UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Sviluppo di un sistema di classificazione basato su Deep Learning per il riconoscimento delle attività di pesca svolte nel Mar Mediterraneo utilizzando dati AIS

Development of a Deep Learning-based classification system for the recognition of fishing activities in the Mediterranean Sea using AIS data

Relatore:
Dr. Galdelli Alessandro
Correlatore:
Prof. Mancini Adriano

Laureando:
Angeli Ludovico

Anno Accademico 2022-2023

Indice

Capitolo 1

Introduzione

La grande crescita delle attività di pesca negli ultimi anni ha reso il monitoraggio e la classificazione delle attività delle navi una sfida aperta nello scenario marino. Il continuo sfruttamento delle risorse ittiche ne ha drasticamente ridotto l'abbondanza, con conseguenze negative sul settore della pesca stesso. La pesca illegale, non dichiarata e non regolamentata provoca il depauperamento degli stock ittici e costituisce quindi una delle più grandi minacce per gli ecosistemi marini, compromette gli sforzi tesi a gestire le risorse alieutiche in modo sostenibile e spinge alcune di esse sull'orlo del collasso. In ottica futura, ad esempio, per preservare la varietà delle specie ittiche, il piano di azione dell'Unione Europea prevede una forte limitazione della pesca a strascico in tutta Europa entro il 2030 e propone la creazione di aree marine protette [1]. Per cui si viene a creare l'esigenza di strumenti di aiuto tecnologici al fine di ridurre l'impatto ambientale delle attività dell'uomo nell'ambito marittimo.

1.1 Il progetto

L'obiettivo di questo lavoro è quello di adattare dei sistemi di classificazione basati sul *Deep Learning* per il riconoscimento delle attività di pesca svolte nel Mar Mediterraneo. Per tracciare le imbarcazioni viene utilizzato il sistema di identificazione automatica (AIS), un sistema di tracciamento automatizzato e autonomo ampiamente utilizzato nel mondo marittimo per lo scambio di informazioni di navigazione [2]. Grazie ad esso, le informazioni statiche e dinamiche sulle imbarcazioni possono essere scambiate elettronicamente tra le stazioni di ricezione AIS (a bordo, a terra o via satellite). Dal dicembre 2004, l'Organizzazione Marittima Internazionale richiede che tutte le imbarcazioni abbiano a bordo un transponder AIS, che trasmette continuamente (ogni 5 minuti) i dati AIS. L'obiettivo è quello di addestrare reti neurali di *Deep Learning* per classificare le attività di pesca e ricostruire le informazione AIS mancanti, utilizzando i dati forniti dal CNR-IRBIM (Consiglio Nazionale delle Ricerche - Istituto per le Risorse Biologiche e le Biotecnologie Marine). Infatti, i dati mancanti sono talvolta dovuti a imbarcazioni che, per svolgere attività di pesca illegali, spengono i loro transponder.

1.2 Struttura della tesi

La tesi viene viene strutturata come segue:

- nel secondo capitolo viene descritto lo stato dell'arte riguardo gli algoritmi di classificazione di *Intelligenza Artificiale* (IA);
- nel terzo capitolo vengono introdotti tutti i principali strumenti software utilizzati, comprensivi del linguaggio di programmazione scelto e delle tecnologie che sono state approfondite durante lo sviluppo del progetto. Si descrive la logica con la quale è stato possibile segmentare il dataset, ovvero i dati acquisiti nella fase preliminare del progetto, analizzando e raccogliendo informazioni dallo stesso mediante l'ausilio di paradigmi di programmazione tradizionale. Si pone particolare enfasi nell'illustrazione delle tematiche di riferimento: *Time Series* e *Deep Learning*. Si illustrano in particolar modo gli script di supporto e la rete utilizzata;
- nel quarto capitolo si descrivono i risultati ottenuti;
- nel quinto ed ultimo capitolo si espongono le conclusioni e le limitazioni del progetto allo stato dell'arte. Infine sono descritti gli sviluppi futuri del progetto.

Capitolo 2

Stato dell'arte

Negli ultimi due decenni, la classificazione delle serie temporali (TSC) è stata considerata uno dei problemi più impegnativi nell'estrazione dei dati (data mining). Con l'aumento della disponibilità di dati temporali, dal 2015 sono stati proposti centinaia di algoritmi di TSC. Grazie al fattore tempo, i dati delle serie temporali sono presenti in quasi tutti i compiti che richiedono un qualche processo cognitivo umano. In effetti, qualsiasi problema di classificazione, che utilizzi dati registrati tenendo conto di un qualche tipo di ordinamento, può essere considerato un problema di TSC. Le serie temporali sono presenti in molte applicazioni del mondo reale, che vanno dalle cartelle cliniche elettroniche e dal riconoscimento delle attività umane, alla classificazione degli scenari acustici e alla sicurezza informatica. Inoltre, la diversità dei tipi di dati presenti nell'archivio UCR/UEA, il più grande archivio di serie di dati temporali, mostra le diverse applicazioni del problema TSC. Data la necessità di classificare accuratamente i dati delle serie temporali, i ricercatori hanno proposto centinaia di metodi per risolvere questo compito. Nei successivi due paragrafi verrà trattato lo stato dell'arte dei due approcci principali di *Intelligenza Artificiale*: *Machine Learning* e *Deep Learning*.

2.1 Machine Learning

Uno degli approcci TSC più popolari e tradizionali è l'uso di un classificatore Nearest Neighbor (NN) abbinato a una funzione di distanza. In particolare, la distanza Dynamic Time Warping (DTW), se utilizzata con un classificatore NN, ha dimostrato di essere una linea di base molto forte. Lines e Bagnall [3] hanno confrontato diverse misure di distanza e hanno dimostrato che non esiste una singola misura di distanza migliore significativamente della DTW. I contributi più recenti si sono concentrati sullo sviluppo di metodi di ensembling che superano significativamente le prestazioni del NN accoppiato al DTW (NN-DTW) [4]. Questi approcci utilizzano un insieme di alberi decisionali (random forest) o un insieme di diversi tipi di classificatori discriminanti (Support Vector Machine (SVM), NN con diverse distanze) su uno o più tipi di funzionalità. La maggior parte di questi approcci supera in modo significativo il NN-DTW e condividono una proprietà comune, ovvero la fase di trasformazione dei dati in cui le serie

temporali vengono trasformate in un nuovo insieme di funzionalità (ad esempio utilizzando la trasformazione shapelets o le funzionalità DTW). Questa nozione ha motivato lo sviluppo di un ensemble di 35 classificatori chiamato COTE (Collective Of Transformation-based Ensembles) che non si limita ad raggruppare diversi classificatori sulla stessa trasformazione, ma invece raggruppa diversi classificatori su diverse rappresentazioni delle serie temporali. Successivamente si è esteso COTE con un sistema di valutazione gerarchica, diventando HIVE-COTE, che ha dimostrato di ottenere un miglioramento significativo rispetto a COTE sfruttando una nuova struttura gerarchica con valutazione probabilistica, includendo due nuovi classificatori e due domini di trasformazione della rappresentazione aggiuntivi. HIVE-COTE è attualmente considerato l'algoritmo più avanzato per la classificazione delle serie temporali, valutato su 85 set di dati dell'archivio UCR/UEA. Per raggiungere la sua elevata accuratezza, HIVE-COTE diventa estremamente intensivo dal punto di vista computazionale e poco pratico da eseguire su un vero problema di big data mining. L'approccio richiede l'addestramento di 37 classificatori e la convalida incrociata di ogni iperparametro di questi algoritmi, il che lo rende inapplicabile in alcune situazioni. Per sottolineare l'inefficacia, si noti che uno di questi 37 classificatori è la Shapelet Transform, la cui complessità temporale è $O(n^2 - l^4)$, dove n è il numero di serie temporali nel dataset e l è la lunghezza di una serie temporale. Alla complessità del tempo di addestramento si aggiunge l'elevato tempo di classificazione di uno dei 37 classificatori: il Nearest Neighbor, che deve analizzare il set di addestramento prima di prendere una decisione al momento del test. Pertanto, poiché il Nearest Neighbor costituisce un componente essenziale di HIVE-COTE, il suo impiego in un ambiente in tempo reale è ancora limitato, se non addirittura impraticabile. Infine, oltre all'enorme tempo di esecuzione di HIVE-COTE, le decisioni prese da 37 classificatori non possono essere interpretate facilmente dagli esperti del settore, poiché i ricercatori hanno già difficoltà a comprendere le decisioni prese da un singolo classificatore.

Un'altra tipologia sono i metodi kernel, algoritmi di apprendimento automatico che consentono trasformazioni non lineari o funzioni decisionali, tra cui le macchine vettoriali di supporto sono probabilmente le più famose e sono state utilizzate con successo in numerose applicazioni [5]. Si basano su una funzione kernel che misura la somiglianza tra qualsiasi coppia di ingressi. Un presupposto fondamentale dei metodi kernel è che il kernel sia positivamente definito. Tuttavia, la DTW non è una distanza perché non soddisfa la disuguaglianza dei triangoli, il che implica che la DTW non può essere utilizzata per definire un kernel positivamente definito. Sebbene la DTW sia stata utilizzata con metodi kernel con alcuni accorgimenti, il fatto che le assunzioni teoriche non siano soddisfatte è un limite importante. È stato proposto un vero kernel positivamente definito per le serie temporali, chiamato kernel di allineamento globale. Il kernel di allineamento globale, denotato con k^{γ}_{GA}, è è definito come la somma di tutti i costi esponenziali negativi su tutti i possibili percorsi di curvatura:

$$k^{\gamma}_{GA} = \sum_{p \epsilon P} \exp\left(-\frac{C_p(X,Y)}{\gamma}\right)$$

dove $C_p(x, y)$ è il costo associato al percorso di warping p, P è l'insieme di tutti

i percorsi di warping e $\gamma > 0$ è un parametro di attenuazione. È importante notare che il kernel di allineamento globale non è un kernel vero e proprio per ogni divergenza locale f. Tuttavia, è possibile dimostrare che, se $1/(1+\exp(f))$ è un kernel positivamente definito, allora k_{GA}^{γ} è un kernel. In particolare, questa condizione è soddisfatta quando f è la funzione differenza quadratica (e più in generale la distanza euclidea al quadrato per le serie temporali multivariate). Il kernel di allineamento globale ha la stessa complessità computazionale del DTW, cioè $O(nm)$, dove n e m sono le lunghezze di entrambe le serie temporali, perché il punteggio tra due serie temporali può essere calcolato utilizzando una relazione di ricorrenza. Le macchine vettoriali di supporto con il kernel di allineamento globale offrono prestazioni predittive migliori rispetto ad altri pseudo kernel basati su DTW per diversi compiti di classificazione di serie temporali multivariate.

In aggiunta è importante citare la shapelet, che è definita come una sottosequenza di osservazioni consecutive di una serie temporale. In alcuni casi d'uso, shapelets specifici possono essere caratteristici delle classi e quindi utili per discriminarle. Diversi algoritmi si basano sugli shapelet, sia estraendo i migliori shapelet dall'insieme dei dati di addestramento, sia imparandoli direttamente.

