)

Se le mappe delle portate specifihe iniziali lungo x o lungo y non sono fornite, il modello assumerà che tali grandezze siano nulle in tutto il dominio. Tali mappe chiaramente non sono fornite partendo da una condizione iniziale di alveo asciutto, se invece si sfrutta una simulazione preliminare per ricavare le condizioni iniziali, i due file potranno essere ottenuti moltiplicando attraverso il comando "Grid Math" di Surfer, il file decodificato DEP(profondità idrica) con i files delle velocità VVX VVY.
Naturalmente è necessario che i file si riferiscano allo stesso istante di tempo nel quale si vuole generare la nuova condizione iniziale.

Condizione al contorno(file.BLN)

Il file .BLN, che racchiude le condizioni al contorno, è un file di testo strutturato nel modo seguente:

- Numero di punti;
- Valore delle coordinate x,y di ogni punto e della condizione al contorno.

Il terzo valore che segue le coordinate è l'indicatore della condizione al contorno da applicare al segmento compreso tra il punto in oggetto e il successivo e può essere:

- 21: portata(onda di piena in ingresso lungo la sezione definita dal segmento);
- 1: muro(l'acqua non può proseguire oltre, in quanto il segmento funge da muro;
- 2: portata specifica;
- 3: livello(si impone un valore di tirante idrico);
- 4: farfield(l'acqua si allontana senza disturbo, come se vi fosse un terreno pianeggiante indefinito e privo di ostacoli);
- 5: scala di deflusso(sulla sezione rappresentata dal segmento è nota la relazione che fornisce l'andamento delle portate in funzione del tirante idrico).

Specifica condizioni al contorno(file.BCC)

Una volta definito il dominio di calcolo e definite quindi le caratteristiche dei punti che individuano il contorno, è necessario specificare attraverso un file BCC l'andamento delle onde di piena, dei livelli o delle scale di deflusso. Tale file è formattato in modo che nella prima riga venga specificato il numero delle condizioni da inserire. Dopodiché, per ciascuna condizione, è necessario indicare il numero segmento interessato(riportato nel file BLN) e il numero di righe necessarie per descrivere la condizione al contorno.

Definizione Brecce(file.BRE)

Questo file fornisce le informazioni relative alla generazione di un'eventuale

breccia lungo un tratto arginale. Generalmente tale istante è valutato in base al passaggio del picco dell'onda di piena oppure quando si potrebbe avere un sormonto arginale che porterebbe prima a cedimenti differenziali del rilevato e poi a un collasso completo.
ALl'interno troviamo quindi i seguenti parametri:

- TIMESTART: ora nella quale si ha l'apertura della breccia;
- TIMEEND: ora nella quale termina l'abbassamento della breccia;
- TIMEENDL: ora nella quale termina l'allargamento della breccia;
- BSTART: larghezza(m) iniziale della breccia;
- BEND: larghezza(m) finale della breccia;
- BETA: angolo d'inclinazione(rad);
- ZSTART: quota iniziale(m slm) formazione della breccia;
- ZEND: quota finale(m slm) raggiungimento della breccia;
- MAXWIDTH: larghezza massima(m) della breccia;
- XBR: coordinata x(m) origine della breccia;
- YBR: coordinata y(m) origine della breccia;
- DIRXBR: versore x della breccia(-);
- DIRYBR: versore y della breccia(-);

Multi risoluzione(file.PTS)

Il modello (PARFLOOD) permette di adottare tre differenti tipologie di dettaglio per l'area di studio.
È possibile infatti discretizzare il dominio attraverso griglie Cartesiane composte da celle quadrate di dimensioni costanti. Tuttavia, tali griglie non

possono essere impiegate per domini molto ampi poiché dovrebbero essere discretizzati utilizzando decine o centinaia di milioni di celle, che non possono essere allocate nella memoria di una GPU.
In alternativa quindi, è preferibile utilizzare un tipo di griglia in grado di utilizzare risoluzioni differenti in diverse porzioni di dominio, in modo da ridurre i tempi di calcolo e restituire risultati dettagliati solamente in specifiche aree di maggior interesse.

1.2.3 Esecuzione della simulazione

Una volta definiti e caricati sul Cluster HPC di Ateneo (sistema operativo Linux) i vari file elencati precedentement è necessario definire un file di input attraverso il quale specificare ulteriori parametri quali la durata della simulazione (hr), di output(hr) etc..
ALl'interno, oltre a parametri strettamente legati al modello sono definiti:

- NOME STRINGA: nome della parte che precede l'estensione dei file;
- START xxx: tempo di inizio della simulazione(inizio alle ore xxx);
- END xxx: tempo in cui termina la simulazione(fine alle ore xxx);
- DTOUTPUT: intervallo di tempo in cui salvare i risultati(risultati salvati e stampati su file ogni xxx ore);
- CR xxx: numero di Courant utilizzato per la definizione del passo di calcolo(<1);
- YEPS xxx: soglia che distingue le celle bagnate da quelle asciutte (valore del livello $<$ xxx la cella è considerata asciutta). Un valore basso dela soglia YEPS comporta errori ridotti ma espone a un rischio maggiore di instabilità numerica;
- VELEPS xxx: soglia per limitare i flussi scambiati tra celle molto piccole, inferiori al valore fissato xxx;

- DTSOGLIA xxx: soglia temporale per il passo di calcolo.

Dati File Output

Le principali mappe restituite in seguito alla fase di calcolo da parte del Modello PARFLOOD, ad intervalli di tempo definiti sono le seguenti:

- Mappe WSE(Water Surface Elevation);
- Mappe MAXWSE(massima quota idrica raggiunta);
- Mappe DEP(tiranti idrici);
- Mappe BTM(eventuali variazioni di batimetria in seguito a brecce);
- Mappe VVX/VVY(componenti della velocità lungo x e y);
- Il file [Nome caso]_output.txt: contiene tante righe quanti sono gli istanti temporali salvati e riassume i tempi di calcolo e di salvataggio.
- Il file Output_[nome caso].txt: descrive in maniera esaustiva l'intero procedimento seguito dal modello di calcolo.

La numerazione degli istanti temporali inizia da 0 e termina con LAST, file contenente gli ultimi risultati prima del termine della simulazione.

Tutti i file di output sono scritti in linguaggio binario e non sono immediatamente visualizzabili.
Per poterli visualizzare è necessario "decodificarli".

Decode

Per poter decodificare le mappe è necessario utilizzare gli strumenti di post-processing del codice PARFLOOD. È infatti possibile ottenere differenti files decodificati variando ad esempio il numero di frame d'interesse,la risoluzione,il range delle dimensioni della mappa o scegliendo di convertire solamente

i file LAST. Sono tre le tipologie fondamentali di dati che si possono ottenere decodificando i file di output:

1. Mappe di livelli,quote idriche,velocità, ecc...;
2. Portate nelle sezioni;
3. Livello nei punti;

1.3 Cluster di Ateneo

All'interno dell'Ateneo si ha a disposizione una macchina **HPC CLUSTER**, sistema operativo Linux, che dispone di **14 schede video GPU(Nvidia Tesla® P100)** il cui utilizzo è regolato attraverso un gestore di code **SLURM**. È possibile interfacciare questa macchina e il proprio computer di lavoro mediante programma WinSCP mentre per gestire le varie operazioni relative alle simulazioni ci si affida al programma Cygwin(pacchetto ssh).

Al fine di effettuare i vari test in maniera piú efficiente è stato messo a disposizione un nuovo nodo di calcolo **wn44.hpc.unipr.it** il quale dispone di:

- 2xCPU: **AMD EPYC 7352 24-Core Processor**
- 4xGPU: **NVIDIA GRID A100 PCIe 40GB**

1.3.1 Keras e Tensorflow

Per lo svolgimento di questa tesi è stato utilizzato TensorFlow, una libreria software open source per il calcolo numerico ad altre prestazioni, che fornisce API native in linguaggio Python, C/C++, Java, GO, e Rust.
Sviluppato originariamente dai ricercatori del team Google Brain, viene fornito con un forte supporto al machine learning, ed è attulmente usato in moltissimi prodotti commerciali Google come il riconoscimento vocale,Gmail,Google

Foto e Ricerca.
Per installarlo su una macchina con Ubuntu 16.04 LTS come sistema operativo, occorre utilizzare il gestore di pacchetti pip di Python.
Se non sono presenti sulla macchina utilizzata sarò necessario intallare Python, il gestore dei pacchetti pip e Virtualenv tramite i comandi:

1. **$ sudo apt update**
2. **$ sudo apt install python3-dev python3-pip**
3. **$ sudo pip3 install -U virtualenv**

Una volta completata tale operazione è possibile creare un ambiente viruale in modo da isolare l'installazione del pacchetto del sistema.

$ **virtualenv –system-site-packages -p python3 ./venv**

Una volta creato l'ambiente, occorre attivarlo usando un comando specifico della shell:

$ **source ./venv/bin/activate # sh, bash, ksh, or zsn**

Per uscire dall'ambiente virtuale:

$ **deactivate**

A questo punto saremo in grado di installare i pacchetti all'interno dell'ambiente virtuale.

$ pip install –upgrade pip

$ pip install tensorflow

Per poter utilizzare la API native di TensorFlow in un programma Python

sarà necessario inserire la seguente riga di codice:

import tensorflow

Keras

Keras è una libreria open source scritta in Python. Si tratta di software open source per l'appredimento automatico e le reti neurali e supporta come back-end Tensorflow.
Keras offre moduli utili per organizzare differenti livelli. Il tipo principale di modello è quello sequenziale, ovvero una pila lineare di livelli.
Keras consente una prototipazione facile e veloce, supporta sia reti convoluzionali(CNN) che reti ricorrenti(RNN) o combinazioni di entrambi, supporta anche schemi di connetività come multi-input e multi-output e funziona sia su CPU che GPU.
Keras ci permette di utilizzare i tensori. Un tensore puó essere scalare, un vettore o un array multi-dimensionale e ha una forma, un rango, e un asse.
Interfacciandosi con Keras, è possibile usa funzioni di attivazione come la ReLU, sigmoide ecc...
Inoltre il backend di Keras, essendo interfacciato con Tensorflow, ci mette a disposizione ulteriori API per la gestione dei dati e la rappresentazione visuale di modelli.

