UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Riconoscimento delle attività di pesca mediante la rete neurale inception time basata su dati AIS

Detection of fishing activities using inception time neural network based on AIS data

Relatore:
Dott. Mancini Adriano

Correlatore:
Ing. Galdelli Alessandro

Laureando:
Latiano Giovanni

Anno Accademico 2019-2020

Indice

Capitolo 1

Introduzione

1.1 Il progetto

Questa tesi descrive il lavoro svolto nei mesi di Maggio, Giugno e Luglio durante il tirocinio riguardate la classificazione delle navi in base alle tipologie di pesca. Si vuole infatti usare delle tecniche di *machine learning* al fine di classificare i dati del *Automatic Identification System* (AIS).

1.2 Motivazioni e Obiettivi

Il progetto di ricerca svolto in collaborazione con l'Istituto per le Risorse Biologiche e le Biotecnologie Marine(CNR IRBIM) di Ancona, consiste nel trattare dati AIS, ovvero i dati del sistema di identificazione automatica delle imbarcazioni, allo scopo di:

- comprendere il comportamento dei pescherecci e classificarli utilizzando tecniche di Machine Learning dai dati della posizione;
- mappare le attività di pesca correlate nel corso degli anni per valutare lo sforzo di pesca in una data regione o area marina;
- individuare eventuali attività illegali.

Il presente lavoro di tesi è strutturato come segue; verranno prima affrontati concetti elementari per poi introdurre più nozioni su tali elementi, ogni parte è propedeutica alla successiva. Viene quindi strutturata come segue:

- nella prima parte verranno introdotti gli strumenti software, il linguaggio di programmazione utilizzato, le tecnologie che sono state approfondite durante lo sviluppo del progetto e le tecniche di pesca che si sono prese in considerazione;
- nella seconda parte verrà approfondito l'approccio in *deep learning*;

- nella terza ed ultima parte verranno esposte le conclusioni e verranno introdotti gli sviluppi futuri;

Capitolo 2

Strumenti utilizzati

2.1 AIS Data

E' l'acronimo di *Automatic Identification System* ed in italiano significa letteralmente "Sistema di Identificazione Automatica". Inizialmente fu sviluppato come uno standard marittimo dalla *International Maritime Organization* (IMO), per aiutare le Autorità portuali ad evitare collisioni fra navi e più in generale a mantenere sotto controllo il traffico marittimo. Il sistema invia e riceve due tipi di dati, quelli statici inseriti dagli operatori e che sono lunghezza, larghezza, pescaggio e stazza oltre al numero assegnato all'imbarcazione chiamato *Marittime Mobile Service Identities* (MMSI) ed il nome dell'imbarcazione. A questi dati vengono sommati i dati dinamici quali la posizione dell'imbarcazione, la destinazione e la rotta. Tutte questi dati statici e dinamici sono scambiati elettronicamente tra stazioni di ricezione AIS (ovunque siano i ricevitori) e permettono alle navi di avere, oltre ai classici sistemi, un ulteriore ausilio alla navigazione. Nel nostro caso la serie di messaggi contenenti i dati (1 ogni 5 minuti circa), che nel chiameremo *ping*, andrà a generare la *time series*, la quale verrà poi studiata mediante le tecniche di *machine learning*. L'AIS è quindi utilizzato dalle navi principalmente per evitare collisioni. Al momento l'Automatic Identification System viene utilizzato anche nel campo della sicurezza in mare. Se ad esempio un membro dell'equipaggio dovesse cadere in mare ed indossasse un giubbetto provvisto di un transponder AIS la sua posizione verrebbe tracciata e ciò faciliterebbe enormemente il suo recupero, specie in condizioni meteo marine avverse o di notte. E' obbligatorio per le navi commerciali e per le navi passeggeri che compiono viaggi internazionali e con una stazza lorda che supera le 300 tonnellate. In figura 2.1 è rappresentato lo schema del sistema AIS.

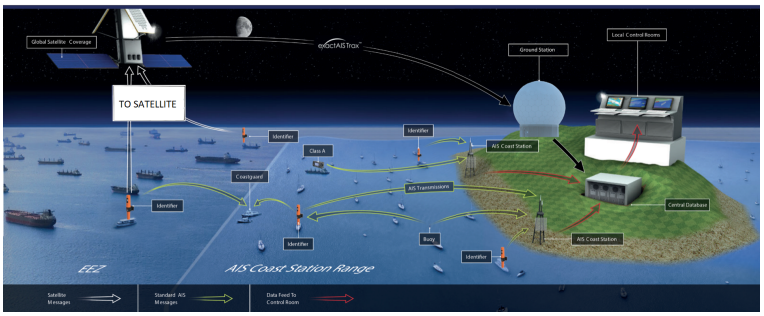

Figura 2.1: Schema AIS.[14]

2.2 Tecniche di pesca

Le tecniche di pesca prese in considerazione sono le seguenti:

- **GILL NETS** - è una tecnica che consente solo alla testa dei pesci di passare attraverso la rete. Quando il pesce cerca di scappare, la copertura delle branchie viene catturata, impigliandosi nella rete. Esistono due metodi di pesca con le gill nets. Uno è una rete fissa che viene ancorata in acqua alla profondità desiderata per evitare qualsiasi movimento della rete. L'altra è una rete galleggiante che viene tenuta a galla con boe, legata alla barca e lasciata alla deriva con le acque correnti;

- **PURSE SEINE** - consiste in una grande rete trainata, solitamente da due barche, che racchiude un banco di pesci e viene poi chiusa in fondo per mezzo di una lenza che ricorda lo spago anticamente usato per chiudere il collo di una borsa o di un sacchetto di denaro;

- **PELAGIC TRAWLS** - sono reti a forma di imbuto che vengono trainate da una o due navi. I pesci vengono raccolti e catturati nell'estremità finale della rete. La cattura di altre specie può essere un problema in alcune zone;

- **BOTTOM TRAWLS** - funzionano in modo simile alle pelagic trawls, ma vengono trascinate lungo i fondali marini. Sono una delle tecniche più utilizzate nella pesca d'altura. Le reti possono danneggiare gli habitat sottomarini come ad esempio le barriere coralline;

- **BEAM TRAWLS** - sono reti a strascico a sacco montate su una pesante trave metallica e trainate lungo il fondale marino. Questa tecnica sconvolge la fauna che vive sul fondale marino;

- **LONG LINES** - consistono in una lunga linea principale, fino a 100 km di lunghezza, con un gran numero di linee brevi (chiamati snodi) che

trasportano migliaia di ami con esca. La cattura involontaria di alcuno specie è un lato negativo di questa tipologia di pesca infatti delfini, squali, tartarughe e uccelli marini spesso rimangono intrappolati sui ganci;

In figura 2.2 sono rappresentate le tecniche di pesca appena descritte.

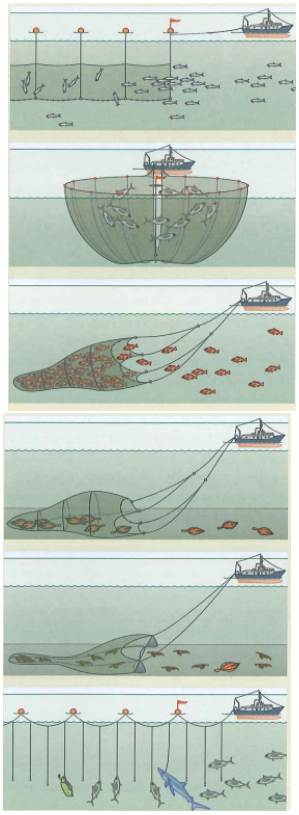

Figura 2.2: In ordine Gill Nets, Purese Seinse, Pelagic Trawls, Bottom Trawls, Beam Trawls, Long Lines. [8]

2.3 InceptionTime Net e TSC

Prima di introdurre l'algoritmo usato è bene descrivere il problema che andremo ad affrontare, ovvero la *time series classification* (TSC), o classificazione delle serie temporali.

