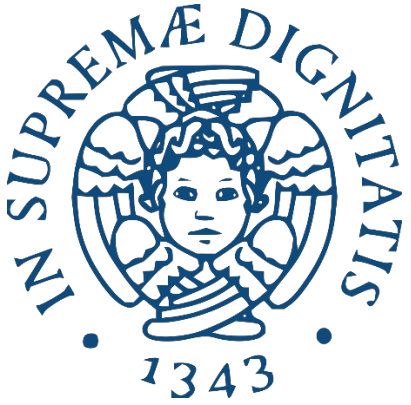

UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea triennale in Ingegneria Informatica

Design, sviluppo e ottimizzazione di una Rete Neurale Convoluzionale per il riconoscimento dei picchi del battito cardiaco dall'elettroencefalogramma

Relatori:

Ing.: Antonio Luca Alfeo

Prof.: Mario G.C.A. Cimino

Candidata:

Rossana Antonella Sacco

ANNO ACCADEMICO 2023/2024

Alla mia famiglia

ABSTRACT

Nell'ambito del *Machine Learning* il riconoscimento dell'attività umana, o *Human Activity Recognition* (di seguito *HAR*), è il problema di determinare l'azione che sta compiendo un essere umano. L'obiettivo dell'*HAR* è quello di identificare e classificare le attività umane da dati grezzi provenienti da sensori, come ad esempio accelerometri o giroscopi presenti in smartphone, smartwatch o dispositivi indossabili.

Tale problema è molto comune nell'ambito del *Machine Learning*, essendo applicabile in diversi contesti e applicazioni, tra cui il monitoraggio della salute, il fitness tracking, la sicurezza e la domotica. Alcuni esempi di attività umane che possono essere riconosciute tramite l'analisi dei dati in questione includono camminare, correre, salire le scale, guidare, sedersi, dormire e molti altri. Questo riconoscimento presenta tuttavia diverse sfide e problematiche a cui prestare attenzione nell'ambito del *Machine Learning* per poter ottenere dei risultati affidabili ed ottimali. Essendo i soggetti da cui provengono i dati degli esseri umani, per natura molto eterogenei in termini di velocità, resistenza e altre caratteristiche, può risultare difficile costruire un modello che riesca a generalizzare concretamente i dati su soggetti così differenti. Altre problematiche potrebbero essere i fattori ambientali che influenzano l'attività, la grande dimensionalità dei dati in esame e l'*overfitting*.

Lo studio condotto si concentra nell'analisi di dati raccolti tramite Elettroencefalogramma (EEG) su un insieme di 26 soggetti. Si utilizza una *Convolutional Neural Network* (anche CNN) *Unidimensionale* (o 1D) come modello per la classificazione e individuazione di picchi del battito cardiaco. Questa tipologia di modello permette di ottenere risultati all'avanguardia su compiti impegnativi come quello in esame, utilizzando funzionalità d'apprendimento su dati grezzi, senza quindi il bisogno di un intervento ingegneristico per l'utilizzo di dati manipolati. Il lavoro, inizialmente si è concentrato sulla valutazione delle prestazioni del modello utilizzando diverse configurazioni di iperparametri nell'architettura e nell'addestramento, utilizzando come metrica di valutazione l'accuratezza, con lo scopo di individuare i valori che fornissero risultati ottimali.
Nello specifico, sono stati testati e individuati i valori ottimali degli iperparametri *"batch_size", "learning_rate", "momentum"* ed è stato individuato l'ottimizzatore più adeguato, considerando lo *Stochastic Gradient Descendent* (anche SGD) e *ADAM.* Per farlo, è stata utilizzata la tecnica di *GridSearch Cross-Validation*, fornita dalla libreria di Python *'scikit-learn'*, con la quale è stato possibile effettuare una ricerca incrociata.
Successivamente, sono stati testati diversi *settings sperimentali* che permettessero di verificare quale fosse il miglior modo di aumentare le capacità di generalizzazione del modello. Nello specifico è stato testato il setting sperimentale *Leave-One-Subject-Out* (anche LOSO) e successivo affinamento dei risultati tramite tecnica di *Fine-Tuning* con diverse configurazioni.

I risultati ottenuti sono stati utili a identificare quali siano i migliori settaggi di un modello sequenziale affinché possa classificare adeguatamente i dati forniti con grandi capacità di generalizzazione.

INDICE

CAPITOLO 1

INTRODUZIONE

L'intelligenza artificiale (o IA) rappresenta una delle più rivoluzionarie e trasformative tecnologie del nostro tempo. Essa ha il potenziale per cambiare radicalmente il modo in cui viviamo, lavoriamo e interagiamo con il mondo attorno a noi. L'IA si riferisce alla capacità delle macchine di imitare l'intelligenza umana, comprese capacità d'apprendimento, ragionamento, problem solving, percezione e linguaggio naturale.

Il *Machine Learning* è una branca dell'IA che si occupa dello sviluppo di algoritmi e modelli informatici capaci di apprendere dai dati e migliorare le proprie prestazioni nel tempo senza essere esplicitamente programmato per farlo. Quest'approccio si basa sull'analisi di grandi quantità di dati, attraverso la quale gli algoritmi imparano a identificare pattern, relazioni o strutture significative al loro interno. I settori in cui queste tecniche posso essere applicate sono molteplici ed il loro effettivo utilizzo è in costante crescita ed evoluzione.

In questo studio, la ricerca svolta si è focalizzata sul *Machine Learn*ing nell'ambito dell'Human *Activity Recognition* (o HAR), il quale si occupa di sviluppare algoritmi e modelli in grado di riconoscere e classificare le attività svolte dagli esseri umani utilizzando dati provenienti da sensori come accelerometri, giroscopi e altri dispositivi indossabili, con lo scopo di identificare le attività specifiche che una persona sta svolgendo in un determinato momento. Lo sviluppo di quest'analisi è estremamente utile in un'ampia varietà di contesti e può portare numerosi benefici sociali, sanitari ed economici:

- ***Monitoraggio della salute e del benessere***. Nel settore della salute, l'*HAR* può essere utilizzato per monitorare l'attività fisica, il sonno, la postura e altri parametri fisiologici, tramite dispositivi indossabili, al fine di valutare lo stato di salute e di benessere di un individuo. I professionisti della salute e dell'assistenza sono

interessati all'utilizzo di sistemi di questo genere per monitorare i pazienti, migliorare la qualità dell'assistenza e prevenire incidenti.

- ***Fitness tracking***. Nell'ambito dello sport, i sistemi di *HAR* offrono la possibilità di tracciare e analizzare le prestazioni degli atleti durante l'allenamento, consentendo un feedback in tempo reale sull'attività svolta e sul raggiungimento degli obiettivi di fitness. Può essere utilizzato per ottimizzare i programmi d'allenamento in base alle attività svolte e i progressi raggiunti.
- ***Sicurezza e monitoraggio***. Nell'ambito della sicurezza, i sistemi di *HAR* possono essere impiegati per rilevare comportamenti sospetti o situazioni d'emergenza, come cadute o incidenti, e attivare le risposte adeguate. Le aziende che operano nei settori della sicurezza, della produzione e della domotica sono interessate allo sviluppo e all'implementazione di questo tipo di sistemi per migliorare la qualità dei loro prodotti e servizi.
- ***Comprendere il comportamento umano***. L'analisi nell'ambito di *HAR* consente di studiare e comprendere il comportamento umano in vari contesti, come l'attività fisica, la mobilità e le attività quotidiane. Ricercatori ed accademici interessati alla comprensione del comportamento umano sono interessati allo sviluppo di nuove tecnologie per il monitoraggio, l'assistenza e la scoperta di pattern utili ai loro studi.

In questi contesti, dunque, l'ambito dell'*HAR* svolge un ruolo cruciale in termini di efficienza, qualità della vita e sicurezza, automatizzando processi e compiti che altrimenti richiederebbero tempo e risorse umane.

Una delle principali problematiche affrontate nell'ambito dell'HAR è la vasta varietà di comportamenti umani. Ogni individuo ha le proprie abitudini, stili di vita e capacità fisiche, il che rende difficile per i sistemi generalizzare correttamente i modelli di attività umane su una vasta gamma di persone. Inoltre, gli ambienti in cui vengono svolte le attività possono variare notevolmente. La luce, la temperatura, la superficie del terreno e altri fattori ambientali possono influenzare le attività e complicare il processo di riconoscimento.

Quando si tratta di analisi di questo tipo, si analizza una vasta quantità di dati, vista la natura delle misurazioni effettuata dai sensori. Parte fondamentale del lavoro, è la

gestione di questi dati. Sono necessarie importanti risorse computazionali, oltre che una particolare attenzione per quanto riguarda l'etichettatura degli stessi, parte fondamentale del processo d'addestramento e di testing. Attraverso l'etichettatura, infatti, il modello identifica, riconosce ed associa un determinato pattern di valori ad una particolare attività. Durante il processo d'addestramento, il rischio più grande è il verificarsi del fenomeno dell'overfitting: un modello si adatta troppo ai dati fornitagli per l'addestramento, perdendo le capacità di generalizzazione.
Risulta quindi fondamentale una corretta gestione dei dati.

L'analisi svolta in questo studio utilizza come dati quelli raccolti tramite Elettroencefalogramma (EEG) su un insieme di 26 soggetti, i quali sono stati sottoposti a tale esame in una prima fase a riposo e successivamente sottoposti a test del freddo, con lo scopo di monitorarne la variazione di attività cardiaca (HVR). L'HVR fornisce informazioni cruciali per comprendere i fenomeni fisiologici. La sua ricostruzione tramite l'analisi dell'EEG (elettroencefalogramma) rappresenta un'importante innovazione nel campo medico, ancora in fase di sviluppo, per lo studio **dell'interazione funzionale tra cervello e cuore**. Le registrazioni EEG sono frequentemente affette da rumore, ma vengono spesso corrette grazie all'analisi delle HRV. Tuttavia, esse non sono sempre disponibili come dati campionati. Questa situazione motiva l'interesse nel tentativo di ricostruire le HRV esclusivamente utilizzando i dati EEG.

Viene utilizzato come modello una *Convolutional Neural Network Unidimensionale*, con un'architettura sequenziale a strati. Si è utilizzata come metrica di valutazione, durante tutta la durata dello studio, l'accuratezza, la quale indica in che percentuale il modello riesce a classificare dei nuovi dati in seguito all'addestramento. Durante la parte iniziale, l'obiettivo dell'analisi è stato individuare i valori ottimali da assegnare agli iperparametri presenti nella configurazione dell'architettura di rete o utilizzati in fase d'addestramento, con lo scopo di migliorare le capacità di generalizzazione del modello, e quindi le performance. Si è utilizzato come setting sperimentale iniziale la suddivisione individuale dei dati raccolti per soggetto. Individuati i parametri ottimali, si è deciso di testare un nuovo setting sperimentale affiancato da una tecnica d'affinamento.