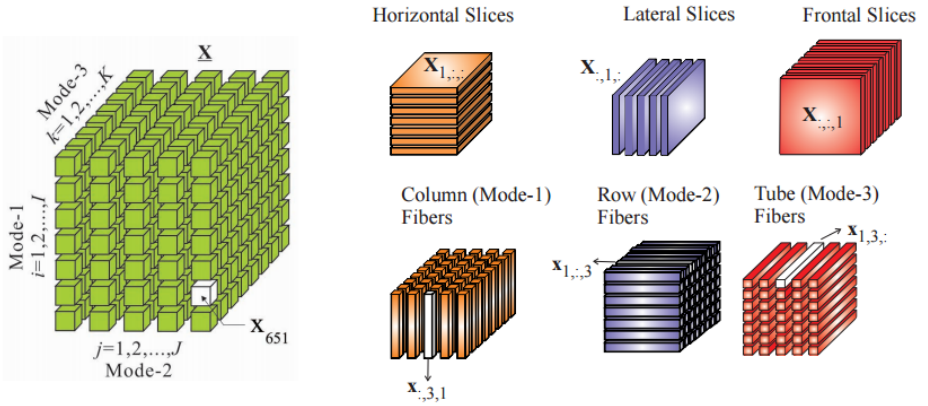

Figure 2: A 3rd-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I \times J \times K}$, with entries $x_{i,j,k} =$

Figura 1.7: La struttura del tensore

Capitolo 2

Obiettivo della tesi

L'obiettivo della tesi è ottenere una rete neurale per la predizione di zone asciutte e bagnate in una alluvione.

2.1 Analisi dei requisiti

Il fiume che prenderemo in considerazione è il Toce.

Al fine di poter analizzare al meglio l'allagamento e permettere una predizione rilevante per la rete neurale sono state eseguite 100 simulazioni con 100 differenti file BCC con differenti valori di portata(m^3/s) al tempo 2.0, che vanno da 0.001840(m^3/s) (Nomefile-000.BCC) a 0.1840(m^3/s) (Nomefile-099.BCC).
È stata inoltre selezionato un valore di soglia di 5mm, in modo da poter considerare le celle al di sotto di tale valore come asciutte.

- START 0
- END 1.4888888888888889E-02
- DTOUTPUT 6.944444444444444E-05

```
toce_ris_V17_50     ! file name BASE with input files for initial conditions, bathymetry, output and boundary conditions
START 0.                    ! t0, beginning time (hours)
END 1.4888888888888889E-02                  ! tlast, simulation time
DTOUTPUT 6.944444444444444E-05     !in hours
CR 0.4                 ! CR, courant number
LIMITER 3                     ! limiter, (1 van leer, 2 van albada, 3 minbee, 4 superbee)
YEPS 1.E-9                ! yeps, water depth treshold for wet/dry cell
VELEPS 1.E-5    !
MUSCL -1                   ! R, MUSCL reconstruction -1 UPWIND, 1 CENTERED
IBINARY 0                    ! IBINARY, input/output (1 for ASCII, 0 for IBINARY)
AL 2                    ! AL 0 SGM puro, 1 DGM puro, 2 WSDGM
SR 1                    ! SR, 1 strang splitting formulation for friction source term, 0 for godunov formulation
DTSOGLIA 0.00                   ! dtsoglia, treshold for the time step (if dt<dtsoglia then WARINING is plotted)
EXPON 1.66667               ! expon, for discharge redistribution at inflow boundary conditions
AW 0.                   ! A_WSDGM, froude number for the beginning of the DGM trasition
BW 2.0                  ! B_WSDGM, end of the DGM transition
METODO 1                    ! metodo, 2 for HLLC fluxes, 1 for slic fluxes
NITER 3                    ! niter, iteration number for correction procedure at wetting/drying fronts
PENDFARFIELD 1 !
```

Figura 2.1: File input simulazione

A questo punto, dopo aver opportunamente decodificato i file ottenuti come risultato delle simulazioni, avremo 100 file "decoded" contenenti 100 file .MAXWSE

Capitolo 3

Progettazione della rete

Per l'apprendimento è stata utilizzata una rete neurale **feed-forward**, ovvero una rete neurale artificiale dove le connessioni tra le unità non formano cicli. In questo tipo di rete neurale le informazioni si muovono in una sola direzione, avanti, rispetto a nodi di ingresso, attraverso nodi nascosti(se esistono) fino ai nodi di uscita,
Non avendo memoria di input avvenuti a tempi precedenti l'output viene determinato a partire dall'attuale input.
Il modello sequenziale risulta il piú appropriato per il tipo di rete neurale che stiamo realizzando.

3.1 Preparazione dei dati

Prima di tutto dovremo organizzare i dati nelle seguenti cartelle:

- input/5mm_maxwse: contenente i file decodificati relativi ai 100 MAXWSE risultanti dalle 100 simulazioni effettuate.

- input/BCC: contenente i file relativi ai 100 BCC generati

Utilizzeremo inoltre uno script python(utils) che ci aiuterà nella fase di recupero dei dati.

3.2 Analisi del codice

In questa sezione andremo ad analizzare la rete neurale.

Imports

Per prima cosa dovremo importare tensorflow e keras nonché Input che ci permetterà di istanziare un tensore.
Dovremo inoltre importare Sequential e Model in quanto la nostra rete adotterà un tipo di modello sequenziale.
Utilizzeremo un ottimizzatore Adam, ovvero un metodo di discesa del gradiente stocastico basato sulla stima adattiva dei momenti del primo e del secondo ordine.
Importeremo infine alcune librerie come numpy,matplotlib e json che ci aiuteranno nella gestione dei dati e delle immagini.

```
import os as os

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Input
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import Adam

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import json
import random
from types import SimpleNamespace

import utils

%matplotlib inline
```

Parse config

Definiremo adesso alcune configurazioni:

- Indicheremo il path relativo aile directory contenenti i nostri file MAXWSE ottenuti dalle simulazioni e i BCC generati
- Specificheremo le epoche di apprendimento e la batchSize
- Indicheremo la cartella per gli output.

```
config = json.loads('''{
    "input": {
        "decodedPath": "input/5mm_maxwse2",
        "bccsPath": "input/BCC"
    },
    "NN": {
        "outputPlot": "images/model_plot.png",
        "epochs": 100,
        "batchSize": 4
    },
    "output": {
        "folderName": "images"
    }
}''', object_hook = lambda d: SimpleNamespace(**d))
```

Retrieve Data

Andremo a recuperare i dati ulizzando un utile script chiamato "utils" cosí che questi ultimi siano finalmente utilizzabili dalla nostra rete.

```
bccs = utils.parse_bccs(config.input.bccsPath)
maxwses = utils.parse_maxwses(config.input.decodedPath)
```

Utils

Di seguito le due funzioni definite nello script "utils" ed utilizzate nella fase di recupero dei dati.

- Funzione parse_maxwses: Questa funzione ci permette di "analizzare" i nostri file MAXWSE selezionando opportunamente solo le righe del file contenenti la matrice e ritornando un array di numpy contenente la matrice stessa.
- Funzione parse_bccs: Questa funzione ci permette di "analizzare" i nostri file BCC anche in questo caso selezionando solo le righe del file utili e ritornando un array di numpy di cui specifichiamo il tipo di dato (float32) grazie alla funzione astype() di numpy.

```
# Parse all file made from a swegpu simulation
def parse_maxwses(directory, line_skip_per_file = 5):
    files = []

    directory = os.path.abspath(directory)
    for f in os.listdir(os.path.abspath(directory)):
        f = directory + "/" + f
        name, file_extension = os.path.splitext(f)
        if os.path.isfile(f) and file_extension.lower() == ".
   maxwse" and "flooding" in name:
            files.append(f) # N.B.: files are put in random
   order

    files.sort()

    matrix = []
    for f in files:
        matrix.append(parse_file(f, line_skip_per_file))

    return np.array(matrix)

def parse_bccs(directory):
```

```
    data = []

    for bcc in os.listdir(directory):
        data.append([
            [
                ' '.join(line.split()).strip("\n").split(" ")
[index]
                for index in [0, 2]
            ]
            for line in open(directory + "/" + bcc, 'r').
readlines()[2:]
        ])

    return np.array(data).astype(np.float32)
```

Reshaping

I file BCC sono della forma { [simulazione1_punto1,simulazione2_punto2,...]...} dato che ogni simulazione potrebbe avere piú di un punto da usare per prevedere il massimo livello di allagamento

```
bccs = [[wave[1] for wave in bcc[1:-1]] for bcc in bccs]
```

Get data shape

Tramite la funzione shape di numpy specificheremo il numero di righe e di colonne dei maxwse noché il numero dei maxwse.
La stessa procedura sarà effettuata per quanto riguarda i file bcc.

```
maxwses_number, maxwse_rows, maxwse_cols = np.shape(maxwses)
bccs_number, wave_points_number = np.shape(bccs)
```

Prepare data for input

A questo punto andremo a preparare i nostri dati.
Dalle 100 simulazioni, è stato scelto un 80% dei risultati ottenuti, che costituirà il training set. Il restante 20% verrò suddiviso in due parti che andranno a formare il test set e l'input che verrà utilizzato per le predizioni e le verifiche dei risultati.
La train_size sarà di 80 immagini mentre la test_size sarà di 20 immagini.

- Inizializziamo la train_size e la test_size
- Utilizzeremo la funzione unique di numpy per ritornare un array di elementi univoci presi da un campione di elementi nel range di bccs_number, ottenuto tramite la funzione random.sample.
- inizializzeremo test_indices e predict_indices ed effettueremo delle stampe per poter visualizzare il campione che utilizzeremo.

```
train_size = int(bccs_number * 0.8)
test_size = train_size // 4

train_indices = np.unique(random.sample(list(range(
    bccs_number)), train_size))

test_indices = [i for i in np.unique(random.sample(list(range
    (bccs_number)), test_size)) if i not in train_indices]
predict_indices = [i for i in list(range(bccs_number)) if i
    not in train_indices and i not in test_indices]

print("Train frames: ", train_indices)
print("Test frames: ", test_indices)
print("Prediction frames: ", predict_indices)
```

Data partitioning

```
x_train = np.array([bccs[index] for index in train_indices])
y_train = maxwses[train_indices]

x_test = np.array([bccs[index] for index in test_indices])
y_test = maxwses[test_indices]

x_predict = np.array([bccs[index] for index in
    predict_indices])
```

Building the Net

Andremo ora a definire la struttura della nostra rete.
Avremo innanzitutto un layer di input, a cui seguiranno tre layer densi composti rispettivamente da 128, 256 e 128 unità a cui alterniamo dei layer di batch normalization al fine di rendere l'apprendimento piú veloce.
Troviamo infine un ultimo layer denso con lo scopo di ottenere la dimensione utilizzata dalle matrici in input.
Troviamo poi la funzione di attivazione ReLu e una funzione reshape che ci servirà per riorganizzare i dati da array a matrice.

