

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

**RICOSTRUZIONE DELLA
TEMPERATURA SUPERFICIALE
MARINA VIA
APPRENDIMENTO AUTOMATICO**

**Relatore:
Chiar.mo Prof.
ANDREA ASPERTI**

**Presentata da:
ANGELO GRECO**

**Sessione II
Anno Accademico 2023/2024**

Abstract

La temperatura superficiale marina (*SST*, *Sea Surface Temperature*) è uno dei principali indicatori climatici ed una variabile essenziale per la previsione meteorologica. Tuttavia, le misurazioni satellitari effettuate per l'SST spesso hanno dati mancanti o di scarsa qualità a causa della presenza di coperture nuvolose o altre ostruzioni, ed i metodi statistici tradizionali come le interpolazioni spaziali e temporali non sono in grado di conservare i dettagli più fini delle dinamiche oceaniche. Questa tesi propone un approccio per la ricostruzione dei dati mancanti basato sull'apprendimento automatico, utilizzando reti neurali convoluzionali (*CNN*, *Convolutional Neural Network*). In particolare, viene mostrato lo sviluppo e l'implementazione di una variante della rete U-Net con blocchi residui, applicata sul Mar Mediterraneo circondante l'Italia. I modelli più performanti hanno ottenuto un RMSE medio di 0.3015 gradi Celsius, dimostrando un miglioramento nella qualità delle ricostruzioni rispetto a metodi tradizionali.

Indice

Abstract	i
Introduzione	1
Struttura della tesi	2
1 Creazione dei dataset e preprocessing dei dati	3
1.1 Processo di download	3
1.2 Caratteristiche dei dati	5
1.3 Preprocessing	5
1.3.1 Land-Sea Mask	6
1.3.2 Normalizzazione e salvataggio dati	7
1.4 Baseline	7
1.5 Suddivisione dei dataset	9
1.6 Generatore	10
1.6.1 Versioni precedenti e copertura artificiale	11
1.6.2 Generatore corrente	13
2 Strumenti e tecnologie usate	16
2.1 Leonardo	16
2.2 Deep Learning	18
2.3 Python	19
2.4 Librerie	20
2.4.1 TensorFlow	20
2.4.2 Keras	20
2.4.3 NumPy	20
2.4.4 Matplotlib	20
3 Implementazione e Valutazione	21
3.1 Loss function	21
3.1.1 Metriche	21
3.2 Architettura della rete	22

3.2.1	Operazione di convoluzione	23
3.2.2	Modello utilizzato	24
3.3	Addestramento	25
3.4	Valutazione	28
3.4.1	Risultati	28
3.4.2	Confronto DINCAE	32
	Conclusioni	34
	Bibliografia	35

Elenco delle figure

1.1	Preview del sito OceanColor	4
1.2	Esempio di dati SST sul Mediterraneo	5
1.3	Land-sea Mask	6
1.4	Esempio di baseline	9
1.5	Esempio di immagine elaborata dal generatore	12
1.6	Prodotto completo del generatore corrente	15
2.1	Preview di Leonardo da terminale	17
2.2	Schema di una rete profonda	19
3.1	Esempio di convoluzione	23
3.2	Schema del modello finale	24
3.3	Grafici di allenamento della ‘val_loss’	27
3.4	Esempi di ricostruzioni	30
3.5	Confronto con metodi tradizionali	31
3.6	Esempio dati DINCAE	33

Introduzione

La temperatura superficiale marina (o SST, *Sea Surface Temperature*) influenza fortemente le masse d'aria nell'atmosfera, fino a una distanza di 35-40 km [1], quindi è un fattore predittivo fondamentale per la circolazione atmosferica e, di conseguenza, per la previsione meteorologica e la modellizzazione climatica nel loro complesso.

La raccolta di dati SST avviene principalmente *via satellite*, con sensori infrarossi. Questo metodo è molto più efficiente rispetto alle misurazioni *in situ* con sensori appositi, tuttavia, questa raccolta è spesso ostacolata da ostruzioni, principalmente dovute alla presenza di nuvole. Queste ostruzioni creano dei 'buchi' nei dati osservati [2], rendendo alcune aree del mare non misurabili per lunghi periodi di tempo. Di conseguenza, si ottengono rappresentazioni incomplete e potenzialmente fuorvianti delle condizioni oceaniche che, se non trattate adeguatamente, possono compromettere la qualità delle previsioni effettuate.

Tradizionalmente, il trattamento di questi dati mancanti è affrontato tramite metodi statistici, che tramite interpolazione spaziale e/o temporale stimano i valori dei dati coperti utilizzando le informazioni disponibili nelle vicinanze [3, 4]. Tuttavia, tali approcci non sono in grado di catturare la complessità delle dinamiche oceaniche, specialmente in presenza di fenomeni oceanografici come correnti, vortici o variazioni termiche rapide [5, 6].

Negli ultimi anni, l'apprendimento automatico, e in particolare il deep learning, ha aperto nuove possibilità per affrontare questa sfida [7, 8, 9, 10]. I modelli di deep learning sono in grado di apprendere rappresentazioni complesse e non lineari dei dati, caratteristica che li rende particolarmente adatti a ricostruire le misurazioni SST offuscate in modo più accurato rispetto ai metodi tradizionali. Una panoramica sui metodi di ricostruzione per dati oceanografici, sia statistici che basati sull'intelligenza artificiale, è disponibile in [11].

Il lavoro discusso in questa tesi è stato effettuato in collaborazione con il collega Alessandro Testa, con l'Istituto di Geofisica e con il CMCC, Centro euro-Mediterraneo

sui Cambiamenti Climatici. Questa tesi esplora l'applicazione di tecniche di deep learning per la ricostruzione dei dati mancanti relativi alla temperatura superficiale marina, con l'obiettivo di migliorare la qualità delle ricostruzioni rispetto ai metodi preesistenti. Verrà posta particolare attenzione ai dati usati nei modelli di deep learning e al loro *preprocessing*. Inoltre, verranno presentati i risultati sperimentali ottenuti tramite l'uso della rete più performante, dimostrando il potenziale di queste tecniche nel fornire ricostruzioni accurate in condizioni reali.

Struttura della tesi

La tesi è strutturata nei seguenti capitoli:

- **Capitolo 1**, dove si mostrano i dati di partenza, le loro caratteristiche e il *preprocessing* effettuato;
- **Capitolo 2**, dedicato a introdurre le tecnologie e gli strumenti utilizzati;
- **Capitolo 3**, dove si mostra l'architettura della rete più performante utilizzata nella sperimentazione, assieme a dettagli sull'addestramento e ai risultati ottenuti.

Capitolo 1

Creazione dei dataset e preprocessing dei dati

I dati utilizzati per la sperimentazione sono stati raccolti dal sensore **MODIS** (*Moderate Resolution Imaging Spectroradiometer*) [12] del satellite **Aqua**, lanciato dalla NASA nel Maggio del 2002. Questo sensore opera su 36 bande spettrali (21 con lunghezza d'onda tra 0.4 e 3.0 μm ; 15 tra 3-14.5 μm) e a diverse risoluzioni spaziali (2 bande a 250 m, 5 bande a 500 m e 29 bande a 1 km). La temperatura della superficie del mare è misurata usando le bande termiche a infrarossi da 11 e 12 μm .

I dati d'interesse catturati da questo sensore sono le misurazioni SST giornaliere diurne e notturne, a una risoluzione spaziale di 4,64 km. In particolare, l'area d'interesse è quella del Mar Mediterraneo: le coordinate selezionate coprono una latitudine compresa tra 30° e 46° e una longitudine tra -5° e 36°, per un totale di 384x984 punti di osservazione. Nel corso della sperimentazione, ci siamo concentrati sull'area attorno alla sola Italia, delineando 256x256 punti di osservazione con latitudine compresa tra 35.33° e 46.0° e longitudine tra 7.92° e 18.58°.

Sono state utilizzate le misurazioni dal 4/7/2002 al 31/12/2023. Tuttavia, la mancanza di alcune di queste misurazioni Aqua-MODIS comporta la presenza di periodi senza dati, come quello che va dal 1/4/2022 al 16/4/2022.

1.1 Processo di download

Per il download dei dati selezionati, effettuato dal sito *OceanColor*¹ della NASA, è necessario creare un account. In seguito, si seleziona dal browser apposito la regione d'interesse di cui scaricare i dati, nonché l'arco temporale delle misurazioni. Dopo aver fatto questo, verrà inviata una mail al proprio account, dove è necessario confermare

¹<https://oceancolor.gsfc.nasa.gov/>

la richiesta di estrazione rispondendo a essa (serve assicurarsi che la mail di risposta appartenga alla mail dell'account che ha richiesto il download).

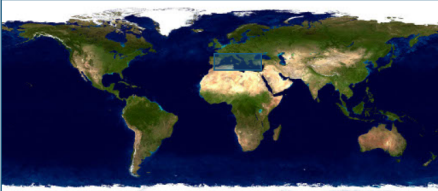
Product Status	Sensor	Product	Period	Resolution
Standard	Aqua-MODIS	Sea Surface Temperature (Daily	4km
Start Date: 2002-07-04		End Date: 2023-12-31		Type: <input type="checkbox"/> Binned <input checked="" type="checkbox"/> Mapped <input type="checkbox"/> PNG
Data Retrieval Method: <input type="radio"/> Download <input checked="" type="radio"/> Extract				
Click and drag the mouse across an area of interest on the map, then click 'Preview Region' to verify your selection. When satisfied, click 'Review Order' to verify your selections.				
<div style="border: 1px solid #ccc; padding: 5px;"> <div style="background-color: #007bff; color: white; padding: 2px 5px; display: inline-block; cursor: pointer;">Preview Region</div>  <div style="margin-top: 10px;"> <p>N: 46</p> <p>W: -5 E: 36</p> <p>S: 30</p> </div> </div>				
<div style="background-color: #28a745; color: white; padding: 5px 15px; display: inline-block; cursor: pointer;">Review Order</div>				

Figura 1.1: Schermata di download del sito OceanColor, con la regione d'interesse selezionata

Una volta fatto ciò, serve scaricare il file `http_manifest.txt` dalla mail successiva, che contiene i dati sotto forma di URL. Per estrarli, serve utilizzare un gestore di download appropriato, come `wget` o `cUrl`. Nel nostro caso, utilizzando `wget`, l'estrazione è effettuata tramite il comando:

```
wget --user=[nome_utente] --password=[password] --auth-no-challenge=
on -P [path\di\destinazione\files] -i http_manifest.txt
```

Questo scarica una cartella compressa `.tar` nel percorso di destinazione indicato. L'ultimo passaggio è l'estrazione di questa cartella con strumenti di decompressione appropriati come **7-Zip** o **WinRAR**, ed i file risultanti sono pronti all'uso.

Tali file sono in formato `.nc`, anche detto **NetCDF** (*Network Common Data Form*) [13]: è un formato di dati progettato appositamente per l'archiviazione e la manipolazione di dati scientifici, in particolare quelli relativi a variabili multidimensionali (come la temperatura, nel nostro caso).

Una delle librerie Python che permettono l'interazione con questi file è `xarray` [14], che supporta nativamente il formato NetCDF, semplificando l'accesso ai dati contenuti, la loro visualizzazione, e la loro modifica attraverso varie operazioni come filtraggio e raggruppamento dati.

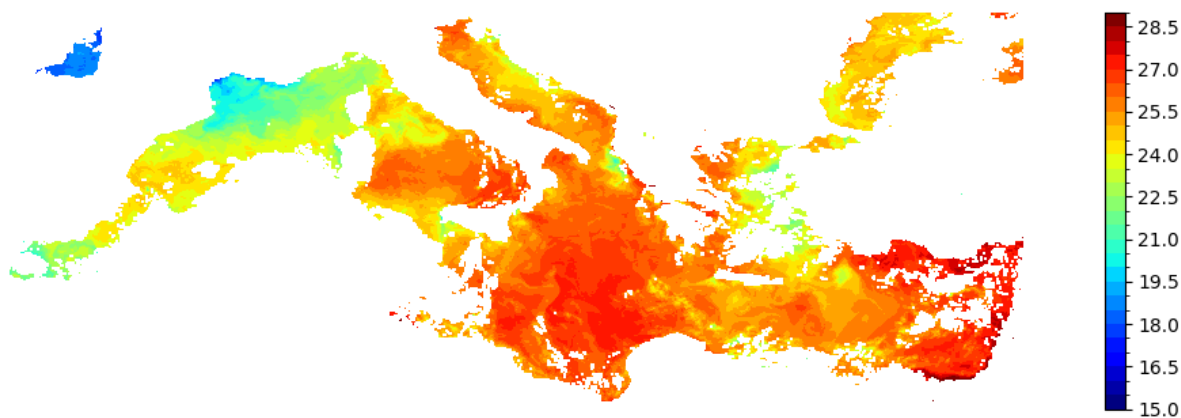


Figura 1.2: Dati SST notturni del 12 Luglio 2002. I valori mancanti (*NaN*) sono rappresentati in bianco.