Il confronto e l'analisi di diversi settings, iperparametri ed ottimizzatori risulta fondamentale per il raggiungimento di performance affidabili e quindi all'ottenimento di modelli utili, affidabili, e applicabili in un contesto reale.

CAPITOLO 2

RELATED WORKS

Il presente capitolo si propone di esaminare e analizzare i lavori correlati al campo di studio, concentrandosi sulle tecniche di apprendimento automatico. Nel contesto di un panorama in costante evoluzione, la ricerca e lo sviluppo di sistemi innovativi per il monitoraggio e la gestione di dati di questo tipo sono diventati sempre più cruciali. Questo capitolo esplorerà le ricerche esistenti mettendo in luce le problematiche relative all'HAR, proponendo le motivazioni che hanno spinto alla ricerca di un approccio differente, che tenti di migliorare le prestazioni dei modelli applicati.

2.1 PROBLEMI DELL'HUMAN ACTIVITY RECOGNITION

Come precedentemente introdotto, il campo dell'Human Activity Recognition si focalizza sullo sviluppo di modelli di Machine Learning capaci di riconoscere, interpretare e classificare le attività umane. Trattandosi di un campo innovativo e in ampia crescita, i problemi ancora presenti al giorno d'oggi sono molteplici [1][2][3]:

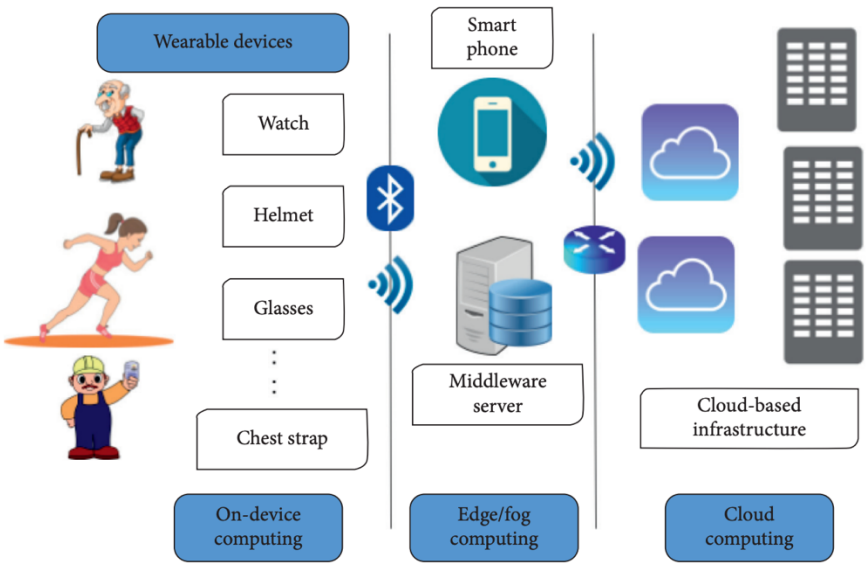

Fig.1: Esempio di raccolta dei dati

- ***Standardizzazione della raccolta dei dati.*** La disponibilità di un'ampia varietà di dispositivi indossabili (tecnica predominante utilizzata in questo campo) può causare un'incoerenza nella qualità della raccolta dei dati in assenza di standardizzazione nella raccolta dei dati e il posizionamento dei sensori sul corpo umano [1].
- ***Privacy dell'utente.*** È necessario che l'utente abbia fiducia che il dispositivo non condivida le proprie informazioni con il pubblico. Un metodo utilizzato per la risoluzione di tale problema, insieme al consumo energetico, si *chiama Edge Computing*: processo locale che garantisce privacy e tempi di risposta simili al tempo reale [1].
- ***Gestione dei dati.*** I dati provenienti da sensori inerziali sono caratterizzati da un'elevata dimensionalità, specialmente se si considerano più sensori o più assi di misura. Gestire ed analizzare una grande mole di dati può richiedere risorse computazionali significative e rendere più complessa l'identificazione di pattern significativi [2].
- ***Individualità dei soggetti.*** Le attività umane possono variare notevolmente tra individui diversi e possono essere eseguite in modi differenti a seconda del contesto e dell'ambiente. Inoltre, le caratteristiche fisiche e comportamentali degli individui possono influenzare la modalità in cui i sensori rilevano e registrano le attività. Gestire questa eterogeneità nei dati può richiedere tecniche di normalizzazione e calibrazione per garantire la coerenza e la comparabilità tra i dati raccolti da diversi soggetti [2][3].

2.1.1 APPROCCI CLASSICI

Ad oggi, tra gli approcci più utilizzati, sono presenti il metodo *Random Forest* e le *Reti Neurali (RN)*:

- ***Random Forest*** è un algoritmo di apprendimento supervisionato basato su una tecnica chiamata "bagging" (Bootstrap Aggregating) e su un insieme di alberi decisionali. Ogni albero decisionale viene addestrato su un sottocampionamento casuale del dataset d'addestramento, utilizzando un subset casuale delle feature.

Consente di valutare l'importanza relativa delle feature ed identificarne le più significative [2].

- Le ***Reti Neurali*** sono modelli di apprendimento automatico che utilizzano una struttura ispirata al funzionamento del cervello umano per elaborare input e generare output. Tuttavia, nel contesto dell'Human Activity Recognition (HAR), queste reti neurali generiche possono incontrare alcuni problemi a causa della rigidità della loro struttura e l'incapacità di gestire grandi quantità di dati.

Nonostante i diversi vantaggi che questi approcci offrono, essi presentano alcuni punti critici. Random Forest, ad esempio, può richiedere risorse computazionali significative e, oltre ad un modellamento dei dati per renderlo applicabile, può presentare diverse difficoltà nell'interpretabilità del modello, non fornendo una relazione diretta tra feature e output. Per le RN il problema principale è rappresentato dalle scarse capacità di generalizzazione su modelli adattati a un particolare insieme di dati di allenamento per funzionare in modo efficace su nuovi dati provenienti da diverse persone o ambienti.

2.2 MOTIVAZIONI DELL'APPROCCIO SCELTO

L'approccio che viene proposto e analizzato in questo studio tenta di minimizzare e/o risolvere i problemi sollevati precedentemente.
Si è scelto di svolgere l'analisi attraverso l'uso di Reti Neurali Convoluzionali (anche CNN), un'architettura di rete neurale profonda. Originariamente progettate per l'elaborazione delle immagini, hanno trovato ampio utilizzo anche nell'ambito dell'Human Activity Recognition:

- ***Apprendimento Gerarchico***. Le CNN sono in grado di apprendere in modo automatico e gerarchico le feature dei dati, partendo da pattern semplici fino ad arrivare ai più complessi. Questa capacità è particolarmente utile nel nostro contesto, dove le azioni possono essere composte da una combinazione di movimenti più semplici.
- ***Convoluzione***. Tale operazione permette una maggiore efficienza computazionale e permette l'addestramento di dataset molto grandi senza la necessità di un'enorme

quantità di memoria. L'operazione di convoluzione consente, inoltre, di catturare pattern spaziali nei dati, come quelle provenienti da sensori indossabili.

- ***Robuste alla traslazione e alla variazione di scala***. Tale robustezza è data dalla presenza delle operazioni di convoluzione e degli strati di pooling. Consente alle CNN di riconoscere le stesse features anche se di dimensioni o posizioni differenti. Tale capacità risulta preziosa nel nostro ambito essendo le azioni eseguibili in posizioni e velocità differenti.
- ***Transfer Learning***. Questa tecnica permette di ottenere buone prestazioni su dataset più piccoli, utilizzando le conoscenze di un modello precedentemente addestrato su un dataset più grande.

Considerando queste caratteristiche, dal punto di vista applicativo, risulterebbe un sistema efficiente ed efficace, applicabile nell'ambito dell'Human Activity Recognition in modo significativo.

CAPITOLO 3

DESIGN E IMPLEMENTAZIONE

Sulla base delle problematiche evidenziate nel capitolo precedente, in questo capitolo viene proposta la soluzione implementata in questo studio, illustrandone dettagliatamente design e implementazione. Verrà infine discusso un caso d'uso a cui questo studio potrebbe essere applicato.

3.1 DESIGN

In questa sezione viene fornita un'approfondita descrizione dell'approccio utilizzato durante le varie fasi dell'analisi, facendo chiarezza in particolar modo sulla tipologia di tecniche utilizzate e degli iperparametri considerati.

3.1.1 CLASSIFICAZIONE

La classificazione nel Machine Learning è il processo di assegnazione dei dati a delle categorie predefinite in base alle loro caratteristiche osservate. L'obiettivo è quello di costruire un modello predittivo che possa generalizzare dalle osservazioni passate tramite l'addestramento e assegnare correttamente le etichette ai nuovi dati in base alle caratteristiche rilevanti.

Nel caso del dataset utilizzato, le etichette dei dati sono binarie, rappresentando picchi e non. Lo studio effettuato si è concentrato sulla classificazione attraverso l'uso di una Rete Neurale Convoluzionale Unidimensionale.

3.1.2 PRE-PROCESSING DEI DATI

Le tecniche di pre-processing sono utilizzate per preparare i dati e renderli adatti agli usi proposti, prima di essere utilizzati nell'addestramento del modello. Questo passaggio risulta fondamentale per garantire che i dati siano nella forma più adatta, permettendo il miglioramento nelle prestazioni del modello.

Nello specifico, sono state eseguite le seguenti operazioni sui dati prima dell'effettivo utilizzo degli stessi nell'analisi:

- Normalizzazione dei dati attraverso l'uso della funzione ***StandardScaler*** della libreria "*scikit-learn*" di Python. Tale funzione applica una trasformazione su ciascuna colonna in modo tale che i dati abbiano media nulla e deviazione standard unitaria, garantendo che i dati abbiano una distribuzione normale.
- Suddivisione dei dati, tramite la creazione di maschere booleane, in set di test e set d'addestramento. Il 10% dei dati sarà utilizzato come **set di test** e il restante 90% come **set d'addestramento**. Se non specificato, la percentuale di dati utilizzata nei rispettivi set durante lo studio sarà invariata.

Questo approccio consente di valutare l'efficacia del modello su dati diversi da quelli utilizzati durante l'addestramento, fornendo una misura più accurata delle capacità di generalizzazione del modello. Inoltre, questa suddivisione aiuta a prevenire il fenomeno di *overfitting,* ossia un apprendimento troppo specifico sui dati d'addestramento che porta ad una diminuzione delle capacità di generalizzazione*,* che potrebbe causare una riduzione delle prestazioni del modello.

3.1.3 RETE NEURALE CONVOLUZIONALE

Le Reti Neurali Convoluzionali (anche CNN) sono un tipo di architettura di rete neurale profonda ampiamente utilizzata nell'ambito nel Machine Learning, le cui principali caratteristiche sono state elencate nel capitolo precedente.