- Il modello utilizzato è sequenziale
- Ci aspettiamo che gli input siamo vettori di dimensione pari a wave_points_number (precedentemente definito in fase di shaping.
- Utilizziamo la funzione BatchNormalization() in modo da standardizzare automaticamente gli input in un layer. Questa funzione, quando utilizzata ha l'effetto di accelerare il processo di allenamento della rete e, in alcuni casi, migliora le performance del modello.
- Gli strati utilizzati sono densi, ad indicare che ogni neurone nel layer denso riceve input da tutti i neuroni del layer precedente.
  Il layer denso effettua una moltiplicazione matrice-vettore e i valori utilizzati nella matrice sono parametri che possono essere allenati e aggiornati con l'aiuto della backpropagation.
  L'output generato da un layer denso è un vettore 'm-dimensionale'.

- La funzione di attivazione utilizzata è la ReLU.

Infine compileremo il modello utilizzando l'ottimizzatore Adam e la "mean-squared-error come loss function al fine di calcolare la media dei quadrati degli errori delle previsioni.

```
model = Sequential([
    Input(shape = (wave_points_number)),
    BatchNormalization(),

    Dense(128),
    BatchNormalization(),

    Dense(256),
    BatchNormalization(),

    Dense(128),
    BatchNormalization(),

    Dense(maxwse_rows * maxwse_cols),

    Activation("relu"),

    Reshape(target_shape = (maxwse_rows, maxwse_cols))
])


model.compile(optimizer = Adam(learning_rate = 0.01), loss =
   "mse", metrics = ["accuracy"])
model.summary()
```

Training

In questa fase avverrà l'allenamento vero e proprio della rete. Chiamiamo la funzione fit() , che addestrarà il modello suddividendo i dati in "batch" di dimensione "batch_size", e ripetutamente iterando sull'intero set di dati per un dato numero di "epoche"

Il validation split ci permette di evitare il rischio di overfitting, in quanto non vogliamo ricampionare il nostro set dopo ogni epoca,se lo facessimo il modello verrebbe addestrato su ogni singolo campione del set di dati causando appunto overfitting.
È opportuno quindi dividere sempre i dati prima del processo di addestramento, motivo per cui l'algoritmo deve essere addestrato utilizzando solo il sottoinsieme dei dati per l'addestramento.

```
history_shuffled = model.fit(
    x_train,
    y_train,
    batch_size = config.NN.batchSize,
    shuffle = True,
    validation_split = 0.2,
    epochs = config.NN.epochs,
    verbose = 2
)
```

Evaluation

Una volta completato il training valuteremo il modello sui dati del test tramite la funzione evaluate().

```
model.evaluate(x_test, y_test)
```

```
plt.title("Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Similarity")
plt.plot(range(config.NN.epochs), history_shuffled.history["
    accuracy"], "r")
plt.show()

plt.title("Loss")
plt.xlabel("Epochs")
```

```
plt.ylabel("Loss")
plt.plot(range(config.NN.epochs), history_shuffled.history["
    loss"], "r")
plt.show()


y_predict = model.predict(x_predict)
```

Check for polarization

Utilizzando la funzione kdeplot di seaborn possiamo tracciare distribuzioni univariate o bivariate utilizzando la stima della densità del kernel.
Questo è un metodo utile per visualizzare la distribuzione delle osservazioni in un set di dati, analogo a un istogramma. KDE rappresenta i dati utilizzando una curva di densità di probabilità continua in una o piú dimensioni.

```
sns.kdeplot(y_predict[0].flatten(), x = "value")
sns.kdeplot(maxwses[0].flatten(), x = "value")
```

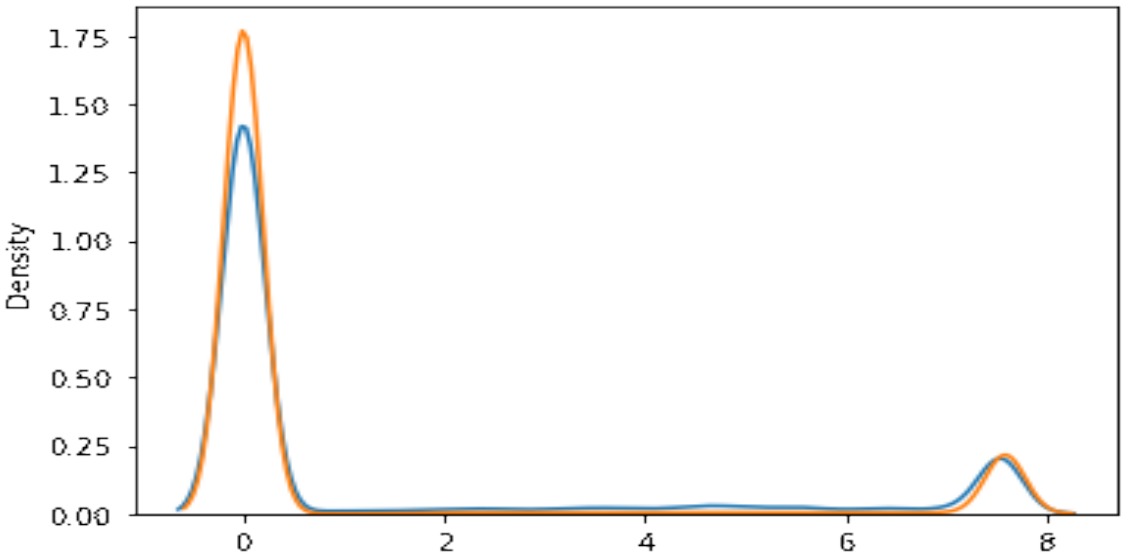

Figura 3.1: Results

Post processing

```
y_predict[y_predict > 0.5] = 1
y_predict[y_predict <= 0.5] = 0
```

Plots

```
xy_predict = zip(maxwses[predict_indices], y_predict)

distances = []
for ((maxwse, predicted), i) in zip(xy_predict,
    predict_indices):
     fig, (maxwse_plot, predicted_plot) = plt.subplots(1, 2)

     maxwse_plot.set_title("Originale " + str(i))
     maxwse_plot.axis("off")
     maxwse_plot.matshow(maxwse)


     predicted_plot.set_title("Previsione " + str(i))
     predicted_plot.axis("off")
     predicted_plot.matshow(predicted, cmap = "cividis")
     plt.show()

     distance = utils.distance(maxwse, predicted)
     distances.append(distance)
     print("Similarity: ", distance)

print("Average Similarity: ", sum(distances) / len(distances)
    )
```

Capitolo 4

Risultati

In questo capitolo andremo ad analizzare i risultati dell'apprendimento e ad effettuare confronti con altri scenari in modo da poter valutare le prestazioni della rete in relazione alla qualità dell'apprendimento.
Le immagini rappresentano i risultati dell'apprendimento mostrando a sinistra il MAXWSE originale e a destra il MAXWSE generato dalla rete.

4.1 Risultati dell'apprendimento

Come risulta dalle immagini, la rete impara progressivamente i frame migliorando ad ogni passo l'accuracy.
La rete dimostra avere difficoltà maggiori nei primi frame, probabilmente a causa di un leggero sbilanciamento del dataset verso gli allagamenti centrali.
Bisogna inoltre specificare che in fase di simulazione, le matrici sono state trasformate da binarie a testuali in modo da poter osservare in modo piú dettagliato la natura delle immagini.