1.2 Caratteristiche dei dati

Ogni file NetCDF include informazioni sulla struttura dei dati al suo interno. Questo significa che non solo memorizza i dati, ma anche i propri **metadati**² (ad esempio, descrizioni delle variabili, delle unità di misura, e delle coordinate) che ne facilitano l'interpretazione. Nel nostro caso troviamo informazioni utili come l'ora di inizio e fine misurazione, e la temperatura minore e maggiore riportata. I dati Aqua-MODIS sono composti principalmente da queste due matrici:

- **sst / sst4** - Le misurazioni SST in gradi Celsius. I punti di osservazione mancanti, per via della presenza di terraferma, nuvole o scarsa qualità, sono rappresentati come NaN, *Not a Number*. Nell'area mediterranea, la temperatura minima e massima riportata è rispettivamente di 0.09° e 34.88°.
- **qual_sst / qual_sst4** - Indicatori di qualità per i punti di osservazione. Vanno da 0 (migliore) a 5 (peggiore). I nostri dati hanno solo indicatori da 0, 1 e 2, che coprono rispettivamente il 68.29%/30.56%/1.14% dei dati diurni e il 79.10%/20.73%/0.17% di quelli notturni.

1.3 Preprocessing

Con **prelaborazione** o *preprocessing* dei dati ci si riferisce al processo di preparazione e trasformazione dei dati 'grezzi' in un formato adatto a essere utilizzato dai modelli di apprendimento automatico. Questa fase è cruciale per migliorare la qualità dei dati

²Descrizione dettagliata su: *Ocean Data Product Users Guide*

e, di conseguenza, le prestazioni del modello stesso (input di scarsa qualità produrranno risultati di scarsa qualità, indipendentemente da quanto il modello usato sia adatto al problema in questione).

Per i nostri dati, una ‘pulizia’ eccessiva non è necessaria, poichè come già menzionato, le misurazioni con indicatori di qualità 3 o superiore (e quindi pessime) sono già assenti. Tuttavia, molte misurazioni registrate non riguardano il mare, ma corpi d’acqua entroterra come fiumi o laghi. Queste misurazioni solitamente sono molto diverse da quelle del mare circostante, quindi risultano non solo irrilevanti ma anche potenzialmente dannose per le performance del nostro modello.

1.3.1 Land-Sea Mask

Al fine di rimuovere queste misurazioni, è necessario delineare quali sono i punti di osservazione che rappresentano il mare, e quali rappresentano invece la terraferma. Questo viene effettuato tramite una **maschera**, una matrice binaria che identifica i punti di nostro interesse e permette di ignorare gli altri. In particolare, nel contesto delle scienze climatiche, oceanografiche e meteorologiche, una maschera che separa sezioni di mare da sezioni di terra prende il nome di *land-sea mask* (maschera terra-mare).

Al fine della nostra sperimentazione, la land-sea mask usata è stata condivisa dal CMCC, Centro euro-Mediterraneo sui Cambiamenti Climatici. Tale maschera è stata creata in modo tale da essere nella stessa risoluzione e sulle stesse coordinate dei nostri dati, coprendo quindi 384x984 punti di osservazione. Come per i dati stessi, la land-sea mask è stata poi ritagliata per inquadrare l’area circostante l’Italia, formando una maschera di 256x256 punti. I punti di osservazione sul mare sono identificati come 1, mentre i punti sulla terraferma sono identificati come 0.



Figura 1.3: A sinistra, la land-sea mask costruita considerando come mare tutti i punti misurati almeno una volta dal satellite Aqua. A destra, la land-sea mask del CMCC, usata nel corso delle sperimentazioni. In percentuale, otteniamo un’area al 59.47% coperta d’acqua.

1.3.2 Normalizzazione e salvataggio dati

La normalizzazione è una tecnica di preprocessing usata per ridimensionare i dati di input in modo che abbiano una scala comune. Consiste nel ridimensionare i valori in un intervallo predefinito (di solito tra 0 e 1), tecnica detta *Min-Max Scaling*, oppure nel dividerli per la loro deviazione standard dopo averne sottratto la media, tecnica detta *Standardization*.

La normalizzazione viene effettuata perchè i modelli di machine learning sono sensibili alle differenze di scala, quindi è fondamentale che tutte le variabili abbiano una scala simile: senza normalizzazione, le variabili con valori molto grandi possono dominare il processo di apprendimento, influenzando in modo sproporzionato il modello pur non essendo le più rilevanti per il problema [15].

Per i nostri dati, inizialmente è stata utilizzata la tecnica *Min-Max Scaling* per la normalizzazione, basata sui valori massimi e minimi misurati sia nei dati diurni che notturni nel Mediterraneo, con un intervallo compreso tra -1 e 1. Tuttavia, l'approccio precedente è stato successivamente sostituito da una tecnica più avanzata, il cosiddetto metodo 'residuale'.

In questo nuovo approccio, il dataset è composto dai **dati 'residui'**, cioè quelli ottenuti sottraendo la *baseline* (una media statistica) ai valori originali delle misurazioni SST. I dati risultanti vengono quindi normalizzati utilizzando la tecnica della *standardizzazione*, cioè dividendo tali dati per la loro deviazione standard dopo averne sottratto la media.

Questa normalizzazione basata sui residui permette al modello di concentrarsi meglio sulle deviazioni della temperatura rispetto alla *baseline*, migliorando quindi l'apprendimento delle variazioni significative nei dati.

I dati vengono infine salvati sotto forma di file **.numpy**, formato binario nativo della libreria Python **NumPy** per il salvataggio e il caricamento di array multidimensionali (*ndarray*) [20]. Si usa il comando `numpy.save(' [path/nome_file.npy] ', [nome_variabile])` per salvare un array multidimensionale in un file e il comando `[nome_variabile] = numpy.load(' [path/nome_file.npy] ')` per caricarlo. Oltre alle misurazioni SST, vengono salvate anche le serie di date delle misurazioni giornaliere diurne e notturne, in seguito usate per la creazione della *baseline* e per mantenere congruenza temporale nell'addestramento dei modelli.

1.4 Baseline

Uno dei dati di maggiore importanza di questo lavoro, passato a lungo come input al modello in fase di addestramento e poi utilizzato per l'aumentazione diretta dei dati, è

la *baseline*, array multidimensionale che rappresenta la media delle misurazioni effettuate per ogni giorno dell'anno nei dati disponibili: la nostra **climatologia giornaliera**. Nonostante si lavori con circa 21 anni di misurazioni giornaliere, se la baseline fosse generata tramite una media semplice e diretta, presenterebbe ancora piccoli 'buchi': aree che, per pura coincidenza, non sono mai state misurate in quel particolare giorno dell'anno.

Per riempire questi buchi, è necessario effettuare una forma di **interpolazione**, tecnica matematica utilizzata per stimare valori sconosciuti all'interno di un intervallo di dati noti. In altre parole, data una serie di punti di dati discreti, l'interpolazione cerca di 'riempire i buchi' stimando i valori intermedi in base a quelli circostanti.

L'interpolazione utilizzata in questo caso è un **blur gaussiano**, una tecnica basata sull'uso dei valori *spazialmente vicini* a quelli mancanti: viene applicato un filtro di convoluzione per 'sfumare' i valori validi circostanti e, grazie alla sfumatura, stimare quelli mancanti.

Il blur gaussiano è particolarmente utile perché, grazie alla sua distribuzione basata su una funzione gaussiana, dà maggiore peso ai valori vicini, riducendo progressivamente l'influenza dei punti più lontani.

Il processo di interpolazione è diviso in fasi:

1. **Sostituzione dei valori mancanti:** I valori mancanti (*NaN*) vengono sostituiti con degli zero nella matrice dei dati.
2. **Applicazione del filtro gaussiano:** Viene applicato un filtro gaussiano alla matrice risultante per creare una versione 'sfumata' dei dati.
3. **Generazione della matrice dei pesi:** Viene creata una seconda matrice che tiene traccia dei punti originariamente noti, assegnando loro un peso (0 ai punti originariamente mancanti, 1 a quelli noti).
4. **Applicazione del filtro gaussiano ai pesi:** Anche la matrice dei pesi viene sottoposta al filtro gaussiano, per distribuire correttamente i pesi dei dati mancanti.
5. **Divisione delle matrici:** La matrice sfumata dei dati viene divisa per la matrice dei pesi: questo compensa l'errore introdotto dalla sostituzione dei *NaN* con zero, ottenendo così una stima ragionevole dei valori originariamente mancanti. I *NaN* riappaiono solo nelle aree dove il filtro non ha dati utili per interpolare, ossia nelle posizioni in cui tutti i valori del filtro erano *NaN*.

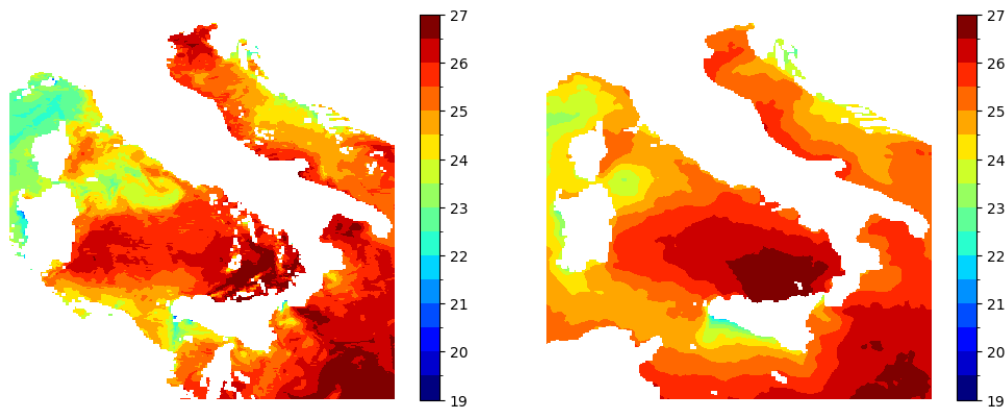


Figura 1.4: A sinistra, i dati SST notturni del 12 Luglio 2002 (*NaN* rappresentati in bianco). A destra, la baseline corrispondente.

L'importanza di una baseline accurata risiede nel suo ruolo di *riferimento statistico*. Essa permette infatti di confrontare le prestazioni del modello di deep learning con un approccio basato su semplici medie, e fornisce una sorta di *'fallback'*, ovvero un valore di riferimento in caso di forte copertura nuvolosa e conseguente assenza di dati sufficienti per effettuare previsioni accurate. Inoltre, è anche un modo per comunicare al modello qual'è il giorno dell'anno in analisi.

Il suo uso come strumento di augmentazione dei dati è già stato discusso nel capitolo **1.3.2**: sottraendo la baseline direttamente ai dati originali, e allenando la rete su questi nuovi dati residui, il modello può fare uso dei benefici della baseline senza la necessità di riceverla in input.

1.5 Suddivisione dei dataset

Nell'addestramento di una rete neurale, è importante non usare tutto il dataset disponibili per allenare la rete, e invece suddividerlo in due o tre insiemi distinti. Questa suddivisione è una pratica comune nel machine learning, e garantisce che il modello apprenda correttamente e sia in grado di generalizzare su dati mai visti prima, evitando fenomeni come l'*overfitting*.

Infatti, se si utilizzasse l'intero dataset per l'addestramento, il modello potrebbe adattarsi perfettamente ai dati a disposizione (con il relativo aumento apparente delle performance), ma fallirebbe nel generalizzare a nuovi dati, rendendosi inefficace in applicazioni reali.