2.3.1 Classificazione delle serie temporali

Seguono alcune definizioni utili per una migliore comprensione del problema.

- **Definizione 1** una serie temporale a variabile singola *(monovariable time series)* $X = [x_1, x_2, ..., x_T]$ è un set ordinato di valori reali. La lunghezza di X corrisponde al numero di valori reali T.
- **Definizione 2** una M-dimensionale MTS *(multivariable time series)* $X = [X^1, X^2, ..., X^M]$ consiste in M differenti serie temporali a variabile singola con $X^i \in \mathbb{R}^T$.
- **Definizione 3** un dataset $D = [(X_1, Y_1), (X_2, Y_2), ..., (X_N, Y_N)]$ è una collezione di coppie (X_i, Y_i) dove X_i può essere la serie temporale singola/multivariabile con Y_i il vettore che indica la classe a cui appartiene. Per un dataset contenente K classi, il vettore Y_i ha lunghezza K dove ogni elemento $j \in [1, K]$ è uguale a 1 se la classe di X_i è j altrimenti è 0.

Il compito della *Time Series Classification* (TSC) è quello di allenare un classificatore con il dataset D per mappare lo spazio dei possibili input allo scopo di riuscire a predire, sulla base statistica, gli output futuri.

2.3.2 Deep learning per la classificazione delle serie temporali

Per risolvere il problema della TSC ci vengono in aiuto le *Deep Neural Networks* (DNN). Una DNN è una composizione di L funzioni parametriche, visualizzate come *layers* (strati), ed ogni layer è considerato come una rappresentazione del dominio di input. Un layer l_i, con $i \in 1, ..., L$, contiene dei "neuroni" ovvero delle piccole unità che computano le informazioni in ingresso. Il layer l_i, prende come input l'output del layer l_{i-1} ed applica una *trasformazione non lineare* per computare l'informazione ed inviarla a sua volta al layer successivo. Il comportamento di queste trasformazioni non lineari di ogni layer è regolato da un set di parametri θ_i, chiamati *pesi*. Questi pesi quindi collegano l'input del layer precedente con l'output del layer attuale. Una volta dato un input x, la rete neurale svolge le seguenti computazioni al fine di 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 trasformazione non lineare applicata al layer l_i.

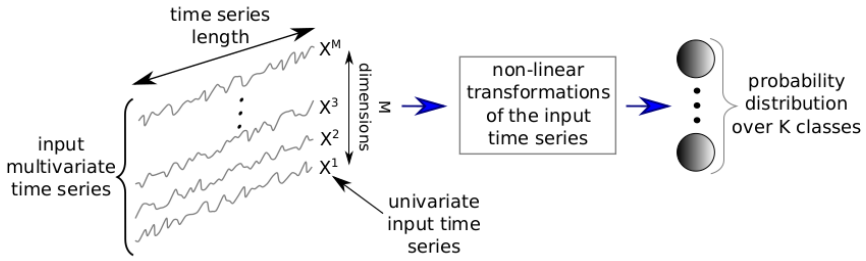

Figura 2.3: Un esempio esplicativo del problema che andremo a trattare.[12]

2.3.3 InceptionTime

La rete neurale usata in questo progetto è la *InceptionTime*, una rete recentemente sviluppata da Hassan Ismail Fawaz ed il suo team [11].
L'InceptionTime consiste in un insieme di 5 differenti *Inception network* inizializzate casualmente. Ogni Inception network contiene due *residual-blocks*. I residual-blocks a loro volta sono formati da tre moduli *Inception*. L'ingresso di ogni residual-block è trasferito tramine una connessione lineare all'input del prossimo residual-block.

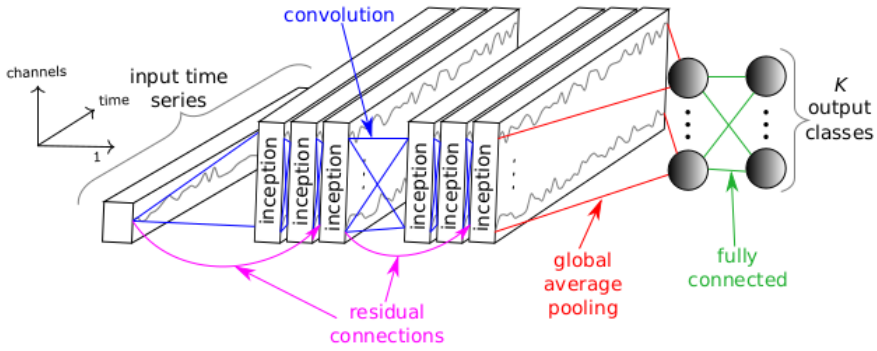

Figura 2.4: Visualizzazione grafica dell'Inception Network.[11]

Consideriamo come input una MTS con dimensione M. Il primo componente del modulo Inception è chiamato *bottleneck* layer. Questo layer effettua un'operazione che trasformerà una MTS di dimensione M ad una MTS con dimensione $m << M$. Ciò permette di ridurre le dimensioni della time series ed inoltre contrasta il problema dell'overfitting con i dataset piccoli. Il secondo componente del modulo Inception è un insieme di filtri di lunghezza

non uniforme applicati agli stessi input della serie temporale. In aggiunta, per dare robustezza contro le piccole perturbazioni, è introdotta un'operazione parallela di *MaxPooling* seguita da un bottleneck layer per ridurne la dimensione. Infine l'output di ogni componente è concatenato per formare l'output MTS del corrente modulo Inception.

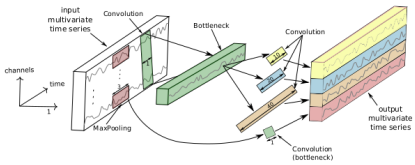

Figura 2.5: Visualizzazione grafica del modulo Inception.[11]

2.3.4 Scelta della InceptionTime

La scelta è ricaduta sulla InceptionTime perchè come è stato dimostrato da Ismail Fawaz ed il suo team [11] la rete ha raggiunto una percentuale di **accuracy** che compete con l'algoritmo leader nel settore, HIVE-COTE[13]. HIVE-COTE è un insieme di 37 algoritmi di TSC con uno schema di volto gerarchico. Nel loro esperimento, per marcare la differenza tra i due algoritmi, sono stati testati sull'archivio UCR[5], composto da 85 datasets. I risultati in figura 2.6 mostrano un Win/Tie/Loss di 40/6/39 in favore dell'Inception Time, ovvero su 40 datasets l'InceptionTime ha avuto una accuracy migliore, a differenza di HIVE-COTE il quale ha performato meglio su 39 datasets. Nei restanti 6 dataset InceptionTime e HIVE-COTE hanno avuto la stessa accuracy.

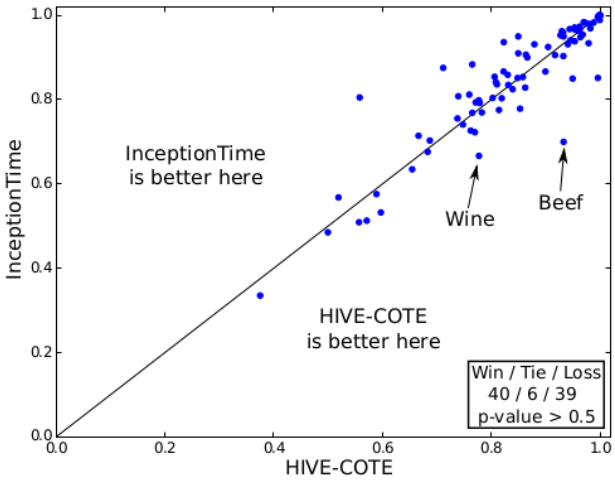

Figura 2.6: Confronto tra HIVE-COTE e InceptionTime sull'archivio UCR.[11]

Possiamo chiaramente vedere in figura 2.7 che la complessità di InceptionTime aumenta quasi linearmente con un aumento della lunghezza della serie temporale, a differenza di HIVE-COTE, la cui esecuzione è quasi due ordini di grandezza più lenta. Risulta evidente quindi come l'InceptionTime

sia più veloce con serie temporali molto lunghe.