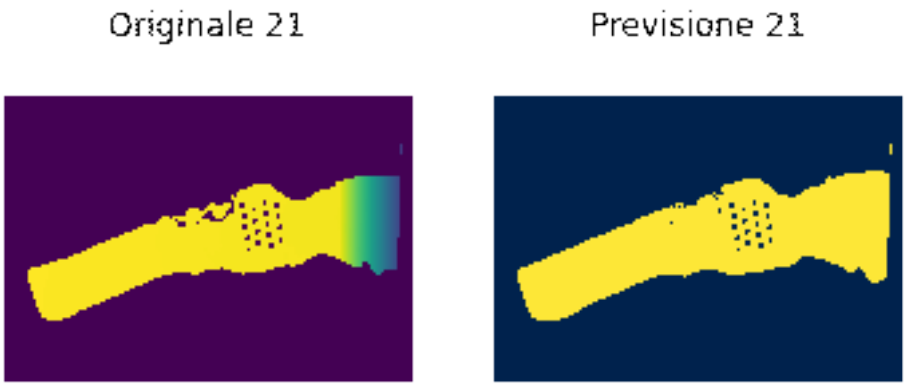

Figura 4.1: Accuracy: 0.9276279807090759

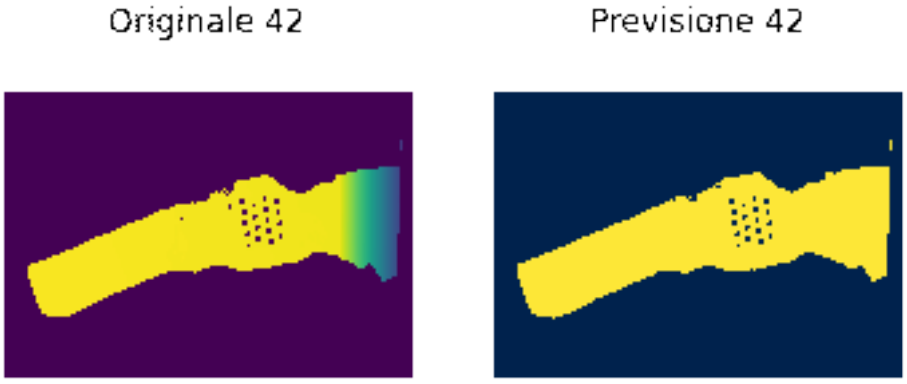

Figura 4.2: Accuracy: 0.9578232765197754

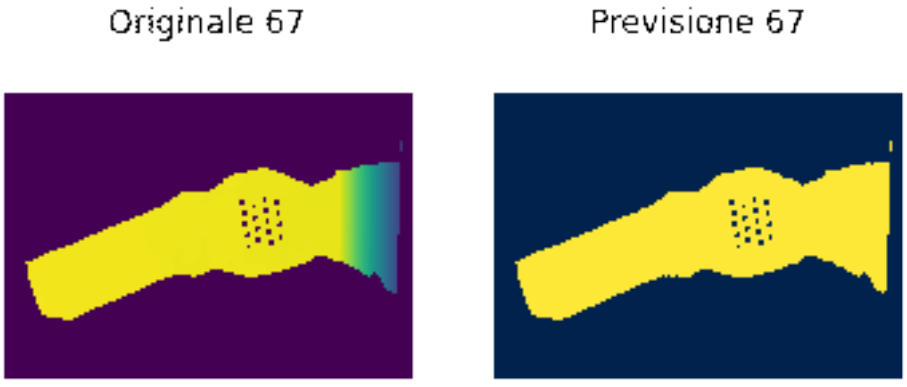

Figura 4.3: Accuracy: 0.9695315957069397

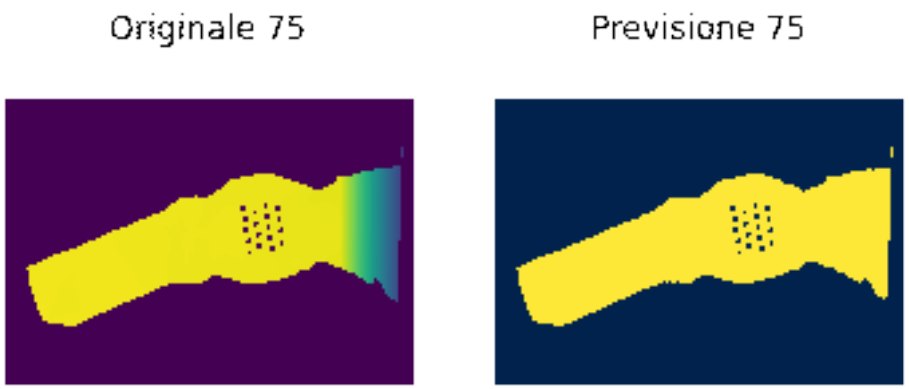

Figura 4.4: Accuracy: 0.9697808623313904

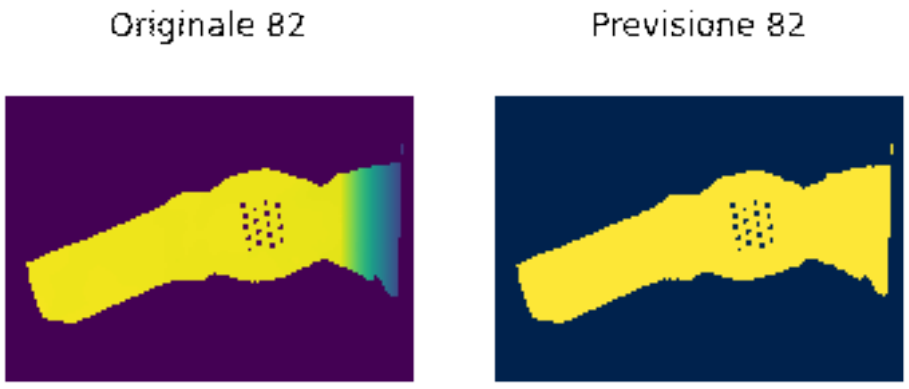

Figura 4.5: Accuracy: 0.9600760936737069

4.2 Precisione della rete

I risultati mostrano come la rete abbia una maggior precisione nei frame iniziali e finali avendo le maggiori incertezze nell'area di fiume dove si presentano gli ostacoli, essendo questa la parte piú complessa da prevedere. La scelta dei frame test può quindi migliorare l'average similarity.

4.3 Test con la rete

Al fine di ottenere una rete neurale il piú precisa possibile, sono stati effettuati vari test studiando anche come i risultati varino al variare della struttura. Per la generazione della matrice binaria degli allagamenti (soglia 5mm) è stata sviluppata anche una rete neurale feed forward che prende in input la matrice BTM(batimetria) + un vettore con tre misurazioni di portata. Questo modello apprende progressivamente aumentando la cosine similarity con la reale matrice binaria.

4.3.1 Rete con Batimetria

Functions

```
import numpy as np
import pandas as pd
import os
from pathlib import Path
import tqdm
import matplotlib.pyplot as plt
import random
import torch
import torch.nn as nn
import torch.optim as optim
import seaborn as sns

%matplotlib inline

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

import keras.layers.advanced_activations as
    advanced_activations
import keras.activations as activations
```

```
import keras.layers as layers
from keras.layers import Dense, concatenate

import scipy.spatial.distance as distance
def plot_mat_accuracy(input, y_hat, y_true):

  input = input
  y_hat = y_hat
  y_true = y_true

  dist = distance.cosine(y_hat.flatten(), y_true.flatten())

  fig, axs = plt.subplots(1,3)
  fig.suptitle('Accuracy: {}'.format(1 - dist))

  axs[0].set_title("Input")
  axs[0].matshow(input)

  axs[1].set_title("Y_hat")
  axs[1].matshow(y_hat)

  axs[2].set_title("Y_true")
  axs[2].matshow(y_true)

  for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

  plt.show()

  return 1 - dist

def get_mat_accuracy(y_hat, y_true):
  dist = distance.cosine(y_hat.flatten(), y_true.flatten())
  return 1 - dist
```

Data import

```
from google.colab import drive
drive.mount('/content/drive/')

%cd drive/MY\ Drive/SWE

portate = np.load('datasets/numpy/portate.npy')
print(portate.shape)

binaries = np.load('datasets/numpy/binaries.npy')
print(binaries.shape)

btms = np.load('datasets/numpy/btms.npy')
print(btms.shape)
```

Data Preprocessing

L'input del modello è formato da 2 tipologie di dato concatenate:

- Matrice BTM(ad 1/3 della risoluzione originale)
- Vettore serie di portate nel tempo

```
portate = portate[:, :, :, 2]
```

Resizing

```
from skimage.transform import resize

binaries = binaries[:, 0, :, :]
btms = btms[:, 0, :, :]

print(binaries.shape)
print(btms.shape)

plt.matshow(binaries[0])
```

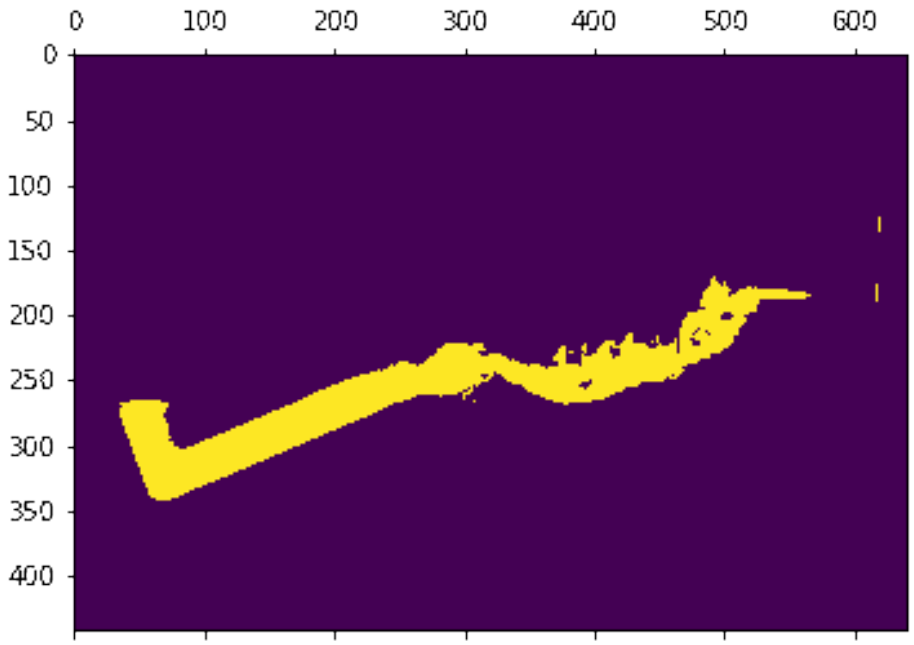

Figura 4.6: Plot dell'immagine binaries[0]

Concatenating

```
model_in_shape = portate.shape[2] + (btms.shape[1] * btms.
    shape[2])
model_out_shape = binaries.shape[1] * binaries.shape[2]

# 100, model_in_shape (es: 300x300 + 3)
inputs = np.zeros((portate.shape[0], model_in_shape))
img_shape = model_in_shape - portate.shape[2]

for i in range(100):
  # Copia immagine flattened
  inputs[i][:img_shape] = btms[i].flatten()
  # Copia dati portata
  inputs[i][img_shape:] = portate[i][0]

TRAIN_SIZE = int(inputs.shape[0] * 0.8)
```

```
x_train, y_train = inputs[:TRAIN_SIZE], binaries[:TRAIN_SIZE]
x_test, y_test = inputs[TRAIN_SIZE:], binaries[TRAIN_SIZE:]
```

Feed Forward

```
model = Sequential(
    [
        layers.Input(model_in_shape),
        layers.BatchNormalization(),

        layers.Dense(128),
        layers.BatchNormalization(),

        layers.Dense(256),
        layers.BatchNormalization(),

        layers.Dense(128),
        layers.BatchNormalization(),

        layers.Dense(model_out_shape),
        layers.Activation(activations.sigmoid),

        layers.Reshape((binaries.shape[1], binaries.shape[2])
    )

    ]
)


model.compile(optimizer="Adam", loss="binary_crossentropy",
    metrics=["cosine_similarity"])
model.summary()

epochs = 100

model.fit(
    x_train,
    y_train,
```

```
    batch_size=4,
    epochs=epochs,
    verbose=2,
    shuffle=1,
    validation_split=0.2,
)
```

Evaluate

```
model.evaluate(x_test, y_test)

# btms 0 + portate 0
index = 0
res = model.predict(np.array([x_test[index]]))

# Noto che la rete ha polarizzato i valori
sns.distplot(res[0].flatten())
```

