UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI **INGEGNERIA**

Corso di Laurea triennale in **Ingegneria Elettronica**

Ottimizzazione di una rete neurale ricorrente per la classificazione di pazienti affetti da Alzheimer tramite pre-processamento del segnale EEG

Optimization of a recurrent neural network for Alzheimer's disease classification through EEG signal preprocessing

Relatore: Chiar.ma

Dott.ssa **Falaschetti Laura**

Correlatore:

Dott. **Alessandrini Michele**

Tesi di Laurea di:

Conti Alessia

A.A. **2022** / **2023**

Indice

Indice delle figure

Indice delle tabelle

Introduzione

Tema

La malattia di **Alzheimer** (Alzheimer's Disease, AD), è la forma di demenza più comune nella popolazione anziana: rappresenta il 50-80% dei casi di demenza [1]. Questa malattia è caratterizzata da un processo degenerativo progressivo che distrugge le cellule del cervello, causando un deterioramento irreversibile delle funzioni cognitive (memoria, ragionamento e linguaggio), fino a compromettere l'autonomia e la capacità di compiere le normali attività giornaliere.
Attualmente, il morbo di Alzheimer non è curabile, e generalmente, dal momento della diagnosi**,** l'aspettativa di vita è tra i 3 e i 10 anni**.** Tuttavia, i farmaci attualmente disponibili sono in grado di migliorare i sintomi della malattia e di rallentarne temporaneamente la progressione, soprattutto se il trattamento avviene durante le prime fasi della malattia [2].
Per questo motivo è importante individuarne la condizione iniziale, definita Declino Cognitivo Lieve o MCI (Mild Cognitive Impairment), in cui la malattia è già presente ma in forma lieve. In generale, la presenza di MCI non indica necessariamente la presenza della malattia ma ne costituisce un fattore di rischio. Pertanto, è molto utile trovare un modo per riuscire ad individuare, in una fase precoce, i soggetti affetti da AD, in modo da fornire i trattamenti adeguati nei giusti tempi [3].

Recentemente, c'è stato un crescente interesse della ricerca nell'utilizzo dell'**elettroencefalogramma** (EEG) come strumento di diagnosi non invasivo per le malattie neurodegenerative come l'AD. L'EEG misura l'attività elettrica cerebrale e può identificare anomalie nelle onde cerebrali legate a determinati disturbi.

Questo metodo permette di registrare il potenziale elettrico generato dalle attività fisiologiche dei neuroni, attraverso elettrodi posizionati sulla testa, in corrispondenza di diverse aree della superficie del cervello (Figura 1). Gli elettrodi sono collegati ad un amplificatore che raccoglie gli impulsi elettrici e li invia a un registratore e a un'apparecchiatura in grado di tradurli in un grafico EEG, che avrà la tensione sulle ordinate e il tempo sulle ascisse, cioè presenterà delle oscillazioni variabili nel tempo [4].

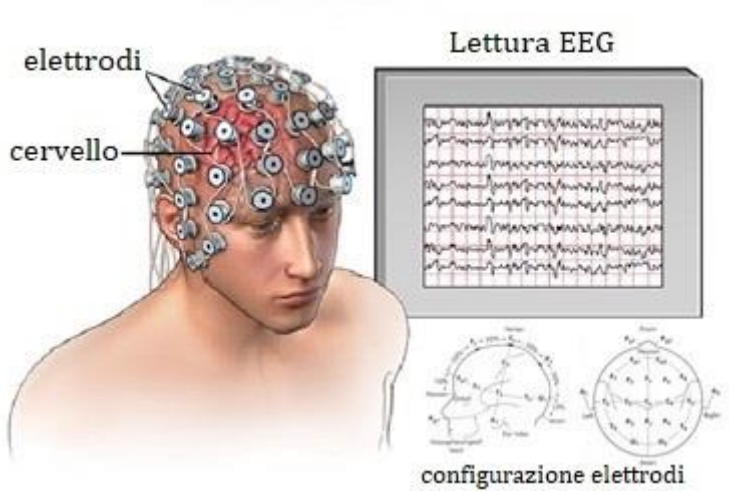

Figura 1: Elettroencefalogramma (EEG)

Questi segnali possono essere analizzati e classificati utilizzando tecniche di apprendimento automatico, come le tecniche di Deep Learning.

Il **Deep Learning** è un sottoinsieme del Machine Learning, e più in generale dell'Intelligenza Artificiale (Figura 2), che è ormai ampiamente diffusa in dispositivi e servizi che utilizziamo quotidianamente. La si trova ad esempio integrata negli smartphone (la fotocamera che riesce a migliorare l'immagine o i sistemi di riconoscimento facciale), negli assistenti vocali come Alexa, nelle auto a guida autonoma, ma anche sulle piattaforme di streaming come Netflix o Spotify, che consigliano cosa guardare in base ai contenuti fruiti in precedenza, oppure sistemi di IA generativa, come il celebre

chatbot ChatGPT, che è in grado di generare un testo, anche complesso, partendo da una breve richiesta da parte dell'utente.
Alla base di queste tecnologie, ci sono le reti neurali: reti di neuroni artificiali interconnessi, che tentano di simulare il funzionamento del cervello umano, permettendo al sistema di "imparare" da grandi quantità di dati e risolvere problemi. A differenza del Machine Learning, il Deep Learning sfrutta delle reti neurali artificiali più complesse per elaborare le informazioni ed eseguire delle attività in totale autonomia. [5]

Gli algoritmi di Deep Learning hanno dimostrato di avere la capacità di estrarre informazioni rilevanti dal segnale EEG chiamate “features” e grazie a queste caratteristiche, sono in grado di distinguere automaticamente i pazienti sani, da quelli affetti da AD, dopo essere state adeguatamente implementate e addestrate.

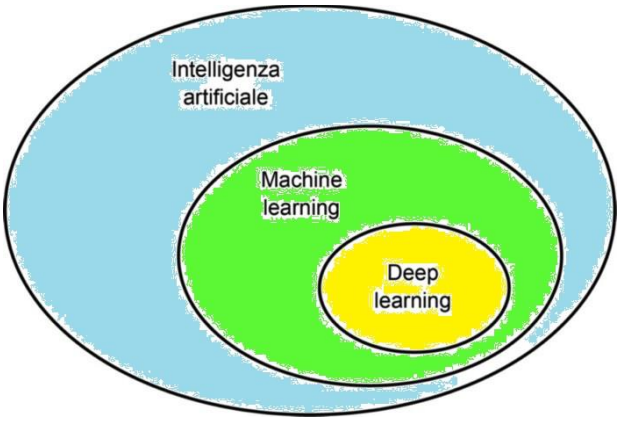

Figura 2: Intelligenza Artificiale, Machine Learning, Deep Learning

Obiettivo

L'obiettivo della tesi è fornire un'analisi delle principali caratteristiche di una rete neurale già implementata e descrivere i metodi utilizzati per migliorare i risultati del modello, in particolare quelli finalizzati al pre-processamento dei segnali EEG.
La rete è stata addestrata a distinguere un paziente sano, da uno malato, utilizzando un dataset ottenuto dalle analisi EEG di 35 pazienti: 20 affetti da Alzheimer, 15 sani. Questi dati provengono dal Dipartimento di Medicina Sperimentale e Clinica di Ancona.
Prima di essere dati in input alla rete, tutti i segnali EEG sono stati convertiti in un formato adeguato, per poi essere pre-elaborati tramite diverse tecniche di riduzione del rumore, con l'obiettivo di rimuovere gli artefatti.

Gli **artefatti** sono parti indesiderate del segnale EEG che non sono state generate dall'attività neuronale del cervello e quindi possono introdurre cambiamenti significativi nei segnali. Essi possono provenire da fonti non fisiologiche, come rumore della linea elettrica o variazioni delle impedenze degli elettrodi, oppure da fonti fisiologiche, come le distorsioni causate da movimenti oculari o muscolari. Pertanto, la necessità di pre-elaborare questi segnali prima di analizzarli è fondamentale. [6]

La tesi è così strutturata.

Nel capitolo 1 verrà illustrata la parte teorica che riguarda le reti neurali, le diverse tipologie e il loro funzionamento.

Nel capitolo 2 verranno analizzati gli strumenti e le diverse tecniche utilizzate per il pre-processamento dei segnali EEG da dare in input alla rete.

Nel capitolo 3 verranno descritti i passaggi dell'algoritmo di classificazione e l'architettura della rete neurale utilizzata.

Nel capitolo 4 verranno messi a confronto i risultati sperimentali ottenuti prima e dopo il pre-processing.

Al termine della tesi, nel capitolo 5, si traggono le conclusioni finali e si ipotizzano delle possibili soluzioni per migliorare la rete in futuri sviluppi.

1. Cenni teorici

1.1 Reti neurali

Le reti neurali sono modelli matematici che cercano di riprodurre il funzionamento biologico del cervello umano, imitando il modo in cui i neuroni biologici si inviano segnali. Una rete neurale artificiale, infatti, è costituita da un insieme di unità collegate tra loro: le unità emulano i neuroni, mentre le connessioni emulano le giunzioni sinaptiche tra i neuroni, come in Figura 3. Neuroni biologici interconnessi formano le nostre reti neurali cerebrali, quelle che permettono a ciascun individuo di ragionare, fare calcoli in parallelo, riconoscere suoni, immagini, volti, imparare e agire secondo uno stato cosciente [7].
L'idea di poter replicare artificialmente il cervello umano, simulandone il funzionamento attraverso delle reti neurali artificiali, ha una storia che inizia da lontano: il primo neurone artificiale fu proposto da McCulloch e Pitts in un famoso lavoro del 1943 intitolato "*A Logical Calculus of the Ideas Immanent in Nervous Activity*" [8].