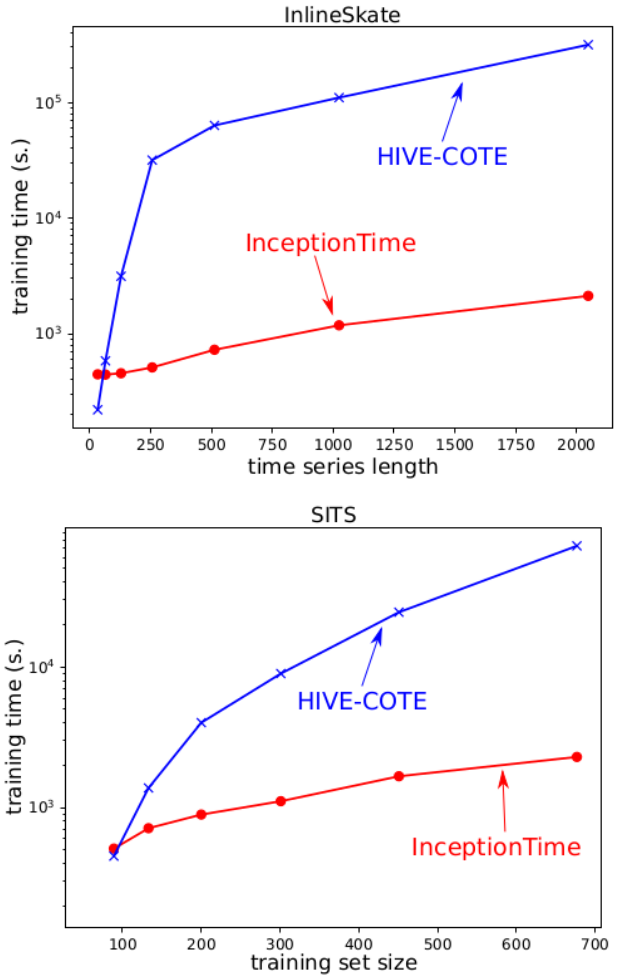

Figura 2.7: Nei due grafici il confronto tra HIVE-COTE e InceptionTime su due archivi appartenenti all'UCR dataset.[11]

Capitolo 3

L'approccio al problema

Il lavoro svolto si divide in due parti. Nella prima abbiamo lavorato con dati AIS per poter generare la serie temporali delle sessioni di pesca, nella seconda abbiamo adattato la rete tramite una modifica per poterle dare in input le serie temporali precedentemente generate ed effettuato i vari test.

3.1 Lavoro con dati AIS

3.1.1 Filtraggio delle sessioni

Il primo lavoro è stato quello di effettuare un **filtraggio** sulle sessioni di pesca. Lo scopo del filtraggio è quello di scartate le sessioni di pesca in cui il sistema AIS è stato lasciato spento per più del 70% del tempo totale delle sessione, questo perché altrimenti la perdita di informazione sarebbe talmente elevata da influire negativamente in fase di addestramento delle rete. Per fare ciò ci siamo serviti dei file *dataSession.csv* contenenti l'elenco di tutte le sessioni di pesca del 2017 suddivisi per mesi. Ogni riga contiene una sessione di pesca, mentre ogni colonna indica un parametro della sessione di pesca(*MMSI, sessione, id inizio/fine, tempo inizio/fine, data inizio/fine, porto partenza/arrivo, paese partenza/arrivo, ...*). Usando python, una sua libreria *Pandas*[1] e facendo un semplice calcolo matematico abbiamo creato uno script che scarta le sessioni di pesca in cui il seguente rapporto è minore di 0.7.

$$\frac{numero\,ping\,emessi}{numero\,ping\,regolari}$$

Dove il numero di ping emessi viene calcolato tramite la differenza $id\,fine - id\,inizio$ mentre il numero ping regolari viene calcolato tramite la differenza $tempo\,fine - tempo\,inizio$ diviso 5, così facendo, considerando che il sistema AIS emette un ping ogni 5 minuti circa, si trova il numero totale di ping

[1] `https://pandas.pydata.org/`

in una sessione in cui il sistema AIS è stato lasciato acceso per il 100% del tempo totale.

```
for i in range(df.shape[0]):
    pings_number.append(df['endid'][i] - df['startid'][i])
    yy1, mon1, dd1, hh1, mm1, ss1 = recover_info(df['starttime'
        ][i], df['startdata'][i])
    yy2, mon2, dd2, hh2, mm2, ss2 = recover_info(df['endtime'][
        i], df['enddata'][i])
    minutes.append(my_time(yy2, mon2, dd2, hh2, mm2, ss2) -
        my_time(yy1, mon1, dd1, hh1, mm1, ss1))
    if (0.7 <= (pings_number[i] / (minutes[i] / 5))): lista.
        append(df.loc[i])
```

Listing 3.1: Una parte del codice sorgente di filtraggio.

Nella porzione di codice in 3.1 si può vedere che se il rapporto tra i ping dentro l'istruzione di `if` è maggiore di 0.7 allora la sessione viene salvata , altrimenti viene scartata. Le funzioni *recover_info()* e *my_time()* servono rispettivamente a recuperare le info di data/ora della sessione e a calcolare il tempo in minuti della sessione.

3.1.2 Bilanciamento delle sessioni

Il secondo lavoro con i dati è stato quello di effettuare un **bilanciamento** delle sessioni di pesca. In questo caso ci siamo serviti di un altro file *session_per_GEARS.csv* il quale in ogni riga contiene le stesse sessioni di pesca dei file *dataSession.csv*, mentre le colonne contengono i parametri principali(*mmsi, sessione, data inizio/fine*) in aggiunta però è presente la classe della tipologia di pesca(*GEAR*). Dopo aver reso coerenti i due file, rimuovendo anche dal file *session_per_GEARS.csv* le sessioni precedentemente scartate tramite il filtraggio, abbiamo provveduto al bilanciamento vero è proprio. Ciò è stato necessario in quanto il numero di sessioni per classe di pesca non è omogeneo, quindi in fase di addestramento la rete potrebbe specializzarsi nel riconoscere una classe piuttosto che un'altra. Abbiamo provveduto quindi, sempre usando python e la sua libreria *pandas*, a creare un piccolo script che per ogni classe scelga in modo casuale tra le sessioni, un numero preimpostato di sessioni per tipologia di pesca(nel nostro caso 340). Le classi di pesca che presentato un numero di sessioni minore di 340 vengono ovviamente prese tutte. Al contrario nelle classi che superano tale soglia vengono prese 340 sessioni scelte in modo casuale.

```
for j in range(len(tipologia)):
    k = 0
    while k < 340 and k < num_tipologia[j]:
        indici_mischiati_bilanciati[j].append(
            indici_tipologia[j][k])
        k += 1
```

Listing 3.2: Una parte del codice sorgente di bilanciamento.

Nel segmento di codice in 3.2 si può vedere come dopo aver preso gli indici di ogni sessione relativi al dataframe, averli suddivisi per classi e poi mischiati, essi vengono prelevati fin tanto che non raggiungono il numero di 340 sessioni o non finiscono.