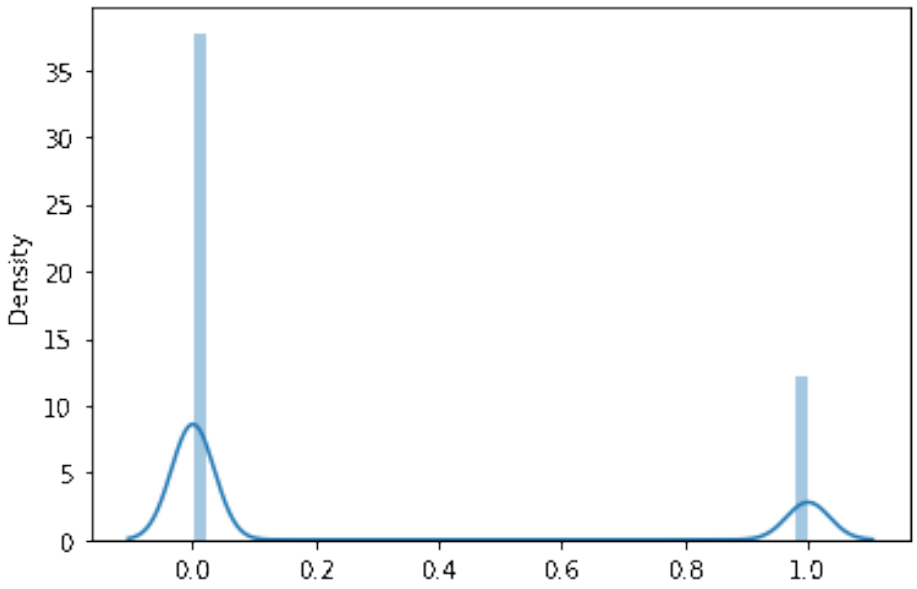

Figura 4.7: Results

```
# Approssimo i valori a binario in base ad una soglia
```

```
THRSHLD = 0.8

res[res >= THRSHLD] = 1
res[res < THRSHLD] = 0

plot_mat_accuracy(
    x_test[index][:img_shape].reshape(btms.shape[1], btms.
   shape[2]),
    res[0],
    y_test[index]
)
```

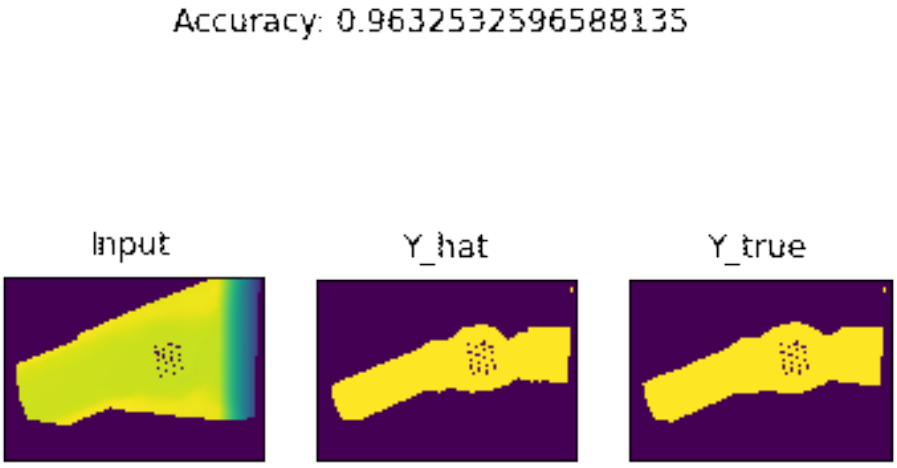

Figura 4.8: Accuracy

```
# Test Loss
test_losses = np.zeros(x_test.shape[0])

for i, x in enumerate(x_test):
  y_hat = model.predict(np.array([x]))
  y_true = y_test[i]

  test_losses[i] = get_mat_accuracy(y_hat, y_true)

# Train loss
```

```
train_losses = np.zeros(x_train.shape[0])

for i, x in enumerate(x_train):
  y_hat = model.predict(np.array([x]))
  y_true = y_train[i]

  train_losses[i] = get_mat_accuracy(y_hat, y_true)


print("Train accuracy: {}".format(train_losses.mean()))
print("Test accuracy: {}".format(test_losses.mean()))

for i, x in enumerate(x_train):
  res = model.predict(np.array([x]))

  plt.imsave("results/feed_binary/{}_diff_prt_{}.png".format(
   i, round(x[img_shape:][1], 3)),
              np.abs(y_train[i] - res[0])
  )
  plt.imsave("results/feed_binary/{}_true_prt_{}.png".format(
   i, round(x[img_shape:][1], 3)), y_train[i])
  plt.imsave("results/feed_binary/{}_pred_prt_{}.png".format(
   i, round(x[img_shape:][1], 3)), res[0])
```

Considerazioni

- L'attivazione ReLU "taglia" dei valori intermedi, la rete impara di conseguenza a prevede solo la matrice binaria finale per tutti gli step
- La BatchNormalization() ad ogni livello migliora le prestazioni
- La sigmoid ad ogni livello migliora le prestazioni
- I valori predetti sono polarizzati tra 0 e 1, è stata applicata una trasformazione per convertirli in binario
- Con troppi neuroni il modello rischia di pensare che chi siano piú celle allagate del previsto

- Con meno neuroni la rete lavora meglio, a dimostrazione del fatto che nonc c'è una relazione lineare tra la dimensione input e la dimensione della rete.

Grid Search			
CASI	**Layers**	**Train loss**	**Test Loss**
Caso 1	2048-4096-2048	0.959	0.744
Caso 2	1024-2048-1024	0.963	0.948
Caso 3	1024-1024-1024	0.960	0.779
Caso 4	256-512-256	0.972	0.972
Caso 5	128-256-128	0.971	0.972
Caso 6	64-128-64	0.973	0.960

Results

I risultati mostrano un campione dei risultati ottenuti.
Da sinistra verso destra troviamo l'immagine originale, la predizione effettuata dalla rete e la matrice differenza che ci è utile per poter evidenziare gli errori.

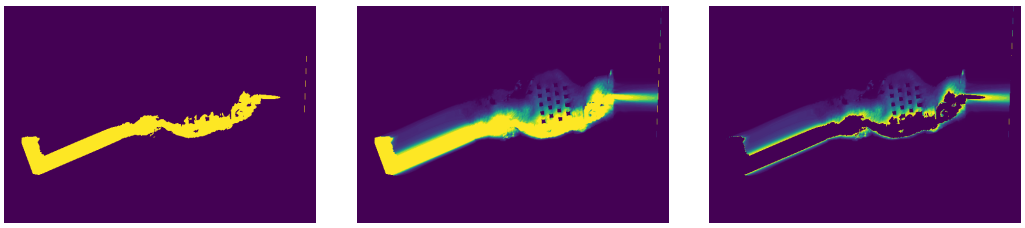

Figura 4.9: **True 1 Prediction 1 Diff 1**

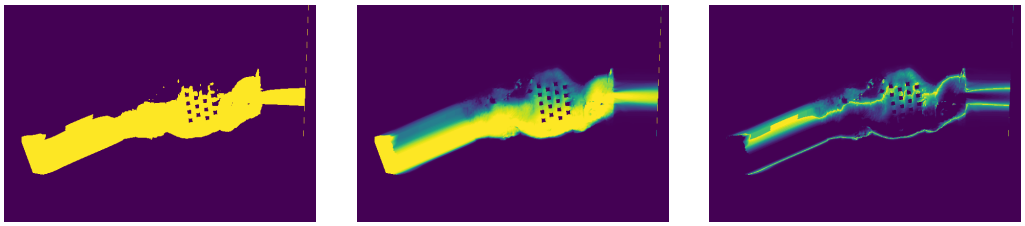

Figura 4.10: **True 25 Prediction 25 Diff 25**

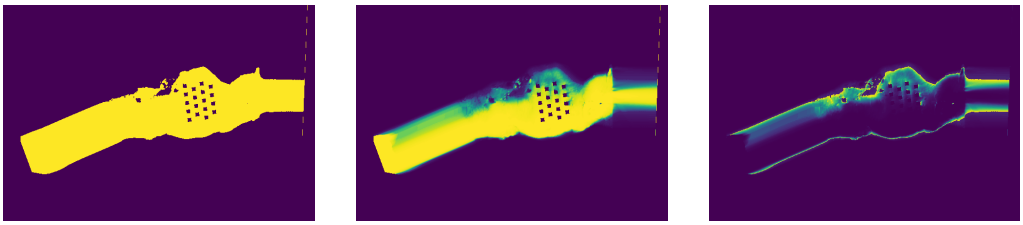

Figura 4.11: **True 50 Prediction 50 Diff 50**

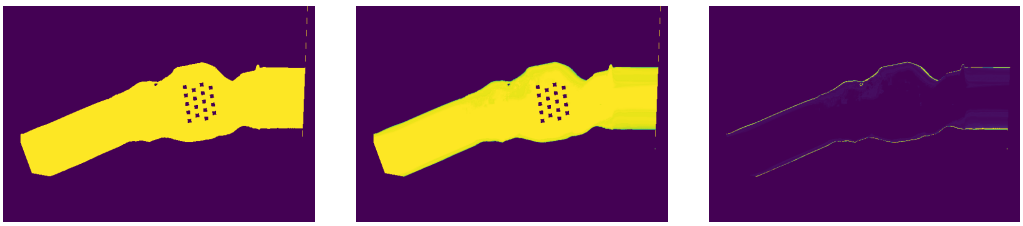

Figura 4.12: **True 60 Prediction 60 Diff 60**

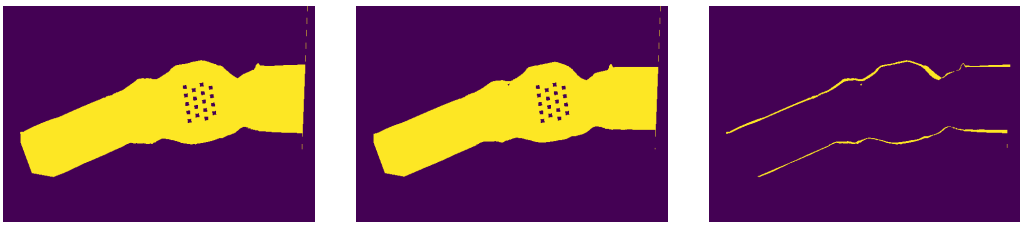

Figura 4.13: **True 75 Prediction 75 Diff 75**

4.4 Confronti con altri scenari

Al fine di poter valutare al meglio le prestazioni della rete, sono stati effettuati training anche su scenari differenti dal fiume toce.
Lo scenario che prenderemo in considerazione è il caso test del torrente Baganza (8 milioni di celle).
Anche per questo scenario sono stati generati 100 file BCC con valori di portata che variano tra 50(BCC-000) E 850(BCC-099).