Una rete neurale, quindi, è costituita da neuroni artificiali interconnessi che ricevono degli stimoli (input) i quali si propagano nella rete che elabora l'informazione e produce dei risultati (output) più complessi. Essa può apprendere dai dati che le vengono forniti, e viene addestrata su molti esempi. In questo modo acquisisce un'esperienza che le consente di rispondere in modo corretto anche quando si troverà davanti dei dati totalmente nuovi e sarà in grado, per esempio, di riconoscere i pattern del parlato o delle immagini, proprio come fa il cervello umano, oppure classificare i dati e prevedere eventi futuri.

Il suo comportamento è definito dal modo in cui sono collegati i suoi singoli elementi e dalla forza, cioè dai pesi (weights), di quelle connessioni. I pesi determinano l'importanza da assegnare ad ogni nodo. Pesi maggiori indicano che quelle specifiche variabili, hanno un'importanza maggiore per il risultato finale. Questi pesi, insieme ad un'altra variabile chiamata bias, vengono regolati automaticamente durante l'addestramento, con una precisa regola di apprendimento, finché la rete neurale non esegue correttamente l'attività desiderata. Per questo si dice che la rete neurale è un sistema "adattivo", in grado, cioè, di generalizzare e modellare un problema nel mondo reale proprio grazie al costante aggiustamento di pesi e bias, che modulano l'output e l'input di ogni singolo neurone, finché la rete non approccia ad una soluzione accettabile [9].

In una rete neurale è importante distinguere i parametri dagli iperparametri.
I **parametri** sono i coefficienti del modello, scelti dal modello stesso: l'algoritmo, durante l'apprendimento, ottimizza questi coefficienti per minimizzare l'errore, come fa ad esempio con pesi e bias.
Gli **iperparametri** sono elementi che, a differenza dei precedenti, è necessario impostare: il modello non li aggiornerà secondo una strategia di ottimizzazione, ma sarà sempre necessario l'intervento manuale del programmatore [10]. Questi ultimi saranno nominati più avanti quando verrà analizzata l'architettura della rete neurale utilizzata.

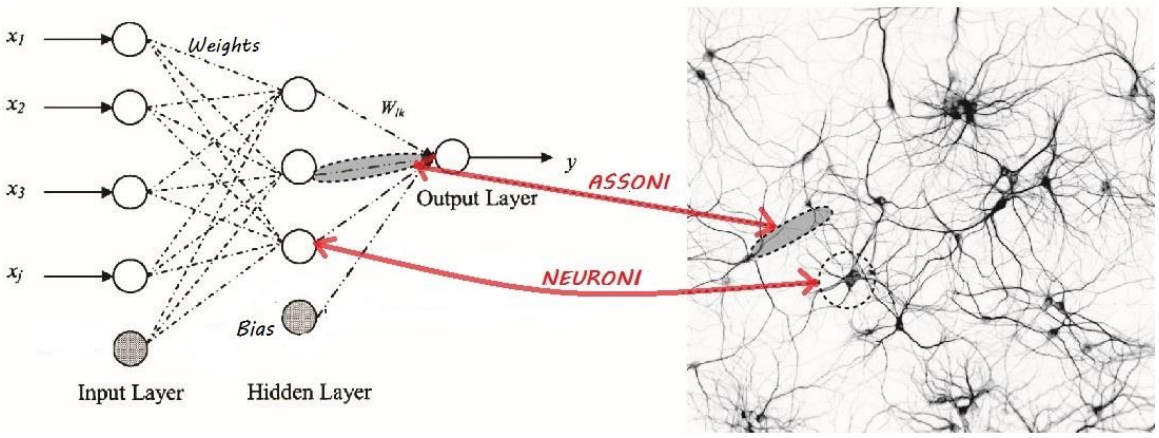

Figura 3: Confronto tra una rete neurale artificiale e una biologica

1.1.1 Percettrone

La rete neurale più semplice è formata da un singolo nodo (un solo neurone) ed è chiamata Percettrone (Figura 4).

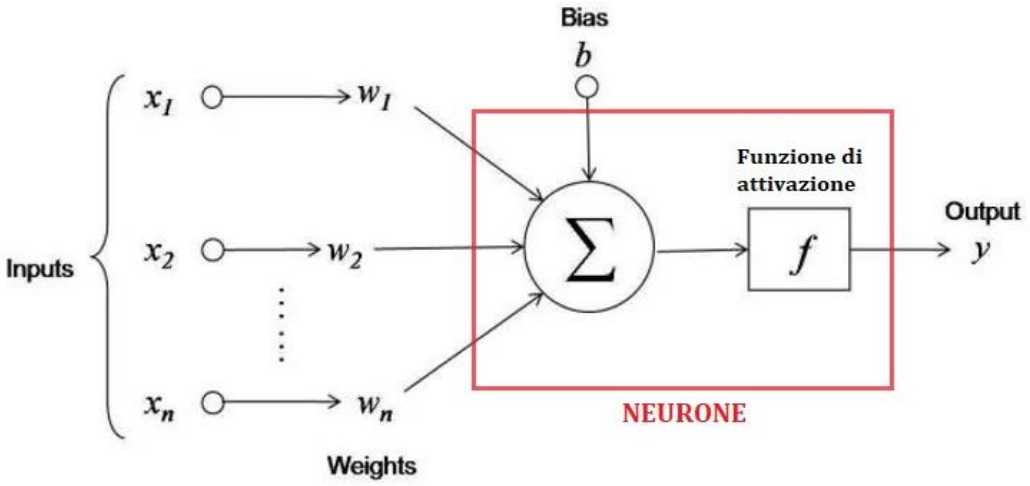

Figura 4: Percettrone

Le operazioni eseguite da ciascun neurone sono le seguenti:

- Innanzitutto, somma il valore di ogni neurone della colonna precedente a cui è connesso. Nella Figura 4, ci sono **n input** (x_1, x_2, … x_n) che arrivano al neurone, quindi n neuroni della colonna precedente sono collegati al nostro neurone.
- Tale valore viene moltiplicato per un'altra variabile denominata "**peso**" (w_1, w_2, … w_n) che determina la connessione tra i due neuroni. Ogni connessione di neuroni ha il suo peso, e questi sono gli unici valori che verranno modificati durante il processo di apprendimento.
- I prodotti così ottenuti vengono sommati tra loro.
- Inoltre, al valore totale calcolato può essere aggiunto un valore di **bias** (b). Non è un valore proveniente da un neurone specifico e viene scelto prima della fase di apprendimento, ma può essere utile per la rete.

- Il neurone applica infine al valore ottenuto una funzione, chiamata "**funzione di attivazione**" (f), che determina se un neurone deve essere attivato o meno, utilizzando alcune semplici operazioni matematiche per stabilire se l'input del neurone alla rete è rilevante o meno nel processo di previsione. Ci sono diverse funzioni di attivazione: la più semplice è la Step Function (funzione gradino), ma solitamente si utilizzano funzioni più complesse, come ReLU e SoftMax [11].

L'**output** della rete neurale, quindi, sarà

$$y = f\left\{ \sum_{i=1}^{n} (x_i \cdot w_i) \; + b \right\}$$

1.1.2 Percettrone multistrato

Il modello visto finora è un modello molto semplice. L'evoluzione del Percettrone è la Rete Multistrato, chiamata anche MLP, Multi-Layer Perceptron, rappresentata in Figura 5 [12].
Queste reti neurali sono formate da tre strati (che però possono coinvolgere migliaia di neuroni e decine di migliaia di connessioni):

- Lo strato degli ingressi (**Input Layer**): è quello che ha il compito di ricevere ed elaborare i segnali in ingresso adattandoli alle richieste dei neuroni della rete;
- Uno o più strati nascosti (**Hidden Layers**): gli strati presenti tra lo strato di input e quello di output, responsabili dell'aggiustamento di pesi e bias. Sono detti "nascosti" proprio perché il processo di elaborazione di pesi e bias che avviene al loro interno non è visibile.
- Lo strato di uscita (**Output Layer**): qui vengono raccolti i risultati dell'elaborazione degli strati nascosti e vengono adattati alle richieste del successivo livello-blocco della rete neurale.

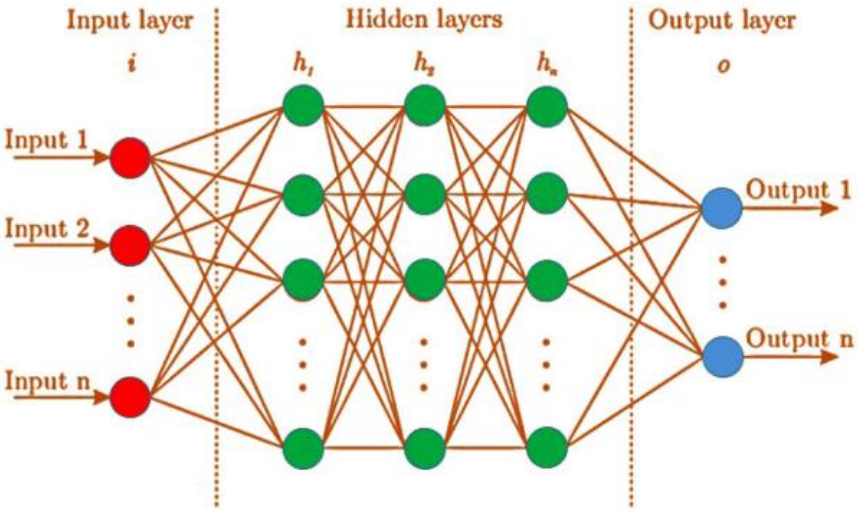

Figura 5: Architettura di una rete neurale multistrato

Durante l'addestramento del modello, inoltre, è utile valutarne l'accuratezza utilizzando una **funzione di costo** (o perdita), comunemente indicata come errore quadrato medio, che indica di quanto l'output della rete differisce dall'output desiderato, per un certo input. L'obiettivo è ridurre al minimo la funzione di costo per garantire la correttezza di adattamento per qualsiasi specifica osservazione. Quando regola i suoi pesi e la sua distorsione, il modello utilizza la funzione di costo per raggiungere il punto di convergenza o il minimo locale.