- **Extrating shapelets**: un algoritmo di questo tipo, sopra citato, è lo Shapelet Transform, che estrae i migliori shapelet da una serie di dati. Sia $X = (x_1, ..., x_n)$ una serie temporale di n osservazioni a valore reale e $S = (s_1, ..., s_l)$ uno shapelet di l numeri reali, con $l \leq n$. La distanza tra lo shapelet S e la serie temporale X, indicata con $d(S, X)$, è definita come il minimo delle distanze euclidee al quadrato tra S e tutti gli shapelet di lunghezza l di X:

$$d(S, X) = \min_{j \epsilon \{0,...,n-l\}} \|S - X_{j:j+l}\|_2^2 = \min_{j \epsilon \{0,...,n-l\}} \sum_{i=1}^{l} (s_i - x_{j+i})^2$$

  L'algoritmo estrae tutti gli shapelets la cui lunghezza appartiene a un intervallo, essendo l'intervallo un iperparametro dell'algoritmo, e seleziona i k migliori shapelets, essendo k un altro iperparametro dell'algoritmo. Questo processo può essere visto come un'estrazione di caratteristiche univariate, dove ogni caratteristica è la distanza tra un dato shapelet e tutte le serie temporali del set di dati. Gli shapelet vengono classificati in base alla statistica F del test di analisi della varianza che confronta le variabilità tra e all'interno della classe. Per estrarre le caratteristiche (cioè gli shapelets) che non sono altamente correlate, vengono rimossi gli shapelets autosimilare; una coppia di shapelets viene considerata autosimilare se proviene dalla stessa serie temporale e ha indici sovrapposti. Una volta identificati i k migliori shapelets e generate le caratteristiche corrispondenti, è possibile applicare a questo nuovo set di dati qualsiasi classificatore standard di apprendimento automatico. Un limite di questo algoritmo è la sua complessità computazionale. Per un set di dati composto da N serie temporali di lunghezza n, esistono $N \times (n - l + 1)$ shapelets di lunghezza l. Poiché vengono esaminati molti valori di $l \ \epsilon \ \{1, ..., n\}$, la complessità

computazionale massima è $O(N \times n^2)$. Inoltre, la definizione di autosimilarità per gli shapelets non tiene conto dei valori degli stessi, il che significa che due shapelets molto simili in termini di distanza euclidea ma estratti da due serie temporali diverse non sono considerati autosimili, anche se producono caratteristiche molto simili.

- **Learning shapelets**: per ovviare alle limitazioni dell'algoritmo della Shapelet Transform, è stato proposto un altro algoritmo che si basa sull'apprendimento delle shapelet, anziché sulla loro estrazione. La distanza tra uno shapelet e una serie temporale definita nell'equazione degli Extrating Shapelets si basa sulla funzione min, che non è differenziabile. Analogamente alla variante soft-DTW della DTW, la funzione minima viene sostituita con una funzione minima regolare, ovvero la funzione LogSumExp, che è differenziabile. L'algoritmo di regressione logistica è utilizzato come classificatore di apprendimento automatico costruito sulla base della trasformazione. Apprendere gli shapelets invece di estrarli presenta diversi vantaggi. In primo luogo, può portare a shapelets che non provengono dall'insieme dei dati, ma che sono discriminativi delle classi. In secondo luogo, non richiede di esaminare l'intero set di dati e quindi può essere più veloce da addestrare, soprattutto con le varianti stocastiche della discesa del gradiente. Tuttavia, l'apprendimento degli shapelets presenta anche degli svantaggi. Poiché devono essere appresi sia gli shapelets sia i coefficienti della regressione logistica, la funzione obiettivo non è convessa. Pertanto, l'algoritmo di ottimizzazione può convergere a un minimo locale negativo. Inoltre, il processo di ottimizzazione è una componente chiave dell'algoritmo e comporta un maggior numero di iperparametri. Infine, poiché l'apprendimento degli shapelets è incorporato nell'algoritmo, potrebbe non essere ottimale provare in seguito altri classificatori oltre alla regressione logistica, mentre l'algoritmo Shapelet Transform è indipendente dal classificatore di apprendimento automatico e quindi la fase di trasformazione può essere calcolata solo una volta e molti classificatori possono essere costruiti su di essa per trovare il classificatore più performante.

Altri algoritmi standard di apprendimento automatico sono quelli basati sugli alberi, come il Random Forest e gli alberi estremamente randomizzati (extremely randomized trees), sono algoritmi popolari che si sono dimostrati potenti. L'uso di tali algoritmi per la classificazione delle serie temporali è stato studiato, dall'estrazione delle caratteristiche utilizzate come input alla modifica della costruzione di ciascun albero dell'ensemble.

Infine è importante citare gli approcci di tipo bag-of-words, noti anche come approcci basati sul dizionario, che consistono nel discretizzare le serie temporali in sequenze di simboli, quindi estrarre le parole da queste sequenze con una finestra scorrevole e infine contare il numero di parole per tutte le parole del dizionario. Questi approcci si dividono in due gruppi: quelli basati sulla discretizzazione delle serie temporali grezze e quelli basati sulla discretizzazione dei coefficienti di Fourier.

2.2 Deep Learning

Dopo aver stabilito l'attuale stato dell'arte del *Machine Learning* per la TSC, parliamo del successo del *Deep Learning* in vari compiti di classificazione che ha motivato il recente utilizzo di modelli di *Deep Learning* per la TSC. Le reti neurali convoluzionali profonde (CNN) hanno rivoluzionato il campo della computer vision [6]. Nel 2015, ad esempio, le CNN sono state utilizzate per raggiungere prestazioni di livello umano nei compiti di riconoscimento delle immagini. Dopo il successo delle reti neurali profonde (DNN) nella computer vision, un'enorme quantità di ricerche ha proposto diverse architetture di DNN per risolvere compiti di elaborazione del linguaggio naturale (NLP) come la traduzione automatica, apprendimento di word embeddings e la classificazione di documenti. I DNN hanno avuto un enorme impatto anche sulla comunità del riconoscimento vocale. È interessante notare che la somiglianza intrinseca tra i compiti di NLP e di riconoscimento vocale è dovuta all'aspetto sequenziale dei dati, che è anche una delle caratteristiche principali delle serie temporali.

Inoltre i dati raccolti dai sensori sono in notevole aumento. Una serie di dati temporali può essere univariata, in cui viene raccolta una sequenza di misurazioni della stessa variabile, o multivariata, in cui viene raccolta una sequenza di misurazioni di più variabili o sensori. Nell'ultimo decennio, la classificazione delle serie temporali multivariate ha ricevuto un notevole interesse. Le classificazioni multivariate delle serie temporali sono applicate all'assistenza sanitaria, alla classificazione dei fonemi, al riconoscimento delle attività, degli oggetti e delle azioni. Nel corso degli anni sono stati sviluppati diversi algoritmi di classificazione delle serie temporali. I metodi basati sulla distanza e i k-Nearest Neighbors (k-NN) si sono dimostrati efficaci nella classificazione di serie temporali multivariate. Numerose ricerche indicano il Dynamic Time Warping (DTW) come la migliore misura basata sulla distanza da utilizzare insieme a k-NN. Oltre alle metriche basate sulla distanza, vengono utilizzati altri algoritmi, ad esempio, gli algoritmi di classificazione basati sulle proprietà. Questi si basano molto sull'estrazione delle proprietà dai dati delle serie temporali. Tuttavia, questa tecnica è ardua perché le proprietà intrinseche dei dati sono difficili da catturare. Per questo motivo, gli approcci basati sulla distanza hanno più successo nella classificazione dei dati delle serie temporali multivariate.

Gli approcci di *Deep Learning* per la TSC possono essere separati in due categorie principali: i modelli generativi e quelli discriminativi [4].

- **Modelli generativi**: i modelli generativi di solito presentano una fase di addestramento non supervisionato che precede la fase di apprendimento del classificatore. Questo tipo di rete è stato definito come classificatore basato su modelli nella comunità TSC. L'obiettivo è trovare una buona rappresentazione delle serie temporali prima di addestrare un classificatore. La fase di pre-addestramento non supervisionato può essere la Stacked Denoising Auto-Encoders (SDAEs). Un modello generativo basato su CNN è il Deep Belief Networks (DBN) utilizzato per modellare le caratteristiche latenti in modo non supervisionato, che vengono poi sfruttate per classificare serie temporali univariate e multivariate. Tramite diversi studi si è

arrivati alla progettazione di un auto-encoder Recurrent Neural Network (RNN) per generare innanzitutto le serie temporali e poi, utilizzando le caratteristiche latenti apprese, si è addestrato un classificatore (come SVM o Random Forest) in cima a queste caratteristiche per prevedere la classe di una data serie temporale in ingresso. Altri studi, hanno utilizzato la modellazione auto-preventiva per la classificazione delle serie temporali, in cui le Echo State Network (ESN) sono state prima utilizzate per ricostruire le serie temporali e poi la rappresentazione appresa nello spazio riservato è stata utilizzata per la classificazione. Altri approcci basati sulle ESN definiscono un kernel sulle caratteristiche apprese seguito da un classificatore SVM o Multilayer Perceptron (MLP).

- **Modelli discriminativi**: un modello di *Deep Learning* discriminativo è un classificatore (o regressore) che apprende direttamente la mappatura tra l'input grezzo di una serie temporale (o le sue caratteristiche ingegnerizzate) e produce una distribuzione di probabilità sulle variabili di classe in un set di dati. Sono state proposte diverse architetture discriminative di *Deep Learning* per risolvere il compito TSC. Questo tipo di modello può essere ulteriormente suddiviso in due gruppi: modelli di *Deep Learning* con caratteristiche modificate e modelli di *Deep Learning* end-to-end. Il metodo di estrazione delle caratteristiche più frequentemente utilizzato e ispirato alla Computer Vision per gli approcci di ingegneria manuale è la trasformazione delle serie temporali in immagini utilizzando metodi di imaging specifici come i campi di Gramian, i diagrammi di ricorrenza e i campi di transizione di Markov. A differenza della trasformazione delle immagini, gli altri metodi di estrazione delle caratteristiche non sono agnostici rispetto al dominio. Queste caratteristiche vengono prima elaborate a mano utilizzando una certa conoscenza del dominio e poi alimentate da una classe discriminativa di apprendimento profondo. Ad esempio, diverse caratteristiche (come la velocità) sono state estratte dai dati del sensore posizionato sulla mano di un chirurgo per determinare il livello di abilità durante l'addestramento chirurgico. In effetti, la maggior parte degli approcci di *Deep Learning* per la TSC con alcune caratteristiche ingegnerizzate della mano sono presenti in compiti di riconoscimento dell'attività umana. A differenza dell'ingegneria delle caratteristiche, il *Deep Learning* end-to-end mira a incorporare il processo di apprendimento delle caratteristiche durante la messa a punto del classificatore discriminativo. Poiché questo tipo di approccio al *Deep Learning* è agnostico rispetto al dominio e non include alcuna fase di pre-elaborazione specifica del dominio, si è deciso di separare ulteriormente questi approcci end-to-end in base alle loro architetture di rete neurale. Si è progettato un MLP per apprendere da zero un classificatore di serie temporali discriminativo. Il problema di un approccio MLP è che le informazioni temporali vengono perse e le caratteristiche apprese non sono più variabili nel tempo. È qui che le CNN sono più utili, apprendendo filtri (o caratteristiche) spazialmente invarianti dalle serie temporali grezze in ingresso. Si è riscontrato che le CNN sono l'architettura più applicata per il problema TSC, probabilmente grazie alla

loro robustezza e al tempo di addestramento relativamente ridotto rispetto ad architetture complesse come le RNN o le MLP. Sono state proposte e convalidate diverse varianti delle CNN su un sottoinsieme dell'archivio UCR/UEA, come le Residual Networks (ResNets) che aggiungono connessioni lineari brevi per gli strati convoluzionali potenzialmente in grado di migliorare l'accuratezza del modello. Più recentemente, le architetture proposte sono state modificate per sfruttare una tecnica di inizializzazione dei filtri basata sui valori Wavelet Daubechies 4. Al di fuori dell'archivio UCR/UEA, il *Deep Learning* ha raggiunto lo stato dell'arte su diversi set di dati in diversi domini. Per problemi di previsione di serie spazio-temporali, come la meteorologia e l'oceanografia, le DNN sono riuscite a rilevare infrazioni miocardiche da dati elettrocardiografici utilizzando le deep CNN. Per il riconoscimento di attività umane da sensori indossabili, il *Deep Learning* sta sostituendo gli approcci di feature engineering, in cui le caratteristiche non sono più disegnate a mano ma apprese da modelli di *Deep Learning* addestrati tramite backpropagation. Un altro tipo di dati di serie temporali è presente nelle cartelle cliniche elettroniche, dove una recente rete generativa in contrapposizione ad una CNN è stata addestrata per la previsione del rischio sulla base delle cartelle cliniche storiche dei pazienti. Le CNN sono state progettate anche per raggiungere prestazioni all'avanguardia per l'identificazione delle competenze chirurgiche. Si è sfruttato un modello CNN per caratteristiche multivariate e lag-feature al fine di raggiungere un'accuratezza allo stato dell'arte sui dati della sfida Prognostics and Health Management (PHM). Infine, una ulteriore evoluzione del *Deep Learning* per la classificazione dei segnali fisiologici ha rivelato che le CNN sono l'architettura più popolare per il compito considerato. Un ultimo tipo di architetture ibride che hanno mostrato risultati promettenti per il compito TSC sui dataset dell'archivio UCR/UEA, dove principalmente le CNN sono state combinate con altri tipi di architetture come le Gated Recurrent Unit e il meccanismo di attenzione.