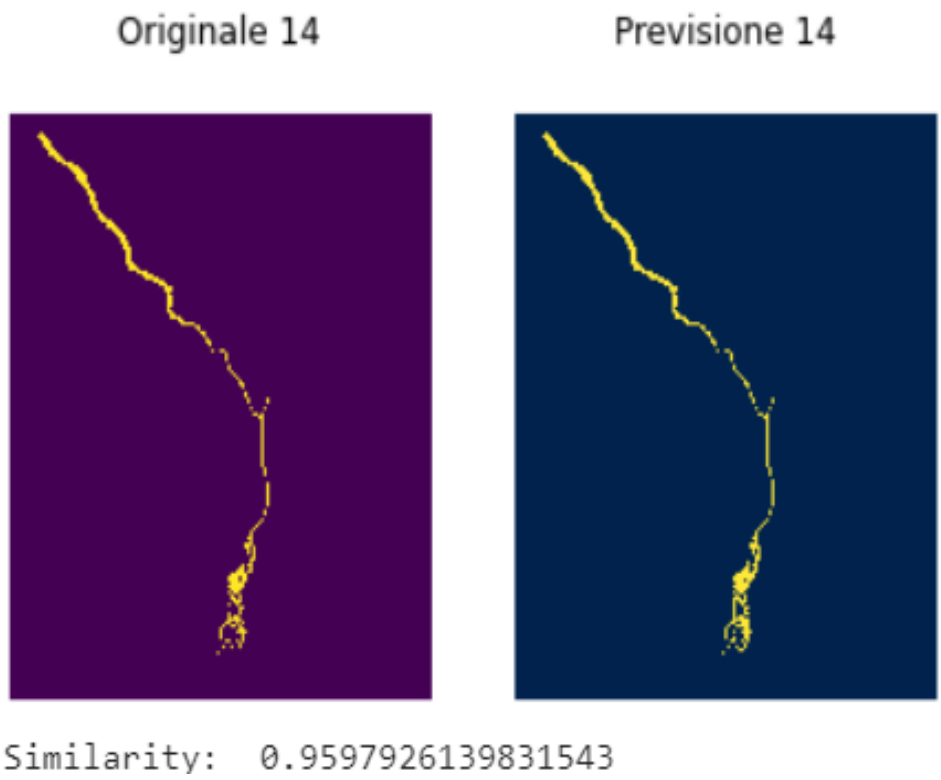

Figura 4.14: Prediction

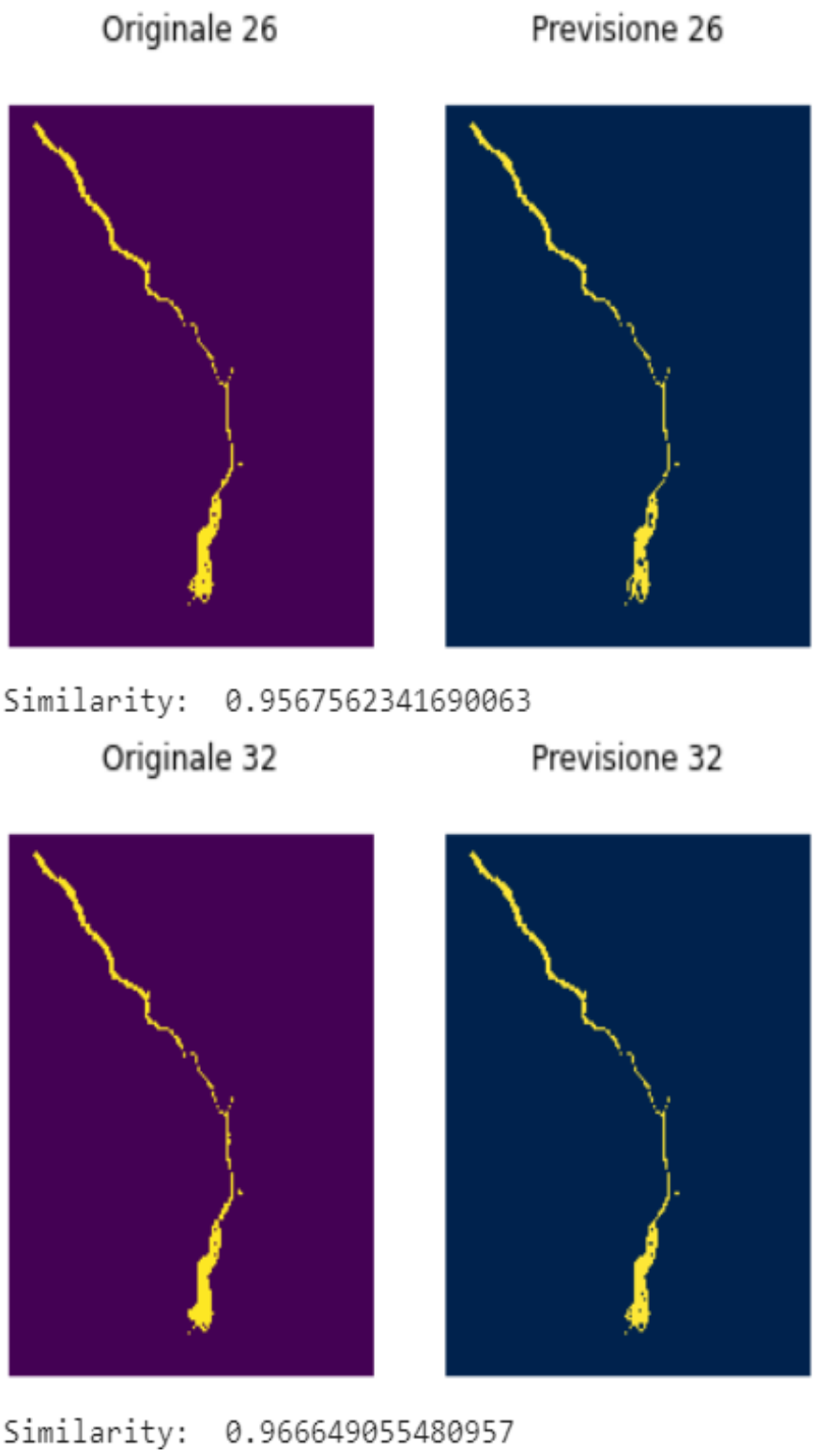

Figura 4.15: Prediction

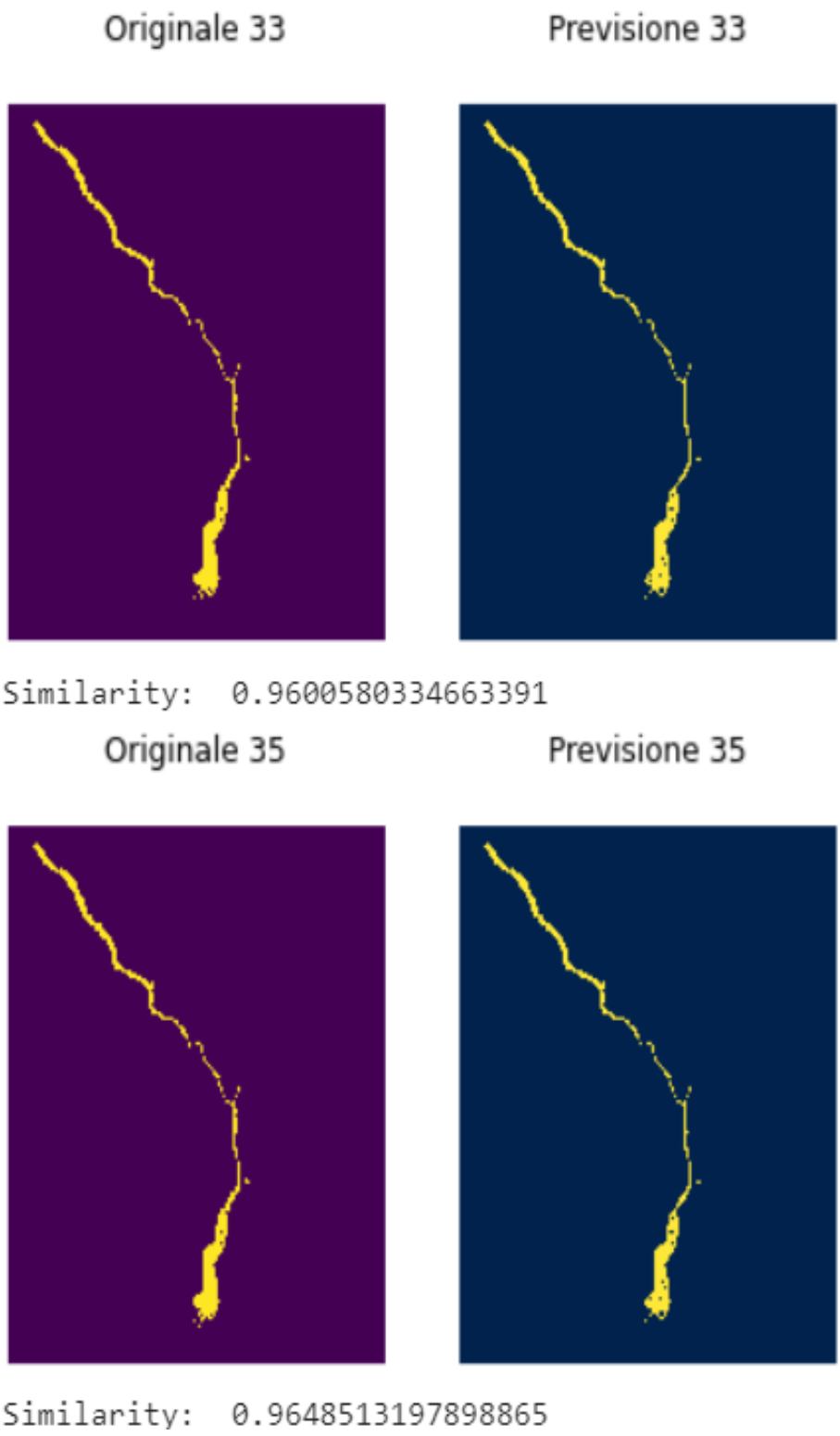

Figura 4.16: Prediction

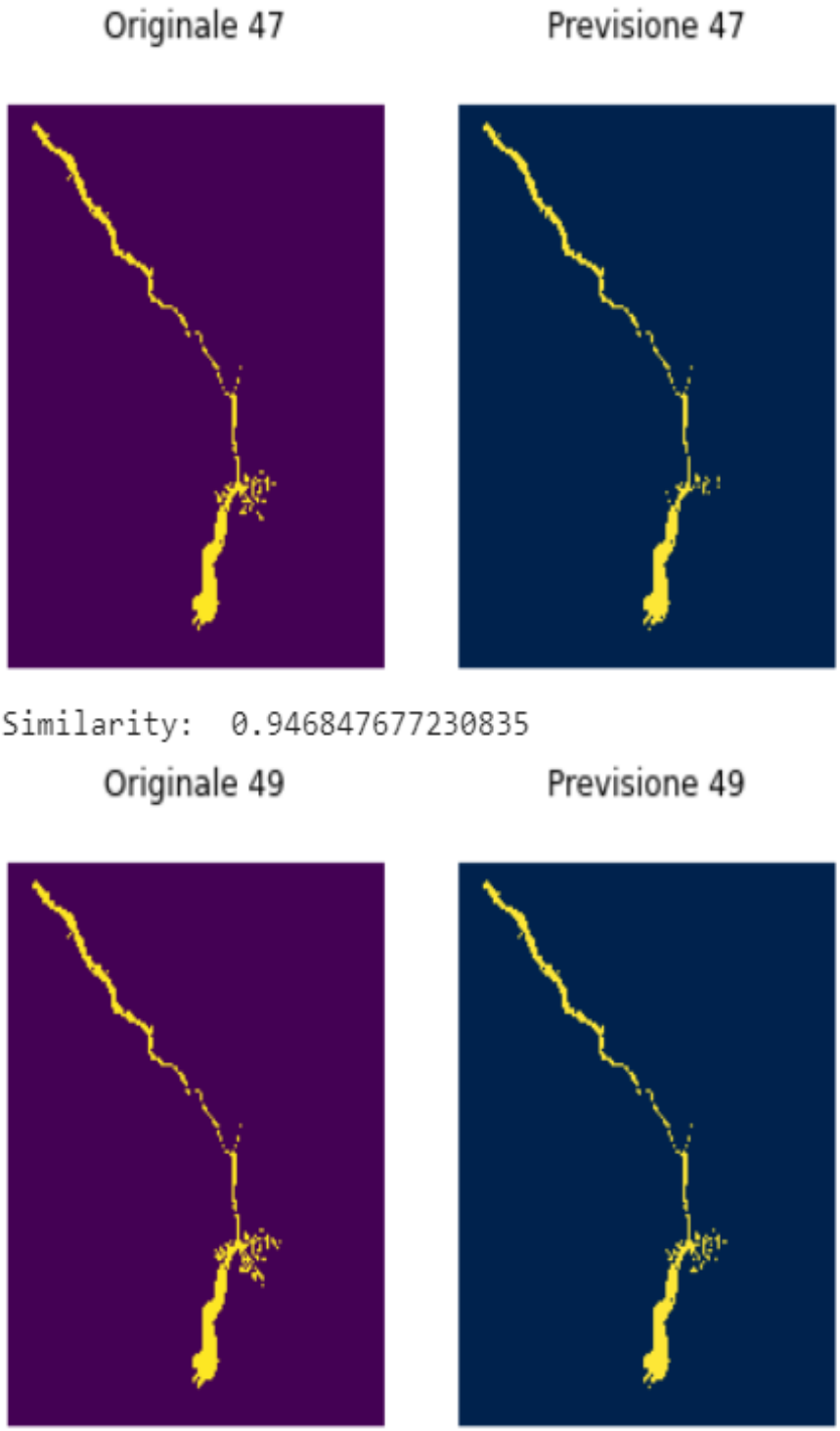

Figura 4.17: Prediction

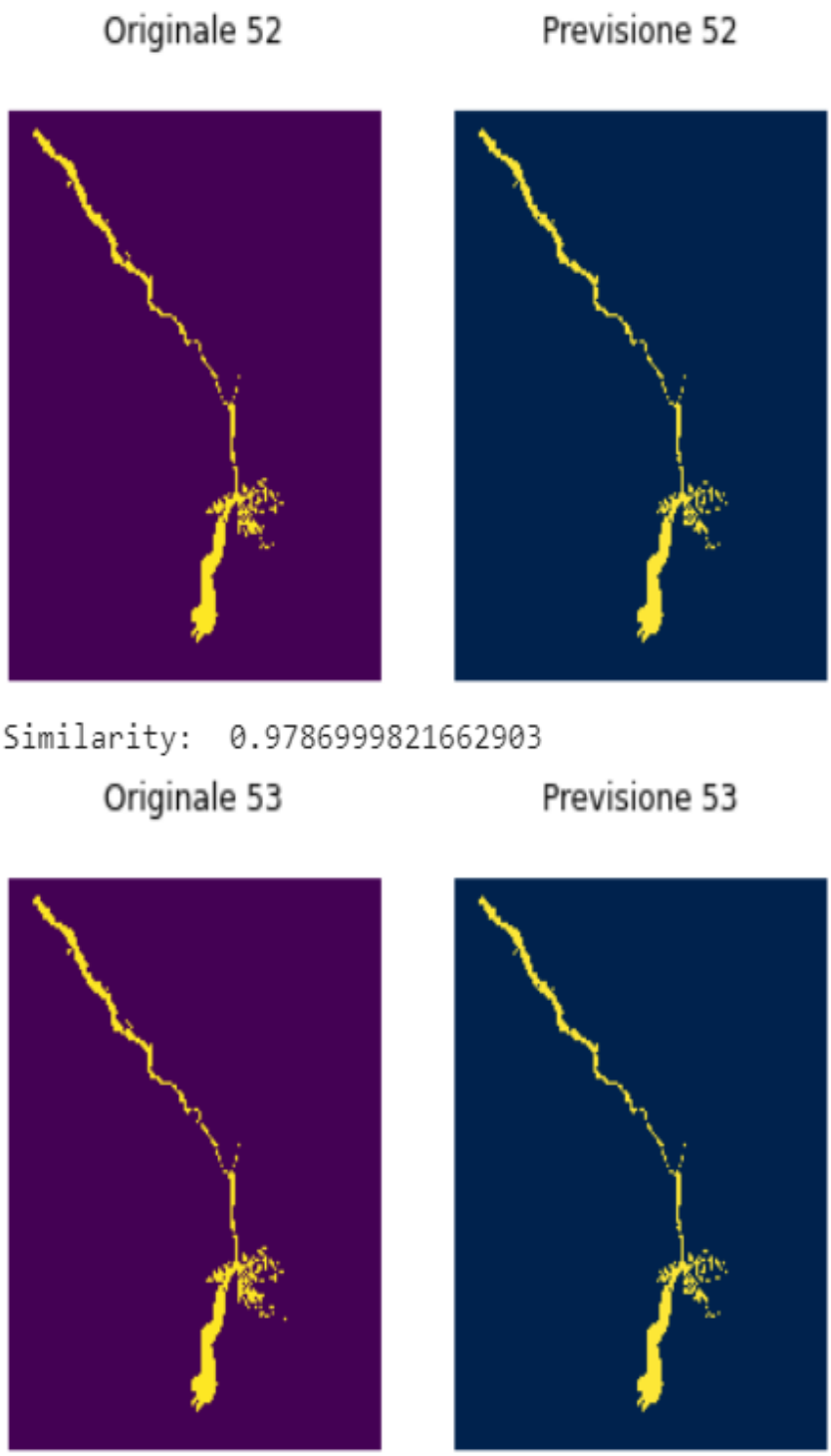

Figura 4.18: Prediction

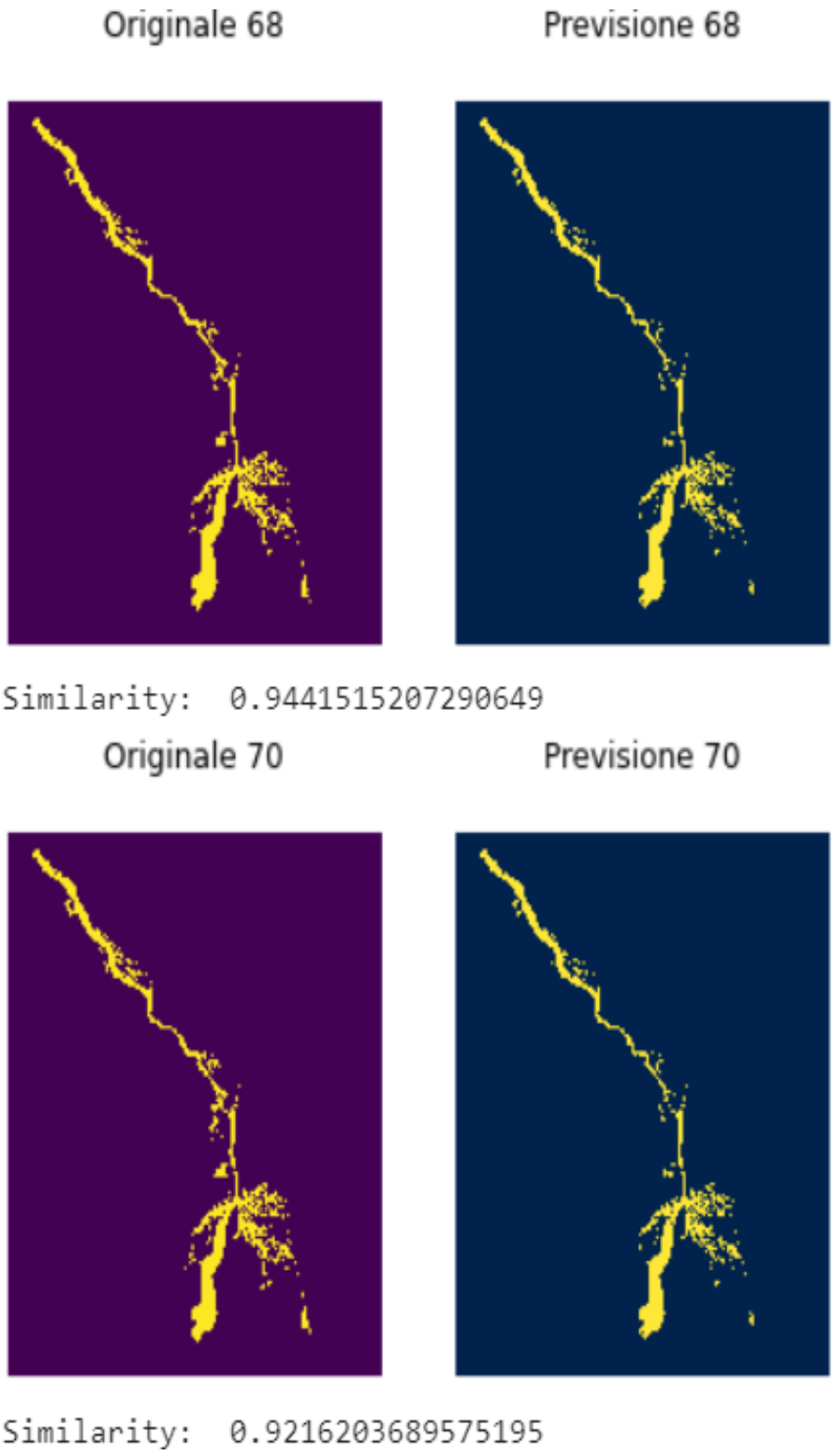

Figura 4.19: Prediction

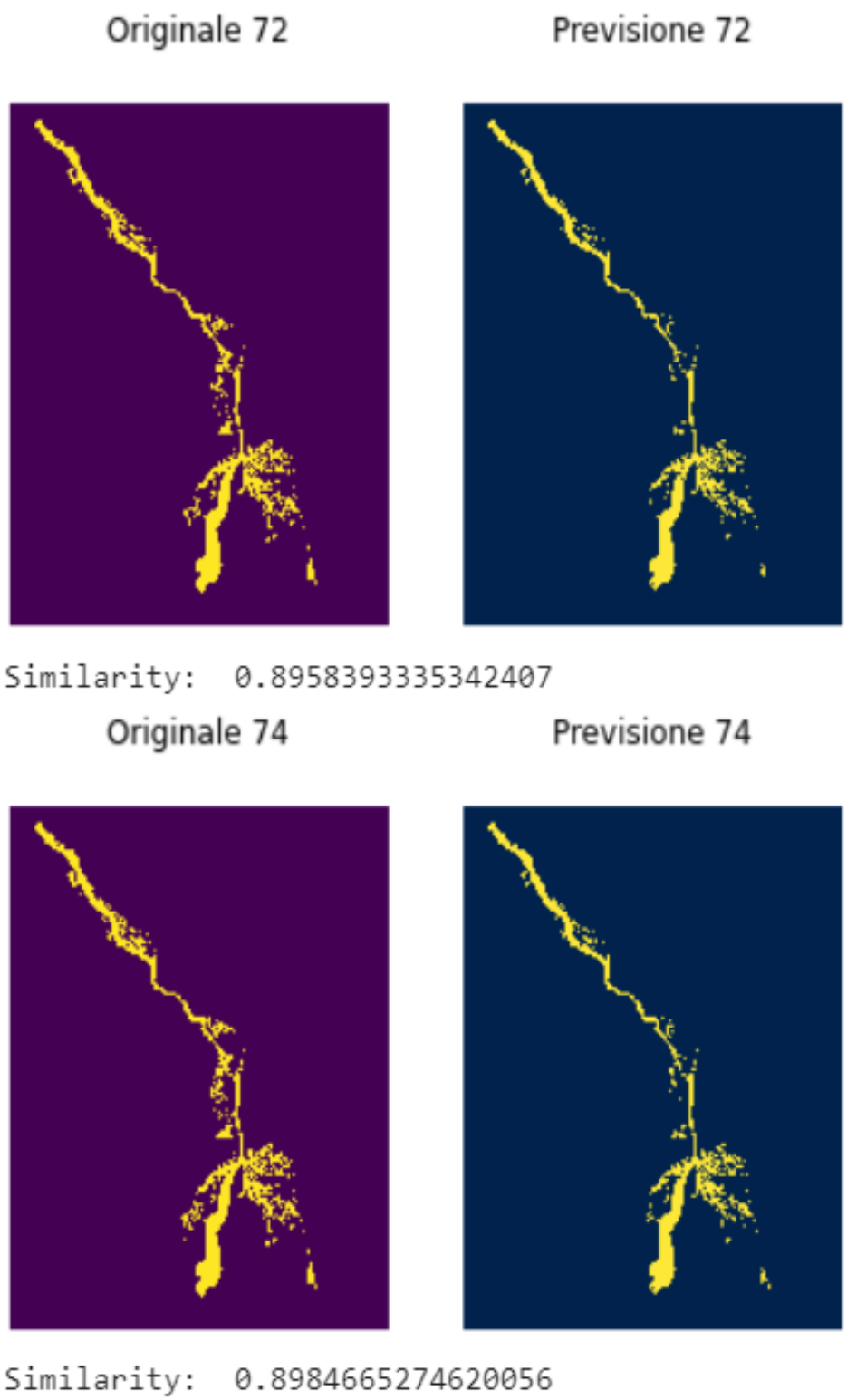

Figura 4.20: Prediction

Come possiamo vedere dalle immagini la rete apprende progressivamente i frame migliorando sempre di piú fino a raggiungere un picco del 98% circa intorno all'immagine 52.
L'accuracy tende a scendere leggermente negli ultimi frame dove la struttura risulta essere molto piú complessa.
A seguito dei test effettuati la previsione della seconda metà dei frame risulta migliorare raddoppiando questi ultimi.

Capitolo 5

Conclusioni

La presente attività di tesi ha avuto come obiettivo lo sviluppo di una rete feed forward per la predizione di zone asciutte e bagnate nelle alluvioni.
Abbiamo quindi descritto la struttura della rete neurale utilizzata analizzandone inoltre diverse configurazioni.
Sono infatti stati effettuati diversi training con configurazioni diversi analizzando come la rete impari con e senza batimetria(BTM) in ingresso.
La rete neurale proposta si è dimostrata efficace anche in relazione a scenari molto complessi, come il caso del torrente Baganza.
L'apprendimento di questo secondo scenario si è rivelato notevolmente piú oneroso in termini di tempo di calcolo, infatti, se da un lato il simulatore impiega un tempo di elaborazione di 6 minuti circa sul cluster HPC per uno scenario come quello del Toce, dall'altro, lo scenario del Baganza (8 milioni di celle) impiega 20 minuti sulla k40.