Questi tre insiemi distinti sono:

- **training set**: la porzione di dati utilizzata per addestrare la rete, aggiornandone i parametri (pesi e bias). Più dati vengono usati per allenare il modello, maggiori

le performance: un training set dev'essere abbastanza grande da permettere alla rete di identificare correttamente le strutture e le caratteristiche dei dati.

- **validation set**: utilizzato per valutare il modello durante l'addestramento, ma senza aggiornare i pesi della rete. Serve a monitorare le prestazioni del modello su dati non visti, consentendo di individuare quando l'allenamento inizia a perdere efficacia e si verifica *overfitting*, evitando quindi di produrre modelli troppo specializzati e fuorvianti. In questa fase, si possono anche ottimizzare iperparametri come il tasso di apprendimento o il numero di epoche di addestramento.
- **test set**: un insieme riservato alla valutazione finale delle prestazioni del modello, utilizzando dati completamente sconosciuti. Questo insieme permette di stimare quanto il modello sia capace di generalizzare su nuovi input, fornendo una misura delle sue performance in applicazioni reali. È importante che il test set non venga mai utilizzato durante l'allenamento, in modo da ottenere una valutazione imparziale.

Nelle prime sperimentazioni, i dati SST giornalieri diurni e notturni erano combinati in un unico dataset, passato in input alla rete. Per avere maggior controllo sui dati passati in input, questo dataset è stato suddiviso nella sua componente diurna e notturna.

In seguito questo dataset combinato è stato suddiviso in training, validation e testing set, con proporzioni del 75%/15%/10% rispettivamente. Questa suddivisione è stata effettuata con la funzione `train_test_split()` della libreria **sklearn**, contenente strumenti utili per l'apprendimento automatico.

Inizialmente i dati venivano assegnati ad ogni set in modo casuale, ma per avere continuità temporale nei dati e assicurare di poter utilizzare i giorni precedenti o successivi rispetto al giorno analizzato, è stato rimosso il fattore di casualità: il dataset è stato suddiviso in modo diretto usando come training set le misurazioni che vanno dal 4/7/2002 al 4/7/2018, come validation set quelle dal 4/7/2018 al 4/7/2021, e come testing set quelle dal 4/7/2021 al 31/12/2023, mantenendo le proporzioni quasi identiche.

1.6 Generatore

Data la loro dimensione, passare i dataset interi come input al modello non è ideale. È per questo che nell'addestramento di reti neurali vengono usati i **generatori**, funzioni speciali che permettono di restituire piccoli lotti di dati (detti *batch*) uno alla volta.

Un generatore è simile a una funzione normale, ma invece di usare `return` per restituire un valore e terminare l'esecuzione, utilizza la parola chiave `yield`. Questo consente alla funzione di 'ricordare' lo stato in cui è stata interrotta, riprendendo l'esecuzione dallo stesso punto alla chiamata successiva.

I generatori producono valori ‘al volo’, un batch alla volta e sotto richiesta. Questo non solo li rende molto efficienti in termini di memoria, ma anche essenziali quando si desidera effettuare operazioni di pre-elaborazione o augmentazione dati in tempo reale. Inoltre, i generatori possono essere eseguiti in parallelo rispetto all’addestramento del modello: mentre il modello elabora un batch di dati, il generatore può preparare il successivo, ottimizzando i tempi di calcolo e riducendo i ‘tempi morti’ durante l’addestramento.

1.6.1 Versioni precedenti e copertura artificiale

Prima di descrivere il generatore usato negli stadi finali, è opportuno discutere dei primi generatori testati e dei motivi che hanno portato alla loro evoluzione, nonché degli approcci utilizzati per *augmentare* i dati.

Inizialmente, il generatore estraeva una misurazione casuale dal dataset mediterraneo, e ne selezionava un’area 128x128 con coordinate casuali. Questo spesso portava il generatore a passare come input al modello aree composte in maggioranza o completamente da terraferma (e quindi matrici senza dati). In questo stadio della sperimentazione, i dati mancanti venivano rappresentati come -1 su una scala di valori normalizzati da 0 a 1, e non venivano aggiunti dati mancanti artificiali via alterazioni della maschera.

In seguito, è stato deciso di concentrarsi su un’unica area di studio: il mare circondante l’Italia. Per catturare quest’area, la dimensione delle immagini in input è aumentata da 128x128 a 256x256. Questa dimensione offre un buon equilibrio tra quantità d’informazione ed efficienza di calcolo. Inoltre, le aree di 256x256 permettono di concentrarsi su fenomeni oceanografici a livello regionale, come vortici e correnti, e data l’influenza di questi fenomeni sulle performance del modello, tale dimensione risulta ottimale.

I generatori successivi hanno iniziato a sperimentare con l’aggiunta di **dati mancanti artificiali**, alterando la maschera che identifica i dati mancanti. Infatti, una delle difficoltà maggiori nell’applicare reti neurali per la ricostruzione di dati mancanti è l’assenza di dati di validazione: la rete non può imparare a ricostruire tali dati se non gli viene mai mostrato un esempio.

A tal fine, uno dei punti chiave di questo lavoro è l’aggiunta di offuscamento artificiale. In questo modo, la rete è in grado di capire come ricostruire una parte dei dati mancanti nelle immagini che riceve come input, e questo col tempo la permette di imparare a ricostruire anche i veri dati mancanti.

Inizialmente la copertura artificiale era creata combinando la maschera del giorno di analisi con quella del giorno precedente. Per garantire un grado di casualità nella copertura aggiunta, il processo di creazione è stato modificato per usare la maschera di

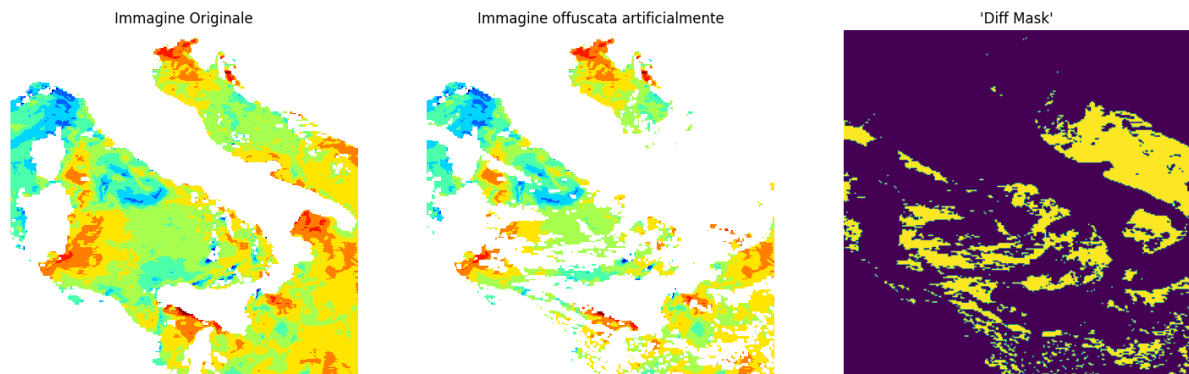


Figura 1.5: A sinistra, l'immagine originale. Al centro, l'immagine offuscata artificialmente, frutto del generatore. A destra, la maschera che traccia l'offuscamento artificiale aggiunto, e quindi l'area su cui la rete può apprendere.

un giorno casuale del dataset. Tuttavia, uno dei problemi più grandi di una casualità totale è che a volte la maschera risultante è identica a quella originale o copre una quantità eccessiva di dati, rischiando anche di coprire del tutto i dati dell'immagine originale, e quindi generando un input del modello composto esclusivamente da *NaN*.

Per compensare questa mancanza, il metodo usato successivamente consisteva nel prendere 3 maschere casuali, combinarle con la maschera originale, e utilizzare quella che ha il grado di copertura intermedio tra di esse. Questo metodo non era in grado di gestire tutti i casi limite, ma riduceva di gran lunga il loro numero.

Uno degli approcci sperimentali usato per aumentare il numero di misurazioni 'pulite', ovvero senza eccessiva copertura, è la creazione di un dataset separato contenente le 10 misurazioni con meno copertura di ogni mese, dove il generatore aveva una chance maggiore di utilizzare una misurazione proveniente da questo dataset al posto di quello normale. Questo metodo aveva l'intenzione di far apprendere alla rete l'aspetto di una misurazione *perlopiù* pulita, ma il suo uso si è rivelato essere irrilevante per le performance del modello.

Un'altro approccio sperimentale, già menzionato, è il passaggio della baseline come input, e in particolare la sua 'calibrazione'. Dato che le condizioni reali sono sempre in cambiamento, per migliorare l'accuratezza del modello, la baseline viene 'calibrata' in base alla temperatura media reale del giorno in analisi. Per farlo, si calcola la temperatura media dell'immagine corrente artificialmente oscurata, considerando solo le aree non coperte, e allo stesso tempo viene calcolata la temperatura media della baseline corrispondente, limitata alle stesse aree 'in chiaro' dell'immagine. Successivamente, viene aggiunta alla baseline la temperatura reale e si sottrae la baseline, facendo in modo che la nuova *baseline calibrata* rispecchi la temperatura media effettiva del giorno in questione.

La maggior parte dei generatori testati utilizzano e producono immagini 256x256. Tuttavia, molti dei modelli testati durante le sperimentazioni accettano input di dimensione inferiore: 128x128. Questo requisito non richiede molte modifiche al generatore stesso: vengono creati quattro generatori, uno per ogni quadrante dell'immagine originale (Nord-Ovest, Nord-Est, Sud-Ovest e Sud-Est), e la cui funzione principale di generazione opera come al solito, ma ognuno opera solo sulla sottosezione dei dati di dimensione 128x128 relativa al proprio quadrante.

Altri esperimenti effettuati per migliorare il generatore (e di conseguenza le performance del modello) sono stati: l'uso dei soli dati notturni, il passaggio dei giorni precedenti come input, l'approccio 'residuale' ed una modifica alla scelta delle misurazioni di partenza e a come le maschere artificiali vengono create partendo da esse. Questi esperimenti sono finalizzati nell'ultimo generatore usato, discusso nel capitolo **1.6.2**.

1.6.2 Generatore corrente

Il generatore corrente rappresenta l'approccio finale utilizzato per la creazione degli input da passare al modello. A differenza delle versioni precedenti, che alternavano tra batch di dati diurni e notturni, il generatore corrente lavora esclusivamente sui dati notturni. Questo consente di utilizzare dati più omogenei, e privi delle distorsioni indotte dalla radiazione solare, come riflessi o variazioni di temperatura superficiale durante il giorno.

Il generatore inizia selezionando il giorno di analisi casualmente dal dataset notturno, con l'obbligo che l'immagine selezionata abbia almeno la parte di dati 'in chiaro' del 40%, per evitare di lavorare su dati che, ancor prima di aggiungere copertura artificiale, sono poco informativi.

A partire da questa immagine, vengono recuperate anche le misurazioni dei quattro giorni precedenti, per un totale di cinque giorni in input. Questo consente alla rete di analizzare non solo i dati attuali, ma anche l'evoluzione temporale delle condizioni atmosferiche, un'informazione critica che permette al modello di ricostruire le immagini incomplete basandosi non solo sulle informazioni spaziali ma anche su quelle temporali. Inoltre, alcune strutture oceaniche coperte nel giorno analizzato possono invece essere scoperte nei giorni precedenti, migliorando le previsioni effettuate.

La maschera artificiale viene sempre creata estendendo la copertura nuvolosa dell'immagine corrente con quella di un giorno casuale, ma, in quest'ultima versione, il generatore cerca un giorno in cui la copertura nuvolosa totale generata combinando la maschera dell'immagine corrente e quella casuale sia tra il 60% e l'85%, assicurando che non vi sia né troppa copertura (che potrebbe nascondere completamente i dati), né trop-

po poca (che non fornirebbe un buon esempio per l'addestramento, dato il basso numero di aree su cui il modello è in grado di apprendere).

Vengono applicate delle maschere artificiali anche sui giorni precedenti. Per mantenere continuità temporale, vengono usate le maschere appartenenti ai giorni precedenti a quello utilizzato per la copertura del giorno corrente.