Nel nostro studio utilizziamo un'architettura sequenziale, avvalendoci della libreria di Python "Keras", costituita da una sequenza lineare di strati neurali, dove ciascuno strato è connesso direttamente ai suoi strati precedenti e successivi, ognuno dei quali esegue una specifica operazione sull'input. Nel nostro caso specifico avremo:

- **Due stati di Convoluzione**: costituiti da filtri, i quali sono formati da "pesi" con cui viene effettuata l'operazione di convoluzione. Il risultato dell'applicazione di un filtro è appunto il filtraggio dell'input ed è chiamato *mappa delle caratteristiche.* I due strati di convoluzione saranno intervallati e susseguiti da uno strato di pooling.

- **Due strati di Pooling**: tale livello opera su ciascuna mappa delle caratteristiche separatamente, le comprime e ne generalizza le rappresentazioni, diminuendo il rischio di overfitting.
- **Strato completamente connesso**: dopo aver ridimensionato le mappe in output dallo strato precedente si crea tale stato definendo il numero di unità nascoste (neuroni presenti). In questo caso applichiamo la *funzione di attivazione "relu"* che permetterà di avere solo input positvi o nulli.
- **Strato di output**: si usa la funzione di attivazione "*sigmoid*", utilizzata per produrre un output binario.

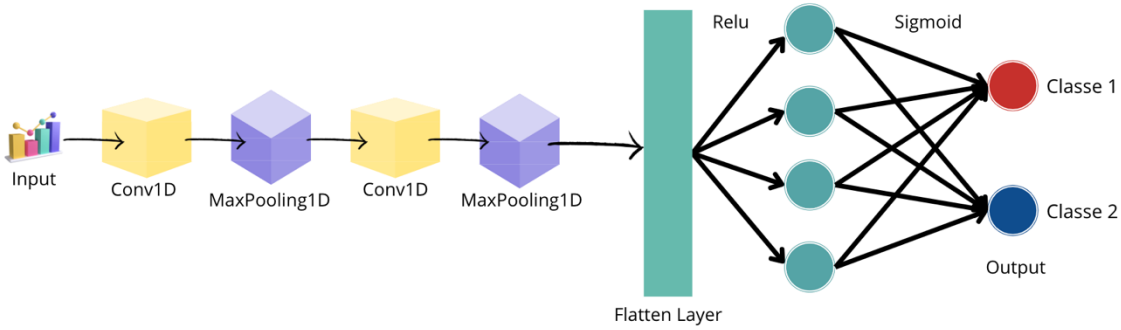

Fig.2: Rappresentazione grafica degli strati della rete neurale convoluzionale utilizzata

3.1.3.1 IPERPARAMETRI DELLA CNN

Parte fondamentale di una valida classificazione dei dati tramite l'uso di CNN, è la corretta scelta dei valori da assegnare agli iperparametri della rete. Tali parametri influenzano in maniera significativa il comportamento e le prestazioni della CNN ed è necessario fissarli prima dell'avvio del processo d'addestramento.

In particolare, i parametri d'interesse in questo studio sono i seguenti:

- ***Batch_size:*** indica il numero di campioni di dati d'addestramento utilizzati in una singola iterazione durante il processo d'addestramento del modello. Determina quindi il numero di campioni elaborati contemporaneamente prima dell'aggiornamento dei pesi del modello.

- ***Learning_rate:*** in italiano Tasso d'Apprendimento, indica la quantità d'aggiornamento che viene applicata ai pesi del modello durante il processo d'addestramento, controlla quindi la velocità con cui il modello si adatta ai dati durante l'ottimizzazione dei pesi.
- ***Momentum:*** utilizzato per velocizzare la convergenza durante l'addestramento di un modello, aggiungendo una frazione della direzione precedente dell'aggiornamento dei pesi al nuovo aggiornamento. Tipicamente assume valori compresi tra *0 e 1*. Un valore più alto indica una maggiore relazione con le direzioni precedenti.

3.1.3.2 OTTIMIZZATORI

Nel contesto delle *Reti Neurali Convoluzionali, l'ottimizzatore* svolge un ruolo critico nel processo d'addestramento del modello. L'obiettivo principale dell'ottimizzatore è aggiornare i pesi dei neuroni della rete in modo da ridurre al minimo la funzione di perdita durante l'addestramento.

La ricerca dell'ottimizzatore più adatto al nostro caso di studio ha portato al confronto tra i seguenti ottimizzatori:

- ***Stochastic Gradient Descent*** **(o SGD):** L'obiettivo di SGD è quello di minimizzare la funzione di perdita, regolando i pesi del modello in modo iterativo. La principale caratteristica di SGD è che esegue l'ottimizzazione utilizzando un campione casuale dei dati di addestramento per calcolare il gradiente della funzione di perdita. Questo approccio rende SGD particolarmente efficace quando si lavora con grandi dataset, poiché riduce la complessità computazionale rispetto all'utilizzo dell'intero dataset di addestramento per ogni iterazione. Matematicamente, l'aggiornamento dei pesi del modello durante l'addestramento con SGD è definito come:

$$\boldsymbol{w_{t+1} = w_t - \eta \cdot \nabla J(w_t)}$$

  - w_t rappresenta i pesi del modello al passo *t*
  - η è il tasso di apprendimento ("learning_rate")
  - $\nabla J(w_t)$ è il gradiente della funzione di perdita rispetto ai pesi al passo *t*

Questo processo viene ripetuto per un numero specificato di epoche o fino a quando non viene raggiunto un certo criterio di arresto.

- **ADAM:** è una variante dell'algoritmo di ottimizzazione SGD che sfrutta il concetto di *"momentum"* dinamicamente per adattare la velocità di apprendimento durante l'addestramento del modello. Il *"momentum"'* è una tecnica che mira ad accelerare la convergenza e ridurre l'instabilità durante l'addestramento, aggiungendo una componente di inerzia agli aggiornamenti dei pesi.

3.1.4 GRIDSEARCH

Al fine di ottimizzare la ricerca dei valori ottimali da assegnare ai parametri del modello, durante il nostro studio abbiamo utilizzato la funzionalità "GridSearch", fornita dalla libreria "scikit-learn" di Python. Si tratta di una tecnica di ottimizzazione dei parametri utilizzata nell'ambito del *Machine Learning* per trovare la combinazione ideale dei parametri di un modello. Questo processo mira a massimizzare le prestazioni del modello su un insieme di dati di valutazione. Se prevista l'analisi e la valutazione del comportamento di più parametri viene utilizzata la tecnica di *cross-validation*, altrimenti, viene effettuato "GridSearch" su tale parametro utilizzando la tecnica dei dati d'addestramento, senza necessità di un'ulteriore suddivisione dei dati per effettuare la *cross-validation.*

3.1.5 SETTINGS SPERIMENTALI

Durante lo svolgimento dello studio sono stati analizzati due diversi setting sperimentali, con il fine di individuare quale portasse alle migliori prestazioni per il modello:

- Addestramento ed analisi delle prestazioni suddividendo il dataset **per soggetto**. Utilizzando tale setting sperimentale il modello viene addestrato e testato utilizzando nei rispettivi set i dati appartenenti ad un singolo soggetto. Questo procedimento viene ripetuto per ogni soggetto e viene analizzata l'accuratezza risultante dalle perfomance ottenute sui singoli test.
- **Leave-One-Subject-Out** (o LOSO): si tratta di un metodo di valutazione incrociata che consente di valutare le prestazioni di un modello utilizzando un approccio iterativo. Con questo metodo, su un totale di *N* soggetti, si utilizzano *N-1* soggetti

come **set d'addestramento** ed il soggetto escluso come **set di test**. Questo metodo aiuta a valutare quanto il modello sia in grado di generalizzare su nuovi soggetti non visti durante l'addestramento.

3.1.5.1 FINE-TUNING

Letteralmente "messa a punto", questo metodo implica l'ulteriore addestramento di un modello pre-addestrato su un ampio set di dati, utilizzando un set più piccolo e specifico per un numero di epoche minore. Lo scopo è specializzare il modello, adattandolo alle caratteristiche del nuovo dataset. Durante il Fine-Tuning, una parte del modello viene messa in uno stato non modificabile, noto come "***freeze***", mentre la parte rimanente, solitamente i livelli superiori responsabili della classificazione finale, viene riaddestrata. Questa tecnica consente al modello di trarre vantaggio dalle conoscenze precedentemente apprese su un ampio insieme di dati, mentre si adatta in modo più preciso alle specifiche esigenze del nuovo compito o dataset.

3.2 IMPLEMENTAZIONE

Nello studio svolto, il sistema prodotto basa le proprie previsioni su dati provenienti da dispositivi indossabili, con lo scopo di riconoscere l'attività eseguita dal soggetto che li indossa. Dunque, tra i vari settori in cui questo sistema risulta applicabile, vi è quello della sicurezza.

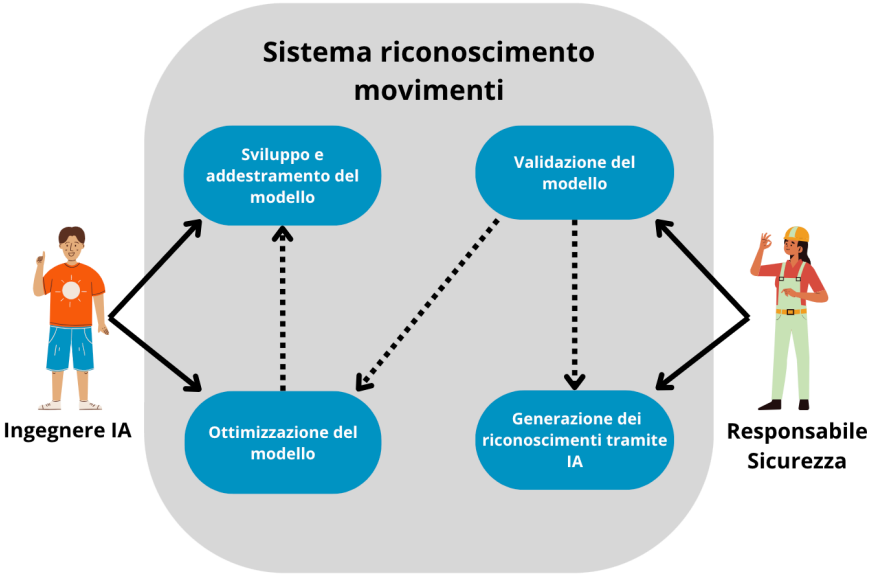

Fig. 3: Caso d'uso nell'ambito della sicurezza

3.2.1 USE CASE

Un possibile caso d'uso è illustrato in *Figura 3.* In questo contesto sono presenti due attori principali:

- ***Ingegnere IA***: Esso è l'attore responsabile dello sviluppo e l'addestramento del sistema. In particolare, si occupa di sviluppare un modello di Machine Learning atto al riconoscimento dei movimenti. Tramite i riscontri ottenuti osservando i risultati, anche in collaborazione con l'utente a cui il servizio è rivolto, si occupa di ottimizzare il modello per ottenere le prestazioni desiderate.
- ***Responsabile della Sicurezza***: rappresenta l'attore principale, ossia colui che usufruisce del servizio. In particolare, il modello potrà essere utilizzato per generare automaticamente riconoscimenti di comportamenti sospetti o pericolosi, come una caduta o un incidente. Egli utilizzerà i risultati ottenuti dall'uso del sistema per determinare le capacità e l'affidabilità del modello sviluppato, partecipando, tramite feedback, all'ottimizzazione del modello

L'utilizzo di tali sistemi permette una risposta rapida e efficace, che in ambiti come quello della sicurezza, rappresenta una svolta cruciale.