Capitolo 3

Strumenti e metodi utilizzati

In questo capitolo si analizzeranno le principali tecnologie e soluzioni software adottate, il dataset utilizzato e si descriverà il modo con il quale si è ottenuto. Verrà inoltre affrontata la descrizione delle *Time Series* e verrà spiegato cosa si intende per *Deep Learning*. Si approfondirà l'algoritmo *InceptionTime* nella sua implementazione e si descriverà il modo in cui vengono prodotti i risultati tramite la matrice di confusione.

3.1 Python

Python è un linguaggio di programmazione ad alto livello orientato ad oggetti e multi-paradigma, particolarmente adatto, tra altri usi, per sviluppare applicazioni distribuite, scripting, computazione numerica e system testing [7]. Nella Figura 3.1 viene mostrato il logo di *Python*.

Figura 3.1: Logo del linguaggio di programmazione *Python*. Fonte: [8].

La sua sintassi elegante lo rende un linguaggio di programmazione estremamente semplice da leggere ed utilizzare. Date le sue caratteristiche e vista la sua versatilità, *Python* è stato prontamente apprezzato dalla community e negli ultimi anni è stato soggetto ad uno sviluppo esponenziale. Questo linguaggio si è arricchito di un elevato numero di librerie di terze parti facilmente integrabili, modularmente, mediante l'utilizzo di un package manager (come ad esempio *pip*). Dovendo affrontare problemi basati sul *Deep Learning*, la scelta del linguaggio di programmazione con il quale procedere è ricaduta proprio su *Python*, in quanto la maggior parte delle librerie che ne implementano gli algoritmi sono disponibili per questo linguaggio, e sono particolarmente ben mantenute con aggiornamenti regolari. Non a caso, a partire dall'anno 2016, *Python* ha regi-

strato un notevole incremento nell'utilizzo da parte dell'utenza, rispetto ad altri linguaggi ottimizzati per le data science, come si vede nella Figura 3.2.

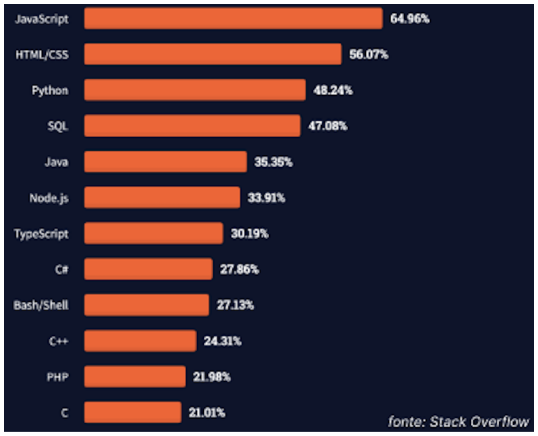

Figura 3.2: Linguaggi di programmazione più usati nel mondo nel 2022. Fonte: [9].

3.1.1 Keras

Keras è una libreria open-source molto popolare scritta in *Python* per l'apprendimento automatico e le reti neurali [10]. Nella Figura 3.3 viene mostrato il logo di *Keras*.

Figura 3.3: Logo della libreria open-source *Keras*. Fonte: [10].

È progettata come un'interfaccia ad un livello di astrazione superiore rispetto ad altre librerie simili, e supporta come back-end principalmente *TensorFlow*, a sua volta un framework per l'apprendimento automatico sviluppato da Google, ma anche la libreria *PyTorch*. Progettata per permettere una rapida prototipazione di reti neurali standard e convoluzionali, *Keras* si concentra sulla facilità d'uso, la modularità e l'estensibilità. La libreria può essere installata utilizzando il package manager:

```
pip install keras
```

Questa libreria contiene numerose implementazioni di blocchi di reti neurali usati comunemente, come i livelli e le funzioni di attivazione, studiati più in dettaglio nel quarto capitolo, oltre che tutta una serie di strumenti utili a semplificare il lavoro con dati di immagini e testo.

3.2 L'Ambiente GIS

Il sistema informatico geografico *GIS* (Geographic Information System) è un sistema computerizzato che permette l'acquisizione, registrazione, analisi, visualizzazione, restituzione, condivisione e presentazione di informazioni derivanti da dati geografici [11]. La tecnologia *GIS* integra in un unico ambiente le più comuni operazioni legate all'uso di database, quali interrogazioni ed analisi statistiche, con l'analisi geografica consentita dalle cartografie numeriche. Infatti, i *GIS* permettono di analizzare una entità geografica sia per la sua completa natura geometrica sia per il suo totale contenuto informativo. I dati possono essere correlati alla loro posizione geografica in due tipologie principali: vettoriali e raster. I dati vettoriali sono costituiti da elementi semplici quali punti, linee e poligoni, codificati e memorizzati sulla base delle loro coordinate. Ad esempio, un punto viene individuato attraverso una coppia di coordinate reali $(x,\ y)$, mentre una linea o un poligono attraverso la posizione dei suoi nodi $(x_0,\ y_0;\ x_1,\ y_1;\ ...)$. A ciascun elemento è associato un record del database che contiene tutti gli attributi dell'oggetto rappresentato. Il dato raster permette di rappresentare il mondo reale attraverso una matrice di celle, generalmente di forma quadrata o rettangolare, dette pixel. A ciascun pixel sono associate le informazione relative a ciò che esso rappresenta sul territorio. La dimensione del pixel è strettamente relazionata alla precisione del dato.

3.2.1 QGIS

QGIS è un'applicazione desktop *GIS* open-source, molto simile nell'interfaccia utente e nelle funzioni ai pacchetti *GIS* commerciali equivalenti. Il software è pubblicato come multi-piattaforma disponibile per i più comuni sistemi operativi come Windows, Linux, Unix, macOS e, sperimentalmente, Android [12]. Nella Figura 3.4 viene mostrato il logo di *QGIS*.

Figura 3.4: Logo di *QGIS*. Fonte: [12].

Paragonato a software della stessa classe, *QGIS* risulta occupare una minore dimensione sul disco e, mediamente ed a parità di operazioni, necessita di una minore quantità di *RAM* (Random Access Memory) per essere eseguito. QGIS permette di far confluire dati provenienti da diverse fonti in un unico progetto di analisi territoriale. I dati, divisi in Layers, possono essere analizzati e da essi viene creata l'immagine mappa con il graficismo che può essere personalizzato dall'utente ed eventualmente rispondere alle analisi tipiche del GIS, ovvero gradazione di colori, sfumatura di colore, valore unico. La mappa può essere arricchita da icone e da etichette dipendenti dagli attributi degli elementi cartografici. Un sistema di scripting può essere invocato per gestire operazioni ripetitive sui dati.

3.3 Altre tecnologie

3.3.1 GitHub

GitHub è un servizio di hosting per progetti software, di proprietà della società GitHub Inc. [13]. Tra i molti soggetti che offrono servizi a livello internazionale che usano *GitHub*, le principali sono Google, Apple, Microsoft, NASA, Meta e molti altri. Nella Figura 3.5 viene mostrato il logo di *GitHub*.

Figura 3.5: Logo di *GitHub*. Fonte: [13].

Il sito è principalmente utilizzato da sviluppatori che caricano il codice sorgente di programmi e lo rendono scaricabile e migliorabile da altre persone. Questi ultimi possono interagire con gli sviluppatori tramite un sistema per inviare segnalazione di bug o funzionalità (issue tracker), un sistema per copiare il software in una versione modificabile (fork), un sistema per proporre modifiche agli sviluppatori originali (pull request) e un sistema di discussione legato al codice del repository. *GitHub* fornisce anche dei report sui contributori più attivi o sul codice.

3.3.2 Colab

Colab o "Colaboratory" permette di scrivere ed eseguire *Python* nel browser senza che sia richiesta una pre-configurazione. Fornisce l'accesso alle GPU (Graphics Processing Unit) e permette una condivisione di codice in maniera rapida [14]. Nella Figura 3.6 viene mostrato il logo di *Colab*.

Figura 3.6: Logo di *Colab*. Fonte: [14].

Un aspetto fondamentale sono i blocchi note che permettono di combinare codice eseguibile e RTF (Rich Text Format) in un unico documento, insieme a immagini, HTML, LaTeX e altro ancora. Quando si creano dei blocchi note *Colab*, questi vengono archiviati nell'account Google Drive di appartenenza. I blocchi note *Colab* sono blocchi note *Jupyter* ospitati da *Colab*. JupyterLab è il più recente ambiente di sviluppo interattivo basato sul web per notebook, codice e dati. La sua interfaccia flessibile consente di configurare e organizzare i flussi di lavoro nei settori della scienza dei dati, dell'informatica scientifica, del giornalismo computazionale e dell'apprendimento automatico. Un design modulare invita a estensioni per espandere e arricchire le funzionalità [15]. Con *Colab* si può inoltre sfruttare tutta la potenza delle librerie *Python* per analizzare

e visualizzare i dati, come ad esempio *numpy* per generare alcuni dati casuali e *matplotlib* per visualizzarli.

3.4 Catalogazione delle tecniche di pesca

Esistono molti tipi di pesca catalogati grazie all'analisi del dataset attraverso gli algoritmi previsionali utilizzati. Alcuni di questi sono consultabili attraverso l'Organizzazione delle Nazioni Unite per l'alimentazione e l'agricoltura. In questo progetto è necessario solo di un insieme più ristretto di attrezzi utilizzati per le varie tecniche di pesca:

- **LLD**: i palangari, utilizzati per la pesca a strascico e la pesca al bottello;
- **LLS**: i palangari fissi per la cattura di specie demersali (naselli, gronchi, corvine, rombi, palombi, saraghi);
- **OTB**: le reti a strascico a divergenti che sono componenti essenziali del traino in quanto assicurano l'apertura orizzontale di tutta l'attrezzatura;
- **TBB**: la sfogliara per la cattura di sogliole, rombi e passere;
- **PS**: le reti da circuizione utilizzate per pescare specie che vivono in banco, da quelli più piccoli come sardine e acciughe a quelli più grandi come quelli di sgombri e tonni;
- **PTM**: le reti a strascico a coppia in acque medie con la caratteristica di essere reti "attive" in quanto catturano il pesce nel loro progressivo avanzamento.

Tutti i tipi elencati rappresentano tipi di pesca legale presenti nel documento ufficiale della comunità europea e sono mostrati nella Figura 3.7.

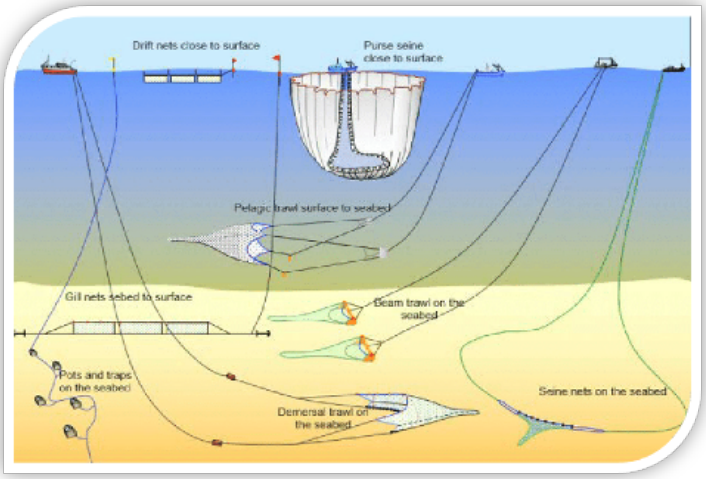

Figura 3.7: Tecniche di pesca. Fonte: [16].