L'output del generatore è diviso in due; il primo output consiste in una matrice tridimensionale di 11 canali (*batch_x*), passata in input al modello e contenente le seguenti informazioni:

- **Canale 0:** Immagine corrente, offuscata artificialmente.
- **Canale 1:** Maschera artificiale applicata all'immagine corrente (1 per i punti di osservazione visibili, 0 per quelli coperti).
- **Canale 2-9:** Le immagini offuscate artificialmente e le rispettive maschere relative ai giorni precedenti. Sono disposte dalla più recente alla meno recente, alternando tra immagine e maschera ogni canale.
- **Canale 10:** La *land-sea mask* (1 per i punti di osservazione nel mare, 0 per la terraferma).

L'altro output (*batch_y*) è una matrice tridimensionale di 3 canali, che contiene l'output previsto assieme alle informazioni per la valutazione delle performance:

- **Canale 0:** L'immagine reale, che rappresenta il target di apprendimento per il modello.
- **Canale 1:** La maschera reale, senza aggiunte artificiali.
- **Canale 2:** La *diff_mask*, ovvero la 'differenza' tra la maschera artificiale e quella reale, che evidenzia le aree coperte dall'offuscamento artificiale su cui il modello è in grado di apprendere.

Per una corretta visualizzazione della temperatura nelle previsioni, è necessario riaggiungere la baseline ai dati denormalizzati. Un modo semplice per farlo è aggiungere la baseline corretta alla *batch_y* o, in alternativa, ritornare gli indici dei giorni usati in ogni batch, e prendere la baseline corrispondente al rispettivo giorno dell'anno.

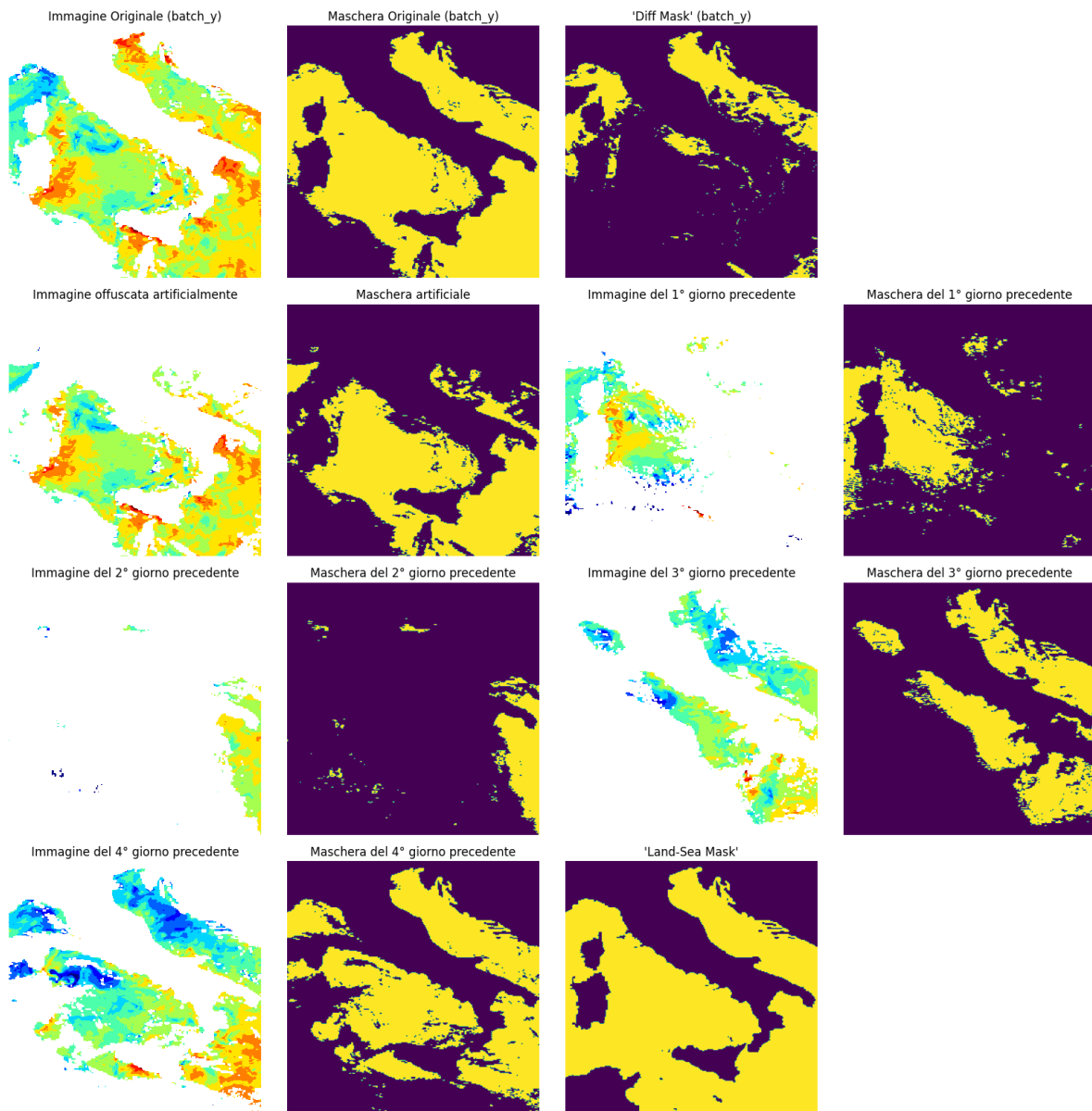


Figura 1.6: Il risultato completo del generatore corrente. Gli elementi appartenenti a $batch_x$ (tutti quelli non appartenenti alla prima riga) verranno usati come input per il modello. Nel caso di modelli 128x128, quattro generatori operano indipendentemente sui quattro quadranti dell'immagine.

Capitolo 2

Strumenti e tecnologie usate

Questo capitolo si concentra sugli strumenti e le tecnologie che hanno permesso la sperimentazione effettuata, nonché utili alla stesura di questa tesi.

2.1 Leonardo

Leonardo è un supercomputer installato nel 2022 a Bologna e gestito dal **Cineca**, reso disponibile sotto richiesta ai ricercatori che hanno bisogno di risorse computazionali elevate. Esso possiede una potenza di calcolo di quasi 250 PFlops ed è dotato di 100 PB di capacità di storage; nella sua partizione ‘Booster’, offre a disposizione 3456 nodi, ciascuno dotato di:

- 1 CPU Intel Xeon 8358 32 cores, 2,6 GHz
- 512 (8 x 54) GB RAM DDR4 3200 MHz
- 2 Nvidia HDR card 100 Gb/s per card
- 4 Nvidia custom Ampere GPU 64GB HBM2

Per usufruire di questo supercomputer è necessario registrarsi sul sito del Cineca e farsi riconoscere come membro del progetto di ricerca. Una volta fatto ciò, l'accesso avviene attraverso terminale, eseguendo il comando

```
step ssh login "[propria_email]" -provisioner cineca-hpc
```

Questo comando reindirizza a una pagina web di accesso in cui serve inserire email, password e una OTP. Una volta eseguito l'accesso in questo modo, si può entrare in Leonardo tramite ssh eseguendo il comando

```
ssh [proprio\_username]@login.leonardo.cineca.it
```



```

#SBATCH -p boost_usr_prod
#SBATCH --time [tempo:limite:esecuzione] # format: HH:MM:SS
#SBATCH -N 1 # 1 node
#SBATCH --ntasks-per-node=4 # 4 tasks out of 32
#SBATCH --gres=gpu:4 # 4 gpus per node out of 4
#SBATCH --mem=123000 # memory per node out of 494000MB
#SBATCH --job-name=my_batch_job

```

```
python3 ./[nome_file.py]
```

Lo script va poi eseguito tramite il comando `sbatch [nome_script]`. Questo crea un ‘*job*’ a cui sarà assegnato un numero basato sulla coda degli altri job eseguiti. L’output sarà generato in un file nominato `slurm-[numero_job].out` creato nella cartella dov’è stato eseguito. Per fermare un job e/o eliminarlo dalla coda d’attesa, si può usare il comando `scancel [numero_job]`.

2.2 Deep Learning

Il Deep Learning, o *Apprendimento Profondo*, è una sottocategoria dell’apprendimento automatico ispirata alla struttura neuronale del cervello: si basa su modelli detti ‘reti neurali artificiali’ (ANN, *Artificial Neural Networks*), che sono costituiti da numerosi **neuroni artificiali** organizzati in livelli interconnessi, detti *layers*.

Il Deep Learning simula la capacità astrattiva del cervello: permette l’emergenza di rappresentazioni astratte di alto livello partendo dai dati grezzi. Questa sua proprietà lo rende adatto a problemi come il riconoscimento di immagini, il riconoscimento vocale, la traduzione linguistica, e la ricostruzione di dati mancanti. [16, 17]

Le reti neurali artificiali sono composte da tre tipi di strati:

- **Strato di input:** Riceve i dati grezzi in ingresso e li trasmette al primo strato nascosto.
- **Strati nascosti (*hidden layers*):** Ogni strato nascosto riceve i dati ritornati dallo strato precedente, li elabora, e restituisce il risultato allo strato successivo, formando rappresentazioni sempre più complesse. La presenza di molteplici strati nascosti è ciò che caratterizza le reti di *Deep Learning*, e che le permette di modellare relazioni complesse.
- **Strato di output:** Riceve l’ultima rappresentazione e ritorna il risultato finale della rete.

Questa struttura stratificata permette alle reti neurali di apprendere in modo gerarchico. Ad esempio, in un’applicazione di riconoscimento di immagini, i primi strati di

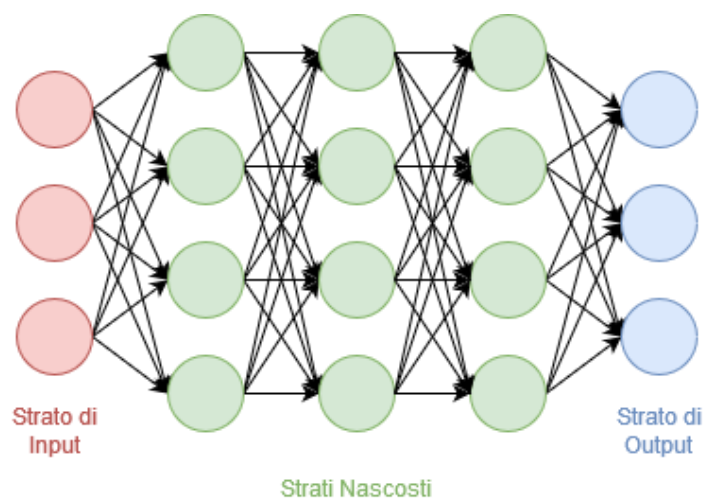


Figura 2.2: Schema rappresentativo di una rete profonda. Ogni elemento rappresenta un neurone artificiale.

una rete convoluzionale possono imparare a riconoscere caratteristiche ‘semplici’ come bordi e texture, mentre gli strati successivi apprendono concetti più complessi come forme e oggetti interi.

L’efficacia del deep learning nella ricostruzione di dati mancanti in vari contesti, tra cui immagini, testi e dati scientifici lo rende adatto al nostro problema: nel caso della temperatura superficiale marina, i dati mancanti possono essere ricostruiti sfruttando le informazioni disponibili nelle aree circostanti o temporalmente vicine, e apprendendo le caratteristiche complesse che governano le dinamiche oceaniche.

2.3 Python

Python è un linguaggio di programmazione ad alto livello che, col tempo, è diventato uno tra i più popolari linguaggi per lo sviluppo di reti e algoritmi di Machine Learning. Questo è dovuto alla sua accessibilità (è un linguaggio dalla sintassi semplice e leggibile anche a programmatori meno esperti) e soprattutto alla vasta gamma di librerie dedicate, come **NumPy**, **Pandas**, **scikit-learn**, **TensorFlow** e **PyTorch**. Queste librerie offrono strumenti potenti per la manipolazione dei dati e la costruzione di reti neurali, semplificando notevolmente il processo di sviluppo e sperimentazione.

2.4 Librerie

In questa sezione si discutono le librerie più importanti utilizzate nella sperimentazione.