CAPITOLO 4

CASE STUDY

Questo capitolo offre una descrizione dettagliata del dataset utilizzato durante lo studio svolto. Verrà inoltre illustrata una panoramica sul codice implementato per il trattamento di esso, facendo particolare riferimento alle librerie utilizzate.

4.1 IL DATASET

Il dataset utilizzato in questo studio è stato creato con l'intenzione di studiare l'interazione funzionale tra cervello e cuore (BHI, Brain-Heart Interplay), in particolare l'interazione tra oscillazioni corticali e simpatico vagali cardiaci, combinando dinamiche neurali corticali (Elettroencefalografia, EEG) e periferiche (simpatico vagali cardiaci).

L'elettroencefalogramma è una tecnica di registrazione elettrofisiologica che misura l'attività elettrica del cervello attraverso elettrodi posizionati sulla superficie del cuoio capelluto. È una delle tecniche più utilizzate per studiare l'attività cerebrale in tempo reale.

Per raccogliere i dati, un gruppo di 32 adulti sani destrimani si è sottoposta a un test del freddo registrando i rispettivi segnali fisiologici. I dati di 26 soggetti (età compresa tra 21 e 41 anni, in media 27 anni, 13 maschi e 13 femmine) sono stati sottoposti a ulteriori analisi a causa della lunghezza e della qualità dei dati. In particolare, i dati di tre soggetti non sono stati considerati a causa della presenza di artefatti nei loro dati fisiologici, mentre i dati di altri tre soggetti sono stati scartati a causa del ritiro precoce della mano dall'acqua fredda. La registrazione di ciascun soggetto comprendeva EEG ad alta densità con 128 canali, attività respiratoria e un ECG a una sola derivazione, campionati a 500 Hz. Prima dell'acquisizione dei dati, ai soggetti è stato chiesto di sedersi comodamente su una sedia per garantire la stabilizzazione emodinamica. Il test consisteva in uno stato di riposo di 3 minuti, seguito da un massimo di 3 minuti di test del freddo e il recupero successivo, durante il quale ai soggetti è stato chiesto di ritirare la mano dal secchio d'acqua ghiacciata. Durante lo svolgimento, ai soggetti è stato chiesto di tenere gli occhi chiusi al fine di

minimizzare gli artefatti. I soggetti sono stati guidati mentre immergevano la mano sinistra fino al polso in un secchio riempito d'acqua ghiacciata (0-4 °C) [4].

Lo spettrogramma EEG è stato calcolato utilizzando la trasformata di Fourier a breve termine con una finestra di Hanning. I calcoli sono stati eseguiti attraverso una finestra temporale scorrevole di 2 secondi con un overlap del 50%, risultando in una risoluzione dello spettrogramma di 1 secondo e 0,5 Hz. Successivamente, le serie temporali sono state integrate all'interno di cinque bande di frequenza (delta: 1–4 Hz, theta: 4–8 Hz, alfa: 8–12 Hz, beta: 12–30 Hz, gamma: 30–45 Hz) [4].

Essendo il modello utilizzato una CNN, ciò ha reso possibile l'utilizzo di un dataset composto da dati "grezzi", ossia direttamente raccolti da sensori, con solo un minimo bisogno di preprocessazione e manipolazione.

Durante il trattamento dei dati, essendo essi di tipo temporale e quindi risultato di misurazioni effettuate nel tempo, eseguiamo il ridimensionamento sugli stessi considerando:

- **Il numero di serie temporali:** sequenze di punti temporali organizzati in ordine cronologico, rappresenteranno il numero di canali. Una serie temporale è una sequenza ordinata di dati registrati a intervalli di tempo regolari.

- **Il numero di gruppi di campioni:** è il conteggio totale delle osservazioni o delle misurazioni effettuate nel set di dati, rappresenteranno il numero di righe. Ogni campione rappresenta un'osservazione unica del fenomeno in studio e può essere costituito da una singola misurazione o da un vettore di misurazioni.

- **Il numero di punti temporali:** si riferiscono ai momenti specifici nel tempo in cui sono state effettuate le osservazioni o le misurazioni all'interno di una serie temporale o di un set di dati temporali. Questi punti rappresentano gli istanti temporali associati alle osservazioni e sono utilizzati per organizzare e indicizzare i dati nel tempo. In questo caso specifico, rappresenteranno il numero di colonne.

L'obiettivo principale dell'analisi di dati di tipo temporale è l'osservazione e identificazione di pattern e tendenze che questi presentano.

Nonostante la possibilità di non dover intervenire approfonditamente sui dati presenti nel dataset, è stato comunque necessario effettuare su di essi, prima di poterli dare in input al modello, una *normalizzazione.* Essa è necessaria per evitare che i dati siano su scale di misura notevolmente diverse, il che potrebbe influenzare negativamente l'efficacia degli algoritmi di apprendimento automatico. In questo caso specifico si è applicata una trasformazione su ciascuna colonna, in modo tale che i dati abbiano media nulla e deviazione standard unitaria.

4.2 LIBRERIE UTILIZZATE

Il codice è stato interamente implementato in Python, sfruttando diverse librerie sia per la creazione e l'addestramento dei modelli, sia per la valutazione delle prestazioni e generazione dei grafici rappresentanti esse.
Qui di seguito sono elencate le principali librerie utilizzate ed il modo in cui sono state impiegate:

- **Keras:** si tratta di una libreria potente e flessibile, ampiamente utilizzata nell'ambito del Machine Learning. Essa fornisce gli strumenti per la creazione, l'addestramento e la valutazione di modelli per l'apprendimento automatico, comprese le Reti Neurali Convoluzionali. Fornisce una vasta gamma di livelli convoluzionali, pooling, normalizzazione, funzioni di attivazione, ottimizzatori e funzioni di perdita, che possono essere utilizzati per costruire architetture CNN complesse.
- **Scikit-learn:** Libreria per la gestione e implementazione di modelli di Machine Learning, utilizzata per la normalizzazione dei dati, tramite l'utilizzo della sua funzione "StandardScaler", e per la cross-validation dei parametri, tramite la funzionalità di "GridSearch".
- **Pandas:** Libreria per la gestione dei dati, utilizzata per caricare i dati dal dataset tramite la funzione "read_csv".
- **Matplotlib:** utilizzata per la creazione dei grafici.

- **Numpy:** Libreria per il calcolo scientifico che consente di svolgere operazioni su array in modo veloce e efficiente.

CAPITOLO 5

RISULTATI SPERIMENTALI

Nel seguente capitolo sono rappresentati i risultati sperimentali ottenuti attraverso l'implementazione e l'esecuzione delle metodologie descritte precedentemente. L'obiettivo principale di questa fase è valutare le prestazioni e l'efficacia delle tecniche proposte.

I risultati qui presentati forniscono una valutazione delle prestazioni del modello di Machine Learning utilizzato, evidenziandone le problematiche e le osservazioni emerse durante l'analisi.

5.1 CONCETTI PRELIMINARI

5.1.1 METRICA UTILIZZATA

Le metriche di valutazione sono strumenti utilizzati per misurare le prestazioni di un modello di Machine Learning confrontando le predizioni del modello con i valori effettivi nel set di dati di test.
Nel presente studio, e per tutta la durata dello stesso, utilizzeremo come metrica di valutazione l'*accuratezza*. Essa è una delle metriche di valutazione più comuni e semplici da utilizzare in questo ambito. Matematicamente:

$$\boldsymbol{Accuratezza} = \frac{Numero\ di\ predizioni\ corrette}{Numero\ totale\ di\ predizioni} \times\ 100\%$$

Dove:

- Il "numero di predizioni corrette" è il numero totale di campioni nel set di dati di test per i quali il modello ha fornito una predizione corretta.
- Il "numero totale di predizioni" è il numero totale di campioni nel set di dati di test.

L'accuratezza, essendo facilmente interpretabile, è intuitivamente comprensibile anche per le persone non esperte di Machine Learning. Presenta però alcune limitazioni, infatti nel

caso di classi non bilanciate e quindi con frequenze molto diverse, un modello potrebbe ottenere un'accuratezza migliore prevedendo sempre la classe maggioritaria, non essendo però in grado di generalizzare quelle minoritarie. Nonostante tali limitazioni, l'accuratezza è ampiamente utilizzata per la valutazione delle prestazioni dei modelli.

5.1.2 SUDDIVISIONE DATASET

La suddivisione del dataset in set d'addestramento e set di test è una pratica fondamentale in questo ambito. Nel nostro studio, se non specificato, useremo una suddivisione dei dati in: 90% set d'addestramento e 10% set di test.

5.2 ANALISI SVOLTA

Di seguito, verranno riportate le sperimentazioni svolte ed i risultati ottenuti, anche graficamente.

5.2.1 IPERPARAMETRO BATCH_SIZE

Il parametro "batch_size" permette di specificare il numero di campioni di dati utilizzati in un singolo passaggio durante il processo d'addestramento del modello. La scelta dei valori per questo parametro ha un impatto significativo sulle prestazioni complessive del modello: valori troppo elevati possono accelerare il processo d'addestramento, poiché i pesi convoluzionali vengono aggiornati meno frequentemente, ma potrebbero anche causare un aumento del rischio di overfitting, in quanto il modello potrebbe adattarsi troppo ai dati d'addestramento. In questa prima fase di studio, l'obiettivo principale è analizzare le prestazioni del modello variando tale parametro, assegnandogli i valori: 8, 16, 32. Per automatizzare il processo di selezione del miglior valore da assegnare al parametro, useremo la funzionalità di "GridSearch". Questo processo può essere computazionalmente complicato e oneroso in termini di tempo, ma risulta essere efficace nella ricerca dei migliori parametri. Come primo passo, viene definita la classe "create_model()", all'interno della quale implementiamo l'architettura del modello che intendiamo analizzare, ossia una Convolutional Neural Network (CNN) con un'architettura a strati.

Per ogni dataset, successivamente, viene eseguita l'attività di normalizzazione e suddivisione dei dati, seguendo la procedura precedentemente introdotta. Dopodiché, viene creato il modello richiamando la funzione definita in precedenza "create_model()",

utilizzando la classe “KerasClassifier” fornita dalla libreria di Python ‘Keras’. L’utilizzo di “KerasClassifier” ci permetterà poi di utilizzare “GridSearch”.