3.5 Pre-elaborazione del Dataset

In questa sezione ci si occuperà, in dettaglio, del set di dati sollecitato. Si elencherà il tipo di dati analizzato e tutti i procedimenti da applicare, compreso il filtraggio, per ottenere un buon dataset. Il dataset utilizzato è costruito a partire da due file csv: RAW e SESSION.

1. **RAW** identifica i dati grezzi, sporchi e con rumore per ogni singola barca. In questo file sono presenti una serie di caratteristiche che devono essere analizzate in dettaglio: la struttura è a colonne, ognuna delle quali inizializza un determinato parametro. Partendo da sinistra verso destra abbiamo:

   - **MMSI** che identifica il numero univoco di un peschereccio;
   - **ID** ovvero il numero di sessione di quella determinata nave in un determinato giorno;
   - **DATE** che identifica la data del ping;
   - **TIME** che indica il tempo del ping;
   - **LATITUDE** e **LONGITUDE** posizione sulla terra in quello specifico momento in base alla latitudine ed alla longitudine;
   - **COURSE** che indica l'angolo di rotazione della nave;
   - **SPEED** ovvero la velocità in quella posizione specifica;
   - **GEOM** cioè le coordinate GIS per poter aprire questo set di dati su un database postGIS, ovvero un programma open source che aggiunge il supporto per gli oggetti geografici al database relazionale a oggetti PostgreSQL. PostGIS segue le specifiche "Simple Features for SQL" dell'Open Geospatial Consortium (OGC) [17];
   - **ETA** ovvero il tempo di durata della pesca;

   I dati grezzi si concentrano su ogni singola sessione di ping della nave. Quando si parla di una singola sessione di ping si intende una sequenza di dati che viene inviata automaticamente ogni 5 minuti a un server che inserisce questi dati in un database.

2. **SESSION** identifica una sessione completa di una certa nave. Tutto ciò che è presente nella tabella RAW è confluito in questa tabella per organizzare e rendere più leggibile e ricercabile il RAW. Infatti, la tabella SESSION mostra la sessione iniziale e finale di ogni singola nave e i punti di partenza da un porto e di ritorno. Il tutto è organizzato in questo modo:

   - **VASSEL** ovvero l'identificazione del peschereccio;
   - **SESSION** cioè la sessione di riferimento del peschereccio;
   - **STARTDATA** che indica la data di partenza da un porto;
   - **ENDDATA** che identifica la data di arrivo al porto e di fine pesca;
   - **GEAR** cioè il tipo di pesca valutato da un team di esperti del settore.

3.6 Time Series

In matematica, una *Time Series* (serie temporale) è una serie di punti di dati indicizzati (o elencati) in ordine temporale. Più comunemente, una serie temporale è una sequenza presa in punti successivi ugualmente distanziati nel tempo. Si tratta quindi di una sequenza di dati a tempo discreto [18]. Esempi di serie temporali sono le altezze delle maree oceaniche o il conteggio delle macchie solari. Sono molto spesso rappresentate da un run chart (un grafico a linee temporali). Vengono utilizzate nella statistica, nell'elaborazione dei segnali, nel riconoscimento dei modelli, nell'ingegneria delle comunicazioni e, in generale, in tutti i settori della scienza applicata e dell'ingegneria che prevedono misurazioni temporali.

L'analisi delle *Time Series* comprende i metodi per analizzare i dati al fine di estrarre statistiche significative e altre caratteristiche dei dati stessi. La previsione consiste nell'utilizzo di un modello per prevedere i valori futuri sulla base dei valori osservati in precedenza. Sebbene l'analisi di regressione venga spesso impiegata per verificare le relazioni tra una o più serie temporali diverse, questo tipo di analisi non viene solitamente definito "analisi delle serie temporali", che si riferisce in particolare alle relazioni tra diversi punti nel tempo all'interno di una singola serie. I dati hanno un ordine temporale naturale. Ciò distingue l'analisi delle serie temporali dagli studi trasversali, in cui non esiste un ordine naturale delle osservazioni (ad esempio, il livello dei salari delle persone in riferimento ai rispettivi livelli di istruzione, in cui i dati degli individui potrebbero essere inseriti in qualsiasi ordine).

L'analisi delle serie temporali si distingue anche dall'analisi dei dati spaziali, in cui le osservazioni si riferiscono tipicamente a luoghi geografici (ad esempio, la contabilizzazione dei prezzi delle case in base alla loro ubicazione e alle caratteristiche intrinseche delle case stesse). Un modello stocastico per una serie temporale rifletterà generalmente il fatto che le osservazioni vicine nel tempo saranno più strettamente correlate rispetto a quelle più distanti. Inoltre, i modelli di serie temporali faranno spesso uso del naturale ordinamento unidirezionale del tempo, in modo che i valori di un dato periodo siano espressi come derivanti in qualche modo dai valori passati, piuttosto che da quelli futuri.

L'analisi delle serie temporali può essere applicata a dati continui a valore reale, a dati numerici discreti o a dati simbolici discreti (cioè sequenze di caratteri come le lettere). Inoltre, le tecniche di analisi delle serie temporali possono essere suddivise in metodi parametrici e non parametrici. Gli approcci parametrici presuppongono che il processo stocastico stazionario sottostante abbia una certa struttura che può essere descritta con un numero ridotto di parametri (ad esempio, utilizzando un modello autoregressivo o a media mobile). In questi approcci, il compito è stimare i parametri del modello che descrive il processo stocastico. Al contrario, gli approcci non parametrici stimano esplicitamente la covarianza o lo spettro del processo senza assumere che il processo abbia una struttura particolare.

Un modello di *Time Series* basato sulla regressione è:

$$y(t) = x(t)\beta + \epsilon(t)$$

dove $y(t) = \{y_t; t = 0, \pm 1, \pm 2, ...\}$ è una sequenza, indicizzata dal pedice temporale t, che è una combinazione di una sequenza di segnali osservabili $x(t) = \{x_t\}$ e una sequenza di rumore non osservabile $\epsilon(t) = \{\epsilon_t\}$ di variabili casuali indipendenti e identicamente distribuite [19]. Un modello più generale, che chiameremo modello generale di regressione temporale, è quello che postula una relazione che comprende un numero qualsiasi di elementi consecutivi di $x(t)$, $y(t)$ e $\epsilon(t)$. Il modello può essere rappresentato dall'equazione

$$\sum_{i=0}^{p} \alpha_i y(t-i) = \sum_{i=0}^{k} \beta_i x(t-i) + \sum_{i=0}^{q} \mu_i \epsilon(t-i)$$

dove di solito si dà per scontato che $\alpha_0 = 1$. Questa normalizzazione del coefficiente principale sul lato sinistro dell'equazione identifica $y(t)$ come sequenza di uscita. Qualsiasi delle somme dell'equazione possono essere infinite, ma se si vuole che il modello sia valido, le sequenze di coefficienti $\{\alpha_i\}$, $\{\beta_i\}$ e $\{\mu_i\}$ possono dipendere solo da un numero limitato di parametri.

Esaminiamo alcune definizioni formali per le TSC:

- una serie temporale univariata $X = [x_1, x_2, ..., x_T]$ è un insieme ordinato di valori reali. La lunghezza di X è pari al numero di valori reali T;
- un Multivariate Time Series M-dimensionale (MTS M-dimensionale), $X = [X^1, X^2, ..., X^M]$ consiste in M diverse serie temporali univariate con $X^i \, \epsilon \, R^T$;
- un dataset $D = (X_1, Y_1), (X_2, Y_2), ..., (X_N, Y_N)$ è un insieme di coppie (X_i, Y_i) dove X_i può essere una serie temporale univariata o multivariata e Y_i è il corrispondente vettore di label One-Hot. Per un dataset contenente K classi, il vettore di label One-Hot Y_i è un vettore di lunghezza K in cui ogni elemento $j \, \epsilon \, [1, K]$ è uguale a 1 se la classe di X_i è j e a 0 altrimenti.

Il compito della TSC consiste nell'addestrare un classificatore su un insieme di dati D per mappare dallo spazio dei possibili input una distribuzione di probabilità sui valori delle variabili di classe (label) [4].

3.6.1 Codifica One-Hot

Approfondiamo la codifica *One-Hot* per capirne il suo necessario utilizzo. I modelli di *Machine Learning* non possono usare direttamente il testo per il loro apprendimento. Occorre trasformare ogni carattere o parola in un numero prima [20].

Facciamo un esempio. Supponiamo di voler creare una rappresentazione numerica delle parole:

- Re;
- Regina;
- Principe;
- Principessa;

Il modo più semplice di codificare queste parole sarebbe di assegnare a ognuna di esse un numero, in maniera sequenziale. Le parole sono state correttamente trasformate in formato numerico, seguendo la mappatura:

```
map = {
    "Re" : 1,
    "Regina" : 2,
    "Principe" : 3,
    "Principessa" : 4
}
```

Ma c'è un problema. Se fornissimo questi dati a qualsiasi modello predittivo, questo andrebbe ad assegnare un valore matematico più alto a principe e principessa, rendendoli più importanti di re e regina. Ovviamente questo andrebbe a fornire informazioni sbagliate al modello, che apprenderebbe relazioni sbagliate. Dobbiamo rendere la nostra rappresentazione numerica più precisa. Per risolvere il problema di rappresentazione numerica descritto sopra, è possibile utilizzare la tecnica del *One-Hot Encoding*. In questo caso, ogni parola sarebbe rappresentata da un vettore numerico di dimensione pari al numero totale di parole che si vuole rappresentare. Il vettore avrebbe tutti i valori pari a zero, tranne uno, che rappresenta la parola specifica. Ad esempio, nel caso delle quattro parole "Re", "Regina", "Principe" e "Principessa", ogni parola sarebbe rappresentata da un vettore di quattro elementi, con il valore "1" nella posizione corrispondente alla parola e "0" in tutte le altre posizioni come mostrato nella Tabella 3.1.

Re	[1, 0, 0, 0]
Regina	[0, 1, 0, 0]
Principe	[0, 0, 1, 0]
Principessa	[0, 0, 0, 1]

Tabella 3.1: Rappresentazione con codifica One-Hot.

Questa tecnica risolve il problema di assegnare un valore matematico più alto a parole che non sono più importanti di altre nella rappresentazione numerica.

3.7 Deep Learning

Il *Deep Learning* (apprendimento profondo) è quel campo di ricerca del *Machine Learning* e dell'*Intelligenza Artificiale* che si basa su diversi livelli di rappresentazione, corrispondenti a gerarchie di caratteristiche di fattori o concetti, dove i concetti di alto livello sono definiti sulla base di quelli di basso come mostrato nella Figura 3.8. In altre parole, per apprendimento profondo si intende un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in maniera sempre più completa [21].

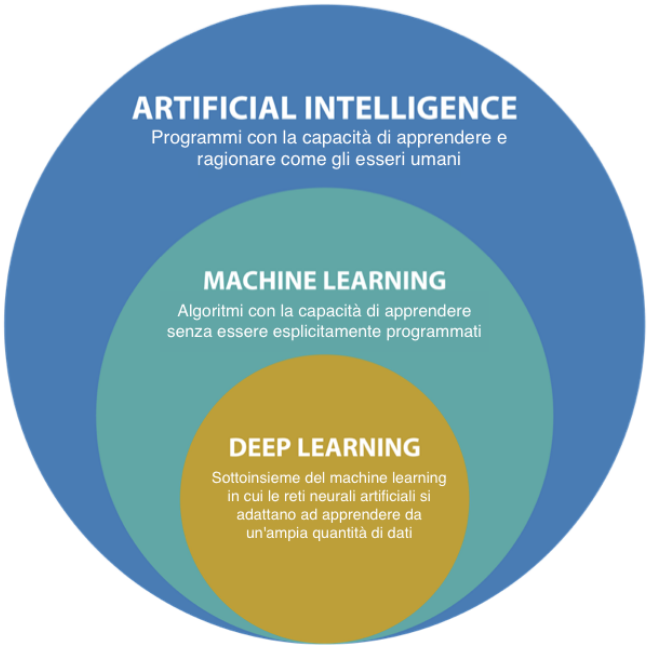

Figura 3.8: Sottoinsiemi dell'*Intelligenza Artificiale*. Fonte: [22].