2.4.1 TensorFlow

TensorFlow è una libreria open source rilasciata da Google che fornisce tutte le risorse necessarie sia alla manipolazione di tensori (da cui il nome), che alla creazione, all'addestramento e all'implementazione di modelli per l'apprendimento automatico [18]. TensorFlow è altamente scalabile ed è in grado di adattarsi a diverse architetture: in base alla disponibilità permette di eseguire modelli su CPU, GPU o TPU, garantendo alta efficienza con qualsiasi tipo di dispositivo e su qualsiasi scala di sperimentazione.

2.4.2 Keras

Keras è una libreria open source costruita su TensorFlow: fornisce un'API intuitiva per la costruzione e la formazione di reti neurali profonde, che permette di creare potenti modelli di apprendimento automatico con poche righe di codice, senza bisogno di una profonda comprensione della matematica e degli algoritmi sottostanti [19]. Keras rende lo sviluppo di modelli di deep learning più accessibile, e la sua flessibilità semplifica la sperimentazione su diversi modelli.

2.4.3 NumPy

NumPy è una libreria Python che contiene strutture dati e funzioni necessarie per il calcolo scientifico e l'analisi dei dati. Supporta grandi array multidimensionali, nonché operazioni matematiche e statistiche di base su tali array. Fondamentale sia per il pretrattamento dei dati prima del loro uso nell'addestramento, sia per effettuare controlli e operazioni finali sui risultati. Negli anni NumPy è diventato uno strumento essenziale per il calcolo scientifico, l'analisi dei dati e l'apprendimento automatico in Python, consentendo agli utenti di eseguire operazioni matematiche complesse e manipolazioni di dati con facilità ed efficienza [20].

2.4.4 Matplotlib

La maggior parte dei grafici presentati in questa tesi, inclusi quelli relativi all'andamento dell'addestramento e ai risultati sperimentali, sono stati generati con Matplotlib, libreria Python dedicata alla creazione di grafici e visualizzazioni [21]. Rappresentare dati in forma visiva è utile a facilitare l'interpretazione dei risultati e delle prestazioni dei modelli.

Capitolo 3

Implementazione e Valutazione

In questo capitolo, verrà descritto il processo di addestramento del modello utilizzato e i parametri scelti, nonché la struttura della rete stessa e i risultati ottenuti una volta applicata su casi reali.

3.1 Loss function

La *loss function* (funzione obiettivo o ‘di perdita’) è la funzione che monitora le performance del modello basandosi su determinati parametri, misurando la differenza tra le previsioni del modello e i valori reali. Nel corso dell’addestramento, il modello modifica i suoi pesi ad ogni iterazione, al fine di minimizzare questa differenza e quindi migliorare la qualità delle previsioni.

Nel contesto di questo progetto, la funzione di perdita personalizzata (*customLoss*) utilizza il **Mean Squared Error (MSE)**, ovvero l’errore quadratico medio. Esso viene espresso matematicamente come:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n \left(y_{\text{true}}^{(i)} - y_{\text{pred}}^{(i)} \right)^2$$

dove y_{true} rappresenta i valori reali e y_{pred} i valori predetti dal modello. Tuttavia, la funzione è costruita in modo da evitare le aree riguardanti la terraferma e gli offuscamenti reali, cioè i punti su cui non si hanno dati da confrontare con le previsioni del modello.

3.1.1 Metriche

Nel corso della sperimentazione sono state utilizzate alcune metriche aggiuntive: parametri che vengono costantemente monitorati ma che non influiscono sull’allenamento come la loss function. Queste metriche sono servite principalmente a monitorare il valore dell’MSE su determinate porzioni di immagine, come il focus sulle sole aree rimaste in

chiaro (*clearMetric*) o il focus sulle sole aree d'interesse, ovvero quelle nascoste artificialmente (*artificialMetric*). A tal ragione, queste metriche risultano identiche alla funzione di perdita, salvo la maschera applicata per identificare la regione su cui viene svolta la misurazione.

Un'altra metrica usata è *RMSEMetric*, che permette il monitoraggio dell'**RMSE** (*Root Mean Squared Error*, radice dell'errore quadratico medio), misurata esclusivamente sulle aree nascoste artificialmente. Questo RMSE è particolarmente utile poichè è il criterio usato al termine dell'allenamento per valutare le performance del modello, in quanto permette di capire non solo quanto la previsione sbaglia in media, ma anche quanto gravi siano gli errori nelle aree più critiche.

3.2 Architettura della rete

Nelle sperimentazioni effettuate sono state provate molteplici tipologie di reti, di varie dimensioni. Una delle prime implementate, e quella che si è dimostrata più performante delle altre, è la cosiddetta '**U-Net**' [22].

L'idea principale di una U-Net è quella di effettuare la segmentazione delle immagini in input, ovvero nel raggruppare i pixel con caratteristiche simili. A tal fine, la rete combina due processi sequenziali per estrarre tali caratteristiche:

1. **Down-scaling (compressione, codifica)**: Inizialmente l'immagine viene compressa progressivamente, riducendo la sua dimensione e perdendo sempre più informazioni, mantenendo solo le caratteristiche più rilevanti.
2. **Up-scaling (espansione, decodifica)**: In seguito l'immagine viene espansa di nuovo, ripristinando la risoluzione originale e riposizionando le caratteristiche estratte nei punti corretti.

È questo processo di compressione e decompressione che dà alla rete la sua caratteristica forma a 'U', da cui prende il nome.

Per evitare la perdita di dettagli importanti durante la codifica, la U-Net utilizza delle **skip connections**, ovvero **connessioni dirette** tra i livelli della fase di codifica e quelli corrispondenti nella fase di decodifica. Queste connessioni permettono di trasmettere una parte delle informazioni originali, evitando che vengano perse durante la compressione, ma consentendo comunque alla rete di apprendere le caratteristiche più significative.

In particolare, la rete U-Net usata è una variante con **residual blocks** (blocchi residui), che riduce il rischio di perdita di informazioni durante l'addestramento e migliora la capacità della rete di apprendere caratteristiche complesse [23].

I blocchi residui consistono nell'inserire una *connessione diretta* tra l'input di uno strato e il suo output, che permette di saltare alcune operazioni e consente all'input originale di essere sommato direttamente al risultato finale del blocco (invece di essere trasformato completamente attraverso convoluzioni). Questo riduce la possibilità che dettagli importanti vengano persi durante il passaggio attraverso i vari strati della rete, rendendo più semplice imparare nuove caratteristiche senza dimenticare quelle precedenti.

Un'altra funzione dei blocchi residui è la facilitazione dell'addestramento per le reti profonde: nelle reti convenzionali, man mano che il numero di strati aumenta, diventa più difficile per il modello aggiornare correttamente i pesi: questo può portare a una diminuzione delle performance nota come il problema del *vanishing gradient*. L'uso di connessioni dirette permette di aggiornare i pesi più facilmente, permettendo di addestrare reti molto profonde con maggiori performance.

3.2.1 Operazione di convoluzione

L'operazione chiave che consente di estrarre le caratteristiche rilevanti dalle immagini in input è detta **convoluzione**. In una convoluzione, un filtro (o *kernel*) viene fatto scorrere sull'immagine, applicando operazioni di moltiplicazione elemento per elemento tra il filtro e una porzione dell'immagine [24]. Il risultato è una nuova mappa di caratteristiche, che evidenzia pattern specifici come bordi o texture.

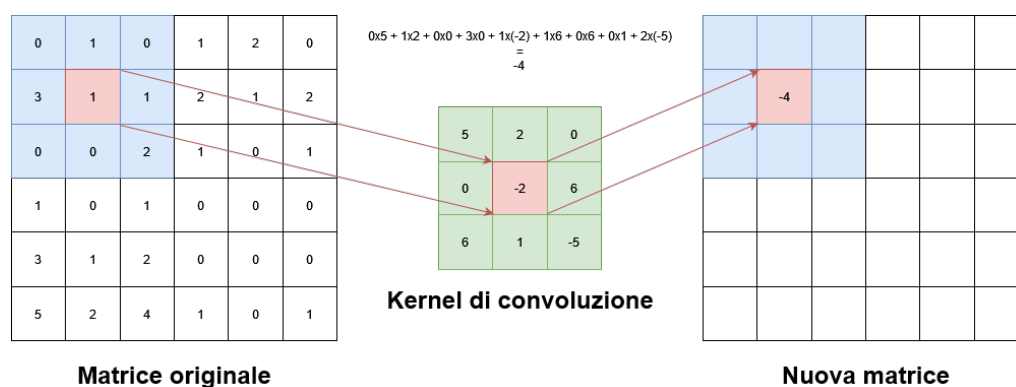


Figura 3.1: Esempio di convoluzione 3x3 applicata su un pixel: il nuovo valore è il risultato della moltiplicazione tra la porzione della matrice originale e il filtro.

L'operazione di convoluzione è fondamentale per il processo di estrazione delle caratteristiche, poiché permette alla rete di riconoscere pattern e strutture importanti nell'immagine, successivamente utilizzate per la previsione.

3.2.2 Modello utilizzato

Il modello più performante allenato opera su immagini di dimensione 128×128 . Quindi, per coprire l'intera area di analisi da 256×256 , sono stati allenati quattro modelli distinti, uno per ogni quadrante.

La rete U-Net con blocchi residui utilizzata ha 4 livelli di profondità, dove ogni livello ha rispettivamente 64, 128, 256 e 512 canali. Inoltre, ogni livello è composto da due blocchi residui consecutivi.

Il modello parte dall'input: un tensore di forma $(128, 128, 11)$, dove 128×128 rappresenta la dimensione delle immagini di partenza, ed 11 rappresenta il numero di canali, ovvero il numero di immagini passate in input (discusse nel capitolo 1.6.2).

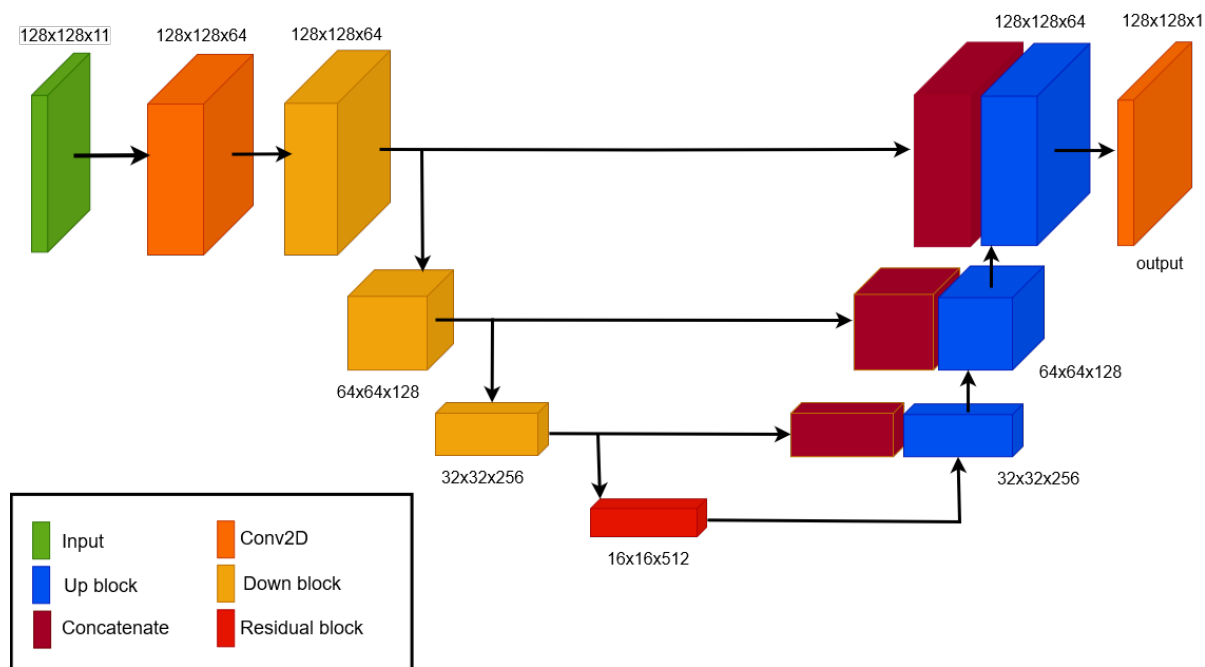


Figura 3.2: Schema che mostra la composizione del modello. Da notare la struttura a 'U' della rete, con connessioni dirette tra i livelli simmetrici della fase di compressione e decompressione.