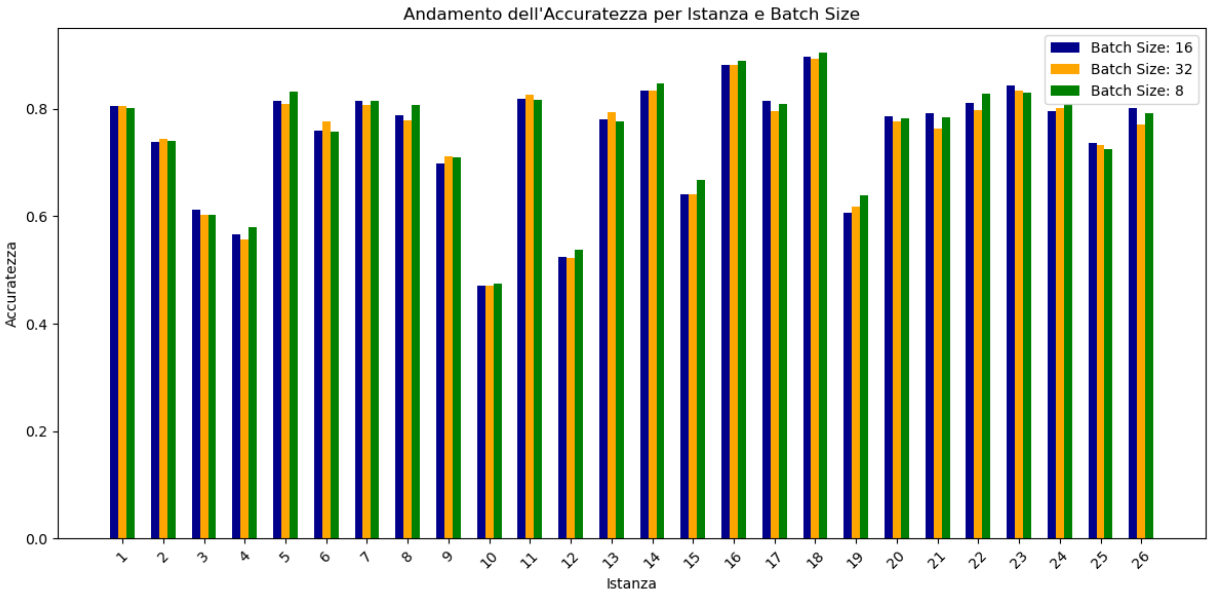

Fig.4: Risultati per ogni dataset dell’accuratezza calcolata con i diversi parametri di “batch_size”.

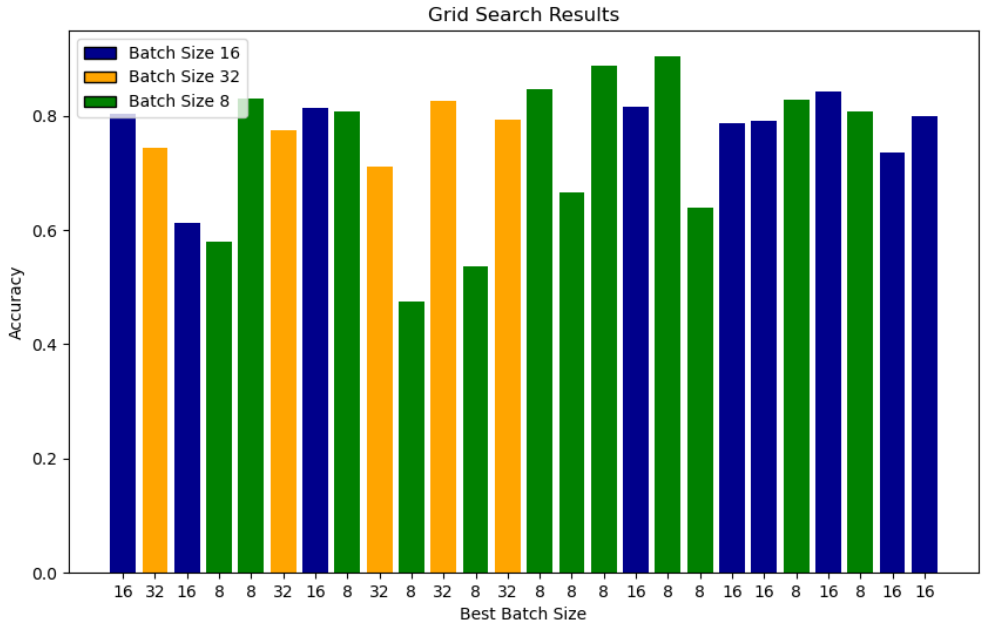

Fig.5: Vediamo, per ogni istanza, quale “batch_size” è risultato essere il migliore

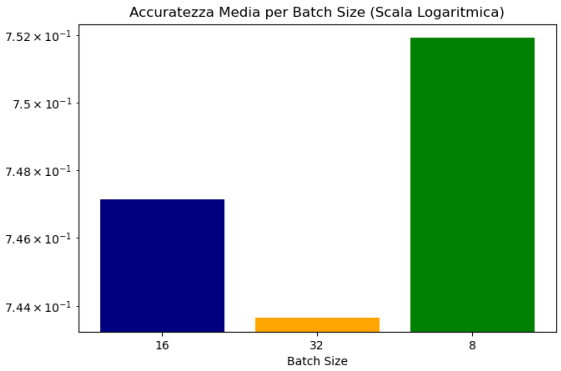

Fig.6: Numero di volte in cui ogni “batch_size” risulta essere il migliore .

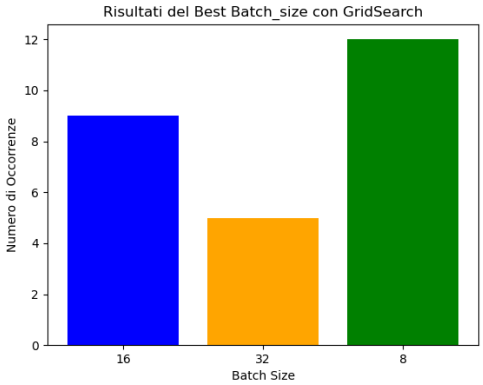

Fig.7: Accuratezza media per ogni valore di è “batch_size”, visualizzato su scala logaritmica.

Analizzando i risultati, considerando i tre valori di accuratezza ottenuti dai diversi valori di “batch_size” per ogni istanza, si nota come le accuratezze ottenute siano diverse ma rimanendo molto simili. Tuttavia, esaminando ulteriormente il tutto, emerge chiaramente che il valore di “batch_size” pari a 8 risulta essere associato alla migliore accuratezza in un numero maggiore di dataset rispetto agli altri valori testati, per tale motivo nei seguenti studi verrà utilizzato tale valore.

5.2.2 OTTIMIZZATORI

Avendo determinato come parametro ottimo della terna “batch_size” pari a 8, lo step successivo di questo studio è valutare le prestazioni del modello cambiando ottimizzatore del modello, passando da SGD a Adam. Di seguito, i risultati ottenuti utilizzando l’ottimizzatore Adam, usando lo stesso valore di “learning_rate” adoperato in precedenza con l’applicazione dell’ottimizzatore SGD.

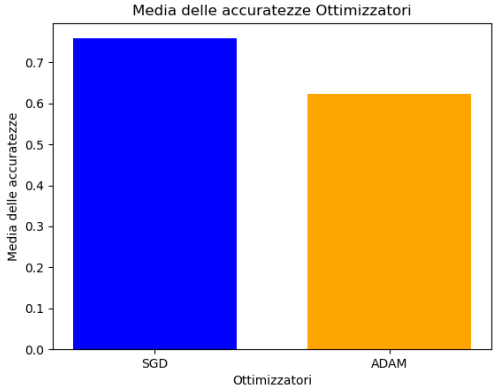

Fig.8: Accuratezza media per ottimizzatore

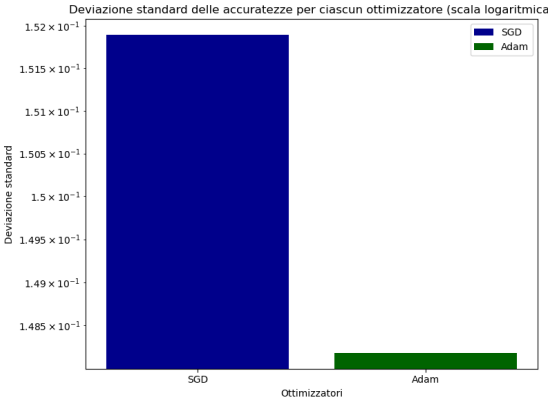

Fig.9: Deviazione standard delle accuratezze calcolate con ciascun ottimizzatore

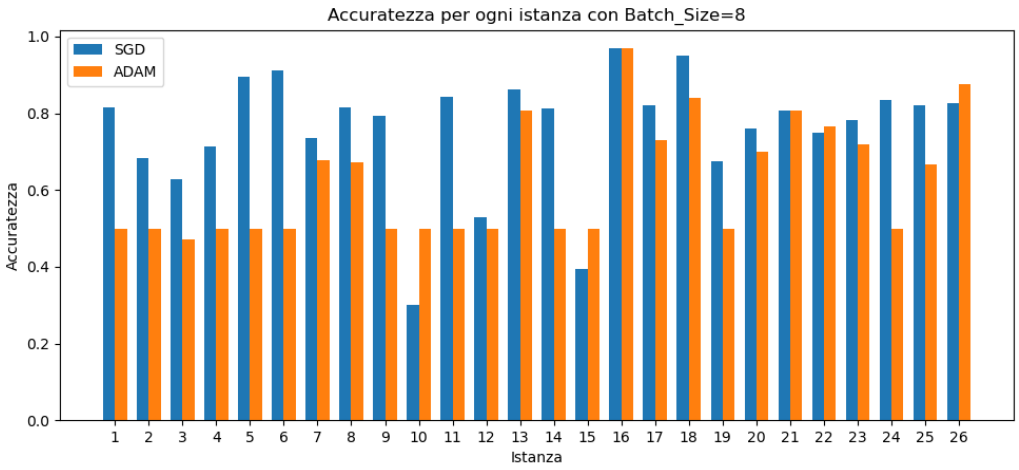

Fig.10: Accuratezza ottenuta con i due ottimizzatori per ogni istanza.

Dall'analisi della Figura 10, che mostra i valori delle accuratezze ottenute utilizzando entrambi gli ottimizzatori applicati al modello, emerge una tendenza generale: l'accuratezza calcolata utilizzando l'ottimizzatore SGD tende ad essere superiore. Tale osservazione trova conferma anche nell'analisi della Figura 8, in cui è riportata la media delle accuratezze di ogni ottimizzatore considerando i 26 dataset.

Tuttavia, analizzando i risultati ottenuti nella Figura 9, che riporta i risultati del calcolo della deviazione standard delle accuratezze per ogni ottimizzatore, rileva una discrepanza. Infatti, notiamo come sia notevolmente più alto il valore della deviazione standard nel caso dell'ottimizzatore SGD, indicando una maggiore variabilità nei risultati dei test sullo stesso

insieme di dati. Per tale motivo procediamo nello studio con l'obiettivo di rendere più stabile l'accuratezza ottenibile con l'ottimizzatore SGD, regolandone gli iperparametri.