Per ottimizzare la modifica dei pesi si applica un'operazione chiamata "**discesa del gradiente**", che consiste nella derivazione della funzione d'errore per trovare il minimo locale, consentendo quindi al modello di determinare la direzione da prendere per ridurre gli errori (o ridurre al minimo la funzione di costo). Con ogni esempio di addestramento, i parametri del modello vengono regolati per convergere gradualmente verso il minimo.

Le reti neurali profonde sono, per la maggior parte, **feedforward**, cioè fluiscono in una sola direzione: dall'input all'output.
Tuttavia, si può addestrare il modello anche con un algoritmo chiamato **back propagation** (di retro-propagazione dell'errore), ossia con un movimento che

va nella direzione opposta: dall'output all'input. La retropropagazione consente di calcolare e attribuire l'errore associato a ciascun neurone e darlo in pasto alla rete assieme all'input, permettendo alla rete di imparare.

1.2 Reti Neurali Ricorrenti

Le reti neurali possono essere classificate in diversi tipi, nel nostro caso la rete neurale utilizzata è una rete neurale ricorrente (Recurrent Neural Network, RNN). Normalmente la rete neurale è un oggetto privo di memoria: dimentica i dati precedenti, e non tiene conto della loro disposizione. La rete neurale di tipo ricorrente invece, tiene conto della sequenza dei dati. Questo algoritmo di apprendimento, infatti, viene sfruttato soprattutto quando si utilizzano dati di serie temporali, per fare delle previsioni su risultati futuri. Una RNN crea una memoria interna alla rete che raccoglie e memorizza informazioni su ciò che il sistema ha calcolato precedentemente, quindi permette di passare l'output di un'esecuzione come input alla successiva, creando un loop di feedback negli strati nascosti.

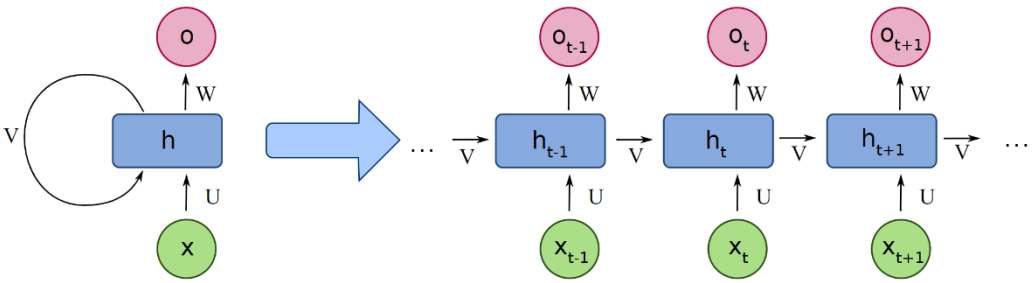

Figura 6: Diagramma di una RNN. A sinistra il diagramma compresso, a destra la sua versione estesa

In Figura 6 sono rappresentati, dal basso verso l'alto, gli stati in input, gli stati nascosti e gli stati di output. U, V e W sono i pesi della rete.

Ogni nodo ha uno stato nascosto h_t e produce un output o_t utilizzando l'ingresso corrente x_t e lo stato nascosto precedente h_{t-1} come segue:

$$h_t = f\{W_h h_{t-1} + V_h x_t + b_h\}$$
$$o_t = f\{W_o h_t + b_o\}$$

dove b è il bias per gli stati nascosti e di uscita e f è una funzione di attivazione. Il risultato di questa architettura è che gli RNN sono in grado di gestire dati sequenziali. Tuttavia, gli RNN sono soggetti a problemi di scomparsa[1] ed esplosione[2] del gradiente [13]. Inoltre, la lunghezza delle sequenze che un RNN può interpretare è piuttosto limitata, soprattutto rispetto agli LSTM [14].

1.3 Reti LSTM

Le reti LSTM (**L**ong **S**hort **T**erm **M**emory, memoria a lungo-breve termine) sono una variante delle reti ricorrenti classiche.

In particolare, le reti LSTM sono state progettate specificamente per superare il problema della scomparsa del gradiente presente nelle RNN, il quale potrebbe, nel peggiore dei casi, impedire alla rete neurale di addestrarsi ulteriormente. Uno dei vantaggi principali delle LSTM è la capacità di apprendere da lunghe sequenze temporali e conservarne la memoria. Queste reti, infatti, sono utilizzate quando si lavora con attività che richiedono la memoria di stati passati, come la traduzione automatica, il riconoscimento vocale, l'analisi dei sentimenti e la previsione delle serie temporali. Esse sono in grado, anche a lungo termine, di elaborare intere sequenze di dati conservando solo le informazioni utili sui dati precedenti nella sequenza, e scartare il resto.

[1] Scomparsa del gradiente: l'errore propagato all'indietro durante il processo di aggiornamento dei parametri diminuisce esponenzialmente ad ogni strato, e ciò non permette ai coefficienti degli ultimi strati (quelli vicini allo strato di input) di aggiornarsi.

[2] Esplosione del gradiente: l'errore propagato all'indietro aumenta esponenzialmente, ciò porta ad un ampio aggiornamento dei parametri che rende il processo instabile.

L'output di una LSTM in un determinato momento dipende da tre elementi:
la memoria a lungo termine della rete, nota come stato della cella (Cell state);
l'output nel momento precedente, noto come stato nascosto precedente;
i dati di input nella fase temporale corrente [15].
Gli LSTM utilizzano una serie di porte (Gates), che possono essere pensate come filtri: controllano il modo in cui le informazioni in una sequenza di dati entrano, vengono memorizzate e lasciano la rete. Una LSTM è caratterizzata da una cella di memoria (Memory Cell) che mantiene e trasporta le informazioni, e tre porte: *forget gate*, *input gate* e *output gate*.

La struttura di una cella LSTM è illustrata in Figura 7, con:
• x_t: Vettore d'ingresso
• x_{t-1}: Output del blocco precedente
• c_{t-1}: Memoria del blocco precedente
• σ: Funzione Sigmoidea
• x: Moltiplicazione
• +: Somma
• *tanh*: Tangente iperbolica

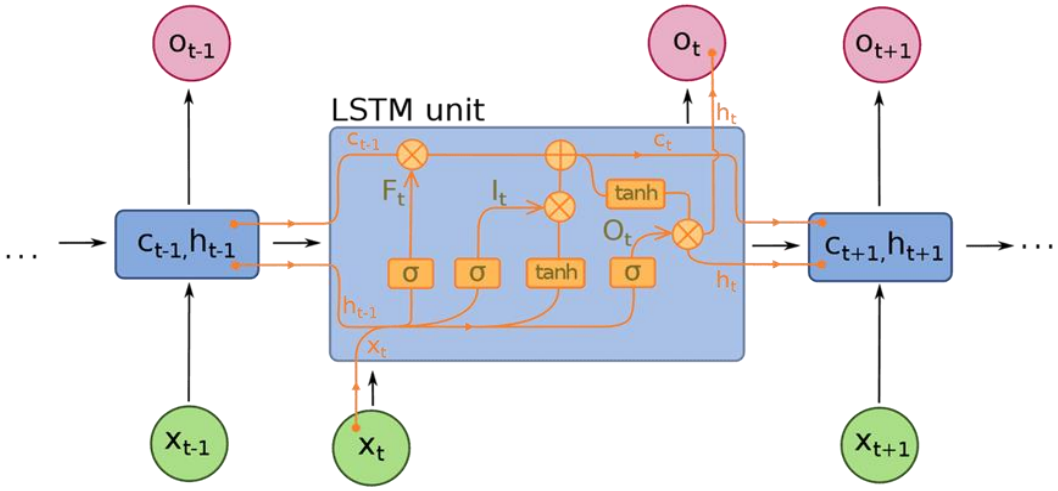

Figura 7: Diagramma di una singola cella LSTM

Ogni cella LSTM genera un nuovo output o_t dopo aver valutato l'input corrente x_t assieme all'output e alla memoria precedente. Ad ogni output generato, la cella ha la capacità di modificare il vettore di stato C, aggiungendo o rimuovendo informazioni. Le porte, o gates, sono costituite da un livello σ e da un'operazione di moltiplicazione e controllano la quantità d'informazione che deve essere lasciata passare.

- La *porta di input* prende decisioni su quali valori sono importanti e dovrebbero essere lasciati passare attraverso il modello. Nella porta di ingresso viene utilizzata una funzione Sigmoide, che determina quali valori trasmettere attraverso la rete ricorrente. Zero abbassa il valore, mentre 1 lo conserva. Anche in questo caso viene utilizzata una funzione TanH, che decide quanto sono importanti per il modello i valori di input, che vanno da -1 a 1.
- Dopo aver tenuto conto degli input correnti e dello stato della memoria, il *gate di output* decide quali valori spingere al passo temporale successivo. Nella porta di uscita, i valori vengono analizzati e viene assegnata un'importanza compresa tra -1 e 1. Questo regola i dati prima che vengano portati al successivo calcolo del passo temporale.
- Infine, il compito del *forget gate* è eliminare le informazioni che il modello ritiene non necessarie per prendere una decisione sulla natura dei valori di input. Il forget gate utilizza una funzione Sigmoide sui valori, emettendo numeri compresi tra 0 (dimentica) e 1 (mantieni) [14].