La rete inizia la fase di compressione con una convoluzione 1×1 , che modifica i canali in ingresso da 11 a 64, corrispondente al primo livello di profondità.

In ogni livello vengono applicati due blocchi residui consecutivi, ognuno che esegue convoluzioni 3×3 e utilizza una connessione diretta per sommare l'input originale al risultato convoluzionale.

Alla fine del livello, la risoluzione dell'immagine viene ridotta tramite *AveragePooling2D*, che dimezza la dimensione spaziale (da 128x128 a 64x64 e così via), e le feature estratte vengono salvate nelle *skip connections* simmetriche, utili per la fase di ricostruzione.

Una volta raggiunto il livello più profondo (e più compresso) della rete, dove la risoluzione è stata ridotta al minimo (16x16 pixel, con 512 canali), vengono applicati due ulteriori blocchi residui per elaborare le 'caratteristiche globali'. Questo strato intermedio aiuta il modello a comprendere le relazioni più astratte tra i vari elementi dell'immagine.

La fase di decompressione inverte il processo della fase precedente: la risoluzione dell'immagine viene gradualmente ripristinata utilizzando *UpSampling2D*. Durante questo processo, il modello utilizza i dati salvati nelle *skip connections* per ripristinare parte delle informazioni perse durante la fase di compressione: i canali vengono concatenati dai livelli di compressione ai livelli di decompressione corrispondenti, preservando i dettagli importanti. Anche in questa fase vengono applicati i blocchi residui, per garantire che le caratteristiche apprese siano integrate in modo efficiente nel risultato finale.

L'ultimo strato del modello è una convoluzione 1x1 che riduce i canali a 1, producendo quindi una singola immagine 128x128 corrispondente alla previsione finale del modello relativa al suo quadrante.

3.3 Addestramento

Il modello è stato compilato con l'ottimizzatore *Adam*, capace di adattare il tasso di apprendimento durante l'addestramento, ed un *learning rate* (tasso di apprendimento) iniziale di 0.0001 ($1e-4$).

L'addestramento è stato effettuato imponendo un limite massimo di 200 epoche. Il numero di aggiornamenti dei pesi effettuato ogni epoca, anche detto *steps per epoch*, è calcolato in base alla dimensione del dataset usato (numero di misurazioni SST dedicate all'allenamento o alla validazione): tale dimensione viene divisa per la *batch_size* usata, ovvero 32. Questa *batch_size* identifica il numero di campioni elaborati per ogni step, quindi ogni epoca elabora all'incirca lo stesso numero di misurazioni contenute nei dataset corrispondenti.

Ogni epoca si compone di:

1. **Fase di addestramento**, dove il modello apprende usando i prodotti del generatore di *training*, *train_gen*, aggiornando i pesi per migliorare la performance su questi dati.

2. **Fase di validazione**, in cui il modello viene valutato sui prodotti del generatore di *validation*, `val_gen`, monitorando la `val_loss` (funzione di perdita calcolata sui dati di validazione) senza aggiornare i pesi, al fine di misurare la capacità del modello di generalizzare ed evitare l'*overfitting*.

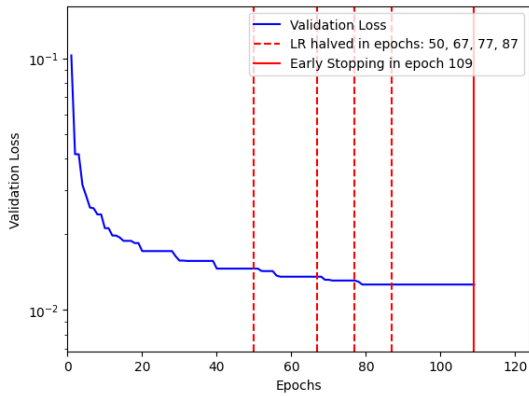
Durante l'addestramento, sono stati utilizzati varie *callbacks* (azioni da svolgere durante il training) per migliorare la stabilità e le performance del modello, nonché effettuare valutazioni '*on the fly*':

- **Early Stopping**: Monitorando la `val_loss`, l'addestramento viene interrotto in anticipo se la perdita non migliora per 30 epoche consecutive. Questo riduce il rischio di *overfitting*, fermandolo quando non ci sono più progressi significativi. Questa callback è il motivo per il quale il limite massimo imposto di 200 epoche di addestramento non viene quasi mai raggiunto.
- **ReduceLRonPlateau**: Se la `val_loss` non migliora per 10 epoche, il tasso di apprendimento viene dimezzato, con un valore minimo impostato a 0.00001 ($1e-5$). Ridurre il tasso di apprendimento consente al modello di effettuare aggiornamenti più piccoli e precisi ai suoi parametri, particolarmente utile nelle fasi finali dell'addestramento, quando il modello si avvicina a una soluzione ottimale e necessita di cambiamenti più delicati per evitare di 'saltare' oltre il punto ottimale.

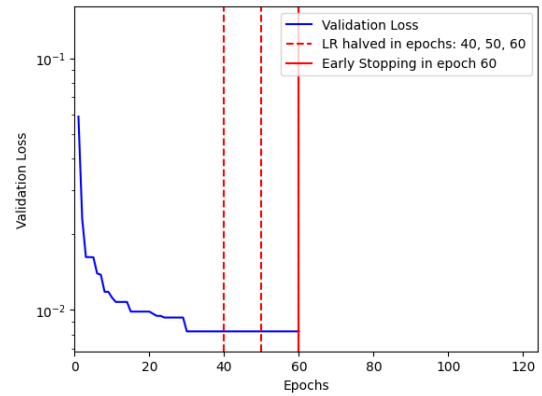
La `val_loss` viene considerata migliorata solo se tale miglioramento è di almeno 0.001 ($1e-3$), evitando che piccole fluttuazioni casuali vengano interpretate come miglioramenti significativi, e quindi garantendo che il tasso di apprendimento venga ridotto solo quando vi è un reale progresso nel ridurre la funzione di perdita, permettendo una convergenza più stabile e accurata.

- **ModelCheckpoint**: A ogni miglioramento della `val_loss`, i pesi del modello vengono salvati in un file per consentire il ripristino del miglior modello addestrato, e continuarne l'addestramento in seguito.
- **TestCallbackGaussian**: Una callback personalizzata creata dal collega Alessandro Testa. Essa effettua automaticamente la valutazione del modello via calcolo dell'RMSE ogni tot epoche, velocizzando il processo di valutazione e consentendo di eseguirlo e monitorarlo in contemporanea con l'allenamento.
- **TestCallbackGaussianDincae**: Simile alla callback precedente, ma la valutazione del modello viene effettuata utilizzando i dati DINCAE, discussi più a fondo nel capitolo 3.4.2.

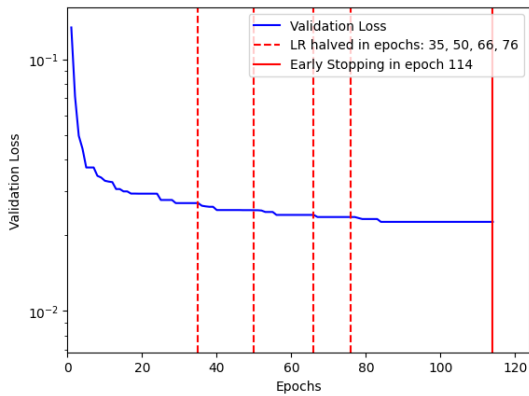
Il modello più performante, come già menzionato, è in realtà la composizione di 4 modelli da 128x128, ognuno allenato su un quadrante diverso dell'area 256x256 di nostro interesse. Per tale ragione, l'allenamento è stato svolto in modo individuale sui 4 modelli. Le informazioni riguardanti il loro allenamento si trovano nei grafici nella seguente figura:



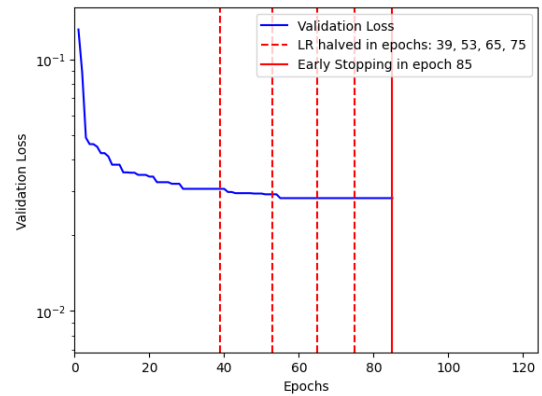
(a) Quadrante NW



(b) Quadrante NE



(c) Quadrante SW



(d) Quadrante SE

Figura 3.3: I miglioramenti della `val_loss` nel corso dell'allenamento per ciascuno dei quadranti. Vengono anche indicate le epoche in cui il *learning rate* viene dimezzato e l'epoca in cui l'allenamento viene terminato a causa della mancanza di miglioramenti (notare la presenza di un *plateau* nelle 30 epoche precedenti la fine di ogni allenamento).

3.4 Valutazione

La valutazione del modello si basa sull'**RMSE** (*Root Mean Squared Error*, radice dell'errore quadratico medio). In particolare, il valore d'interesse per valutare quanto il modello sia performante nel ricostruire i dati SST mancanti è l'RMSE calcolato sulle aree offuscate artificialmente.

Per ogni batch di immagini testate, le previsioni del modello vengono denormalizzate, e le differenze tra le previsioni e i valori reali vengono calcolate solo nelle aree corrispondenti alla *diffMask*, cioè ai punti offuscati artificialmente.

Viene calcolato l'errore quadratico medio (MSE) per ogni pixel rilevante, e la radice quadrata di tale errore fornisce il valore RMSE per ciascuna immagine. Per ogni batch, questi valori vengono aggregati per ottenere:

- **RMSE medio**: rappresenta l'errore medio del modello nel ricostruire i valori SST mancanti.
- **RMSE std**: la *deviazione standard* dell'RMSE, che indica la 'dispersione' dei risultati ottenuti.

Il processo di valutazione viene ripetuto su 50 batch, e si calcola la media dei risultati totali ottenuti. Questo processo viene a sua volta ripetuto 10 volte per migliorare l'affidabilità dei risultati. In ogni ripetizione viene memorizzato l'RMSE medio e si calcola la deviazione standard su queste 10 ripetizioni. Si considera come RMSE finale la media di queste 10 ripetizioni, più o meno la deviazione standard. In questo modo si stima con maggiore precisione la performance del modello, assieme alla sua robustezza.

3.4.1 Risultati

I risultati del modello più performante, basati sull'RMSE medio e la sua deviazione standard, sono riassunti nella seguente tabella. Le misure sono calcolate sulle aree offuscate artificialmente per ciascun quadrante del dominio di previsione.

Quadrante	RMSE medio \pm deviazione standard
NW	0.3049 \pm 0.0030
NE	0.2829 \pm 0.0024
SW	0.3042 \pm 0.0031
SE	0.3139 \pm 0.0036
Avg	0.3015 \pm 0.0114

Tabella 3.1: Risultati finali dell'RMSE per ciascun quadrante, con media totale.

Come si può osservare da questi risultati, il quadrante NE ha ottenuto il miglior risultato complessivo, con un RMSE medio di 0.2829 e una deviazione standard di 0.0024.

Questo indica che il modello ha avuto più successo nel ricostruire i dati mancanti nella regione del Nord Adriatico, con una minore variabilità rispetto agli altri quadranti.

Al contrario, il quadrante SE ha registrato l'RMSE più elevato, pari a 0.3139, e la maggiore deviazione standard, indicando che nel Mar Ionio il modello ha avuto maggiori difficoltà.

I quadranti NW e SW (regione del Mar Tirreno) mostrano valori molto simili tra loro sia per l'RMSE medio che per la deviazione standard (entrambi intorno a 0.304 ± 0.003).