5.2.3 LEARNING_RATE E MOMENTUM

Anche in questo caso, utilizzeremo la funzionalità "GridSearch" per la ricerca ottimizzata dei migliori iperparametri. I valori che prenderemo in esame sono i seguenti:

- **Learning_rate: [0.01, 0.005, 0.1]**

Il "**learning_rate"** determina ogni quanto i pesi verranno aggiornati durante la fase di addestramento, tipicamente compreso tra *0.0 e 1.0*. Un "**learning_rate"** troppo elevato può far si che il modello converga troppo rapidamente verso una soluzione non ottimale, mentre un "**learning_rate"** troppo piccolo può causare il blocco del processo. In questo caso vengono scelti per la valutazione:

1) **0.01**: Valore di default dell'iperparametro se non definito esplicitamente.
2) **0.005**: Valore utilizzato nei precedenti studi.
3) **0.1**: Valore moderato che potrebbe portare a prestazioni ottime.

- **Momentum: [0, 0.8, 0.9]**

Il "**momentum"** è utilizzato per velocizzare la convergenza durante l'addestramento di un modello, aggiungendo una frazione della direzione precedente dell'aggiornamento dei pesi al nuovo aggiornamento. Tipicamente assume valori compresi tra *0 e 1*. Un valore più alto indica una maggiore relazione con le direzioni precedenti. In questo caso vengono scelti per la valutazione:

1) **0**: Valore di default dell'iperparametro se non definito esplicitamente.
2) **0.8**: Valore utilizzato nei precedenti studi.
3) **0.9**: Valore comunemente utilizzato.

Di seguito, i risultati.

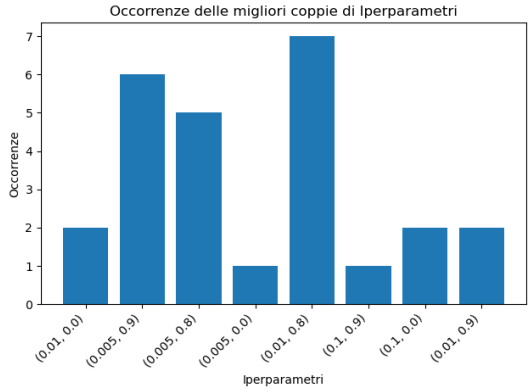

Fig.11: Occorrenze delle coppie coppie.

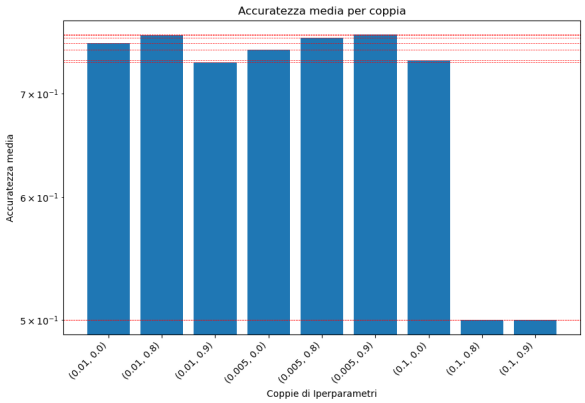

Fig.12: Accuratezza media delle risultate come ottime

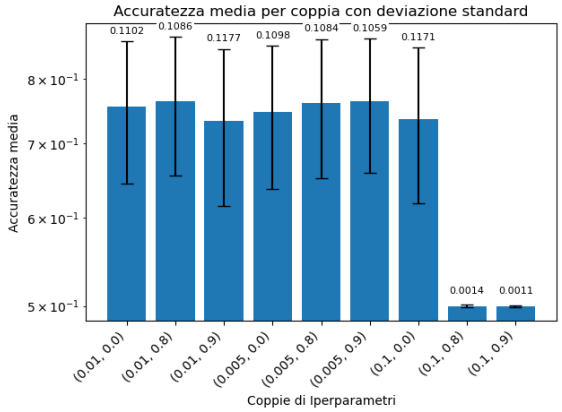

Fig.13: Deviazione standard calcolo dell'accuratezza media per coppia

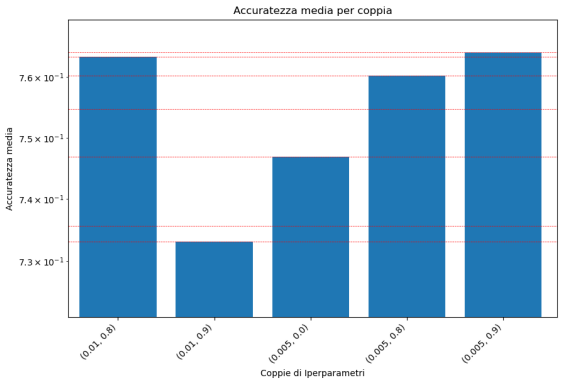

Fig.14: Accuratezza media delle coppie vista nel dettaglio

Otteniamo come coppia ottima i valori: "**learning_rate" pari a 0.01 e "momentum" pari a 0.8.** Osservando l'accuratezza media e la deviazione standard delle coppie dalle Figure 11, 12 e 13, possiamo notare come le prestazioni della coppia [0.01, 0.8] siano molto simili, se non peggiori, a quelle risultanti dall'utilizzo della coppia [0.005, 0.9], ma in maniera trascurabile. Viene quindi scelta, visti i risultati di "GridSearch", [0.01, 0.8] come coppia di valori ottimi per effettuare i test seguenti.

È possibile constatare come tali valori, effettivamente, rendano più stabili le misurazioni effettuate utilizzando l'ottimizzatore SGD, tramite i seguenti risultati.

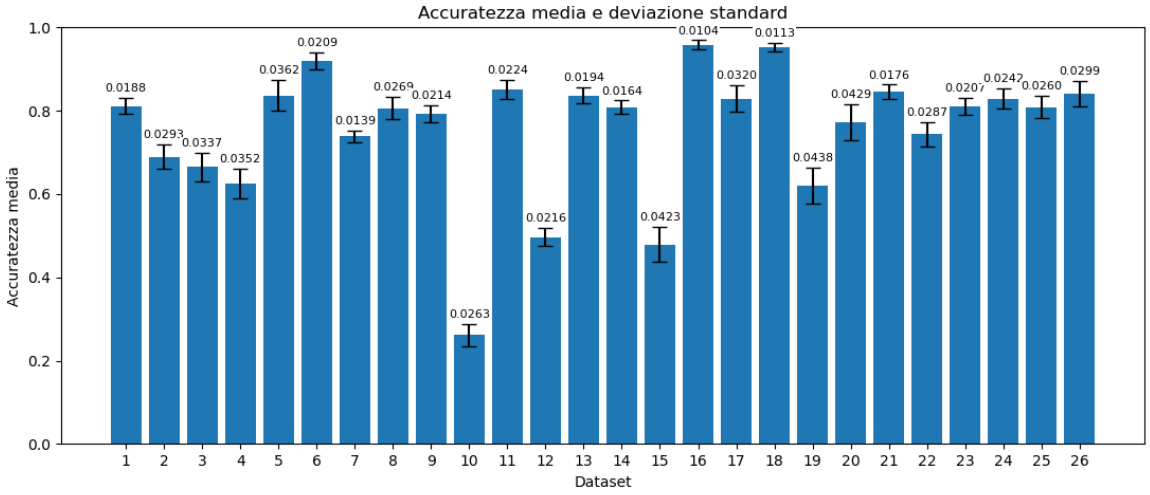

Fig.15: Accuratezza media e deviazione standard utilizzando SGD con iperparametri Ottimi

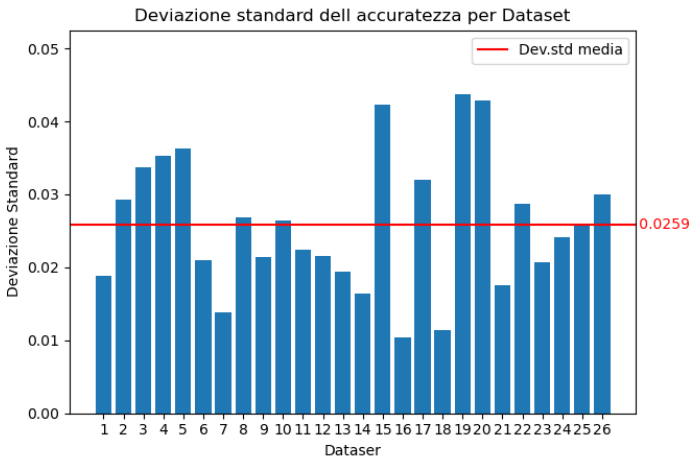

Fig.16: Deviazione standard media e per dataset

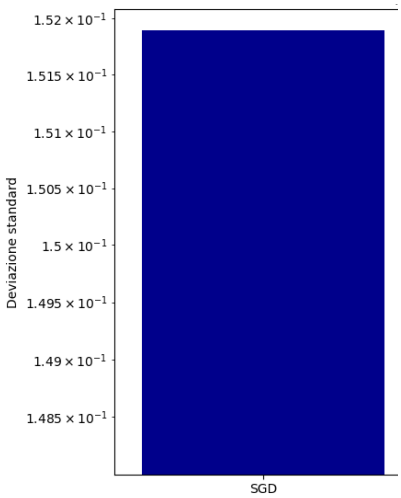

Fig.17: Deviazione standard dei calcoli dell'accuratezza senza iperparametri ottimi effettuata nello studio precedente

5.2.4 SETTING SPERIMENTALE LOSO

Nello studio svolto fino a questo momento, l'addestramento e la valutazione del modello venivano effettuati considerando i dati di un singolo soggetto individualmente, suddividendo in set d'addestramento e set di test i dati relativi allo stesso.

A questo punto, volendo validare il modello tenendo conto della variabilità individuale dei soggetti, decidiamo di utilizzare un nuovo setting sperimentale: Leave-One-Subject-Out (o LOSO). In un setting LOSO, l'idea fondamentale è quella di valutare le prestazioni del modello su un soggetto alla volta, escludendo quel soggetto dal set di addestramento e utilizzandolo come set di test. Quindi, per ogni soggetto nel dataset, viene addestrato un modello utilizzando tutti gli altri soggetti tranne quello in questione, e il modello viene quindi valutato sui dati del soggetto escluso. Per poter realizzare tale sperimentazione, è stato necessario un ridimensionamento dei dati, effettuato tramite l'operazione di interpolazione che evita la perdita di qualsiasi informazione.