Tra le architetture di apprendimento profondo si annoverano le reti neurali profonde, la convoluzione di reti neurali profonde e le reti neurali ricorsive, che sono state applicate nella visione artificiale, nel riconoscimento automatico del discorso, nell'elaborazione del linguaggio naturale, nel riconoscimento audio e nella bioinformatica. L'apprendimento profondo è definito come una classe di algoritmi di apprendimento automatico che:

- usano vari livelli di unità non lineari a cascata per svolgere compiti di estrazione di caratteristiche e di trasformazione. Ciascun livello successivo utilizza l'uscita del livello precedente come input. Gli algoritmi possono essere sia di tipo supervisionato sia non supervisionato e le applicazioni includono l'analisi di pattern (apprendimento non supervisionato) e classificazione (apprendimento supervisionato);
- sono basati sull'apprendimento non supervisionato di livelli gerarchici multipli di caratteristiche (e di rappresentazioni) dei dati;
- le caratteristiche di più alto livello vengono derivate da quelle di livello più basso per creare una rappresentazione gerarchica;
- fanno parte della più ampia classe di algoritmi di apprendimento della rappresentazione dei dati all'interno dell'apprendimento automatico;
- apprendono multipli livelli di rappresentazione che corrispondono a differenti livelli di astrazione; questi livelli formano una gerarchia di concetti.

Ciò che queste definizioni hanno in comune sono i livelli multipli di unità non lineari e l'apprendimento (supervisionato o non supervisionato) in ogni livello della rappresentazione di caratteristiche, in cui i livelli formano una gerarchia

delle caratteristiche stesse. La composizione di ciascun livello di unità non lineari usata in un algoritmo di apprendimento profondo dipende dal problema che deve essere risolto. Nell'apprendimento profondo possono venire usati livelli nascosti di una rete neurale artificiale e insiemi di formule proposizionali. Esistono alcuni tipi di reti neurale, tra i quali:

- la **rete neurale convoluzionale** (Convolution Neural Network, CNN) che è un metodo di scelta per elaborare dati visuali e dati di tipo 2D. Una CNN è composta da uno più strati convoluzionali con strati completamente connessi verso l'alto. Usa anche pesi e strati comuni (pooling layer). Quest'architettura permette alle CNN di avere dei vantaggi dalle strutture 2D di ingresso. Sono particolarmente efficaci nell'area delle immagini e di riconoscimento del discorso. Possono essere allenate anche con la backpropagation standard. Sono inoltre facili da allenare rispetto ad altre reti neurali profonde o feed-forward ed hanno molti meno parametri da stimare;

- le **reti neurali ricorsive** (Recurrent Neural Networks, RNN) nascono con il tentativo di rendere le reti neurali Turing complete aggiungendo una componente di memoria. Le reti neurali feed-forward rispondono in modo costante agli stessi input, senza però poter fare collegamenti tra input diversi, come potrebbe essere utile nell'ambito della semantica. Le reti neurali ricorsive invece pesano ogni input in base allo stato precedente e lo stesso input potrebbe dare output diversi a seconda del contesto in cui è inserito. Nonostante questo le RNN, a meno che non vengano inseriti elementi casuali, restano reti deterministiche in quanto la stessa sequenza di input porterà sempre alla stessa sequenza di output. Lo stato di una RNN è implementato aggiungendo a un layer dei neuroni ricorsivi che avranno come valore ad ogni istante di tempo il valore all'istante precedente sommato a una variazione data dagli input dei neuroni feed-forward. Le reti neurali di questo tipo diventano impossibili da allenare per retropropagazione dell'errore, quindi viene utilizzato come stratagemma quello di svolgere le ricorsività considerando il loro funzionamento per ogni singolo istante di tempo. Mentre nel caso di percorsi a ritroso molto brevi questo funziona, per i casi in cui bisogna percorrere diversi step a ritroso si incorre nel problema del Vanishing Gradient o dell'Exploding Gradient nel caso la funzione di costo abbia derivata sempre minore di uno (ad esempio sigmoide e tanh) o sempre maggiore di uno. Questo infatti porta a una convergenza a zero esponenziale nel primo caso e a un'esplosione esponenziale del gradiente nel secondo. In nessun caso il gradiente è più computabile anche solo dopo pochi passi. Per risolvere questo problema si fa in modo di utilizzare una funzione rettificatrice.

3.7.1 Deep learning per la classificazione delle serie temporali

Ora ci concentriamo sulle TSC utilizzando le DNN (Deep Neural Networks), considerate modelli di apprendimento automatico complessi. La Figura 3.9 illustra

un quadro generale di *Deep Learning* per le TSC. Queste reti sono progettate per apprendere rappresentazioni gerarchiche di dati.

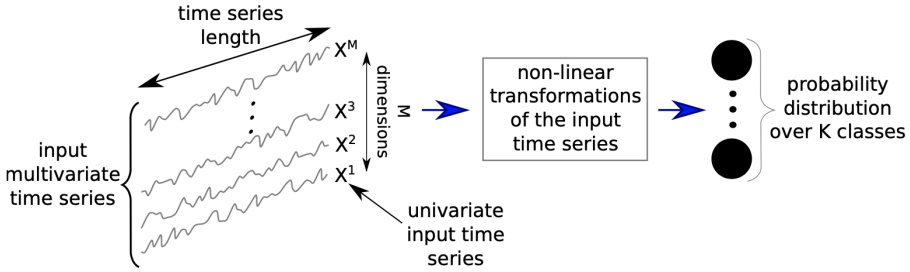

Figura 3.9: Un framework unificato di *Deep Learning* per la classificazione delle serie temporali. Fonte: [4].

Una rete neurale profonda è una composizione di L funzioni parametriche denominate strati, dove ogni strato è considerato una rappresentazione del dominio di ingresso. Uno strato l_i, come $i \epsilon 1...L$, contiene neuroni, che sono piccole unità che calcolano un elemento dell'output dello strato. Lo strato l_i prende in ingresso l'uscita dello strato precedente l_{i-1} e applica una non linearità (come la funzione sigmoide) per calcolare la propria uscita. Il comportamento di queste trasformazioni non lineari è controllato da un insieme di parametri θ_i per ogni strato. Nel contesto delle DNN, questi parametri sono chiamati pesi che collegano l'ingresso dello strato precedente all'uscita dello strato corrente. Quindi, dato un input x, una rete neurale esegue i seguenti calcoli per predire la classe:

$$f_L(\theta_L, x) = f_{L-1}(\theta_{L-1}, f_{L-2}(\theta_{L-2}, ..., f_1(\theta_1, x)))$$

dove f_i corrisponde alla non linearità applicata allo strato l_i. Questo processo viene definito propagazione a *feed-forward*. Durante l'addestramento, alla rete viene presentato un certo numero di *input-output* noti (ad esempio un set di dati D). In primo luogo, i pesi vengono inizializzati in modo casuale, anche se un'alternativa potrebbe essere quella di prendere un modello pre-addestrato su un set di dati di partenza e metterlo a punto sul set di dati di arrivo. Dopo l'inizializzazione del peso, viene applicato un passaggio in avanti attraverso il modello: utilizzando la funzione f viene calcolata l'uscita di un ingresso x. L'output è un vettore le cui componenti sono le probabilità stimate di x appartenenti a ciascuna classe. La perdita di predizione del modello viene calcolata utilizzando una funzione di costo, per esempio il log likelihood negativo. Quindi, utilizzando la discesa del gradiente, i pesi vengono aggiornati in un passaggio all'indietro per propagare l'errore. In questo modo, eseguendo iterativamente un passaggio in avanti seguito da una retro propagazione, i parametri del modello vengono aggiornati in modo da minimizzare la perdita sui dati di addestramento. Durante il test, il classificatore probabilistico (il modello) viene testato su dati non visibili, che viene anche chiamato fase di inferenza: un passaggio in avanti su questo input non visibile seguito da una predizione della classe. La predizione corrisponde alla classe la cui probabilità è massima. Per misurare le prestazioni del modello sui dati di prova, si può usare la misura di accuratezza. Un vantaggio delle DNN rispetto ai classificatori non probabilistici (come NN-DTW) è

che la rete prende una decisione probabilistica, consentendo così di misurare la fiducia di una certa previsione fornita da un algoritmo.

3.7.2 Reti neurali convoluzionali

Una convoluzione può essere vista come l'applicazione e lo scorrimento di un filtro sulla serie temporale. A differenza delle immagini, i filtri presentano una sola dimensione (il tempo) anziché due (larghezza e altezza). Il filtro può anche essere visto come una generica trasformazione non lineare di una serie temporale. Concretamente, se stiamo convolvendo (moltiplicando) un filtro di lunghezza 3 con una serie temporale univariata, impostando i valori del filtro come $[1/3, 1/3, 1/3]$, la convoluzione risulterà nell'applicazione di una media mobile con una finestra scorrevole di lunghezza 3. Una forma generale di applicazione della convoluzione per un timestamp centrato t è data dalla seguente equazione:

$$C_t = f(\omega * X_{t-l/2:t+l/2} + b) \; \forall \; t \; \epsilon \; [1, T]$$

dove C denota il risultato di una convoluzione (simbolo $*$), come mostrato nella Figura 3.10, applicata a una serie temporale univariata X di lunghezza T con un filtro ω di lunghezza l, un parametro di bias b e una funzione finale non lineare f, come ad esempio l'unità lineare rettificata (ReLU). Il risultato di una convoluzione (un filtro) su una serie temporale di ingresso X può essere considerato come un'altra serie temporale univariata C sottoposta a un processo di filtraggio. Pertanto, applicando più filtri a una serie temporale si otterrà una serie temporale multivariata le cui dimensioni sono pari al numero di filtri utilizzati.

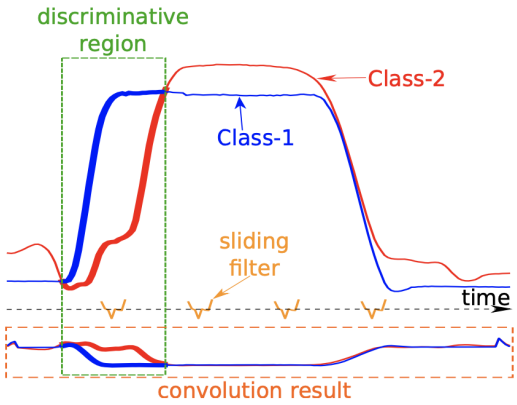

Figura 3.10: Il risultato dell'applicazione di una convoluzione discriminativa acquisita in un dataset di prova. Fonte: [4].