3.1.3 Generazione delle serie temporali

Dopo aver effettuato il filtraggio ed il bilanciamento è ora possibile ricavare le serie temporali. Per fare ciò questa volta abbiamo usato i file *dataAis.csv*. In questo caso ogni singola riga indica il singolo ping emesso dall'imbarcazione, mentre le colonne contengono, come detto precedentemente nel paragrafo 2.1, i dati statici e dinamici dell'imbarcazione *MMSI, nome imbarcazione, velocità, posizione imbarcazione, rotta, destinazione* e così via. Per generare la serie temporale si prende dunque la serie di ping relativa ad una sessione. Nel nostro caso prendiamo i tutti i ping relativi alle sessioni rimaste dopo aver effettuato il filtraggio ed il bilanciamento. Più avanti affronteremo nel dettaglio questo argomento.

3.2 Lavoro con la rete

3.2.1 Far partire l'InceptionTime

Il primo lavoro con l'InceptionTime è stato quello di farla partire e replicare i risultati ottenuti in [11]. Per fare ciò ci siamo serviti di Google Colab, una piattaforma utile ad eseguire codice sul Cloud. Dopo aver impostato l'UCR Archives come input e impostato le varie directory nel codice sorgente della rete[10], per far partire la rete basta digitare nella cella di codice di Colab il seguente comando:

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

In un primo momento, l'esecuzione del comando veniva interrotta da due errori. Il primo era:

```
Traceback (most recent call last):
  File "main.py", line 117, in <module>
    fit_classifier()
  File "main.py", line 49, in fit_classifier
    classifier.fit(x_train, y_train, x_test, y_test, y_true)
  File "/content/cloned-repo/classifiers/inception.py", line
      110, in fit
    if len(keras.backend.tensorflow
        _backend._get_available_gpus()) == 0:
  File "/usr/local/lib/python3.6/dist-packages/keras/
    backend/tensorflow_backend.py", line 506, in
        _get_available_gpus
    _LOCAL_DEVICES = tf.config.experimental_list_devices()
AttributeError: module 'tensorflow._api.v2.config' has no
   attribute 'experimental_list_devices'
```

Il quale è stato possibile risolverlo modificando il file nella directory di Colab `/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py` alla riga 506 sostituendo

```
_LOCAL_DEVICES = tf.config.experimental_list_devices()
```

con

```
devices = tf.config.list_logical_devices()
_LOCAL_DEVICES = [x.name for x in devices]
```

Il secondo errore era:

```
Traceback (most recent call last):
File "main.py", line 117, in
fit_classifier()
File "main.py", line 49, in fit_classifier
classifier.fit(x_train, y_train, x_test, y_test, y_true)
File "content/InceptionTime/classifiers/inception.py", line
    145, in fit
plot_test_acc=plot_test_acc)
File "content/InceptionTime/utils/utils.py", line 211, in
    save_logs
df_best_model['best_model_train_acc'] = row_best_model['acc']
File "/usr/lib/python3/dist-packages/pandas/core/series.py",
    line 623, in getitem
result = self.index.get_value(self, key)
File "/usr/lib/python3/dist-packages/pandas/core/indexes/base.
    py", line 2574, in get_value
raise e1
File "/usr/lib/python3/dist-packages/pandas/core/indexes/base.
    py", line 2560, in get_value
tz=getattr(series.dtype, 'tz', None))
File "pandas/_libs/index.pyx", line 83, in pandas._libs.index.
    IndexEngine.get_value
File "pandas/_libs/index.pyx", line 91, in pandas._libs.index.
    IndexEngine.get_value
File "pandas/_libs/index.pyx", line 139, in pandas._libs.index.
    IndexEngine.get_loc
File "pandas/_libs/hashtable_class_helper.pxi", line 1265, in
    pandas._libs.hashtable.PyObjectHashTable.get_item
File "pandas/_libs/hashtable_class_helper.pxi", line 1273, in
    pandas._libs.hashtable.PyObjectHashTable.get_item
KeyError: 'acc'
```

Il quale è stato possibile risolverlo aggiungendo alla riga 209 del file `utils.py` della rete il seguente frammento di codice:

```
'if 'acc' in row_best_model.index:
    df_best_model['best_model_train_acc'] = row_best_model['acc
        ']
else:
    df_best_model['best_model_train_acc'] = 0
```

Dopo aver risolto i due errori l'esecuzione è stata eseguita correttamente ed i risultati usando l'archivio URC sono stati replicati correttamente.

3.2.2 Adattare la rete ad una *multivariable time series*

Il secondo lavoro con l'InceptionTime è stato quello di adattare la rete ad una serie temporale multivariabile (MTS). Infatti nel primo test con la rete si è usato l'archivio URC, il quale contiene 85 datasets monovariabili. La rete è già predisposta ad accettare in input delle MTS, è stato necessario però apportare delle piccole modifiche affinché la MTS venga presa in input correttamente. Nel file `utils.py` appartenente alla rete, abbiamo quindi riscritto da zero la funzione **readucr** affinché in seguito la rete sia in grado di capire e prendere in ingresso automaticamente serie temporali mono/multivariabili senza dover specificare ogni volta il numero di variabili della serie temporale.

```
import pandas as pd
import numpy as np

def readucr(file_name):
          df = pd.read_csv(file_name, header=None)
          data = np.array(df)
          index = 0
          n_variable = 1

          for i in range(data.shape[1]):
               if ':' in str(data[0, i]):
                    if n_variable == 1:
                         index = i
                    n_variable += 1
               elif n_variable == 1 and i+1 == data.shape[1]:
                    index = data.shape[1]-2

          lista0, big_l = [], []

          for j in range(data.shape[0]):
               for i in range(data.shape[1]-1):
                    if ':' in str(data[j,i]):
                         d1, d2 = str(data[j,i]).split(':')
                         lista0.append(float(d1))
                         lista0.append(float(d2))
                    else:
                         lista0.append(float(data[j,i]))
                    if i == data.shape[1]-2 :
                         big_l.append(lista0)
                         lista0 = []

          print(n_variable)
          big = np.array(big_l)
          lista1, lista2, lista3 = [],[],[]

          for j in range(df.shape[0]):
               for i in range(index+1):
                    lista1.append(big[j,i])
                    if n_variable > 1:
```

```
                              lista1.append(big[j,i+index+1])
                        if n_variable > 2:
                              lista1.append(big[j,i+2*(index+1)])
                        if n_variable > 3:
                              lista1.append(big[j,i+3*(index+1)])
                        if n_variable > 4:
                              lista1.append(big[j,i+4*(index+1)])
                        lista2.append(lista1)
                        lista1 = []
                        if i == index:
                              lista3.append(lista2)
                              lista2 = []

            X = np.array(lista3)
            Y = data[:, -1]
            return X, Y
```

Listing 3.3: L'adattamento effettuato.

Nel primo ciclo `for` viene stabilito il numero delle variabili della serie temporale. Il numero delle variabili viene incrementato ogni qual volta è presente il carattere `":"` nella stringa contenente la serie temporale. Questo perché nelle serie temporali che andremo ad utilizzare, i valori di ogni variabile sono delimitati da una `","`, mentre le variabili sono delimitate proprio dal carattere`":"`, la classe(*label*) di appartenenza viene invece ottenuta leggendo l'ultimo valore presente nella riga(nel nostro caso sarà un numero).

$$
\begin{gathered}
a_{11}, a_{12}, .., a_{1n} : b_{11}, b_{12}, .., b_{1n}, l_1 \\
a_{21}, a_{22}, ..., a_{2n} : b_{21}, b_{22}, ..., b_{2n}, l_2 \\
. \\
. \\
a_{m1}, a_{m2}, ..., a_{mn} : b_{m1}, b_{m2}, ..., b_{mn}, l_m
\end{gathered}
$$

Figura 3.1: Lo schema del dataset che andremo ad usare, con n lunghezza della serie temporale ed m il numero di sessioni. In questo esempio stiamo usando due variabili (A,B).