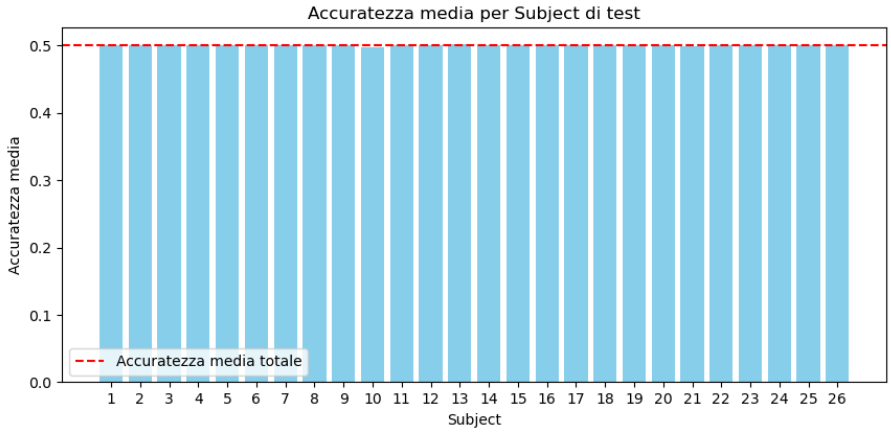

Fig.18: Accuratezza media per Subject. Il Subject considerato è quello utilizzato come set di test durante lo svolgimento della sperimentazione.

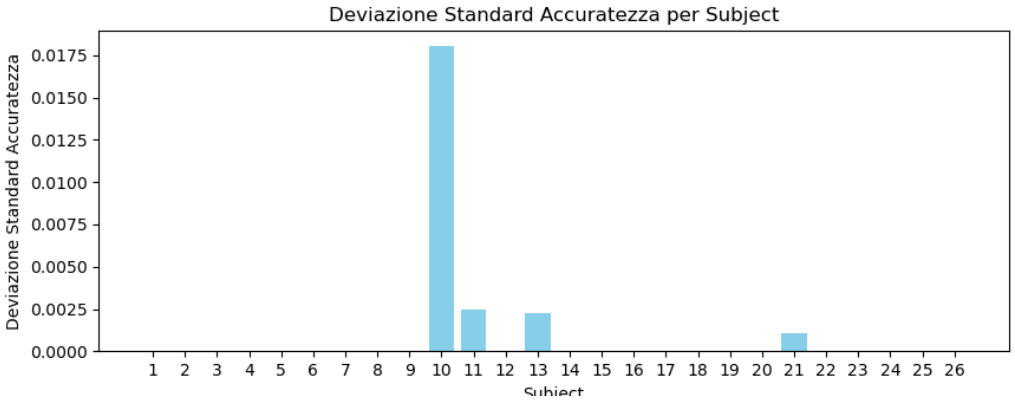

Fig.19: Deviazione Standard del calcolo dell'accuratezza per Subject. Il Subject considerato è quello utilizzato come set di test durante lo svolgimento della sperimentazione.

Essendo un'accuratezza del 50% interpretabile come un risultato casuale o equiprobabile, si può affermare che, utilizzando il setting sperimentale LOSO, la capacità del modello di effettuare previsioni affidabili, e quindi le sue prestazioni, sono risultate scarse ed inferiori alle prestazioni ottenute utilizzando i metodi precedenti.

CAPITOLO 6

CONCLUSIONI

Nel corso di questo studio dedicato al riconoscimento delle attività umane (HAR), abbiamo esplorato l'efficacia delle Reti Neurali Convoluzionali (CNN) nell'affrontare questa sfida complessa. Utilizzando approcci sperimentali rigorosi, abbiamo investigato su una serie di fattori chiave, tra cui gli iperparametri, gli ottimizzatori e i setting sperimentali, al fine di massimizzare le prestazioni del nostro modello. La nostra valutazione si è concentrata sull'accuratezza come metrica principale, che rappresenta la capacità del modello di classificare correttamente le attività umane. Le conclusioni che emergono da questo studio forniscono preziose informazioni sulle migliori pratiche e strategie per affrontare efficacemente il problema dell'HAR utilizzando CNN.

Percorrendo le vari fasi della nostra sperimentazione, possiamo concludere:

- Attraverso l'analisi dei migliori iperparametri, abbiamo identificato le configurazioni ottimali per la nostra CNN, tenendo conto di fattori come il tasso d'apprendimento, settings sperimentali e altre caratteristiche chiave. Questo ci ha permesso di ottenere un equilibrio tra complessità del modello e capacità di generalizzazione, migliorando significativamente le prestazioni complessive del sistema di HAR.
- Abbiamo confrontato due diversi ottimizzatori, l'SGD (Stochastic Gradient Descent) e l'ADAM, valutandone l'impatto sul processo di addestramento e sulle prestazioni del modello. I risultati indicano che l'ADAM ha dimostrato di essere particolarmente efficace nel convergere rapidamente e con risultati stabili ma scarsi. Per questo abbiamo deciso di procedere con la stabilizzazione di SGD, che ci ha permesso di ottenere risultati ottimali.
- Utilizzando il setting sperimentale LOSO (Leave-One-Subject-Out), abbiamo esaminato come la variabilità intra-soggetto influenzi le prestazioni del modello. I

risultati suggeriscono che il LOSO risulta, nel nostro caso, essere un setting sperimentale più debole del precedente per quanto riguarda la generalizzazione.

In particolare, l'HVR fornisce informazioni cruciali per comprendere i fenomeni fisiologici. La sua ricostruzione tramite l'analisi dell'EEG rappresenta un'importante innovazione nel campo medico, ancora in fase di sviluppo, per lo studio dell'interazione funzionale tra cervello e cuore.

Complessivamente, l'uso di CNN ottimizzate e configurate correttamente ha portato a significativi miglioramenti nelle prestazioni del sistema di HAR. Grazie all'approccio sperimentale rigoroso e alla selezione oculata dei parametri, siamo riusciti a ottenere un modello altamente accurato e affidabile per il riconoscimento delle attività umane, con importanti implicazioni per applicazioni reali nel campo della salute, della sicurezza e del benessere.

APPENDICE A

Quest'appendice contiene il codice prodotto durante l'implementazione dello studio svolto.

1. CODICE PRINCIPALE

Di seguito il codice su cui è stato effettuato il test dei diversi valori di iperparametri.

```
import time
import numpy as np
import pandas as pd
from glob import glob
import tensorflow as tf
from keras.optimizers import SGD
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
BatchNormalization, ReLU, Dropout
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.wrappers.scikit_learn import KerasClassifier
import warnings

# Disable all warnings
warnings.filterwarnings("ignore")

base_path = '/Users/rossana/Desktop/tesi/'

for i in range(1, 27):  #numero di file csv che dobbiamo valutare
    for j in range(1, 10): #10 prove ciascuno
        file_name = glob(base_path + 'data/subject_' + str(i) +
'_MADts_*.csv')[0]
        X = pd.read_csv(file_name).values

        # Normalizzazione e split
        scaler = StandardScaler()
        X = scaler.fit_transform(X)

        # Dimensioni dell'input e reshape dei dati per la CNN 1D, la
prima meta' dei samples sono peak
        # e il resto no
        n_series = int(file_name[-6:-4])  # Numero di serie temporali per
gruppo
        n_samples = int(file_name[-10:-7])  # Numero di gruppi (samples
peak e no-peak)
        n_points = int(file_name[-20:-17])  # Numero di punti temporali
per serie

        X = X.reshape(n_samples, n_points, n_series)
        y = np.array([1] * (n_samples // 2) + [0] * (n_samples //
2)).astype(int)

        test_idx_end = int((n_samples // 2) + int(n_samples * 0.1))
        test_idx_begin = int((n_samples // 2) - int(n_samples * 0.1))
```

```
        # Create a boolean mask to select elements to keep for training
and testing sets
        mask_train = np.ones(X.shape[0], dtype=bool)
        mask_train[test_idx_begin:test_idx_end] = False
        X_train = X[mask_train, :, :]
        y_train = y[mask_train]

        print(y_train.shape)


        mask_test = np.zeros(X.shape[0], dtype=bool)
        mask_test[test_idx_begin:test_idx_end] = True
        X_test = X[mask_test, :, :]
        y_test = y[mask_test]

        print(y_test.shape)


    # Costruzione del modello
        model = Sequential()
        model.add(Conv1D(filters=64, kernel_size=10, activation='relu',
input_shape=(n_points, n_series))) #64 filtri lunghi 10 elementi
        model.add(MaxPooling1D(pool_size=3))
        model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))

        sgd = SGD(learning_rate=0.01, momentum=0.8)
        model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])

    # Training e testing
        model.fit(X_train, y_train, epochs=150, batch_size=8,
validation_data=(X_test, y_test), verbose=0)

        loss, accuracy = model.evaluate(X_test, y_test)
        res = 'Subject: ' + str(i) + ', accuracy: ' + str(accuracy) +
'\n'

        print(res)

        with open(base_path + "results/performanceTS_iperSGD_%s.txt" %
(time.strftime("%Y%m%d")), "a",
                  encoding="utf-8") as file_object:
            file_object.write(res)
            file_object.close()
```

2. CODICE PER L'UTILIZZO DI GRIDSEARCH

```
import time
import numpy as np
import pandas as pd
from glob import glob
import tensorflow as tf
from keras.optimizers import SGD
```

```
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
BatchNormalization, ReLU, Dropout
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from scikeras.wrappers import KerasClassifier
import warnings

# Disable all warnings
warnings.filterwarnings("ignore")

base_path = '/Users/rossana/Desktop/tesi/'

param_grid = {
        'model__learning_rate': [0.01, 0.005, 0.1],
        'model__momentum': [0, 0.8, 0.9]
    }


def create_model(learning_rate, momentum):
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=10, activation='relu',
input_shape=(n_points, n_series))) #64 filtri lunghi 10 elementi
    model.add(MaxPooling1D(pool_size=3))
    model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    sgd = SGD(learning_rate=learning_rate, momentum=momentum)
    model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])
    return model


for i in range(1, 27):  #numero di file csv che dobbiamo valutare
    file_name = glob(base_path + 'data/subject_' + str(i) +
'_MADts_*.csv')[0]
    X = pd.read_csv(file_name).values

    # Normalizzazione e split
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    # Dimensioni dell'input e reshape dei dati per la CNN 1D,
considera che la prima meta' dei samples sono peak
    # e il resto no
    n_series = int(file_name[-6:-4])  # Numero di serie temporali per
gruppo
    n_samples = int(file_name[-10:-7])  # Numero di gruppi (samples
peak e no-peak)
    n_points = int(file_name[-20:-17])  # Numero di punti temporali
per serie

    X = X.reshape(n_samples, n_points, n_series)
    y = np.array([1] * (n_samples // 2) + [0] * (n_samples //
2)).astype(int)
```

```
    test_idx_end = int((n_samples // 2) + int(n_samples * 0.1))
    test_idx_begin = int((n_samples // 2) - int(n_samples * 0.1))

    # Create a boolean mask to select elements to keep for training
and testing sets
    mask_train = np.ones(X.shape[0], dtype=bool)
    mask_train[test_idx_begin:test_idx_end] = False
    X_train = X[mask_train, :, :]
    y_train = y[mask_train]

    mask_test = np.zeros(X.shape[0], dtype=bool)
    mask_test[test_idx_begin:test_idx_end] = True
    X_test = X[mask_test, :, :]
    y_test = y[mask_test]

    # Costruzione del modello

    model1 = KerasClassifier(model=create_model, epochs=150,
batch_size=8, verbose=0)

    grid = GridSearchCV(estimator=model1, param_grid=param_grid,
n_jobs=-1, cv=3)
    grid_result = grid.fit(X_train, y_train)

    print("Best_%d: %f using %s" % (i, grid_result.best_score_,
grid_result.best_params_))
    means = grid_result.cv_results_['mean_test_score']
    stds = grid_result.cv_results_['std_test_score']
    params = grid_result.cv_results_['params']
    for mean, stdev, param in zip(means, stds, params):
        print("%f (%f) with: %r" % (mean, stdev, param))

    with open(base_path + "results/gridSearch_IperSGD_2BS_%s.txt" %
(time.strftime("%Y%m%d")), "a",
        encoding="utf-8") as file_object:
        file_object.write("\nBest_%d: %f using %s\n" % (i,
grid_result.best_score_, grid_result.best_params_))
        # Stampa i punteggi medi, gli scarti standard e i parametri
per ciascuna configurazione
        for mean, stdev, param in zip(means, stds, params):
            file_object.write("%f (%f) with: %r\n" % (mean, stdev,
param))
```