L'intuizione alla base dell'applicazione di più filtri a una serie temporale in ingresso è quella di apprendere più caratteristiche discriminative utili per il compito di classificazione. La stessa convoluzione sarà utilizzata per trovare il risultato per tutti gli intervalli di tempo $t \; \epsilon \; [1, T]$. Questa è una proprietà molto potente (chiamata condivisione dei pesi) delle CNN, che consente loro di apprendere filtri invarianti nella dimensione temporale. Infine, invece di impostare manualmente i valori del filtro ω, questi valori dovrebbero essere appresi

automaticamente, poiché dipendono fortemente dal set di dati in questione. Ad esempio, per un dataset il filtro ottimale sarebbe pari a $[1, 2, 2]$, mentre per un altro dataset il filtro ottimale sarebbe pari a $[2, 0, -1]$. Per filtro ottimale intendiamo un filtro la cui applicazione permetterà al classificatore di discriminare facilmente tra le classi del dataset. Per acquisire automaticamente un filtro discriminativo, la convoluzione deve essere seguita da un classificatore discriminativo, che di solito è preceduto da un'operazione di pooling che può essere locale o globale. Il pooling locale, come il pooling medio o massimo, prende una serie temporale in ingresso e ne riduce la lunghezza T aggregandola su una finestra scorrevole della serie temporale. Ad esempio, se la lunghezza della finestra scorrevole è pari a 3, la serie temporale risultante avrà una lunghezza pari a $T/3$ (questo è vero solo se il passo è pari alla lunghezza della finestra scorrevole). Con un'operazione di pooling globale, le serie temporali verranno aggregate sull'intera dimensione temporale, ottenendo un unico valore reale. In altre parole, è simile all'applicazione di un pooling locale con una finestra scorrevole di lunghezza pari alla lunghezza della serie temporale in ingresso. Di solito un'aggregazione globale viene adottata per ridurre drasticamente il numero di parametri in un modello, diminuendo così il rischio di overfitting (si ha quando un modello statistico molto complesso si adatta ai dati osservati perché ha un numero eccessivo di parametri rispetto al numero di osservazioni). Nella Figura 3.11 è mostrata un'architettura CNN per la TSC con tre strati convoluzionali.

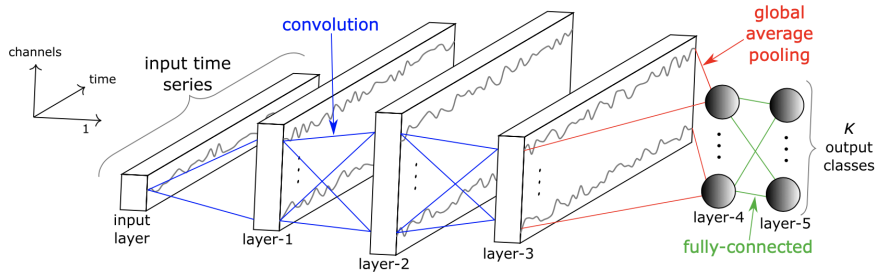

Figura 3.11: Architettura CNN per la TSC con tre strati convoluzionali. Fonte: [4].

3.8 La scelta della rete

Terminata la realizzazione degli strumenti utili a migliorare il risultato di addestramento di un modello, è necessario scegliere la rete neurale basata su *Deep Learning*, che possa realizzarlo. E' stata effettuata un'analisi dello stato dell'arte di reti già sviluppate che, seppur sia fondamentale modificarle per adattarle al caso d'uso, possano restituire un ottimo risultato. L'obiettivo, infatti, è quello di produrre un modello che, a partire dalla serie dei dati catturati dai sensori posti sulle barche, classifichi le varie tecniche di pesca.

3.8.1 Architettura: InceptionTime

La rete considerata allo scopo è la *InceptionTime*. E' un'architettura innovativa per la TSC. La composizione di un classificatore di rete Inception contiene due

diversi blocchi residui. Ogni blocco è composto da tre moduli Inception anziché dai tradizionali strati completamente convoluzionali. L'input di ogni blocco residuo viene trasferito tramite una connessione lineare per essere aggiunto all'input del blocco successivo, mitigando così il problema dell'annullamento del gradiente, consentendo un flusso diretto [23]. La Figura 3.12 illustra l'architettura di una rete Inception, che mostra 6 diversi moduli Inception impilati uno dopo l'altro.

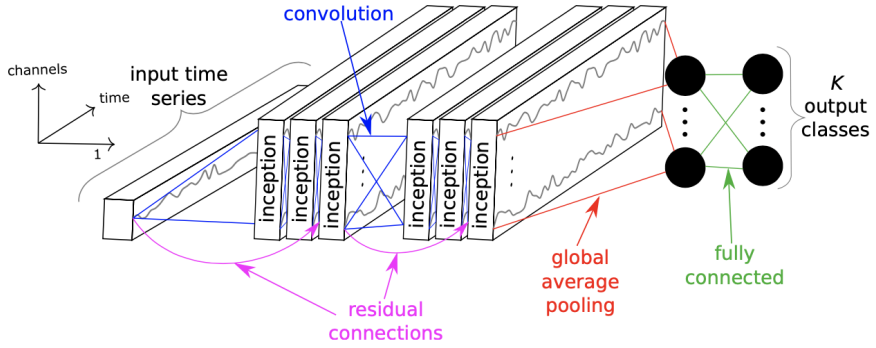

Figura 3.12: La rete Inception per la classificazione delle serie temporali. Fonte: [23].

Per quanto riguarda il modulo Inception, la Figura 3.13 illustra i dettagli interni di questa operazione. Consideriamo che l'ingresso sia un MTS con M dimensioni. Il primo componente principale del modulo Inception è chiamato strato "collo di bottiglia". Questo strato esegue un'operazione di scorrimento di m filtri di lunghezza 1 con uno stride pari a 1. Questo trasformerà la serie temporale da un MTS con M dimensioni a un MTS con $m << M$ dimensioni, riducendo così in modo significativo la dimensionalità della serie temporale e la complessità del modello e mitigando i problemi di overfitting per i piccoli insiemi di dati.

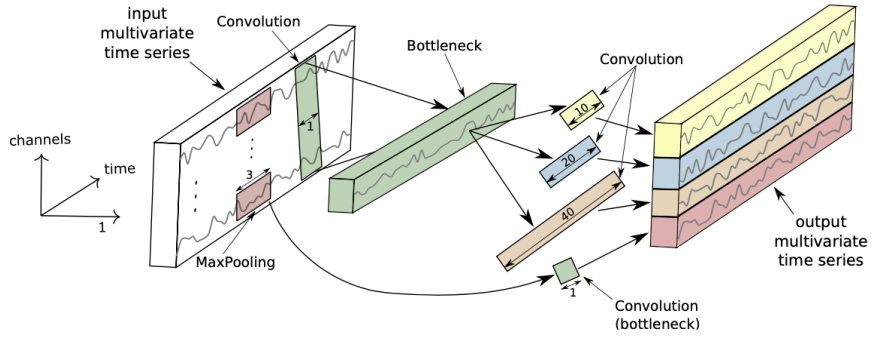

Figura 3.13: All'interno del modulo Inception per la classificazione delle serie temporali. Fonte: [23].

Si noti che, a scopo di visualizzazione, la Figura 3.13 illustra uno strato a collo di bottiglia con $m = 1$. E' opportuno ricordare che questo strato a collo di bottiglia è stato creato con un'unica dimensione. Il secondo componente principale del modulo Inception è lo scorrimento simultaneo di più filtri di lunghezza diversa sulla stessa serie temporale in ingresso. Ad esempio, nella Figura 3.13,

tre diverse convoluzioni di lunghezza ϵ $(10, 20, 40)$ sono applicate all'MTS di ingresso, che tecnicamente è l'uscita dello strato a collo di bottiglia. Inoltre, per introdurre la possibilità di avere un modello invariante a piccole perturbazioni, si introduce un'altra operazione di MaxPooling parallela, seguita da uno strato a collo di bottiglia per ridurre la dimensionalità. L'output della finestra di MaxPooling scorrevole viene calcolato prendendo il valore massimo in questa finestra di serie temporali. Infine, l'uscita di ogni convoluzione parallela indipendente/MaxPooling viene concatenata per formare l'MTS di uscita del modulo Inception corrente. Impilando più moduli Inception e addestrando i pesi (valori dei filtri) tramite backpropagation, la rete è in grado di estrarre caratteristiche gerarchiche latenti di più risoluzioni grazie all'uso di filtri di varia lunghezza. Per completezza, specifichiamo il numero esatto di filtri per il modulo Inception proposto: 3 insiemi di filtri ciascuno con 32 filtri di lunghezza $l \epsilon (10, 20, 40)$ con l'aggiunta di MaxPooling, rendendo così il numero totale di filtri per strato pari a $32 \times 4 = 128 = M$, la dimensionalità dell'MTS di uscita.

3.8.2 Un insieme di reti neurali per la TSC

Il modello *InceptionTime* è un insieme di 5 reti di Inception, a ogni previsione viene attribuito un peso equivalente. L'utilizzo di una singola rete Inception presenta un'elevata deviazione standard nell'accuratezza. Questa variabilità deriva sia dai pesi inizializzati casualmente sia dal processo di ottimizzazione stocastica stesso. La seguente equazione spiega l'ensembling delle previsioni fatte da una rete con diverse inizializzazioni:

$$y_{i,c} = \frac{1}{n} \sum_{j=1}^{n} \sigma_c(x_i, \theta_j) \ \forall \ c \ \epsilon \ [1, C]$$

con $y_{i,c}$ che indica la probabilità di uscita dell'ensemble di avere la serie temporale in ingresso x_i appartenente alla classe c, che è uguale all'uscita logica σ_c mediata sugli n modelli inizializzati in modo casuale.

3.8.3 Estratto del codice in *Python* dell'algoritmo

Nel file *.ipynb* del Jupyter Notebook una volta chiamata l'esecuzione del **main.py** tramite il comando

```
!python3 main.py InceptionTime
```

viene eseguito il codice del *main* che dopo aver preparato il dataset e le directory chiama la funzione *fit_ classifier()* il cui codice è il seguente:

```
1 def fit_classifier():
2     input_shape = x_train.shape[1:]
3
4     classifier = create_classifier(classifier_name, input_shape, nb_classes,
      output_directory)
5
6     classifier.fit(x_train, y_train, x_test, y_test, y_true)
```

Questa funzione crea il classificatore chiamando a sua volta la *create_ classifier()*:

```
def create_classifier(classifier_name, input_shape, nb_classes,
    output_directory, verbose=False, build=True):
    if classifier_name == 'inception':
        from classifiers import inception
        return inception.Classifier_INCEPTION(output_directory, input_shape,
        nb_classes, verbose, build=build)
```

Qui avviene l'inizializzazione del classificatore tramite la *__init__()* presente nel file **inception.py**:

```
class Classifier_INCEPTION:
    def __init__(self, output_directory, input_shape, nb_classes,
    verbose=True, build=True, batch_size=64, lr=0.001, nb_filters=32,
    use_residual=True, use_bottleneck=True, depth=6, kernel_size=41,
    nb_epochs=1500):

        self.output_directory = output_directory

        self.nb_filters = nb_filters
        self.use_residual = use_residual
        self.use_bottleneck = use_bottleneck
        self.depth = depth
        self.kernel_size = kernel_size - 1
        self.callbacks = None
        self.batch_size = batch_size
        self.bottleneck_size = 32
        self.nb_epochs = nb_epochs
        self.lr = lr
        self.verbose = verbose

        if build == True:
            self.model = self.build_model(input_shape, nb_classes)
            if (verbose == True):
                self.model.summary()
            self.verbose = verbose
            self.model.save_weights(self.output_directory + 'model_init.hdf5')
```

Successivamente viene costruito il modello tramite la libreria *Keras* utilizzato dalla *fit()*; dopo la verifica della presenza della GPU questa funzione restituisce tramite la *save_logs()* presente nel file **utils.py** i valori delle metriche trattate nel paragrafo 3.9.2. Il codice è il seguente:

```
def fit(self, x_train, y_train, x_val, y_val, y_true):
        if not tf.test.is_gpu_available:
            print('error no gpu')
            exit()
        # x_val and y_val are only used to monitor the test
        # loss and NOT for training

        if self.batch_size is None:
            mini_batch_size = int(min(x_train.shape[0] / 10, 16))
        else:
            mini_batch_size = self.batch_size

        start_time = time.time()

        hist = self.model.fit(x_train, y_train, batch_size=mini_batch_size,
        epochs=self.nb_epochs, verbose=self.verbose,
        validation_data=(x_val, y_val), callbacks=self.callbacks)

        duration = time.time() - start_time

        self.model.save(self.output_directory + 'last_model.hdf5')

        y_pred = self.predict(x_val, y_true, x_train, y_train, y_val,
        return_df_metrics=False)
```

```
25
26          # save predictions
27          np.save(self.output_directory + 'y_pred.npy', y_pred)
28
29          # convert the predicted from binary to integer
30          y_pred = np.argmax(y_pred, axis=1)
31
32          df_metrics = save_logs(self.output_directory, hist, y_pred, y_true,
33          duration)
34
35          keras.backend.clear_session()
36
37          return df_metrics
```

3.9 Alcuni concetti importanti

Ora vedremo due concetti fondamentali per l'analisi dei dati nell'ambito dell'*Intelligenza Artificiale*.