Per quanto riguarda il tempo di addestramento, il caso del Toce impiega circa 30 minuti mentre il caso del baganza impiega circa 50 minuti. Questo fatto potrebbe essere oggetto di studi futuri.

5.1 Ipotesi migliorative

Come abbiamo avuto modo di notare, aumentare il numero di frames nell'area piú complessa del terreno oggetto di studio può migliorare notevolmente l'accuracy di apprendimento. Infatti come si ha avuto modo di osservare, i primi allagamenti (gli ultimi nel caso del Baganza) presentano caratteristiche molto diverse dagli altri, e dato che queste caratteristiche spariscono con l'avanzare delle immagini, queste spariscono dal dataset stesso. Nel nostro studio ci siamo concentrati nello sviluppo di reti neurali feed-forward, la realizzazione di reti piú complesse potrebbe avere applicazioni interessanti in relazione all'obiettivo della tesi. Vi sono inoltre molti altri problemi di interessante studio in futuro come prevedere l'evoluzione temporale della simulazione con buona approssimazione del simulatore ma con tempi di calcolo previsti 1000 volte piú veloci. Potrebbe inoltre essere di interessante studio la previsione delle matrici velocità VVX e VVY.

Bibliografia

[1] Andrea Missinato: *Reti neurali demistificate*
https://www.spindox.it/it/blog/ml1-reti-neurali-demistificate/

[2] *Surfer;* https://www.goldensoftware.com/products/surfer

[3] *Qgis:* https://qgis.org/it/site/

[4] *Matlab:* https://it.mathworks.com/products/matlab.html

[5] *Notepad++:* https://notepad-plus-plus.org/downloads/