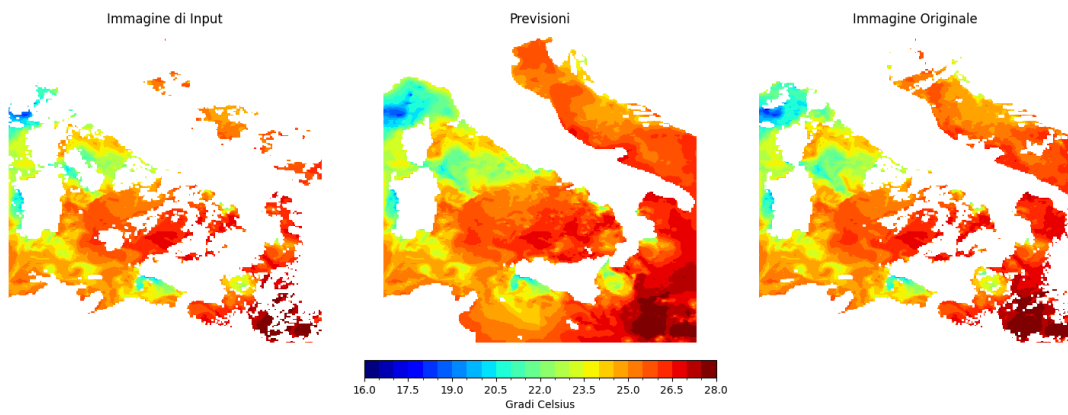
La media totale delle valutazioni per i 4 modelli utilizzati nella nostra area d'interesse è di **0.3015**, con una variazione standard di **0.01140**: questo significa che le previsioni effettuate dal modello ricostruiscono le aree nascoste artificialmente con una differenza dai dati reali di circa 0.3015 °C. La stabilità delle previsioni dimostra che il modello ha un buon livello di generalizzazione, e può essere considerato un valido strumento per la previsione della temperatura superficiale marina in assenza di misurazioni di buona qualità.

Per dimostrare le performance del modello, verranno presentate nella figura **3.4** un paio di esempi di ricostruzioni effettuate. A scopo dimostrativo, le quattro immagini da 128x128 sono unite in una singola immagine da 256x256, e viene analizzato lo stesso giorno con la stessa copertura artificiale in ogni quadrante. Tuttavia è importante ricordare che i modelli sono allenati individualmente e, non essendo a conoscenza di ciò che accade negli altri quadranti, i bordi delle previsioni potrebbero non combaciare perfettamente: ogni quadrante dev'essere confrontato solo con la sua rispettiva porzione di immagine reale.

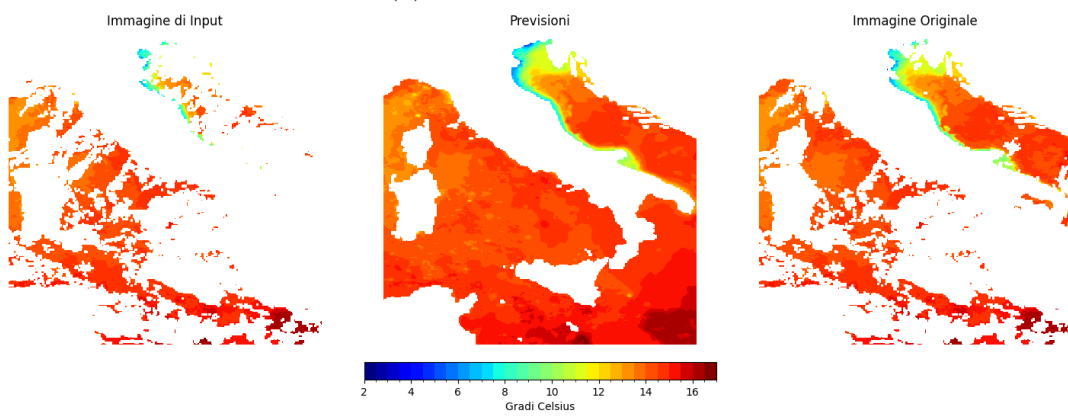
Dalle figure si può notare che, come atteso, le performance del modello sono relative alla quantità di offuscamento presente nel giorno in analisi. Nella prima immagine, con una quantità moderata di dati mancanti, le previsioni del modello risultano ben allineate con i valori reali: le temperature nelle zone ricostruite sono coerenti con quelle osservate, con solo minime deviazioni visibili. In regioni dove il modello ha meno informazioni di partenza, come nel Mar Adriatico, queste deviazioni sono invece maggiori, ma pur sempre accettabili.

Nella seconda immagine, caratterizzata da un'alta copertura nuvolosa, il modello riesce comunque a mantenere una discreta accuratezza. Tuttavia, nelle aree largamente mascherate le ricostruzioni risultano più piatte, con alcune discrepanze rispetto ai dati reali.

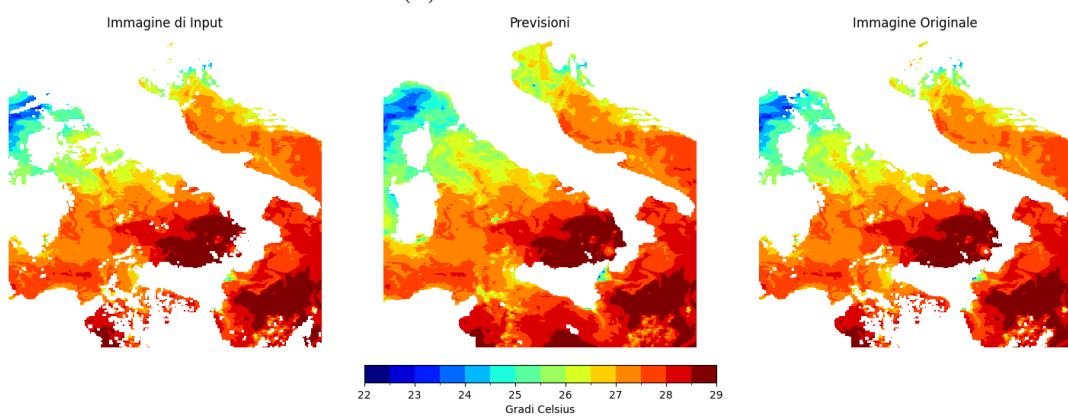
Infine, nella terza immagine, con bassa copertura nuvolosa, si osservano performance ottimali: dato il basso numero di offuscamenti, e data la loro area contenuta, le previsioni risultano quasi indistinguibili dalle misurazioni reali.



(a) Offuscamento medio



(b) Offuscamento alto



(c) Offuscamento basso

Figura 3.4: A sinistra, le immagini passate in input ai modelli. Al centro, le rispettive previsioni effettuate, che ricostruiscono le aree con i dati mancanti. A destra, le misurazioni reali del giorno in analisi.

Le previsioni prodotte dal modello su diverse condizioni di copertura mostrano che il modello è ben calibrato per gestire situazioni reali. Anche se la precisione diminuisce in condizioni di scarsità di dati, il livello complessivo di accuratezza rimane alto, rendendolo uno strumento affidabile per la ricostruzione della temperatura superficiale marina su regioni estese e con copertura eterogenea.

Un importante obiettivo di performance di questo lavoro è il raggiungimento di ricostruzioni più accurate di quelle effettuate via metodi statistici tradizionali. A tal fine, effettuiamo anche un confronto visivo tra le previsioni del nostro modello e i dati ‘L4’, ovvero dati già trattati via interpolazione. In questo test, il modello riceve in input le immagini e le maschere originali senza offuscamento artificiale, quindi possiamo osservare le previsioni risultanti passando in input misurazioni inalterate.

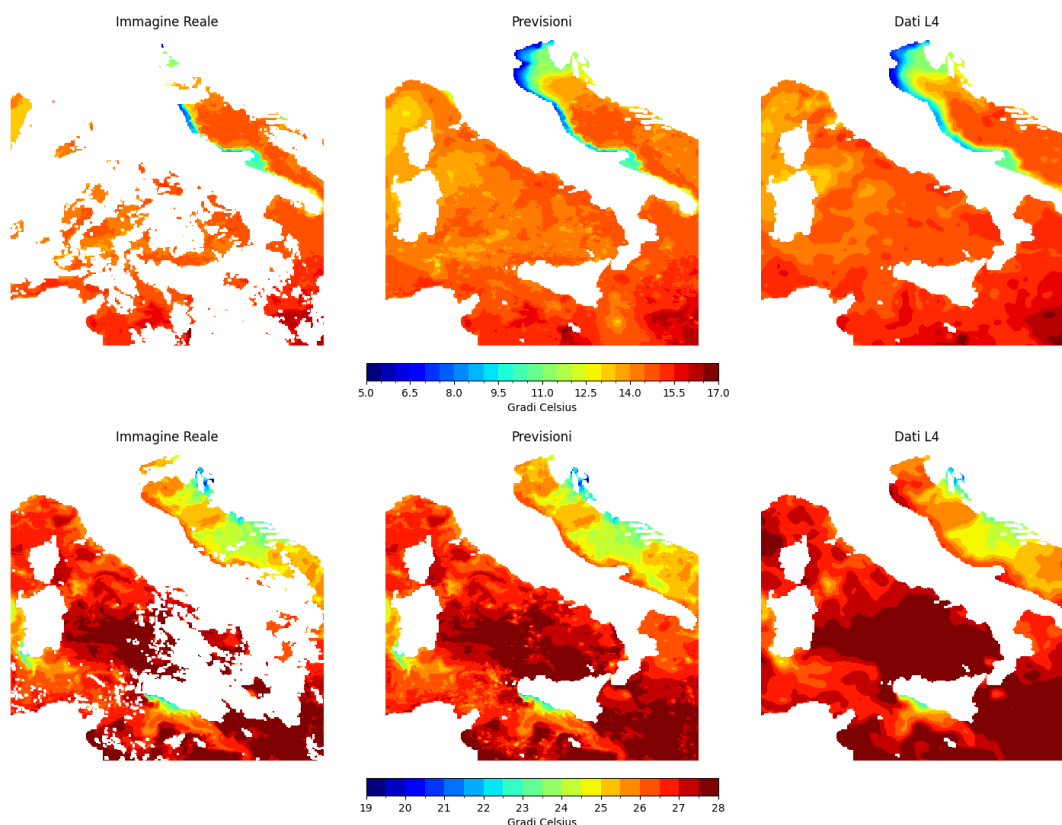


Figura 3.5: A sinistra, le misurazioni originali, passate in input ai modelli. Al centro, le rispettive previsioni effettuate. A destra, i dati L4 del giorno in analisi.

I dati L4 di confronto sono stati offerti dal CMCC, Centro euro-Mediterraneo sui Cambiamenti Climatici, e si basano sui dati misurati dal sensore MODIS presente sia sul satellite Aqua che su quello Terra. Vanno dal 4/7/2021 al 31/12/2023, con il periodo

dal 1/4/2022 al 16/4/2022 rimosso in modo da combaciare con il test set dei dati (a cui, come già menzionato, mancano questi 16 giorni di misurazioni).

Come si può notare dalla figura **3.5**, le previsioni risultano più dettagliate rispetto ai dati L4. In particolare, le aree con gradienti di temperatura più complessi e le zone costiere mostrano una maggior definizione nelle ricostruzioni effettuate rispetto a quelle interpolate con i metodi tradizionali. Questo risultato dimostra la capacità della rete neurale di catturare pattern locali e quindi ricostruire strutture più realistiche nelle aree mancanti.

3.4.2 Confronto DINCAE

Per paragonare i risultati ottenuti con altre reti *State-of-Art*, il modello allenato è stato applicato sui dati utilizzati dal DINCAE (*Data INterpolating Convolutional Auto-Encoder*) in [8].

I dati di confronto sono giornalieri e nella stessa risoluzione dei dati su cui il modello è stato allenato, provenienti dalle misurazioni SST notturne del satellite Terra-MODIS dal 1/1/2003 al 31/12/2016. Questi dati sono divisi in due dataset: uno con coperture artificiali e l'altro coi dati originali, usati rispettivamente come parte della `batch_x` e della `batch_y` del generatore.

Dato che il test DINCAE si occupa del Nord del Mar Adriatico, l'area di analisi è diversa da quella usata dal modello, ma si sovrappone in maggior parte: quest'area si estende in latitudine da 40° a 46° e in longitudine da 12° a 19° , quindi per rientrare nelle coordinate della nostra area di analisi (latitudine compresa tra 35.33° e 46.0° e longitudine tra 7.92° e 18.58°), dei 144×168 punti di osservazione originali viene tagliata via una fascia verticale alla destra larga 10 pixel, ottenendo un'area sovrapposta di dimensione 144×158 .