3.9.1 Tensore

Un tensore è un oggetto matematico definito in spazio vettoriale V e quindi non dipendente da un particolare sistema di riferimento [24]. I tensori sono la naturale generalizzazione del concetto di vettore ovvero di un oggetto che ha dimensioni e direzioni. Un array in informatica, indica una struttura dati complessa, statica e omogenea, ed un vettore è definibile come un array monodimensionale, ovvero una lista di numeri $V = [1\ 3\ 5\ 8]$. I vettori possono a loro volta essere suddivisi in elementi a cui si fa riferimento con gli indici. Dunque con V_i si indica l'elemento i-esimo. Una matrice invece è un array bidimensionale:

$$\begin{bmatrix} 1 & 2 & 9 \\ 3 & 8 & 9 \\ 1 & 2 & 2 \end{bmatrix}$$

In una matrice M_{ij} indica l'elemento della riga i-esima e della colonna j-esima. Così facendo, aggiungendo una dimensione, potremmo disegnare una matrice cubica. Per analogia si arriva così alla definizione di tensore.

Il tensore è un array multidimensionale, pertanto sia il vettore che la matrice sono esempi di tensori a dimensione rispettivamente 1 e 2. A questo punto possiamo introdurre due proprietà fondamentali dei tensori:

- **Rank** indica il numero di dimensioni del tensore, ad esempio una matrice cubica ha grado 3 e una matrice ha grado 2;
- **Shape** rappresenta la forma del tensore e viene espresso nel seguente formato $n \times m \times k$ nel caso di una matrice cubica. Ad esempio una matrice cubica può avere forma $1 \times 1 \times 1$ o $2 \times 5 \times 7$.

Alcuni esempi:

(a) Un tensore con $Rank = 2$ e $Shape = 2 \times 4$ è una matrice con 2 righe e 4 colonne

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

(b) Un tensore con $Rank = 2$ e $Shape = 4 \times 2$ è una matrice di 4 righe e 2 colonne

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}$$

(c) Un tensore di $Rank = 2$ e $Shape = 1 \times 4$ è un vettore $V = [1\ 3\ 5\ 8]$.

In *Python* per rappresentare un tensore si può usare la libreria *numpy*. Ad esempio se vogliamo rappresentare la matrice di $Rank = 2$ e $Shape = 2 \times 3$

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \end{bmatrix}$$

in *Python* si utilizzano le seguenti istruzioni:

```
>>> M = np.array([[1,2,3],[3,4,5]])

>>> np.shape(M)

(2, 3)

>>> np.rank(M)

2
```

3.9.2 Matrice di confusione

La matrice di confusione è uno strumento per analizzare gli errori compiuti da un modello di *Machine Learning*. E' utile per valutare la qualità delle previsioni del modello di classificazione. In particolar modo, la matrice mette in evidenza dove sbaglia il modello, in quali istanze risponde peggio e in quali meglio [25].

Nelle righe sono indicate le classi effettive. Sono le classi delle risposte corrette. Nelle colonne sono indicate le classi di previsione. Sono le classi delle risposte del modello. Nella Figura 3.14 è mostrata la struttura di una matrice di confusione.

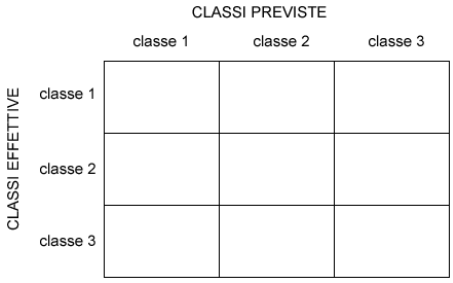

Figura 3.14: Esempio di una matrice di confusione. Fonte: [25].

All'interno delle caselle sono inserite le risposte del modello alle varie istanze. Nella diagonale si trovano le risposte corrette, nelle altre posizioni le errate. Gli errori e le risposte corrette però non sono tutte uguali. Possono verificarsi quattro

casi che esamineremo prendendo in considerazione 2 classi come mostrato nella Figura 3.15:

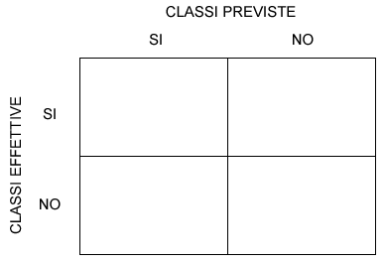

Figura 3.15: Matrice di confusione con 2 classi. Fonte: [25].

- **True Positive (TP)**: se la classe prevista è SI ed è uguale alla classe effettiva, si tratta di un caso di true positive. Il modello ha risposto correttamente SI;
- **True Negative (TN)**: se la classe prevista è NO ed è uguale alla classe effettiva, si tratta di un caso di true negative. Il modello ha risposto correttamente NO;
- **False Positive (FP)**: se la classe prevista è SI ma è diversa dalla classe effettiva, si tratta di un caso di false positive. Il modello ha sbagliato a rispondere SI;
- **False Negative (FN)**: se la classe prevista è NO ma è diversa dalla classe effettiva, si tratta di un caso di false negative. Il modello ha sbagliato a rispondere NO.

Dalla matrice di confusione si possono ottenere facilmente diverse metriche:

- **il tasso di errore** (error rate) misura la percentuale di errore delle previsioni sul totale delle istanze. Varia da 0 (migliore) a 1 (peggiore):

$$ERR = \frac{FP + FN}{TP + TN + FP + FN}$$

- **l'accuratezza** (accuracy) misura la percentuale delle previsioni esatte sul totale delle istanze. E' l'inverso del tasso di errore. Varia da 0 (peggiore) a 1 (migliore):

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} = 1 - ERR$$

- **la precisione** (precision) è la percentuale delle previsioni positive corrette (TP) sul totale delle previsioni positive del modello (giuste TP o sbagliate FP):

$$PR = \frac{TP}{TP + FP}$$

- **il richiamo o sensitività** (o recall o sensitivity) è la percentuale delle previsioni positive corrette (TP) sul totale delle istanze positive. Varia da 0 (peggiore) a 1 (migliore):

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

3.10 Elaborazione dei dati

Per l'utilizzo delle reti neurali per i dati temporali, viene fatto riferimento ad un'importante pubblicazione di Johannes Furnkranz [4], che ha anche un repository su *GitHub*, sul quale sono presenti una serie di classificatori, che è possibile scegliere e lanciare su dataset già pronti, come quello del caffè, della carne, dell'ECG, ecc. È strutturato in 3 parti fondamentali: *main.py*, *utils* e *classifiers*. Con il codice implementato e reso disponibile a chiunque, tutti possono utilizzare questa CNN con tutti i classificatori messi a disposizione.

- *Main.py* contiene il codice principale per poter eseguire un esperimento.
- *Utils* contiene le funzioni necessarie per leggere i set di dati e visualizzare i grafici.
- *Classifiers* contiene tutti i classificatori.

Nel repo sono presenti tutti i prerequisiti necessari per avviare e preparare l'ambiente di lavoro e, inoltre, sono presenti i risultati, su una serie di set di dati, riportati in forma tabellare. Innanzitutto viene testata la rete e il suo funzionamento sui dataset sui quali è già stato effettuato il training, per vedere come interagisce con i dati e se i risultati sono coerenti con quanto riportato in tabella. Come serie temporale della rete viene utilizzata *InceptionTime* [26]. Il funzionamento di questo classificatore è il seguente: lavora su un flusso di dati che rappresenta un solo parametro raccolto nel tempo e viene fatto elaborare dalla rete con un vettore aggiuntivo contenente il tipo di classe su cui fa riferimento quella collection. La rete si allena su una serie di dati di tipo univariato, questo perché la scelta del tipo di classe è determinata da un unico tipo di variabile. Tuttavia, questo modo di lavorare non funziona bene per il set di dati a disposizione, che raccoglie una serie di valori nel tempo su più variabili. È quindi necessario cambiare e trasformare l'approccio di tipo univariato in uno multivariato. Per modificare questo aspetto, si inserisce il codice del classificatore Inception.

Vengono apportate piccole modifiche alla struttura dei dati che vengono forniti alla rete, cambiando l'approccio da una singola variabile a più di una, per essere precisi a tre (SPEED, COURSE e TIME). Nella sezione *utils*, le variabili sono riportate su un singolo vettore per ogni tipo di classe. L'obiettivo è quello di modificare questo approccio cambiando la lettura del file di dati e generando un array di lunghezza $3 \times N$, dove 3 è il numero di variabili su cui si fa affidamento e N è il numero della lunghezza massima dell'array dovuta ai ping raccolti in mare durante la sessione di pesca.

Come si può intuire, i ping sono diversi per ogni sessione e anche per le stesse sessioni effettuate in momenti diversi della giornata o in porzioni di mare diverse. La rete Inception non vuole una serie di array di dimensioni diverse, ma tutto deve essere riconducibile allo stesso. Si utilizza quindi un approccio chiamato riempimento a zero, in cui tutte le matrici vengono portate alla stessa dimensione aggiungendo 0 in testa fino a quando la dimensione non corrisponde alla lunghezza massima effettiva standardizzata. Fatto questo, si alimenta la serie di array della stessa dimensione con l'array che rappresenta la tipologia alla rete e i risultati sono sorprendenti come analizzato nel prossimo capitolo.

Capitolo 4

Risultati

4.1 Dataset finale

Il dataset va bilanciato, infatti, si ha più dell'80% delle sessioni attribuibili solo a una sessione di pesca. Queste traggono in inganno il classificatore che mostra l'intero mondo come un'unica classe. La soluzione è quella di rimuovere le varie istanze organizzandole in un nuovo file così da passare da più di 34 mila righe a circa 3 mila in cui il dataset è bilanciato e robusto.

Le 1500 imbarcazioni note sono distribuite come segue:

- 900 per il train, suddivise nel seguente modo:
  - 180 LongLine LL = LLD + LLS;
  - 202 OTB;
  - 199 PS;
  - 200 PTM;
  - 199 TBB.
- 500 per il test, suddivise nel seguente modo:
  - 177 LongLine LL = LLD + LLS;
  - 261 OTB;
  - 21 PS;
  - 89 PTM;
  - 102 TBB.