2. Preprocessamento segnale EEG

2.1 Strumenti utilizzati

2.1.1 EDFbrowser

EDFbrowser è un software che permette di visualizzare graficamente ed analizzare i file di archiviazione di serie temporali come EEG, EMG, ECG, ecc. In questo caso l'ho utilizzato per visualizzare i segnali EEG in formato EDF. Una volta caricato il segnale EEG, l'applicazione offre informazioni sul soggetto, la data di registrazione e la durata, e consente di selezionare, aggiungere o rimuovere segnali [16].

2.1.2 Google Collaboratory

Colaboratory, o in breve "Colab", è un prodotto di Google Research; è un IDE che consente a qualsiasi utente di scrivere codice sorgente nel suo editor ed eseguirlo dal browser. Nello specifico, supporta il linguaggio di programmazione Python ed è orientato a compiti di machine learning. Questo servizio è totalmente gratuito, vi si accede tramite l'account GMail, e non richiede alcuna configurazione o installazione. È una macchina virtuale che offre risorse di elaborazione per poter modificare e testare il codice, come le GPU, per sfruttarne la capacità di parallelizzazione, cioè la capacità di eseguire più operazioni in parallelo (threads). Per questo l'esecuzione risulta più rapida [17].

2.1.3 Librerie di Python

Python dispone di una collezione di moduli pronti all'uso, contenuti nella Standard Library di Python. Ogni modulo è come una libreria (un insieme) di funzioni già scritte, che forniscono soluzioni standardizzate ai problemi che si verificano più spesso nella programmazione.
I moduli principalmente usati in questo progetto sono: **Numpy** per eseguire calcoli numerici e manipolare array multidimensionali e matrici; **Matplotlib** per la creazione di grafici; **SciPy** per il calcolo scientifico; **Scikit-learn** per la realizzazione di modelli predittivi di machine learning.
A queste si aggiunge anche **TensorFlow**: una libreria software open source che oggi è utilizzata in molti ambiti scientifici e aziendali per svolgere funzioni in deep learning. Viene usata ad esempio da Google negli algoritmi di riconoscimento delle immagini, oppure per la lettura e la digitalizzazione delle frasi scritte a mano, per il riconoscimento dei numeri, degli oggetti e dei simboli in una foto [18] [19].

2.1.4 EEGLAB

Uno strumento per il preprocessamento dei segnali è EEGLAB: il toolbox di Matlab utilizzato per la visualizzazione e l'elaborazione di biosegnali, come l'EEG. EEGLAB fornisce un'interfaccia utente grafica interattiva (GUI) che consente agli utenti di elaborare in modo flessibile e interattivo i segnali EEG (Figura 8). Il pacchetto che utilizza EEGLAB è **BioSignal Toolbox** [20], una libreria software open source per l'elaborazione di segnali biomedici con Matlab, Octave, C/C++ e Python. Sono supportati circa 50 diversi formati di dati, tra cui EDF, MAT e CSV [21].

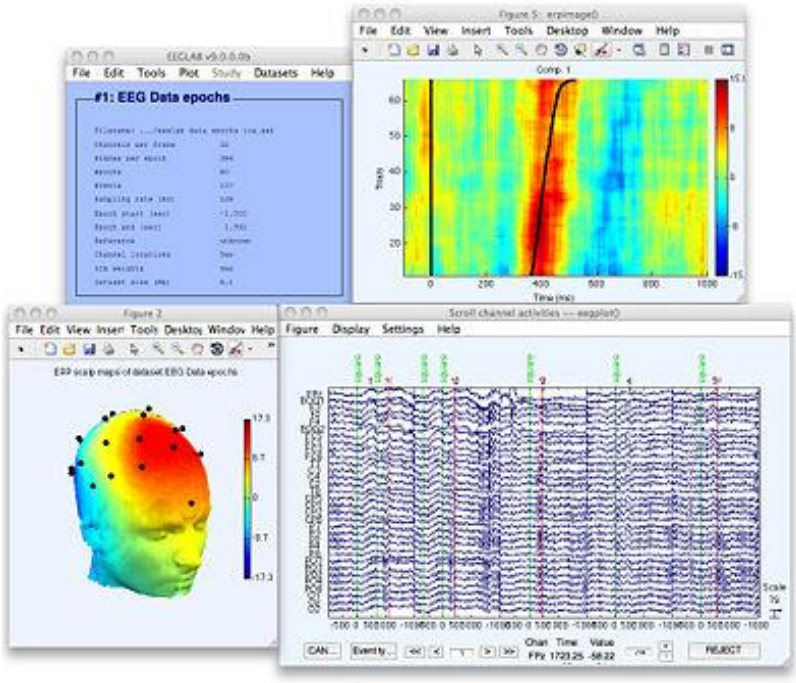

Figura 8: Interfaccia grafica del toolbox EEGLAB in MATLAB [22]

2.2 Formato del dataset

Il dataset a disposizione è costituito dai segnali EEG di 35 pazienti, 20 affetti da Alzheimer (AD) e 15 sani (N). I dati sono stati forniti in formato **EDF** (European Data Format), un formato utilizzato in ambito medico per lo scambio e l'archiviazione di segnali biologici e fisici, che permette di memorizzare serie temporali e dati multicanale. Ogni segnale EEG acquisito, infatti, ha più canali (da 21 a 23 canali) poiché ad ogni elettrodo corrisponde una regione ben precisa del cervello, quindi l'EEG di ogni paziente sarà caratterizzato da più segnali provenienti da aree cerebrali diverse. Alcuni esempi: i segnali Fp (cioè fronto-parietali), F (cioè provenienti dal lobo frontale), C (centrali), T (lobo temporale), P (lobo parietale), O (lobo occipitale) [23].

Un esempio è mostrato in Figura 9: è un segnale EEG con 22 canali (i nomi dei 22 segnali visibili a sinistra) e presenza di artefatti.

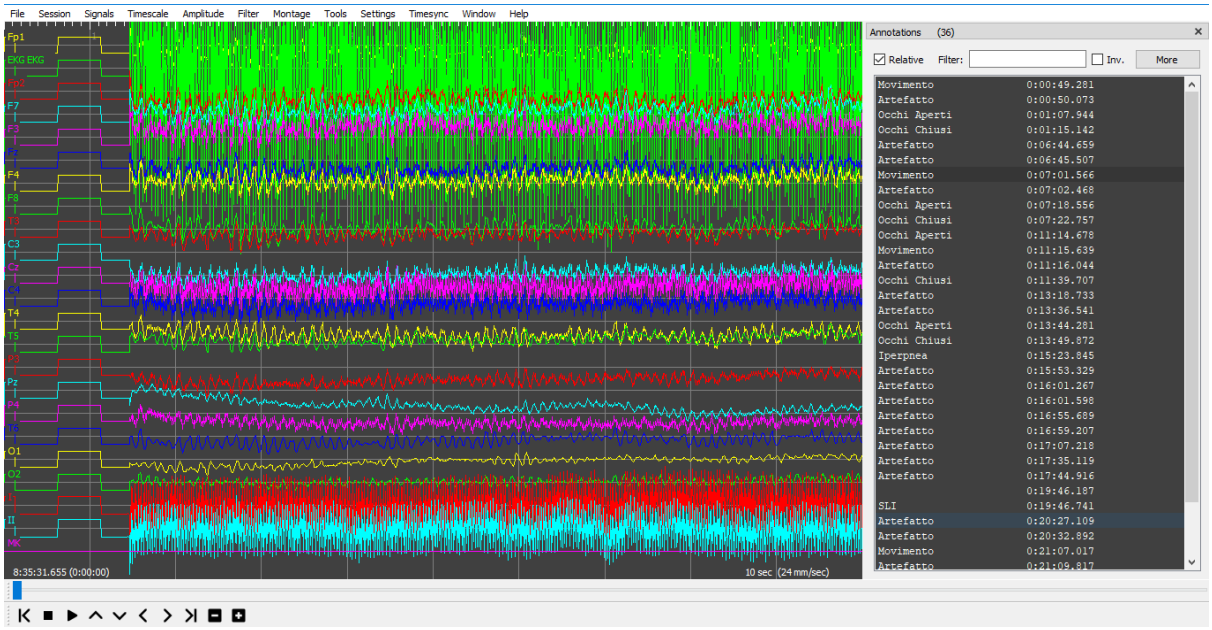

Figura 9: Segnale EEG visualizzato in formato EDF con EDFbrowser

A partire dal formato EDF iniziale, è stato fatto un preprocessing iniziale per avere un set di dati in input alla rete coerente e ordinato: sono stati riordinati e rimossi alcuni canali per far sì che tutti i segnali in input abbiano lo stesso numero di canali, 16, e nello stesso ordine [24].

In questo processo i dati sono stati salvati in formato **NPY**. Un file NPY è un file di array NumPy creato con la libreria NumPy di Python. Il file NPY memorizza tutte le informazioni necessarie per ricostruire correttamente l'array. Per ogni segnale, inoltre, è stato creato un file in formato **NPZ** (NumPy Zipped Data), cioè un file zip contenente più file NPY, uno per ogni array [25].