3. CODICE PER SETTING LOSO

```
import time
import numpy as np
import pandas as pd
from glob import glob
import tensorflow as tf
from keras.optimizers import SGD
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
BatchNormalization, ReLU, Dropout
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
from keras.wrappers.scikit_learn import KerasClassifier
from scipy.interpolate import interp1d
import warnings
import math

# Disable all warnings
warnings.filterwarnings("ignore")

base_path = '/Users/rossana/Desktop/tesi/'

def ridimensiona_righe(data, new_length):
    # Crea un array di valori x
    x = np.arange(len(data))

    # Calcola la nuova lunghezza dell'array x
    new_x = np.linspace(0, len(data) - 1, new_length)

    # Interpolazione dei dati lungo l'asse delle x
    f = interp1d(x, data, axis=0)

    # Valori interpolati per la nuova lunghezza
    new_data = f(new_x)

    return new_data


n_series = 200

for n in range(1, 27): #mettilo globale e vedi se funziona
    #trovo i canali minimi
    file_name = glob(base_path + 'data/subject_' + str(n) +
'_MADts_*.csv')[0]
    n_series = n_series if int(file_name[-6:-4]) > n_series else
int(file_name[-6:-4])# Numero di serie temporali per gruppo**


def LOSO_function(escluso):

    n_points_glb = 150
    scaler = StandardScaler()

    train_data = np.empty((0, n_points_glb, n_series), dtype=float)
    test_data = np.empty((0, n_points_glb, n_series), dtype=float)

    label_train = np.empty((0,), dtype=int)
    label_test = np.empty((0,), dtype=int)

    for k in range(1, 27):

        file_name = glob(base_path + 'data/subject_' + str(k) +
'_MADts_*.csv')[0]

        data = pd.read_csv(file_name).values

        n_samples = int(file_name[-10:-7])

        dim_des = n_samples * n_series
        data = ridimensiona_righe(data, dim_des)
        data = scaler.fit_transform(data)
        data = data.reshape(n_samples, n_points_glb, n_series)
```

```
        label = np.array([1] * (n_samples // 2) + [0] * (n_samples //
2))

        if k != escluso:
            label_train = np.concatenate((label_train, label))
            train_data = np.vstack((train_data, data))
        else:
            label_test = label
            test_data = data

    return train_data, test_data, label_train, label_test,
n_points_glb, n_series


for i in range(1, 27):  #numero di file csv che dobbiamo valutare, qui
i sarà l'escluso
    for j in range(1, 11): #10 prove ciascuno

        X_train, X_test, y_train, y_test, n_points, n_series =
LOSO_function(i)

    # Costruzione del modello
        model = Sequential()
        model.add(Conv1D(filters=64, kernel_size=10,
activation='relu', input_shape=(n_points, n_series))) #64 filtri
lunghi 10 elementi
        model.add(MaxPooling1D(pool_size=3))
        model.add(Conv1D(filters=32, kernel_size=3,
activation='relu'))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    #sgd = SGD(learning_rate=0.005, momentum=0.8)
    #adam_opt = Adam(learning_rate=0.005)
        sgd = SGD(learning_rate=0.01, momentum=0.8)
        model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])

    # Training e testing
        model.fit(X_train, y_train, epochs=150, batch_size=8,
validation_data=(X_test, y_test), verbose=0)

        loss, accuracy = model.evaluate(X_test, y_test)
        res = 'Subject_test: ' + str(i) + ', accuracy: ' +
str(accuracy) + '\n'

        print(res)

        with open(base_path + "results/performanceTS_LOSO_%s.txt" %
(time.strftime("%Y%m%d")), "a",
                  encoding="utf-8") as file_object:
            file_object.write(res)
            file_object.close()
```

4. CODICE PER FINE-TUNING

```
import time
import numpy as np
import pandas as pd
from glob import glob
import tensorflow as tf
from keras.optimizers import SGD
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
BatchNormalization, ReLU, Dropout
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.wrappers.scikit_learn import KerasClassifier
from scipy.interpolate import interp1d
import warnings
import math

# Disable all warnings
warnings.filterwarnings("ignore")

base_path = '/Users/rossana/Desktop/tesi/'

def ridimensiona_righe(data, new_length):
    # Crea un array di valori x
    x = np.arange(len(data))

    # Calcola la nuova lunghezza dell'array x
    new_x = np.linspace(0, len(data) - 1, new_length)

    # Interpolazione dei dati lungo l'asse delle x
    f = interp1d(x, data, axis=0)

    # Valori interpolati per la nuova lunghezza
    new_data = f(new_x)

    return new_data


n_series = 200

for n in range(1, 27):
    #trovo i canali minimi
    file_name = glob(base_path + 'data/subject_' + str(n) +
'_MADts_*.csv')[0]
    n_series = n_series if int(file_name[-6:-4]) > n_series else
int(file_name[-6:-4])# Numero di serie temporali per gruppo**


def LOSO_function(escluso):

    n_points_glb = 150
    scaler = StandardScaler()

    train_data = np.empty((0, n_points_glb, n_series), dtype=float)
    test_data = np.empty((0, n_points_glb, n_series), dtype=float)

    label_train = np.empty((0,), dtype=int)
    label_test = np.empty((0,), dtype=int)
```

```
    for k in range(1, 27):

        file_name = glob(base_path + 'data/subject_' + str(k) +
'_MADts_*.csv')[0]

        data = pd.read_csv(file_name).values

        n_samples = int(file_name[-10:-7])

        dim_des = n_samples * n_series
        data = ridimensiona_righe(data, dim_des)
        data = scaler.fit_transform(data)
        data = data.reshape(n_samples, n_points_glb, n_series)

        label = np.array([1] * (n_samples // 2) + [0] * (n_samples //
2)).astype(int)

        if k != escluso:
            label_train = np.concatenate((label_train, label))
            train_data = np.vstack((train_data, data))
        else:
            label_test = label
            test_data = data

    return train_data, test_data, label_train, label_test,
n_points_glb, n_series


for i in range(1, 27):  #numero di file csv che dobbiamo valutare, qui
i sarà l'escluso
    for j in range(1, 11): #10 prove ciascuno

        X_train, X_test, y_train, y_test, n_points, n_series =
LOSO_function(i)

    # Costruzione del modello
        model = Sequential()
        model.add(Conv1D(filters=64, kernel_size=10,activation='relu',
input_shape=(n_points, n_series))) #64 filtri lunghi 10 elementi
        model.add(MaxPooling1D(pool_size=3))
        model.add(Conv1D(filters=32, kernel_size=3,
activation='relu'))
        model.add(MaxPooling1D(pool_size=2))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    #sgd = SGD(learning_rate=0.005, momentum=0.8)
    #adam_opt = Adam(learning_rate=0.005)
        for layer in model.layers:
            layer.trainable = True

        sgd = SGD(learning_rate=0.01, momentum=0.8)
        model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])

    # Training e testing
        model.fit(X_train, y_train, epochs=150, batch_size=8,
validation_data=(X_test, y_test), verbose=0)
```

```
    #da qui in poi FINETUNING
        file_name = glob(base_path + 'data/subject_' + str(i) +
'_MADts_*.csv')[0]
        n_samples = int(file_name[-10:-7])

        test_idx_end = int((n_samples // 2) + int(n_samples * 0.20))
        test_idx_begin = int((n_samples // 2) - int(n_samples * 0.20))

        mask_train = np.ones(X_test.shape[0], dtype=bool)
        mask_train[test_idx_begin:test_idx_end] = False
        X_FT_Train = X_test[mask_train, :, :]
        y_FT_Train = y_test[mask_train]

        mask_test = np.zeros(X_test.shape[0], dtype=bool)
        mask_test[test_idx_begin:test_idx_end] = True
        X_FT_Test = X_test[mask_test, :, :]
        y_FT_Test = y_test[mask_test]


        #CONGELO TUTTI I LAYER TRANNE 6 E 7
        for layer in model.layers[:5]:
            layer.trainable = False

        sgd = SGD(learning_rate=0.01, momentum=0.8)
        model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])

        model.fit(X_FT_Train, y_FT_Train, epochs=10, batch_size=8,
validation_data=(X_FT_Test, y_FT_Test), verbose=0)
        #VEDO LE PERFORMANCE

        loss, accuracy = model.evaluate(X_FT_Test, y_FT_Test)

        res = 'Subject_test: ' + str(i) + ', accuracy: ' +
str(accuracy) + '\n'

        print(res)

        with open(base_path +
"results/performanceTS_LOSO_FineTuning_20%.txt", "a",
                  encoding="utf-8") as file_object:
            file_object.write(res)
            file_object.close()
```

BIBLIOGRAFIA

[1] Binh Nguyen, Yves Coelho, Teodiano Bastos, Sridhar Krishnan. Trends in human activity recognition with focus on machine learning and power requirements, Machine Learning with Applications, Volume 5, (2021).

[2] Rung-Ching Chen, Christine Dewi, Su-Wen Huang, Rezzy Eko Caraka. Selecting critical features for data classification based on machine learning methods, Volume 7, article number 52, (2020).

[3] Farida Sabry , Tamer Eltaras , Wadha Labda , Khawla Alzoubi , Qutaibah Malluhi. Machine Learning for Healthcare Wearable Devices: The Big Picture (2021).

[4] Diego Candia-Rivera, Vincenzo Catrambone, Riccardo Barbieri, Gaetano Valenza, Functional assessment of bidirectional cortical and peripheral neural control on heartbeat dynamics: A brain-heart study on thermal stress, NeuroImage, Volume 251, (2022).