Nel secondo ciclo `for` vengono acquisiti i valori di tutta la serie temporale è vengono inseriti in un'unica matrice. Viene però salvato l'indice della posizione in cui finisce la prima variabile ed inizia la seconda variabile.

$$
\begin{bmatrix}
a_{11}, a_{12}, .., a_{1n}, b_{11}, b_{12}, .., b_{1n} \\
a_{21}, a_{22}, .., a_{2n}, b_{21}, b_{22}, .., b_{2n} \\
. \\
. \\
a_{m1}, a_{m2}, .., a_{mn}, b_{m1}, b_{m2}, .., b_{mn}
\end{bmatrix}
$$

Nel terzo ciclo `for` viene finalmente generata la matrice che prenderà in ingresso la rete. La matrice è formata da coppie(a seconda del numero di variabili) di valori delle variabili della serie temporale.

$$\begin{bmatrix} (a_{11}, b_{11}), (a_{12}, b_{12}), .., (a_{1n}, b_{1n}) \\ (a_{21}, b_{21}), (a_{22}, b_{22}), .., (a_{2n}, b_{2n}) \\ . \\ . \\ (a_{m1}, b_{m1}), (a_{m2}, b_{m2}), ..., (a_{mn}, b_{mn}) \end{bmatrix}$$

Si può vedere nelle ultime tre righe del codice che la matrice X di sarà formato dai valori delle TS, il vettore Y sarà formato dalle classi di appartenenza.
Dopo aver effettuato la modifica, abbiamo testato la rete con il dataset AtrialFibrillation, un dataset di registrazioni ECG a due canali (quindi due variabili) creato dai dati utilizzati nella Computers in Cardiology Challenge 2004[9], una competizione aperta con l'obiettivo di sviluppare metodi automatizzati per prevedere l'interruzione spontanea della fibrillazione atriale. Il risultato di **accuracy** che ci si aspettava era del 33% circa, il quale è stato confermato dalla rete.

3.2.3 Generare il dataset

Dopo aver testato la rete con un dataset monovariabile e averla adattata ad un ingresso multivariabile, è arrivato il momento di generare il dataset delle sessioni e testarlo. Come detto in precedenza, ci siamo serviti dei file *dataAIS.csv* i quali in ogni riga contengono i ping singoli emessi dal sistema AIS. Nel nostro lavoro ci si è occupati di addestrare la rete principalmente con due parametri del dato AIS, **speed**(velocità) e **course**(rotta). Abbiamo quindi prima prelevato i ping dai file *dataAIS.csv* delle sessioni filtrate e bilanciate salvandoli in dei *numpy array*, per poi generare le serie temporali delle sessioni di pesca. La scelta di questa suddivisione in due è stata dettata dal fatto che il numero di ping contenuti nei file dataAIS.csv è molto elevato, circa 13milioni, quindi nella fase di reperimento dei ping relativi alle sessioni filtrate e bilanciate il costo in termini di risorse fisiche(RAM,CPU) e temporali è stato molto elevato. Per fare ciò abbiamo creato uno script in Python, usando le librerie *Pandas* e *Numpy*[2].

```
DATA_MISS = True
if DATA_MISS:
    df0 = pd.read_csv('session_temp.csv')
    df1 = pd.read_csv('/home/giovanni/Documenti
        /Tirocino_AIS/DATASETS/Raw/dataAIS1.csv')
    df2 = pd.read_csv('/home/giovanni/Documenti
        /Tirocino_AIS/DATASETS/Raw/dataAIS2.csv')
    df3 = pd.read_csv('/home/giovanni/Documenti
        /Tirocino_AIS/DATASETS/Raw/dataAIS3.csv')
    df4 = pd.read_csv('/home/giovanni/Documenti
        /Tirocino_AIS/DATASETS/Raw/dataAIS4.csv')
    df5 = pd.read_csv('/home/giovanni/Documenti
```

[2] https://numpy.org/

```
13          /Tirocino_AIS/DATASETS/Raw/dataAIS5.csv')
14      df6 = pd.read_csv('/home/giovanni/Documenti
15          /Tirocino_AIS/DATASETS/Raw/dataAIS6.csv')
16      df7 = pd.read_csv('/home/giovanni/Documenti
17          /Tirocino_AIS/DATASETS/Raw/dataAIS7.csv')
18      df8 = pd.read_csv('/home/giovanni/Documenti
19          /Tirocino_AIS/DATASETS/Raw/dataAIS8.csv')
20      df9 = pd.read_csv('/home/giovanni/Documenti
21          /Tirocino_AIS/DATASETS/Raw/dataAIS9.csv')
22      df10 = pd.read_csv('/home/giovanni/Documenti
23          /Tirocino_AIS/DATASETS/Raw/dataAIS10.csv')
24      df11 = pd.read_csv('/home/giovanni/Documenti
25          /Tirocino_AIS/DATASETS/Raw/dataAIS11.csv')
26      df12 = pd.read_csv('/home/giovanni/Documenti
27          /Tirocino_AIS/DATASETS/Raw/dataAIS12.csv')
28      frames = [df1, df2, df3, df4, df5, df6, df7, df8, df9, df10
            , df11, df12]
29      df = pd.concat(frames, ignore_index=True)
30
31      indice, max_lunghezza_ts, array_lunghezza, = 0, 0, []
32      GEARS = {'LLD':1, 'LLS':1, 'OTB':2, 'PS':3, 'PTM':4, 'TBB'
            :5  }
33      speed, course, acceleration, GEARS_, time= [], [], [], [],
            []
34      print('Lunghezza Dataframe: ' + str(df.shape[0]) + ' righe!
            ')
35
36      for i in range(df0.shape[0]):
37          if df0['GEAR'].loc[i] != 'LLS_gtr' and df0['GEAR'].loc[
                i] != 'LLS_prob':
38              array_lunghezza.append(df0['endid'].loc[i] - df0['
                    startid'].loc[i])
39              max_lunghezza_ts = maggiore(array_lunghezza[indice
                    ], max_lunghezza_ts)
40              sp, co, ac , ti = [], [], [0], []
41              q, old_time, old_speed = 0, 0, 0
42              j = df[(df0['startid'].loc[i] == df['id']) & (df0['
                    mmsi'].loc[i] == df['mmsi'])
43              & (df0['starttime'].loc[i] == df['time'])].index[0]
44              for l in range(array_lunghezza[indice]):
45                  act_id = df['id'].loc[j+q]
46                  sp.append(float(df['speed'].loc[j + q]))
47                  co.append(float(df['course'].loc[j + q]))
48                  ti.append(my_time(*recover_info(df['time'].loc[
                        j + q],
49                      df['date'].loc[j + q])))
50                  if  l > 0 :
51                      ac.append(((float(df['speed'].loc[j+q]) -
                            float(sp[l-1]))*1.852)/
52                      ((my_time(*recover_info(df['time'].loc[j+q]
53                      ,df['date'].loc[j+q])) - ti[l-1])*0.017))
54                  if act_id + 1 != df['id'].loc[j+q+1] and df['
                        mmsi'].loc[j+q] !=  df['mmsi'].loc[j+q+1]
                        and
```

```
                l != array_lunghezza[indice]-1 :
                    j = df[(act_id + 1 == df['id']) & (df0['
                        mmsi'].loc[i] == df['mmsi'])].index[0]
                    q = 0
                else : q += 1
            speed.append(sp)
            course.append(co)
            acceleration.append(ac)
            GEARS_.append(GEARS[df0['GEAR'].loc[i]])
            time.append(ti)
            indice += 1
            print('Sessione n.' + str(i) + ' acquisita!')
        else : continue
    #saving data
    Speed, Course, Acceleration, _GEARS_, Time,
        Max_lunghezza_ts = np.array(speed), np.array(course),
        np.array(acceleration), np.array(GEARS_), np.array(time
        ), np.array(max_lunghezza_ts)
    to_save = True
    if to_save:
        np.save('Array Numpy Salvati/numpy_speed.npy', np.array
            (speed))
        np.save('Array Numpy Salvati/numpy_course.npy', np.
            array(course))
        np.save('Array Numpy Salvati/numpy_acceleration.npy',
            np.array(acceleration))
        np.save('Array Numpy Salvati/numpy_GEARS_.npy', np.
            array(GEARS_))
        np.save('Array Numpy Salvati/numpy_time.npy', np.array(
            time))
        np.save('Array Numpy Salvati/numpy_max_len.npy', np.
            array(max_lunghezza_ts))
        print('Vettori salvati correttamente!')
```

Listing 3.4: Script Python usato per l'estrazione dei ping che andranno a formare le serie temporali.