La fase successiva è quella di organizzare le sessioni archiviate con dati RAW, organizzandole tutte in un unico file mostrato in Figura 4.1. Viene effettuato un join tra le due tabelle RAW e SESSION e si scartano i dati non rilevanti ottenendo una nuova tabella con i seguenti campi:

- **MMSI** che identifica il numero univoco di un peschereccio;
- **STARTDATA** che indica la data di partenza da un porto;

- **ENDDATA** che identifica la data di arrivo al porto e di fine pesca;
- **STARTID**;
- **ENDID**;
- **GEAR** cioè il tipo di pesca valutato da un team di esperti del settore;
- **CSVSESSION** che indica il nome del file a cui quella data si riferisce alla sessione;
- **SPEED** array con tutte le velocità di quella nave in una certa sessione;
- **COURSE** array con tutte le velocità angolari di una certa nave in una certa sessione;
- **TIME** array di tutti gli orari di ogni singolo ping;
- **GEOM** array di tutte le coordinate di tipo GIS.

mmsi	startdata	enddata	startid	endid	GEAR	csvSession	speed	course	time	geom
247051920	03/01/2017	04/01/2017	450	744	TBB	1	['4.1', '7.7'	['42.4', '19	['21:03:14	0102000000270100005 2B81E85EB9128404E6210
247051920	08/01/2017	10/01/2017	745	1306	TBB	1	['0', '0', '0',	['190.9', '0	['22:30:38	0102000000320200006F8104C58F91284087A757
247051920	11/01/2017	13/01/2017	1310	1754	TBB	1	['5.6', '8.8'	['357', '71.	['22:51:37	0102000000BD010000C442AD69DE912840143FC
247051920	20/01/2017	20/01/2017	1760	1960	TBB	1	['4.5', '6.8'	['38.5', '14	['06:07:49	0102000000C90000006DC5FEB27B9228406A4DF:
247051920	22/01/2017	24/01/2017	1965	2404	TBB	1	['4', '8.4', '	['322.3', '3	['22:50:01	0102000000B8010000FB3A70CE889228404DF38E
247051920	03/01/2017	04/01/2017	450	744	TBB	1	['4.1', '7.7'	['42.4', '19	['21:03:14	0102000000270100005 2B81E85EB9128404E6210
247051920	08/01/2017	10/01/2017	745	1306	TBB	1	['0', '0', '0',	['190.9', '0	['22:30:38	0102000000320200006F8104C58F91284087A757
247051920	11/01/2017	13/01/2017	1310	1754	TBB	1	['5.6', '8.8'	['357', '71.	['22:51:37	0102000000BD010000C442AD69DE912840143FC
247051920	20/01/2017	20/01/2017	1760	1960	TBB	1	['4.5', '6.8'	['38.5', '14	['06:07:49	0102000000C90000006DC5FEB27B9228406A4DF:
247051920	22/01/2017	24/01/2017	1965	2404	TBB	1	['4', '8.4', '	['322.3', '3	['22:50:01	0102000000B8010000FB3A70CE889228404DF38E
247051920	03/01/2017	04/01/2017	450	744	TBB	1	['4.1', '7.7'	['42.4', '19	['21:03:14	0102000000270100005 2B81E85EB9128404E6210
247051920	08/01/2017	10/01/2017	745	1306	TBB	1	['0', '0', '0',	['190.9', '0	['22:30:38	0102000000320200006F8104C58F91284087A757
247051920	11/01/2017	13/01/2017	1310	1754	TBB	1	['5.6', '8.8'	['357', '71.	['22:51:37	0102000000BD010000C442AD69DE912840143FC
247051920	20/01/2017	20/01/2017	1760	1960	LLD	5	['4.5', '6.8'	['38.5', '14	['06:07:49	0102000000C90000006DC5FEB27B9228406A4DF:
247051920	22/01/2017	24/01/2017	1965	2404	LLD	5	['4', '8.4', '	['322.3', '3	['22:50:01	0102000000B8010000FB3A70CE889228404DF38E
247051920	03/01/2017	04/01/2017	450	744	LLD	5	['4.1', '7.7'	['42.4', '19	['21:03:14	0102000000270100005 2B81E85EB9128404E6210
247051920	08/01/2017	10/01/2017	745	1306	LLD	5	['0', '0', '0',	['190.9', '0	['22:30:38	0102000000320200006F8104C58F91284087A757
247051920	11/01/2017	13/01/2017	1310	1754	LLD	5	['5.6', '8.8'	['357', '71.	['22:51:37	0102000000BD010000C442AD69DE912840143FC
247051920	20/01/2017	20/01/2017	1760	1960	LLD	5	['4.5', '6.8'	['38.5', '14	['06:07:49	0102000000C90000006DC5FEB27B9228406A4DF:
247051920	22/01/2017	24/01/2017	1965	2404	LLD	5	['4', '8.4', '	['322.3', '3	['22:50:01	0102000000B8010000FB3A70CE889228404DF38E
247051920	03/01/2017	04/01/2017	450	744	LLD	5	['4.1', '7.7'	['42.4', '19	['21:03:14	0102000000270100005 2B81E85EB9128404E6210
247051920	08/01/2017	10/01/2017	745	1306	LLD	5	['0', '0', '0',	['190.9', '0	['22:30:38	0102000000320200006F8104C58F91284087A757
247051920	11/01/2017	13/01/2017	1310	1754	LLD	5	['5.6', '8.8'	['357', '71.	['22:51:37	0102000000BD010000C442AD69DE912840143FC
247051920	20/01/2017	20/01/2017	1760	1960	LLD	5	['4.5', '6.8'	['38.5', '14	['06:07:49	0102000000C90000006DC5FEB27B9228406A4DF:
247051920	22/01/2017	24/01/2017	1965	2404	LLD	5	['4', '8.4', '	['322.3', '3	['22:50:01	0102000000B8010000FB3A70CE889228404DF38E

Figura 4.1: Struttura del Dataset.

Infine, alcune navi in alcune sessioni non rilasciano i loro ping esattamente ogni 5 minuti, a volte ci sono dei vuoti di ore. Questo può essere dovuto a un malfunzionamento del dispositivo di bordo o allo spegnimento intenzionale del dispositivo. Quindi, per non creare problemi al classificatore, vengono eliminate le sessioni il cui totale del numero dei ping non superi il 60% del totale atteso dall'inizio alla fine della sessione. Dopo questa modifica si ottiene il dataset ufficiale che viene utilizzato nell'analisi da parte del classificatore.

4.2 Risultati con l'InceptionTime

Vengono effettuati una serie di test sullo stesso dataset, considerando un numero di sessioni di pesca di circa 1500 (900 per l'addestramento e 600 per il test). In totale sono eseguiti 4 tipi di addestramento e successivi test: il primo basato sull'utilizzo di un dataset con 3 tipi di variabili (SPEED, COURSE e TIME), il secondo con l'eliminazione del tempo di pesca, il terzo con la sola variabile velocità (SPEED) e infine, l'ultimo, con la sola angolazione della barca (COURSE) durante la sessione.

Dopo l'esecuzione dei test con le varie combinazioni, dall'analisi delle metriche relative alle matrici di confusione si nota che la rete ha una forte dipendenza dalla variabile SPEED per decidere il tipo di pesca. Nella Figura 4.2 vengono mostrati i valori percentuali di precisione, accuratezza e richiamo per ogni tipo di dataset.

Type	Precision	Accuracy	Recall
1	92%	97%	98%
2	91%	97%	98%
3	94%	98%	98%
4	79%	87%	88%

Figura 4.2: Valori percentuali dei parametri per ogni tipo di dataset.

Si ricorda che il Tipo 1 è il dataset con tutte e tre le variabili presenti, il Tipo 2 è quello senza TIME, il Tipo 3 è quello con la sola SPEED e infine il Tipo 4 è quello con la sola COURSE. Si nota quanto detto in precedenza: la rete si basa maggiormente sulla velocità. Le altre tipologie hanno minore peso e la loro combinazione riduce le prestazioni. Il motivo per cui la velocità è una metrica su cui si può fare affidamento si evince dalla differenza tra due tipi di pesca: il palangaro e la pesca a strascico. Il palangaro può andare più lontano degli altri, portando la velocità iniziale molto più in alto e per molto tempo, mentre la pesca a strascico ha variazioni di velocità una volta arrivati sul luogo di pesca, dove avvengono frequenti accelerazioni per abbassare o riprendere gli ami da pesca.

Nella Figura 4.3 viene mostrata la matrice di confusione generata dopo l'elaborazione dei dati da parte di *InceptionTime*.

Class	PTM	OTB	PS	TBB	LL
PTM	86	0	3	0	0
OTB	1	260	0	0	0
PS	0	0	21	0	0
TBB	0	0	0	102	0
LL	1	1	10	1	164

Figura 4.3: Matrice di confusione generata con l'utilizzo di *InceptionTime*.

Capitolo 5

Conclusioni

L'uso delle reti neurali per le serie temporali ha sicuramente un ruolo di primo piano in questo lavoro. *InceptionTime* ha un ruolo fondamentale per la classificazione di enormi quantità di dati. È sicuramente una rete da utilizzare quando si parla di elaborare grandi quantità di dati accumulati nel tempo, come quelli dei sensori o di altri dispositivi di tipo IoT. Svolge un ruolo cruciale nella classificazione, con un'elevata precisione anche per pochi campioni. Tuttavia nella fase di training è necessaria l'analisi di molte sessioni di pesca e questo è un difetto da considerare se non si ha a disposizione una sufficiente quantità di dati. Il dataset comprende 37 mila sessioni di pesca, di cui solo 900 sono impiegate per l'addestramento. Nonostante ciò, la risposta del classificatore è estremamente buona, fornendo metriche a dir poco sorprendenti. Infatti, sull'intero set di dati, dopo l'addestramento e l'estrazione del modello, l'accuratezza è dell'88%, con una recall del 97%, a causa del forte squilibrio tra i vari tipi di pesca.

In conclusione, la rete *InceptionTime* si dimostra una rete neurale efficace per la classificazione dei tipi di pesca in mare con l'utilizzo di 3 variabili: velocità, rotta e tempo di pesca. Gli approcci per il futuro saranno sicuramente quelli di inserire ulteriori valori identificativi, per creare un modello più accurato, preciso e puntuale, su dataset più robusti e corposi.

Bibliografia

[1] Corte dei conti europea. Special report 20/2022: Eu action to combat illegal fishing. https://www.eca.europa.eu/lists/ecadocuments/sr22_20/sr_illegal_fishing_it.pdf, 2022.

[2] D'Agostino Lorenzo, Lanciotti Antonio, and Olivieri Davide. Classification and reconstruction of fishing trips through deep neural networks.

[3] Lines J and Bagnall A. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29:565–592, 2015.

[4] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 2019.

[5] Johann Faouzi. Time series classification: A review of algorithms and implementations. https://inria.hal.science/hal-03558165, 2022.

[6] Fazle Karim, Somshubra Majumdar, Houshang Darabi and Samuel Harford. Multivariate lstm-fcns for time series classification. https://www.sciencedirect.com/science/article/abs/pii/S0893608019301200, 2019.

[7] Wikipedia. Python. https://it.wikipedia.org/wiki/Python, 2023.

[8] The python logo. https://www.python.org/community/logos/.

[9] Riccardo Esposito. Linguaggi di programmazione: più usati, richiesti e ben pagati oggi. https://seoriented.it/linguaggi-programmazione-richiesti/, 2022.

[10] Wikipedia. Keras. https://it.wikipedia.org/wiki/Keras, 2021.

[11] Wikipedia. Geographic information system. https://it.wikipedia.org/wiki/Geographic_information_system, 2023.

[12] GitHub contributors. Qgis. https://github.com/qgis/QGIS, 2023.

[13] Wikipedia. Github. https://it.wikipedia.org/wiki/GitHub, 2023.

[14] Google. Colaboratory. https://colab.research.google.com/?hl=it, 2023.

[15] Project Jupyter. Jupyter. https://jupyter.org/, 2023.

[16] Rignanese Pietro, Polenta Andrea, and Bocci Francesco Pio. Classification of fishing activities from ais data. 2020.

[17] Wikipedia. Postgis. https://en.wikipedia.org/wiki/PostGIS, 2023.

[18] Wikipedia. Time series. https://en.wikipedia.org/wiki/Time_series, 2023.

[19] D. Stephen G. Pollock. The mathematics of time-series analysis. https://www.le.ac.uk/users/dsgp1/COURSES/THIRDMET/MYLECTURES/3XMETHODS.pdf.

[20] Andrea D'agostino. Rappresentazioni vettoriali per il machine learning. https://www.diariodiunanalista.it/posts/rappresentazioni-vettoriali-per-il-machine-learning/, 2023.

[21] Wikipedia. Apprendimento profondo. https://it.wikipedia.org/wiki/Apprendimento_profondo, 2023.

[22] Iot Worlds. Qual è il ruolo dell'apprendimento automatico nell'iot? https://www.iotworlds.com/it/what-is-the-role-of-machine-learning-in-iot/.

[23] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. 2019.

[24] Domenico Soriano. Tensori, matrici, vettori e python. https://www.domsoria.com/2019/10/tensori-matrici-vettori-e-python/#google_vignette, 2019.

[25] Andrea Minini. Matrice di confusione. https://www.andreaminini.com/ai/machine-learning/matrice-di-confusione, 2023.

[26] Hassan ISMAIL FAWAZ. Inceptiontime. https://github.com/hfawaz/InceptionTime, 2023.

Elenco delle figure

Elenco delle tabelle