Per effettuare il confronto su questi dati senza la necessità di cambiare dimensioni o coordinate degli input, i dataset di DINCAE sono ritagliati e posizionati in una matrice 256×256 nelle loro coordinate corrispondenti, e il resto della matrice viene considerato offuscato. Quando vengono effettuate previsioni e valutazioni, viene quindi considerata solo la parte rilevante ai dati inseriti.

Nel caso di modelli con dimensione spaziale 128×128 , viene usato il modello del quadrante Nord-Est, e per ottenere un'immagine delle giuste dimensioni vengono tagliati via anche 10 punti di osservazione dal basso e 30 dalla sinistra (oltre ai 10 punti di osservazione sempre rimossi dalla destra).

I dati vengono normalizzati 'al volo': usando un generatore apposito, viene selezionata e sottratta ai dati la porzione d'interesse della baseline, e si procede con la loro standardizzazione, per poi invertire il processo nella valutazione.

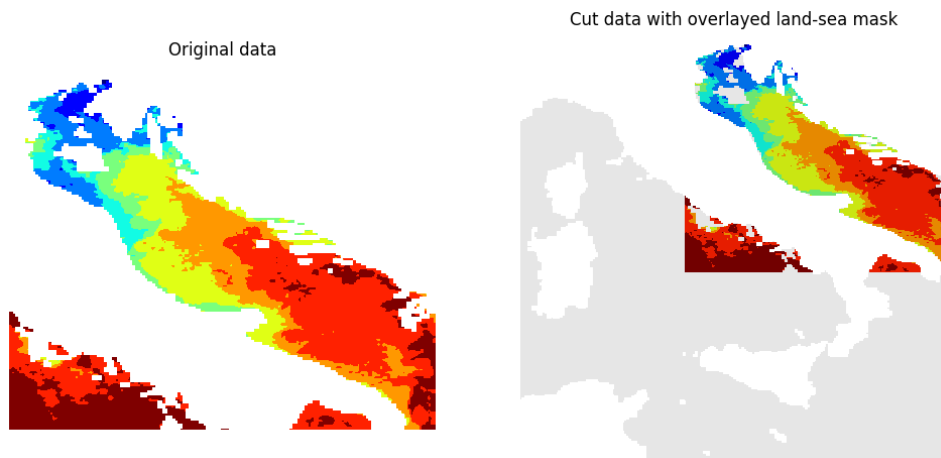


Figura 3.6: A sinistra, la misurazione SST DINCAE del 15 aprile 2003. A destra, tale misurazione viene tagliata e ripozizionata in una matrice 256x256, mantenendo le coordinate su cui il modello si è allenato.

Le performance del modello sono state misurate calcolando, anche in questo caso, la radice dell'errore quadratico medio (RMSE) nelle aree offuscate artificialmente. Questa valutazione è stata effettuata iterando su ogni immagine del dataset di confronto una singola volta.

Con i risultati ottenuti dal modello più performante (unet di input 128x128, rete di 4 strati con canali da 64 a 512, e con 4 giorni precedenti passati in input), l'RMSE medio ottenuto sulle aree coperte artificialmente è di **0.4201**, con una deviazione standard di $9.1720e-05$. Il modello ha mostrato in media prestazioni inferiori rispetto ai propri dati, ma pur sempre soddisfacenti.

Questi risultati sottolineano l'importanza di adattare i modelli in base alle caratteristiche specifiche del dataset d'interesse: seppur efficaci una volta calibrati, i modelli di apprendimento automatico dipendono sempre dai dati di partenza, e devono essere quindi addestrati su dati rappresentativi dell'ambito in cui vengono applicati per ottenere prestazioni elevate.

Conclusioni

L'obiettivo di questo lavoro era di sviluppare un modello in grado di ricostruire i dati mancanti nelle misurazioni della temperatura superficiale marina (SST), utilizzando tecniche di apprendimento automatico. A tal fine, sono stati mostrati i dati utilizzati e il loro trattamento, nonché gli strumenti utilizzati per le sperimentazioni effettuate. Successivamente è stato implementato un sistema di reti neurali convoluzionali: quattro modelli basati su varianti di U-Net, ciascuno allenato su una porzione del Mar Mediterraneo.

L'errore medio complessivo di $0.3015 (\pm 0.0114)$ gradi Celsius è indicativo di una performance solida dei modelli, ma la variazione nelle prestazioni tra i vari quadranti dev'essere tenuta in considerazione per future espansioni dell'area di analisi.

Le implementazioni effettuate in questa tesi presentano alcune limitazioni, e c'è ampio margine di miglioramento. Ogni rete U-Net è stata addestrata separatamente, senza prendere in considerazione le informazioni 'globali' al di fuori del proprio quadrante, perciò questa suddivisione, seppur efficiente, porta alla nascita di discontinuità ai bordi delle regioni scelte. Inoltre una dimensione spaziale ristretta non permette di catturare alcune correlazioni spaziali su larga scala.

Un altro possibile miglioramento riguarda la struttura del modello stesso: le reti utilizzate, benché efficaci, sono relativamente semplici. In futuro, l'adozione di architetture più sofisticate e specializzate alla ricostruzioni di dati oceanografici potrebbe migliorare ulteriormente la qualità delle ricostruzioni.

Infine, un'altra direzione promettente per il proseguimento di questo lavoro è l'integrazione di dati aggiuntivi, di varia natura e proveniente da diverse fonti (ad esempio la velocità e direzione del vento), che potrebbero arricchire ulteriormente l'addestramento del modello e fornire quindi maggior accuratezza.

In conclusione, il lavoro svolto in questa tesi rappresenta un passo avanti nell'utilizzo dell'apprendimento automatico per la ricostruzione di dati oceanografici mancanti. Nonostante le limitazioni discusse, i risultati ottenuti dimostrano che i modelli di *deep learning* possono fornire previsioni accurate e stabili dei dati mancanti, offrendo quindi un supporto alla gestione e all'analisi delle misurazioni della temperatura superficiale marina e all'oceanografia in generale.

Bibliografia

- [1] Inoue, J., Kawashima, M., Fujiyoshi, Y. et al. *Aircraft Observations of Air-mass Modification Over the Sea of Okhotsk during Sea-ice Growth*. *Boundary-Layer Meteorol* 117, 111–129 (2005). <https://doi.org/10.1007/s10546-004-3407-y>
- [2] Wylie, Donald & Jackson, Darren & Menzel, W. & Bates, John. (2005). *Trends in Global Cloud Cover in Two Decades of HIRS Observations*. *Journal of Climate - J CLIMATE*. 18. 3021-3031. [10.1175/JCLI3461.1](https://doi.org/10.1175/JCLI3461.1).
- [3] Azcarate, Aïda & Barth, Alexander & Sirjacobs, Damien & Lenartz, Fabian & Beckers, Jean-Marie. (2011). *Data Interpolating Empirical Orthogonal Functions (DINEOF): a tool for geophysical data analyses*. *Mediterr. Mar. Sci.*. 12. [10.12681/mms.64](https://doi.org/10.12681/mms.64).
- [4] Jung, S.; Yoo, C.; Im, J. *High-Resolution Seamless Daily Sea Surface Temperature Based on Satellite Data Fusion and Machine Learning over Kuroshio Extension*. *Remote Sens.* 2022, 14, 575. <https://doi.org/10.3390/rs14030575>
- [5] Fablet, Ronan & Viet, Phi & Lguensat, Redouane. (2017). *Data-Driven Models for the Spatio-Temporal Interpolation of Satellite-Derived SST Fields*. *IEEE Transactions on Computational Imaging*. PP. 1-1. [10.1109/TCI.2017.2749184](https://doi.org/10.1109/TCI.2017.2749184).
- [6] Toshio Michael Chin, Jorge Vazquez-Cuervo, Edward M. Armstrong, *A multi-scale high-resolution analysis of global sea surface temperature*, *Remote Sensing of Environment*, Volume 200, 2017, Pages 154-169, ISSN 0034-4257, <https://doi.org/10.1016/j.rse.2017.07.029>.
- [7] Barth, A., Alvera-Azcárate, A., Licer, M., and Beckers, J.-M.: *DINCAE 1.0: a convolutional neural network with error estimates to reconstruct sea surface temperature satellite observations*, *Geosci. Model Dev.*, 13, 1609–1622, <https://doi.org/10.5194/gmd-13-1609-2020>, 2020.
- [8] Barth, A., Alvera-Azcárate, A., Troupin, C., and Beckers, J.-M.: *DINCAE 2.0: multivariate convolutional neural network with error estimates to reconstruct sea*

- surface temperature satellite and altimetry observations*, *Geosci. Model Dev.*, 15, 2183–2196, <https://doi.org/10.5194/gmd-15-2183-2022>, 2022.
- [9] Han Z, He Y, Liu G, Perrie W. *Application of DINCAE to Reconstruct the Gaps in Chlorophyll-a Satellite Observations in the South China Sea and West Philippine Sea*. *Remote Sensing*. 2020; 12(3):480. <https://doi.org/10.3390/rs12030480>
- [10] Goh, E., Yepremyan, A. R., Wang, J., and Wilson, B.: *MAESSTRO: Masked Autoencoders for Sea Surface Temperature Reconstruction under Occlusion*, *EGU sphere* [preprint], <https://doi.org/10.5194/egusphere-2023-1385>, 2023.
- [11] Ćatipović L, Matić F, Kalinić H. *Reconstruction Methods in Oceanographic Satellite Data Observation—A Survey*. *Journal of Marine Science and Engineering*. 2023; 11(2):340. <https://doi.org/10.3390/jmse11020340>
- [12] P. Werdell, B. Franz, S. Bailey, G. Feldman, E. Boss, V. Brando, M. Dowell, T. Hirata, S. Lavender, Z. Lee, H. Loisel, S. Maritorena, F. Mélin, T. Moore, T. Smyth, D. Antoine, E. Devred, O. d’Andon, and A. Mangin, *Generalized ocean color inversion model for retrieving marine inherent optical properties*, *Appl. Opt.* 52, 2019-2037 (2013).
- [13] R. Rew and G. Davis, *NetCDF: an interface for scientific data access*, in *IEEE Computer Graphics and Applications*, vol. 10, no. 4, pp. 76-82, July 1990, doi: 10.1109/38.56302.
- [14] Hoyer, S. and Hamman, J. (2017) *xarray: N-D labeled Arrays and Datasets in Python*, *Journal of Open Research Software*, 5(1), p. 10. Available at: <https://doi.org/10.5334/jors.148>.
- [15] Muhammad Ali, Peshawa & Faraj, Rezhna. (2014). *Data Normalization and Standardization: A Technical Report*. 10.13140/RG.2.2.28948.04489.
- [16] LeCun, Yann & Bengio, Y. & Hinton, Geoffrey. (2015). *Deep Learning*. *Nature*. 521. 436-44. 10.1038/nature14539.
- [17] Jürgen Schmidhuber, *Deep learning in neural networks: An overview*, *Neural Networks*, Volume 61, 2015, Pages 85-117, ISSN 0893-6080, <https://doi.org/10.1016/j.neunet.2014.09.003>.
- [18] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, *TensorFlow: A system for large-scale machine learning*, 2016, <https://arxiv.org/abs/1605.08695>.

- [19] Jeff Heaton, *Applications of Deep Neural Networks with Keras*, 2022, <https://arxiv.org/abs/2009.05673>.
- [20] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). <https://doi.org/10.1038/s41586-020-2649-2>
- [21] Barrett, Paul & Hunter, J. & Miller, J.T. & Hsu, J.-C & Greenfield, P.. (2005). *matplotlib – A Portable Python Plotting Package*.
- [22] Olaf Ronneberger, Philipp Fischer, Thomas Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, arXiv, 2015, <https://arxiv.org/abs/1505.04597>.
- [23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition*, arXiv, 2015, <https://arxiv.org/abs/1512.03385>.
- [24] Vincent Dumoulin, Francesco Visin, *A guide to convolution arithmetic for deep learning*, 2018, <https://arxiv.org/abs/1603.07285>.

A CHI MI HA SEMPRE SUPPORTATO:
*Dal profondo del mio cuore,
Grazie.*