Nella prima parte di codice i file *dataAIS.csv* vengono concatenati ed inseriti nel dataframe **df**. La variabile **df0** contiene il dataframe con le sessioni filtrate e bilanciate e relativa classe. Nel ciclo `for` viene prima selezionata la sessione corrispondente alla riga con `i`, vengono poi incrociati i dati con il dataframe **df0** per estrarre i ping relativi alla sessione attualmente selezionata. Vengono quindi generati i 2 vettori(**speed, course**) contenenti rispettivamente la velocità e la rotta della sessione attualmente puntata da `i`. Oltre ai dati di speed/course viene creato anche un vettore contenente il tempo, questo vettore serve a calcolare l'accelerazione media tra due ping, un altro parametro che useremo per addestrare la rete. L'accelerazione media viene calcolata come segue:

$$Acceleration_{ij} = \frac{Speed_{ij+1} - Speed_{ij}}{Time_{ij+1} - Time_{ij}}$$

Nell'ultima parte di codice si può vedere come i dati acquisiti vengono salvati nei numpy array. Avremo quindi le matrici così formate:

$$Speed = \begin{bmatrix} s_{11}, s_{12}, .., s_{1k} \\ s_{21}, s_{22}, .., s_{2p} \\ . \\ . \\ s_{m1}, s_{m2}, .., s_{mg} \end{bmatrix} \qquad Course = \begin{bmatrix} c_{11}, c_{12}, .., c_{1k} \\ c_{21}, c_{22}, .., c_{2p} \\ . \\ . \\ c_{m1}, c_{m2}, .., c_{mg} \end{bmatrix}$$

Con k non per forza uguale a p o g. Questo perché la lunghezze delle sessioni di pesca (quindi la lunghezze della serie temporali) sono indipendenti le une dalle altre quindi avremo lunghezze diverse per ogni sessione. Le matrici così composte non possono essere usate per creare il dataset così come in figura 3.1. Per risolvere questo problema ricorriamo allo *zero padding*. Lo zero padding consiste nell'inserimento di valori al fine di rendere la matrice di dimensioni omogenee. In pratica dopo aver determinato la sessione con numero di ping maggiore(*lunghezza massima*), si provvede ad aggiungere per ogni sessione un numero "*lunghezza massima* − *lunghezza sessione i-esima*" di zeri.

$$\begin{bmatrix} A & B & C & & \\ D & E & F & G & \\ H & I & & & \\ L & M & N & O & P \end{bmatrix} \rightarrow \begin{bmatrix} A & B & C & 0 & 0 \\ D & E & F & G & 0 \\ H & I & 0 & 0 & 0 \\ L & M & N & O & P \end{bmatrix}$$

Figura 3.2: Esempio di zero padding

Abbiamo quindi effettuato lo zero padding mettendo gli zero una volta davanti, ed una volta dietro. Fatto ciò è stato finalmente possibile creare il dataset con lo schema in 3.1 unendo le matrici:

$$[Speed : Course, Label]$$

con Label vettore contenete la classe di ogni sessione. Una volta ottenuto il dataset è però necessario una ripartizione del dataset in *_TRAIN* e *_TEST*. Con dataset_TRAIN avviene l'addestramento della rete, mentre con dataset_TEST viene testata la rete dopo l'addestramento. La ripartizione da noi effettuata è stata 70/30, ovvero il 70% delle sessioni per ogni classe è stata usata per l'addestramento ed il restante 30% per i test. Per fare ciò abbiamo creato il seguente script in python:

```
1 import pandas as pd
2 import random
3
4 path = '/home/giovanni/Documenti/Tirocino_AIS/codici_tirocinio/
    Dataset_/'
5 suf_path = [ 'Speed_Course_zero_dietro',
6              'Speed_Course_zero_davanti'
```

```
            ]
for q in range(len(suf_path)):
    df = pd.read_csv(path + suf_path[q], header=None, index_col
        =None)
    dimensioni = []
    c = 0
    indici_classi = []
    list_temp = []

    for i in range(df.shape[0]):
        c += 1
        list_temp.append(i)
        if i + 1 == df.shape[0]:
            indici_classi.append(list_temp)
            dimensioni.append(c)
            break
        if df.loc[i, df.shape[1]-1] != df.loc[i+1, df.shape
            [1]-1]:
            indici_classi.append(list_temp)
            dimensioni.append(c)
            list_temp = []
            c = 0

    train_list = []
    test_list = []

    for i in range(len(dimensioni)):
        t_70 = dimensioni[i] * 70 // 100
        random.shuffle(indici_classi[i])
        for j in range(dimensioni[i]):
            if j <= t_70:
                train_list.append(df.loc[indici_classi[i][j],
                    :].values)
            else:
                test_list.append(df.loc[indici_classi[i][j],
                    :].values)

    TRAIN = pd.DataFrame(train_list, index=None)
    TEST = pd.DataFrame(test_list, index=None)

    TRAIN.to_csv('/home/giovanni/Documenti/Tirocino_AIS/
        codici_tirocinio/archives_/TSC/' + suf_path[q] + '/' +
        suf_path[q] + '_TRAIN', header=None, index=None)
    TEST.to_csv('/home/giovanni/Documenti/Tirocino_AIS/
        codici_tirocinio/archives_/TSC/' + suf_path[q] + '/' +
        suf_path[q] + '_TEST', header=None, index=None)
    print( str(q) + 'DONE!')
```

Listing 3.5: Script Python usato per ripartire il dataset in TRAIN e TEST.

3.2.4 Inizio esperimento

Il primo dataset che abbiamo usato per l'esperimento è quello con la combinazione di **Speed e Course** con zero padding sia davanti che dietro, i risultati sono stati fin da subito ottimi con un'**accuracy** del **99%**.
D'ora in avanti consideriamo sottinteso l'uso dello zero padding sia davanti che dietro.
La seconda combinazione è stata quella **Speed, Course, Time**. Abbiamo quindi 3 variabili, con la matrice Time come contatore incrementale, il quale ad ogni ping viene incrementato di uno. Nel caso dello zero padding dietro, la riga viene riempita con la ripetizione dell'ultimo valore di Time, nel caso dello zero padding avanti vengono invece inseriti degli zeri. Avremo quindi rispettivamente per lo zero padding davanti e dietro:

$$Time_i = [0, 0, 0, 0, .., 0, 0, 0, 0, 1, 2, 3, 4, 5, .., 122, 123]$$

$$Time_i = [1, 2, 3, 4, 5, 6, 7, .., 122, 123, 123, 123, .., 123]$$

La forma del dataset è la seguente:

$$[Speed : Course : Time, Label]$$

Anche in questo caso i risultati sono stati ottimi, con un'accuracy intorno al **98%** sia per il dataset con lo zero padding davanti che dietro.
La terza combinazione è stata quella con **Speed, Course, Acceleration**, anche qui abbiamo 3 variabili e la matrice Acceleration precedentemente ricavata, viene ora inserita nel dataset, il quale ha la seguente forma:

$$[Speed : Course : Acceleration, Label]$$

L'accuracy è stata intorno del **98%**
La quarta combinazione è stata quella con **Speed, Course, Time, Acceleration**. In questo caso abbiamo quindi 4 variabili vengono inserite le matrici Time e Acceleration, ed il dataset ha la seguente forma:

$$[Speed : Course : Time : Acceleration, Label]$$

L'accuracy in questo caso è stata del **98%**.
Prima di procedere è bene introdurre il **Jerk** o in italiano lo **Strappo**. In meccanica classica si definisce Jerk la derivata dell'accelerazione rispetto al tempo, o la derivata di terzo ordine della posizione rispetto al tempo. La formula che andiamo ad usare è la seguente:

$$Jerk_{ij} = \frac{Acceleration_{ij+1} - Acceleration_{ij}}{Time_{ij+1} - Time_{ij}}$$