Per processare i segnali con le funzioni dell'EEGLAB, però, è necessario convertire il formato NPZ in formati supportati dall'EEGLAB, per questo progetto sono stati scelti i formati **MAT** e **CSV**. Quindi, dopo aver caricato il set di segnali in formato NPZ, ho creato una funzione per convertire rapidamente tutti i segnali (20 AD + 15 N), in entrambi i formati. I codici di conversione sono scritti interamente in Python, e sono riportati in Figura 10.

```python
def npz_to_CSV (filename):
  # carico il file .npz dalla cartella 'eeg2'
  data = np.load('/content/drive/MyDrive/eeg_DNN-Conti/eeg2/' + filename + '.npz')
  newfile= filename+'.csv'

  #salvo il file .csv nella cartella 'eeg2_csv'
  working_dir = '/content/drive/MyDrive/eeg_DNN-Conti/eeg2_csv/'+ newfile
  arr=data.files[0]
  csv=np.savetxt(working_dir, data[arr], delimiter=",")
  return data, csv
```

Figura 10: Codice di conversione npz/csv

Per verificare che la conversione sia stata eseguita correttamente ho confrontato i segnali di formati diversi, visualizzandoli graficamente, con l'apposita libreria di Python Matplotlib, verificando che abbiano lo stesso andamento nel tempo (Figura 11). Ogni segnale EEG è una matrice di dimensione nx16, con n numero di istanti temporali e 16 numero di canali. Dal grafico di sinistra è visibile la sovrapposizione dei 16 segnali. Utilizzando gli indici da 0 a 15, come nel grafico di destra, è possibile visualizzare ogni canale singolarmente.

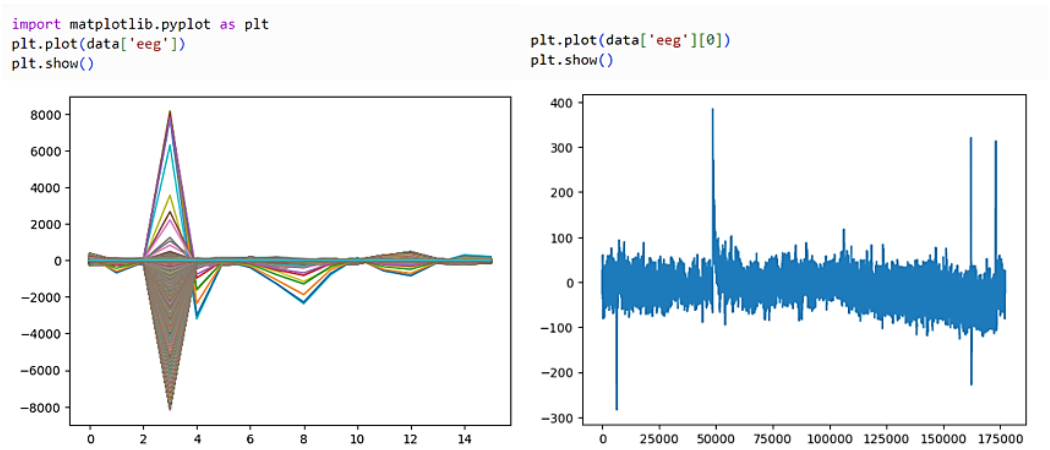

Figura 11: Grafici del segnale EEG

2.3 EEGLAB in Python

Dopo aver portato i segnali in un formato adeguato, il passo successivo è quello di applicare le funzioni dell'EEGLAB ai segnali del dataset per eliminarne le distorsioni.
L'EEGLAB però non funziona nativamente in Python, ma lavora in MATLAB e sulla piattaforma open source Octave. Inoltre, lavora principalmente con il formato **SET**, che è un formato proprietario del MATLAB, quindi non integrabile in codici open source come Python. Oltre a questo, dato che il preprocessing dei dati deve essere effettuato in un'architettura di rete già implementata in Python, per automatizzare la procedura si è deciso di integrare la parte EEGLAB direttamente in Python. Per questi motivi è stato necessario cercare un collegamento tra EEGLAB e Python. [26]

Per chiamare e quindi eseguire le funzioni dell'EEGLAB in Python, la soluzione trovata è installare Octave e usare il pacchetto Python Oct2py (Figura 12). Questa libreria Python converte le strutture dati Python in strutture dati MATLAB/Octave e viceversa. Tutte le funzioni di elaborazione del segnale EEGLAB, infatti, funzionano anche su Octave [27].

```
!apt-get install octave
!pip install oct2py
```

Figura 12: EEGLAB in Python

Oltre al pacchetto Oct2py sono stati aggiunti altri pacchetti non presenti in Octave, necessari per l'utilizzo delle funzioni: *eeglab*, *dipfit*, *firfilt*, *clean_rawdata* e *cleanline* (Figura 13).

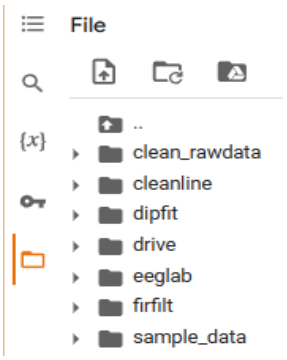

Figura 13: Pacchetti installati

Sono state importate le funzioni desiderate ed è stato caricato un segnale di prova. Su questo è stato applicato un filtro passa alto alla frequenza di 0.2 Hz utilizzando la funzione pop_eegfiltnew del pacchetto firfilt, che contiene tutte le funzioni necessarie ad implementare diversi tipi di filtri.
Un'altra funzione utilizzata per il filtraggio è pop_cleanline, dal pacchetto CleanLine ("linea pulita"), che rimuove gli artefatti dai componenti ICA, in particolare il rumore di linea, che può derivare da fluttuazioni della linea di alimentazione (ad es. rumore di linea con ripple a 50/60 Hz). La funzione per implementare l'ICA con l'EEGLAB, invece, è pop_runica, contenuta nel pacchetto popfunc, tra le 'functions' di EEGLAB.
L'obiettivo era quello di implementare tutto il preprocessing con l'EEGLAB, ma alcune funzioni EEGLAB non sono ancora supportate in Python.

Un'altra soluzione trovata per poter lavorare in Python è **MNE**, un pacchetto Python per l'esplorazione, la visualizzazione e l'analisi dei dati neurofisiologici, tra cui l'EEG. Questo software open source fornisce algoritmi implementati in Python per il preprocessing, il filtraggio e l'ICA. Lo svantaggio di questo metodo è che tra i formati supportati non ci sono quelli di nostro interesse, cioè NPZ o CSV, ma solo l'EDF o il SET [28].

Quindi il preprocessing è stato elaborato sempre in Python ma utilizzando un altro pacchetto: scikit-learn, che permette di utilizzare i dati in formato NPZ, già pre-elaborati.

2.4 Scikit-learn: FastICA

Scikit-learn è una libreria di machine learning per Python che supporta apprendimento supervisionato e non supervisionato. Fornisce inoltre vari strumenti per adattamento del modello, pre-elaborazione dei dati, selezione del modello, valutazione del modello, e molte altre utilità.
La funzione presa da questa libreria, utilizzata nella pre-elaborazione, è **FastICA**: un algoritmo veloce per eseguire l'analisi indipendente dei componenti (ICA, Independent Component Analysis) [29].

L'**ICA** è un metodo di elaborazione del segnale che permette di rimuovere/sottrarre artefatti incorporati nei dati senza rimuovere le parti di dati interessate, poiché riesce a separare i segnali misti. Il segnale EEG, infatti, è un segnale oscillatorio con caratteristiche stocastiche, ed è una miscela di segnali utili (quelli provenienti dal cervello) e segnali provenienti da sorgenti di disturbo. L'ICA è una tecnica di analisi statistica, in grado di individuare e separare le diverse sorgenti, poiché di solito sono indipendenti da un punto di vista statistico [30]. (Per approfondimenti: problema del cocktail-party [31]).

La funzione FastICA, dato che fa parte di una libreria Python, riceve in input i segnali direttamente in formato npz, quindi a differenza di quello che è stato fatto per poter utilizzare le funzioni dell''EEGLAB, in questo caso non c'è bisogno di convertire il formato dei dati a disposizione.
La funzione è applicata nella parte iniziale di codice, in cui viene elaborato il dataset per migliorare il più possibile i dati in input alla rete.
L'implementazione è mostrata in Figura 14.

```
transformer= sklearn.decomposition.FastICA(n_components=None,  algorithm='parallel',
                                           whiten='arbitrary-variance', fun='cube',
                                           fun_args=None, max_iter=1000, tol=0.0001,
                                           w_init=None, whiten_solver='eigh',
                                           random_state=None)
eeg= transformer.fit_transform(eeg)
```

Figura 14: Implementazione dell'algoritmo FastICA

I parametri principali che caratterizzano la funzione sono:

- ***n_components***: Specifica il numero di componenti da utilizzare.
  Di default vengono utilizzati tutti (default='*None*').
- ***algorithm***: Specifica l'algoritmo da utilizzare per FastICA,
  '*parallel*' o '*deflation*' (default='parallel').
- ***whiten***: Specifica la strategia di sbiancamento[3] da utilizzare.
  'arbitrary-variance': sbiancamento con varianza arbitraria.
  '*unit-variance*': varianza unitaria (default=unit-variance).
  '*False*': i dati sono già considerati sbiancati e non viene eseguito lo sbiancamento.
- ***fun*:** La funzione utilizzata nella stima della matrice di demiscelazione[4].
  Potrebbe essere *'logcosh'*, *'exp'* o *'cubo'* (default='logcosh').
- ***max_iter*:** Numero massimo di iterazioni durante l'adattamento (default=200).
- ***tol***: Uno scalare positivo che fornisce la tolleranza alla quale si considera che la matrice di demiscelazione sia convergente (default=0,0001).
- ***whiten_solver*:** Il risolutore da utilizzare per lo sbiancamento.
  '*eigh*' o '*svd*' (default='svd').

[3] sbiancamento (whitening): è un'importante fase di pre-elaborazione prima di eseguire l'ICA, e serve per ridurre la ridondanza nei dati di input. Il termine sbiancamento, infatti, deriva dal rumore 'bianco', caratterizzato da campioni indipendenti tra loro (non correlati). Lo sbiancamento trasforma quindi un vettore casuale in un vettore con componenti non correlate. Oltre a questo algoritmo, un altro metodo utilizzato più avanti nel codice per ottenere lo sbiancamento è la PCA.