[6] *WinSCP:* https://winscp.net/eng/download.php

[7] *Cygwin64("openssh"):* https://cygwin.com/packages/summary/openssh.html

[8] Kingma, Diederik P. & Ba, Jimmy
Adam: A Method for Stochastic Optimization, International Conference on Learning Representations, Dec. 2014

[9] *Keras:* https://keras.io/

[10] *Tensorflow:* https://www.tensorflow.org/

[11] Renato Vacondio and Alessandro Dal Palù and Alessia Ferrari and Paolo Mignosa and Francesca Aureli and Susanna Dazzi
A non-uniform efficient grid type for GPU-parallel Shallow Water Equations models, Enviromental modeling & software

[12] Ioffe, Sergey & Szegedy, Christian.
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Proceedings of Machine Learning Research, Feb. 2015

Ringraziamenti

Desidero ringraziare il Professor Dal Palú per tutti gli insegnamenti ricevuti durante questi 3 anni e per avermi dato l'opportunità di svolgere un lavoro di tirocinio e tesi per me molto interessanti.
Desidero inoltre ringraziare il Dott. Ing. Renato Vacondio che mi ha fornito dati e supporto al tirocinio mostrandosi sempre disponibilie a consigli e feedback.
Un sentito ringraziamento ai miei genitori, Giuseppe e Barbara e a mio fratello Leroy, che non solo mi hanno permesso di poter svolgere gli studi, ma mi hanno anche sempre mostrato il massimo supporto. Ringrazio anche i miei zii, Emiliano e Linda, mia cugina Asia e le mie nonne, Rita e Pina.
Un ringraziamento speciale và ai miei amici e compagni di corso Francesco, Valerio, Cristian, Elias, Giulio, Marco, Luca e Sergio, che mi hanno accompagnato in questi 3 anni e li hanno resi per me davvero speciali.