La quinta combinazione è stata quella con **Speed, Course, Jerk**, il dataset finale ha la seguente forma:

$$[Speed : Course : Jerk, Label]$$

L'accuracy è stata del **97%**.
Un altro concetto da introdurre è lo *smoothing* o smussamento. Lo smoothing consiste nell'applicazione di una funzione di filtro il cui scopo è evidenziare i pattern significativi, attenuando il rumore generato da artefatti ambientali, elettrici, elettronici, informatici o fisiologici oppure altri fenomeni di disturbo legati a fattori di scala molto piccoli (ad es. i movimenti millimetrici di un paziente nel neuroimaging che a causa dell'elevata risoluzione provocano effetti di traslazione) o a fenomeni ad alta velocità. Praticamente si tratta di fare una media tra valori contigui oppure molto vicini nello spazio (2D, 3D, 4D) oppure nel tempo.
Nei dataset successivi abbiamo usato lo smoothing. Per fare ciò abbiamo creato uno script in *Matlab* usando la funzione *smooth*[3] con parametro *"sgolay-filter"*.

```
max = length(time{1});                    % (0,0) (1,0) (0,1) (1,1)
zero_dietro = boolean(0);                  %(smooth(speed), smooth(acceleration)) (boolean)
for i = 2:1:length(time)
    if max < length(time{i})
      max = length(time{i});
    end
end

speed_1 = double.empty(0, max);
speed_0 = double.empty(0 , max);
acceleration_11 = double.empty(0, max);
acceleration_10 = double.empty(0, max);
acceleration_01 = double.empty(0, max);
acceleration_00 = double.empty(0, max);
jerk_11 = double.empty(0, max);  %(1,1)
jerk_10 = double.empty(0, max); %(1,0)
jerk_01 = double.empty(0, max); %(0,1)
jerk_00 = double.empty(0, max); %(0,0)

if zero_dietro == true
    zero_dietro
    %array speed 0 e 1
    for i = 1:1:length(speed)
        speed_0 = [speed_0; speed{i} zeros(1, max-length(speed{i}))];
        speed_1 = [speed_1; transpose(smooth(speed{i}, 'sgolay')) zeros(1, max-length(speed{i}))];
    end

    %array acceleration 11 10 01 00
    for i = 1:1:length(acc)
        acceleration_00 = [acceleration_00; acc{i} zeros(1,max-length(acc{i}))];  %0,0
        temp = [0;0;0];   %[01, 10, 11]
```

[3] https://www.mathworks.com/help/curvefit/smooth.html

```
32          for j = 1:1:length(acc{i})-1   %1,0
33              temp(1,j+1) = (speed_0(i,j+1)-speed_0(i,j))/(time{i
                    }(j+1) - time{i}(j));
34              temp(2,j+1) = (speed_1(i,j+1)-speed_1(i,j))/(time{i
                    }(j+1) - time{i}(j));
35          end
36          acceleration_01 = [acceleration_01; transpose(smooth(
                temp(1,:),'sgolay')) zeros(1, max-length(acc{i}))];
37          acceleration_10 = [acceleration_10; temp(2,:), zeros(1,
                 max-length(acc{i}))];
38          acceleration_11 = [acceleration_11; transpose(smooth(
                temp(2,:),'sgolay')) zeros(1, max-length(acc{i}))];
39      end
40
41      %array smooth 11 10 01 00
42      for i = 1:1:length(acc)
43          temp = [0 0; 0 0; 0 0; 0 0];
44          for j = 2:1:length(acc{i})-1
45              temp(1,j+1) = (acceleration_00(i,j+1)-
                    acceleration_00(i,j))/(time{i}(j+1) - time{i}(j
                    ));
46              temp(2,j+1) = (acceleration_10(i,j+1)-
                    acceleration_10(i,j))/(time{i}(j+1) - time{i}(j
                    ));
47              temp(3,j+1) = (acceleration_01(i,j+1)-
                    acceleration_01(i,j))/(time{i}(j+1) - time{i}(j
                    ));
48              temp(4,j+1) = (acceleration_11(i,j+1)-
                    acceleration_11(i,j))/(time{i}(j+1) - time{i}(j
                    ));
49          end
50          jerk_00 = [jerk_00; temp(1,:) zeros(1,max-length(temp
                (1,:)))];
51          jerk_10 = [jerk_10; temp(2,:) zeros(1,max-length(temp
                (2,:)))];
52          jerk_01 = [jerk_01; temp(3,:) zeros(1,max-length(temp
                (3,:)))];
53          jerk_11 = [jerk_11; temp(4,:) zeros(1,max-length(temp
                (4,:)))];
54      end
55      save('/home/giovanni/Documenti/Tirocino_AIS/
            codici_tirocinio/Array Matlab Salvati/Post/
            array_zero_dietro/matlab_speed_0.mat', 'speed_0')
56      save('/home/giovanni/Documenti/Tirocino_AIS/
            codici_tirocinio/Array Matlab Salvati/Post/
            array_zero_dietro/matlab_speed_1.mat', 'speed_1')
57      save('/home/giovanni/Documenti/Tirocino_AIS/
            codici_tirocinio/Array Matlab Salvati/Post/
            array_zero_dietro/matlab_acceleration_00.mat', '
            acceleration_00')
58      save('/home/giovanni/Documenti/Tirocino_AIS/
            codici_tirocinio/Array Matlab Salvati/Post/
            array_zero_dietro/matlab_acceleration_01.mat', '
            acceleration_01')
```

```
59    save('/home/giovanni/Documenti/Tirocino_AIS/
          codici_tirocinio/Array Matlab Salvati/Post/
          array_zero_dietro/matlab_acceleration_10.mat', '
          acceleration_10')
60    save('/home/giovanni/Documenti/Tirocino_AIS/
          codici_tirocinio/Array Matlab Salvati/Post/
          array_zero_dietro/matlab_acceleration_11.mat', '
          acceleration_11')
61    save('/home/giovanni/Documenti/Tirocino_AIS/
          codici_tirocinio/Array Matlab Salvati/Post/
          array_zero_dietro/matlab_jerk_00.mat', 'jerk_00')
62    save('/home/giovanni/Documenti/Tirocino_AIS/
          codici_tirocinio/Array Matlab Salvati/Post/
          array_zero_dietro/matlab_jerk_01.mat', 'jerk_01')
63    save('/home/giovanni/Documenti/Tirocino_AIS/
          codici_tirocinio/Array Matlab Salvati/Post/
          array_zero_dietro/matlab_jerk_10.mat', 'jerk_10')
64    save('/home/giovanni/Documenti/Tirocino_AIS/
          codici_tirocinio/Array Matlab Salvati/Post/
          array_zero_dietro/matlab_jerk_11.mat', 'jerk_11')
```

Listing 3.6: Script Matlab con funzione *smooth*

Tramite lo script in 3.6, siamo riusciti a generare le matrici di **Speed, Acceleration, Jerk** con le varie combinazioni di smoothing. Le combinazioni con cui sono stati generati i dataset sono:

- smooth solo sulla Speed quindi calcolo Acceleration ed in seguito calcolo del Jerk(Jerk10 come notazione). Il dataset è così formato:

  $$[SpeedSmooth : Course : Jerk10, Label]$$

  L'accuracy è stata del **98%**;

- smooth solo sull'Acceleration, quindi calcolo del Jerk dopo aver effettuato lo smooth(Jerk01 come notazione). Il dataset è così formato:

  $$[Speed : Course : Jerk01, Label]$$

  L'accuracy è stata del **99%**, miglior risultato registrato nel corso dell'intero esperimento;

- smooth prima sulla Speed, calcolo Acceleration, smooth sull'acceleration quindi calcolo del Jerk(Jerk11 come notazione). Il dataset è così formato:

  $$SpeedSmooth : Course : Jerk11, Label]$$

  L'accuracy è stata del **97%**;

Negli ultimi test abbiamo cercato di determinare quale delle due variabili principali(*Speed, Course*) abbia avuto un peso maggiore negli ottimi risultati

ottenuti. Per fare ciò abbiamo creato il dataset usando le variabili Speed, Course prima da sole e poi combinate rispettivamente con la variabile Time (contatore Incrementale). Le combinazioni sono state quindi:

$$[Speed, Label]$$

con un'accuracy del **98%** circa.

$$[Course, Label]$$

con un'accuracy del **90%** circa.

$$[Speed : Time, Label]$$

con un'accuracy del **78%** circa.

$$[Course : Time, Label]$$

con un'accuracy del **90%** circa.

3.2.5 Risultati

Di seguito la tabella con i risultati dettagliati dell'addestramento della rete effettuato con le varie combinazioni[4].

DATASET	*ACCURACY*	*DURATION(sec)*
Speed	**98.71% 97.85%**	**4823 4821**
Course	88.84% 90.12%	8679 8670
Speed/Time	58.15% 97.85%	5144 5140
Course/Time	87.3% 90.55%	5148 5154
Speed/Course	**99,35%**	**5150**
Speed/Course/Time	97.85% 98.06%	9188 9261
Speed/Course/Acceleration	98.28% 98.28%	9321 9314
Speed/Course/Time/Acceleration	98.74%	9523
Speed1/Course/Jerk11	96.99% 97.42%	9150 9141
Speed1/Course/Jerk10	98.71% 98.06%	9199 9184
Speed/Course/Jerk01	**98.06% 99.78%**	**5173 5175**
Speed/Course/Jerk00	97.42%	9283

Si può vedere in tabella che l'addestramento con la sola variabile *Speed*, ha restituito un'accuracy (98/97%) ed un tempo di addestramento migliori (4823/4821sec) rispetto all'addestramento con la sola variabile *Course* (88/90%) (8679/8670sec).
Le coppie *Speed/Time* e *Course/Time* hanno restituito dei buoni risultati in termini di accuracy, ma comunque sotto la media dei risultati con le altre combinazioni.
Il dataset formato dalla coppia *Speed/Course* è stato il punto di partenza, ed ha restituito fin da subito un'ottima accuracy con un buon tempo di addestramento.
La combinazione della coppia *Speed/Course* rispettivamente con *Time*, *Acceleration* e *Time/Acceleration* ha restituito un'accuracy che non si discosta molto da quella della coppia *Speed/Course*, la differenza sta nell'aumento del tempo di addestramento. Ciò può essere dovuto all'aumento del numero delle variabili.
L'introduzione della variabile *Jerk* ed in seguito l'uso dello smoothing ci hanno permesso di creare il dataset *Speed/Course/Jerk* con le relative combinazioni. I risultati restituiti dall'addestramento con queste tre variabili non si discostano molto né in accuracy né in tempo di addestramento dagli ottimi risultati già visti. La differenza la si ha però con il dataset *Speed/-Course/Jerk01*. Questo dataset è stato ottenuto tramite la combinazione di Speed (senza smooth), Course, e Jerk. In questo caso il Jerk è stato ottenuto

[4]il primo valore della cella corrisponde al risultato con zero padding davanti, il secondo al risultato con zero padding dietro

calcolando prima l'Acceleration tramite la Speed (senza smooth), in seguito effettuando lo smooth sull'Acceleration per poi calcolare il Jerk. I risultati restituiti da questo dataset sono stati i migliori, arrivando a toccare il $99,78\%$ di accuracy e con un tempo di addestramento relativamente minore ($5173/5175 sec$) rispetto agli altri dataset a tre variabili.

Capitolo 4

Conclusioni e sviluppi futuri

I risultati ottenuti hanno superato le aspettative iniziali ed aprono la strada ad un nuovo approccio nell'ambito del *precision fishing*. Osservando la tabella dei risultati risulta evidente come la variabile **Speed** abbia avuto un peso maggiore rispetto agli altri parametri, restituendo da sola un'accuracy del 98/97% e con un tempo di addestramento minore rispetto alle altre combinazioni di variabili. La coppia (**Speed**, **Course**) da sola ha restituito una accuracy del 99.35%, classificandosi come secondo miglior risultato. L'utilizzo delle variabili di **Jerk** e **Acceleration** insieme alla coppia (**Speed**, **Course**), nel dataset *Speed/Course/Jerk01*, ci ha restituito un'accuracy del 99,78% classificandosi come miglior risultato. Il lavoro pone le basi per mappatura delle attività di pesca con lo scopo di monitorare i relativi impatti per una gestione sostenibile delle risorse marine. In futuro il lavoro potrà essere portato ulteriormente avanti utilizzando magari altre variabili del dato AIS come ad esempio la *Latitudine* e la *Longitudine*.

Bibliografia

[1] Galdelli Alessandro. *AIS Data Processing*. 2019.

[2] Adriano Mancini Alessandro Galdelli et al. *A Cloud Computing Architecture To Map Trawling Activities Using Position Data*. 2019.

[3] Gianluca Baldassarre. *Introduzione alle reti neurali*. https://archive.org/details/BaldassarreIntroduzioneAlleRetiNeurali. 2018.

[4] Jason Brownlee. *How to Develop LSTM Models for Time Series Forecasting*. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting. 2018.

[5] Yanping Chen et al. *The UCR Time Series Classification Archive*. www.cs.ucr.edu/~eamonn/time_series_data/. Lug. 2015.

[6] Guillaume Chevalier. *LSTMs for human activity recognition*. 2016.

[7] devforfu. *A Simple LSTM-Based Time-Series Classifier*. https://www.kaggle.com/purplejester/a-simple-lstm-based-time-series-classifier. 2018.

[8] *Different fishing techniques and their impacts on the environment*. https://worldoceanreview.com/en/wor-2/fisheries-policy/mangement/different-fishing-techniques-and-their-impacts-on-the-environment/.

[9] Moody GB. «Spontaneous Termination of Atrial Fibrillation: A Challenge from PhysioNet and Computers in Cardiology 2004». In: *Computers in Cardiology* (2004).

[10] hfwaz. *InceptionTime*. https://github.com/hfawaz/InceptionTime. 2019.

[11] Hassan Ismail Fawaz et al. «InceptionTime: Finding AlexNet for Time Series Classification». In: *Data Mining and Knowledge Discovery* (2020).

[12] Weber Ismail Fawaz Forestier. «Deep learning for time series classification: a review». In: *Data Min Knowl Disc 33* (2019).

[13] Jason Lines, Sarah Taylor e Anthony Bagnall. «HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles for Time Series Classification». In: 2016. DOI: 10.1109/ICDM.2016.0133.

[14] Fabio Portesan. *Tutto sul sistema AIS*. https://www.hinelson.com/blog/tutto-sullais/. 2020.

Elenco delle figure

Listings