[4] matrice di demiscelazione: matrice (A) che lega le misurazioni (x) ai segnali sorgente originali (s): $s = A \cdot x$

Per verificare che la funzione abbia agito correttamente, ho applicato l'ICA su un solo segnale e ho confrontato il plot iniziale del segnale, con quello dopo l'ICA (Figura 15). Nei grafici in alto, prima dell'ICA, sono visibili dei picchi molto ampi causati dagli artefatti, sovrapposti al segnale utile. Nei grafici in basso, ottenuti dopo l'ICA, si può osservare come l'ampiezza dei picchi sia notevolmente ridotta. Questo è evidente sia nel plot del segnale con 16 canali (sin), sia in quello con singolo canale (dx).

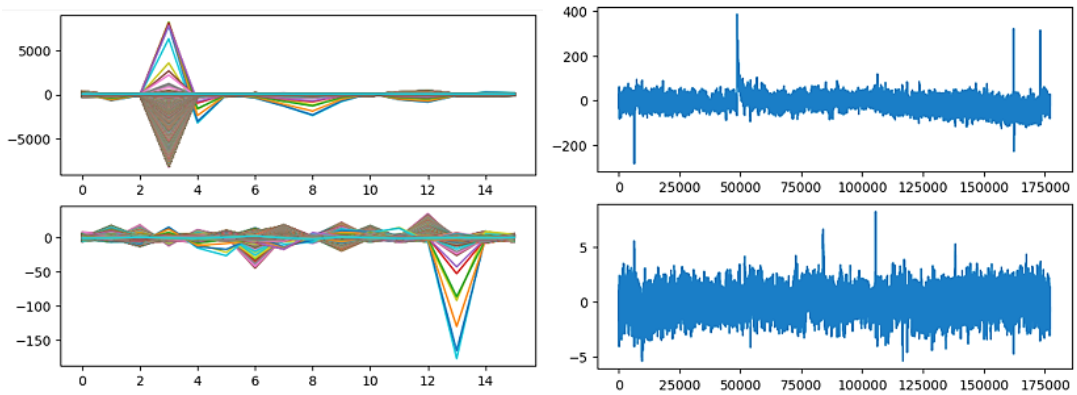

Figura 15: Confronto: prima e dopo ICA

3. Descrizione del modello

3.1 Algoritmo di classificazione

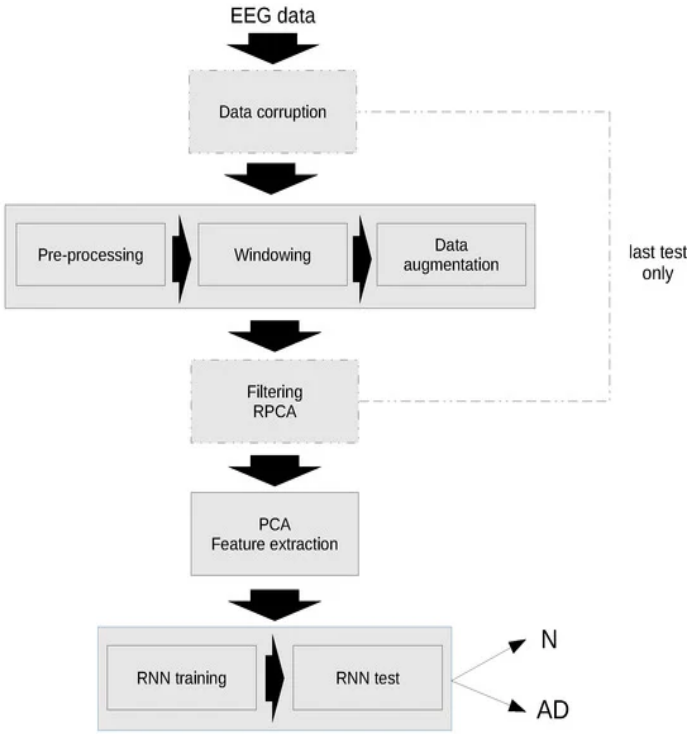

Figura 16: Diagramma dell'algoritmo di classificazione [24]

In Figura 16 è rappresentato un diagramma schematico dell'algoritmo di classificazione tratto dall'articolo [24].

I segnali EEG acquisiti sono rumorosi e sporchi, quindi prima di essere dati in input alla rete neurale sono necessari diversi step per renderli più puliti e ottenere una migliore accuratezza del modello.

Il primo passaggio è il **pre-processing**, cioè la pulizia dei dati tramite diversi filtraggi e rimozione di artefatti, con le tecniche di cui si è precedentemente discusso.

Il passo successivo è il **windowing**: i dati di input vengono suddivisi lungo l'asse temporale in N finestre di dimensioni fisse, che si sovrappongono di una determinata quantità. La lunghezza della finestra (w) e la sovrapposizione (overlap) sono iperparametri stabiliti sperimentalmente al fine di ottimizzare l'accuracy (w=256, overlap=50%).

Poi c'è bisogno di un **aumento dei dati**: poiché il numero di ingressi appartenenti alle due diverse classi non è equamente rappresentato, la rete potrebbe essere sbilanciata verso una classe specifica. Una soluzione a questo problema è il **sovracampionamento**, che permette di aumentare i dati duplicando quelli delle classi con meno occorrenze, così i dati utilizzati per l'addestramento sono distribuiti in modo più uniforme tra le diverse classi. Ai dati duplicati è stato aggiunto un rumore casuale gaussiano, in modo da non avere dati identici nel set di addestramento. Il sovracampionamento è stato applicato prima di ogni fase di addestramento al sottoinsieme di soggetti utilizzati per l'addestramento, lasciando inalterati i soggetti del test.

Successivamente viene eseguita l'**analisi delle componenti principali** (**PCA**). L'obiettivo principale della PCA è quello di ridurre la dimensione del dataset: seleziona un certo numero di 'componenti principali', cioè solo quelle che contengono le caratteristiche (features) più importanti, scartando le variabili che danno uno scarso contributo all'informazione totale che i dati contengono. La riduzione della dimensionalità non solo semplifica il calcolo, ma migliora il rapporto segnale/rumore e anche l'accuratezza finale del modello. Dal punto di vista matematico la PCA permette di proiettare un dataset n-dimensionale in un piano dimensionale più basso, conservando il maggior numero di informazioni rilevanti (cioè selezionando le componenti che raccolgono al meglio la varianza dei dati originali) [32]. Nel nostro caso il dataset originale ha 3 dimensioni (è un tensore): N×w×16, dove N è il numero di finestre, w la lunghezza della finestra e 16 il numero di canali. Con la PCA il tensore 3D viene ridotto in una matrice bidimensionale. Il numero di componenti principali selezionate (p) è un'iperparametro scelto sperimentalmente (p=50).

Prima della PCA invece, viene applicata la **MSPCA**, un'altra tecnica di filtraggio che oltre ad avere la capacità di estrarre le features principali riducendo la dimensionalità come la PCA, è anche un metodo efficace per la rimozione del rumore. [24]

Ora i segnali sono pronti per essere dati in input alla rete neurale RNN. L'architettura di questa rete è illustrata nel capitolo successivo.

3.2 Architettura della rete RNN

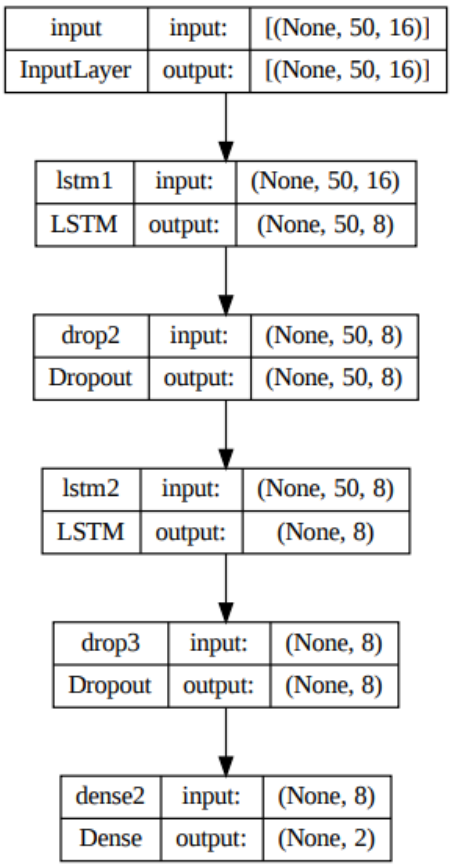

Figura 17: Architettura della rete utilizzata [24]

In Figura 17 viene mostrata una rappresentazione grafica della struttura della rete neurale utilizzata. La rete è di tipo **ricorrente (RNN)**, ed è costituita da 6 strati: 1 di input, 4 nascosti e 1 di output.
Il nucleo della rete neurale ricorrente sono i due strati **LSTM** in cascata.
Ogni livello LSTM è seguito da un livello **Dropout**. Il dropout è una tecnica di regolarizzazione, utile per ridurre il problema di overfitting, che consiste nel rimuovere un certo numero di nodi della rete in input ad ogni iterazione della fase di addestramento. Questo metodo viene implementato dopo lo strato sul quale viene applicato, come se fosse un nuovo strato della rete, e i nodi da scartare vengono selezionati in modo casuale [11].
Tutti gli strati intermedi hanno dimensione 8, questo iper-parametro è stato scelto sperimentalmente, ottimizzando l'accuracy.
Infine, l'ultimo è uno strato **Denso**, cioè completamente connesso, di dimensione 2. Questo livello esegue l'attività di classificazione in base alle features estratte dal segnale tramite i livelli precedenti, utilizzando la funzione di attivazione Softmax per classificare gli input in modo appropriato, e produce una probabilità da 0 a 1. È un modello di classificazione binaria: in output alla rete, infatti, ci saranno due possibili esiti: AD (soggetto malato) o N (soggetto sano).

3.2.1 Matrice di confusione

Uno strumento utile nelle reti neurali per valutare la qualità della classificazione svolta, è la matrice di confusione (confusion matrix), che permette di analizzare gli errori compiuti dal modello per migliorarne le prestazioni.
Quella in Figura 18 è la matrice di confusione del modello descritto precedentemente. Come già detto, la rete è un classificatore binario: riceve in input i segnali di 35 pazienti e restituisce in output una delle due possibili classi di appartenenza: sano (N) o malato (AD).

Nelle righe della matrice ci sono le classi effettive, cioè le classi delle risposte corrette (True label). Nelle colonne ci sono le classi di previsione, cioè le classi delle risposte del modello (Predicted label).

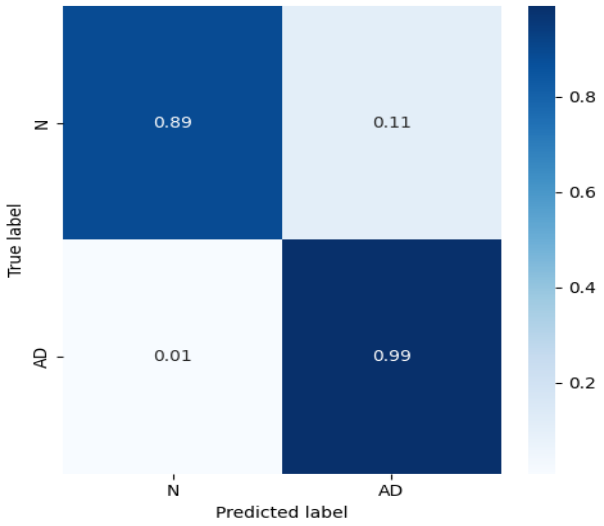

Figura 18: Matrice di Confusione

I risultati raggiunti dalla rete sono i seguenti:
Sani classificati correttamente (True Positive TP): (0,89x100) /2 = **44,5%**
Malati classificati correttamente (True Negative TN): (0,99x100) /2 = **49,5%**
Sani classificati come malati (False Positive FP): (0,11x100) /2 = **5,5%**
Malati classificati come sani (False Negative FN): (0,01x100) /2 = **0,5%**
Analisi corrette: 44,5% + 49,5% = **94%**
Analisi errate: 5,5% + 0.5% = **6%**

Dalla matrice si evince che nel 94% dei casi il modello classifica correttamente mentre nel 6% sbaglia. Dalla matrice di confusione, inoltre, si possono ottenere diversi parametri che valutano la precisione del modello, tra cui:
Il **tasso di errore** (error rate): misura la percentuale di errore della previsione sul totale delle istanze. Varia da 0 (migliore) a 1 (peggiore).

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

L'**accuratezza** (accuracy) misura la percentuale delle previsioni esatte sul totale delle istanze. È l'inverso del tasso di errore. Varia da 0 (peggiore) a 1 (migliore) [33].

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

3.2.2 Accuratezza della rete

Solitamente, per validare la qualità di una rete neurale, questa va testata su un set di dati non presentato durante l'addestramento. Per far ciò è stato diviso il dataset iniziale in due subset, uno per l'addestramento ed uno per il test (**test set**). Il set di addestramento va suddiviso a sua volta in altri due insiemi: uno per l'addestramento effettivo (**train set**), e uno per la convalida (**validation set**), di solito con un rapporto del 70-80% per l'addestramento e del 20-30% per la valutazione. Nel nostro caso sono stati utilizzati 30 soggetti per l'addestramento e la validazione e 5 per i test.

La suddivisione del dataset è descritta in Figura 19. Ogni subset di dati contribuisce ad addestrare meglio il modello:

- ✓ **Train set**: è il campione di dati effettivo usato per addestrare il modello. Il modello vede e apprende da questi dati.
- ✓ **Validation set**: è il campione di dati usato per valutare l'adattamento del modello al set di dati di training durante l'ottimizzazione degli iperparametri. Quindi il modello vede occasionalmente questi dati, ma non "impara" mai da questi. Questo set di dati influisce indirettamente sul modello durante la fase di "sviluppo", poiché i risultati del set di convalida vengono usati per aggiornare gli iperparametri.
- ✓ **Test set**: è il campione di dati usato per valutare l'adattamento finale del modello al set di dati di training, ed è il riferimento utilizzato per valutare il modello. Viene utilizzato solo dopo che un modello è stato completamente addestrato (usando i set di training e convalida).

I parametri principali che forniscono una valutazione dell'adattamento del modello al dataset sono i valori delle accuracy sui rispettivi subset di dati a cui si riferiscono: la train accuracy (sul train set), la val accuracy (sul validation set) e la test accuracy (sul test set).

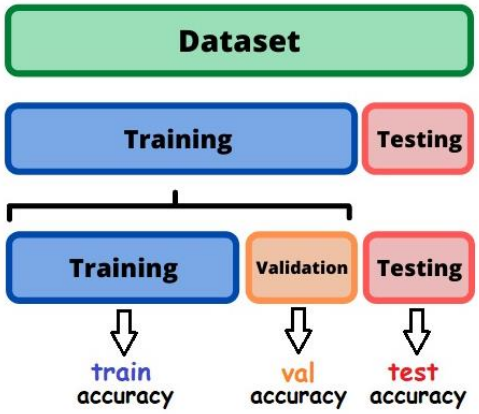

Figura 19: Suddivisione del dataset

3.2.3 Underfitting e Overfitting

La suddivisione di un set di dati potrebbe anche essere importante per rilevare se il modello soffre di uno dei due problemi molto comuni, denominati underfitting e overfitting:

- L'underfitting è la conseguenza dell'incapacità di un modello di acquisire le relazioni tra i dati. Questi modelli avranno probabilmente prestazioni scadenti sia con i set di addestramento che con quelli di test.
- L'overfitting di solito si verifica quando un modello ha una struttura eccessivamente complessa e apprende sia le relazioni esistenti tra i dati che il rumore: si adatta perfettamente ai dati di addestramento e, di conseguenza, quando viene sottoposto al test set, cioè a dati mai osservati in precedenza, non generalizza bene. Sebbene funzioni bene con i dati di training, in genere produce prestazioni scadenti con i dati di test. [34]

4. Risultati finali

4.1 Confronto dei risultati ottenuti

Per confrontare le prestazioni della rete prima e dopo l'applicazione dell'algoritmo FastICA, ho analizzato i valori di train val e test accuracy, di cui discusso nel precedente capitolo, e le loro variazioni, in seguito a modifica di alcuni iperparametri della rete.

4.1.1 Rete di partenza senza ICA

Parametri	train accuracy	val accuracy	test accuracy
PCA=true epochs=20	0.9831	0.9716	0.9657
PCA=true epochs=100	0.9940	0.9897	0.8925
PCA=false epochs=20	0.9259	0.9277	0.4282

Tabella 1: Rete senza ICA

4.1.2 Rete con algoritmo FastICA

Ho modificato i parametri della funzione FastICA lasciando invariato il resto del codice, per valutare le prestazioni della rete. Nella prima riga sono presenti i valori di default.

Parametri ICA	train accuracy	val accuracy	test accuracy
n_components= None max_iter=200 tol=0.0001	0.9876	0.9857	0.4404 ↓
n_components= 16 max_iter=200 tol=0.0001	0.9898	0.9884	0.5775
n_components= None max_iter=200 **tol=0.001**	0.9899	0.9889	0.5355
n_components= None **max_iter=1000** tol=0.0001	0.9913	0.9887	0.5882 ↑
n_components= None **max_iter=2000** tol=0.0001	0.9892	0.9844	0.4140 ↓
n_components= 16 **max_iter=1000** tol=0.0001	0.9886	0.9863	0.4224
n_components= 16 **max_iter=1000** **tol=0.001**	0.9879	0.9846	0.5678
whiten='arbitrary-variance'	0.9893	0.9872	0.4882
whiten=False	0.9998	0.9992	0.3231 ↓
whiten_solver='eigh'	0.9897	0.9867	0.4407
fun='cube'	0.9900	0.9862	0.4730
fun='exp'	0.9922	0.9799	0.2010 ↓

whiten='arbitrary-variance' fun='cube' whiten_solver='eigh'	0.9919	0.9892	0.5142
whiten='arbitrary-variance' fun='cube' whiten_solver='eigh' max_iter=1000	0.9915	0.9882	0.5497

Tabella 2: Parametri FastICA

In seguito, ho modificato i parametri della rete (PCA e numero di epoche) lasciando nell'algoritmo FastICA i parametri di default, e ho ottenuto:

Parametri	train accuracy	val_accuracy	test accuracy
FastICA=True PCA=True epochs=50	0.9945	0.9905	0.5150
FastICA=True PCA=True epochs=100	0.9969	0.9912	0.5434
FastICA=True PCA=False epochs=20	0.7518 ⬇	0.7568 ⬇	0.4625
FastICA=True PCA=False epochs=100	0.9936	0.9919	0.3384 ⬇

Tabella 3: Rete con ICA

4.2 Analisi dei risultati ottenuti

In Tabella 1 sono riportati in verde i valori ottimali, raggiunti con la rete di partenza senza ICA, con l'algoritmo della PCA e con un adeguato numero di epoche. L'obiettivo è quello di raggiungere risultati migliori aggiungendo un ulteriore preprocessing sui dati, cioè l'ICA.

Come si vede dalla Tabella 2, però, l'ICA porta a un crollo del valore della test accuracy, mentre la train e val accuracy restano invariate. Ho eseguito diverse prove modificando i parametri della funzione FastICA, ma la test accuracy non migliora. Con i valori di default l'ICA non converge. Con tol=0,001 scompare il warning sulla non convergenza dell'ICA, ma si abbassa ulteriormente la val accuracy. Il valore migliore con l'ICA si ottiene portando il numero di iterazioni a 1000.

Nella Tabella 3 invece ho modificato i parametri della rete, provando a non eseguire la PCA e aumentando il numero di epoche.
Un'epoca consiste in un ciclo completo di allenamento sul train set. Una volta che tutti i campioni del set sono stati visti, la rete ricomincia, segnando l'inizio della seconda epoca. Quindi all'aumentare del numero di epoche, il modello sarà più allenato nella predizione dei risultati, ma il rischio è quello di un overfitting, cioè una scarsa capacità di adattamento a nuovi dati mai visti prima. Questo potrebbe essere il motivo per cui, aumentando il numero di epoche il modello non diventa più preciso ma, come si vede dalla Tabella 3, la test accuracy diminuisce ancora.

5. Conclusioni e futuri sviluppi

Dopo aver aggiunto l'algoritmo FastICA al preprocessing dei dati, i valori di train e val accuracy si mantengono alti, invece quello della test accuracy è molto minore. Questo potrebbe accadere per due motivi.

Una prima ipotesi è che il modello sia soggetto a **overfitting**, fenomeno descritto nel capitolo 3.2.3. Infatti, quando si tenta di valutare il modello sul test set, dai risultati sperimentali si nota che le prestazioni sono molto peggiori di quelle che ci si aspetta. Una causa di questo fenomeno potrebbe essere che il dataset acquisito dal Dipartimento sia insufficiente.
Per un futuro sviluppo del progetto, quindi, la prima soluzione sarebbe quella di ampliare il dataset per avere un train set più esteso su cui far addestrare nuovamente la rete.

La seconda ipotesi potrebbe riguardare l'applicazione dell'ICA.
L'ICA, come visto in precedenza, riesce a separare il rumore dal segnale utile: dopo aver rilevato le componenti che rappresentano gli artefatti, qualsiasi segnale sorgente che ha una caratteristica di artefatto viene rimosso (moltiplicando per 0) e trasformato di nuovo in EEG.
Questo metodo però potrebbe avere alcune limitazioni:

- L'**aggressività** dell'algoritmo: un componente sorgente che si presume sia un artefatto viene rimosso, in realtà non sappiamo se l'algoritmo sta davvero rimuovendo una componente rumorosa o se insieme alla componente presunta ci sono anche importanti informazioni neurali che potrebbero essere utili per la modellazione predittiva.

L'aggressività di qualsiasi algoritmo di rimozione degli artefatti nell'EEG è distruttiva per la modellazione predittiva. Quindi sarebbe necessario un controllo visivo per determinare quali componenti rappresentino gli artefatti, per poterli eliminare correttamente.

- La **convergenza** effettiva di ICA: dovuta al fatto che, per stimare la matrice di demiscelazione, l'ICA lavora sulla minimizzazione delle informazioni reciproche tra i componenti indipendenti dei segnali sorgente, ma Potrebbero non esserci segnali sorgente indipendenti. C'è sempre un problema con la convergenza della matrice.
- Durante la rimozione di qualsiasi componente, introduce componenti di frequenza extra, che possono essere osservate nello spettro (grafico del periodogramma). Nel dominio del tempo queste componenti corrispondono a dei **picchi,** cioè brusche variazioni del segnale [35].

Un'altra soluzione, quindi, potrebbe essere quella di utilizzare altri algoritmi in sostituzione all'ICA, come ad esempio **ATAR** [36] [37] o **MARA**.

Bibliografia

[1] «alz.org,» Alzheimer's Association, [Online]. Available: https://www.alz.org/it/cosa-e-il-morbo-di-alzheimer.asp.

[2] « IRCCS Istituto Clinico Humanitas,» 2023. [Online]. Available: https://www.humanitas.it/malattie/alzheimer/.

[3] MSD Italia S.r.l., «Manuale Msd,» 2023. [Online]. Available: https://www.msdmanuals.com/it-it/professionale.

[4] M. R. F. B. L. Vicchietti, «Metodi computazionali di analisi dei segnali EEG per la classificazione della malattia di Alzheimer.,» *Scientific Reports,* p. 8184 , 2023.

[5] N. Boldrini, «ai4business,» NetworkDigital360, 2 luglio 2023. [Online]. Available: https://www.ai4business.it/intelligenza-artificiale/deep-learning/.

[6] M. Mancini, «MM,» 15 Gennaio 2021. [Online]. Available: https://www.mancinimarco.com/it/articoli.

[7] D. Parisi, «Reti neurali e vita artificiale,» in *Frontiere della Vita, Istituto dell'Enciclopedia Italiana*, Roma, D. Amit e G. Parisi, 2000.

[8] McCulloch, S. Warren e W. Pitts, "A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 1943.

[9] «MATLAB & Simulink,» MathWorks, [Online]. Available: https://it.mathworks.com/discovery/neural-network.html.

[10] «Reti neurali: parametri, iperparametri e strategie di ottimizzazione,» ICHI.PRO, [Online]. Available: https://ichi.pro/it/reti-neurali-parametri-iperparametri-e-strategie-di-ottimizzazione.

[11] G. Gullo, «Deep Learning e reti neurali con Python: il corso completo,» Profession AI. [Online]. [Consultato il giorno aprile 2023].

[12] «Tipi di architetture di rete neurale nel deep learning,» Intelligenza Artificiale Italia, [Online]. Available: https://www.intelligenzaartificialeitalia.net/post/tipi-di-architetture-di-rete-neurale-nel-deep-learning.

[13] F. Lollato, «Deep learning per la classificazione di immagini: un approfondimento sulle reti neurali profonde,» Università degli Studi di Padova, 2023.

[14] D. Nelson, «Cosa sono gli RNN e gli LSTM nel Deep Learning?,» Unite.ai, 23 agosto 2020. [Online]. Available: https://www.unite.ai/it/cosa-sono-rnns-e-lstms-nel-deep-learning/.

[15] «Reti LSTM | Una spiegazione dettagliata,» ICHI.PRO, [Online]. Available: https://ichi.pro/it/reti-lstm-una-spiegazione-dettagliata.

[16] «EDFbrowser,» Softpedia, [Online]. Available: https://www.softpedia.com/get/Multimedia/Graphic/Graphic-Viewers/EDFbrowser.shtml.

[17] «Google Colab,» [Online]. Available: https://colab.google/.

[18] «Python,» [Online]. Available: https://www.python.org/.

[19] M. Saba, «programmareinpython.it,» [Online]. Available: https://www.programmareinpython.it/video-corso-python-base/i-moduli-della-standard-library/.

[20] «File Exchange,» MathWorks, [Online]. Available: https://it.mathworks.com/matlabcentral/fileexchange/79427-biosig-a-toolbox-for-biomedical-signal-processing.

[21] «The BioSig Project,» [Online]. Available: https://pub.ista.ac.at/~schloegl/biosig/TESTED.

[22] «Swartz Center for Computational Neuroscience,» [Online]. Available: https://sccn.ucsd.edu/eeglab/index.php.

[23] «To the Science & Beyond!,» YouTube, [Online]. Available: https://youtu.be/ka5fEdkIZOc.

[24] M. Alessandrini, G. Biagetti, P. Crippa, L. Falaschetti, S. Luzzi e C. Turchetti, «Riconoscimento della malattia di Alzheimer basato su EEG utilizzando una rete neurale ricorrente robusta-PCA e LSTM.,» Sensors 2022, [Online]. Available: https://doi.org/10.3390/s22103696.

[25] «NumPy,» [Online].
Available: https://numpy.org/doc/stable/index.html.

[26] «Swartz Center for Computational Neuroscience,» [Online].
Available: https://eeglab.org/others/EEGLAB_and_python.html.

[27] «Swartz Center for Computational Neuroscience,» [Online]. Available: https://eeglab.org/others/Running_EEGLAB_on_Octave.html.

[28] «MNE-Python,» 22 11 2023. [Online].
Available: https://mne.tools/stable/index.html.

[29] Pedregosa, «Scikit-learn: Machine Learning in Python,» pp. pp. 2825-2830, 2011.

[30] E. Cordovani, *Sviluppo di metodi per l'analisi delle componenti indipendenti per la classificazione di potenziali evento-correlati di segnali elettroencefalografici.,* 2006.

[31] Parande, G. Poorva e T. G. Thomas, «A study of the cocktail party problem,» in *International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, 2017 .

[32] «Diario Di Un Analista | Data Science, Machine Learning & Analytics,» 2023. [Online]. Available:
https://www.diariodiunanalista.it/posts/introduzione-alla-pca-in-python-con-sklearn-pandas-e-matplotlib/.

[33] A. Minini, «online personal knowledge base,» [Online]. Available: https://www.andreaminini.com/ai/machine-learning/matrice-di-confusione.

[34] M. Stojiljković, «Real Python,» [Online]. Available:
https://realpython.com/train-test-split-python-data/#training-validation-and-test-sets.

[35] N. βajaj, «Artifacts in EEG and how to remove them: ATAR, ICA,» Medium, 26 agosto 2022. [Online].
Available: https://medium.com/@nikeshbajaj/artifacts-in-eeg-and-how-to-remove-them-atar-algorithm-ica-fbb91ea8485a.

[36] N. Bajaj, «Automatic and tunable algorithm for EEG artifact removal using wavelet decomposition with applications in predictive modeling during auditory tasks,» in *Biomedical Signal Processing and Control*, 2020, p. 101624.

[37] N. Bajaj, C. Requena e Jesús, «SpKit: Signal Processing Toolkit,» Python Package, [Online]. Available: https://spkit.github.io.