

UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea Triennale in Ingegneria Informatica

**Sviluppo di una Web App per la
ricostruzione del battito cardiaco
partendo dall'EEG tramite una Rete
Neurale Convoluzionale**

12 Dicembre 2024

Candidato:

Lorenzo Monaci

Relatori:

Prof: Antonio Luca Alfeo

Prof: Mario G.C.A. Cimino

Indice

1	Introduzione	7
1.1	Reti Neurali Convoluzionali, un approccio vantaggioso	7
1.2	Impieghi nella realtà, l'IA a supporto della scienza	8
2	Related Works	11
2.1	L'analisi delle serie temporali	11
2.1.1	Continous Wavelet Transform	11
2.1.2	CNN e DeepLearning	11
2.1.3	CNN multi-scala	12
2.2	Applicazione a segnali fisiologici	12
2.3	Limitazioni	13
3	Design e Implementazione	15
3.1	Use Cases	15
3.1.1	Utente	15
3.2	Il nostro approccio	16
3.2.1	Rappresentazione dell'output	16
3.3	Costruzione del modello di rete neurale	17
3.3.1	Conv1D	17
3.3.2	MaxPooling1D	17
3.3.3	Flatten	18
3.3.4	Dense	18
3.3.5	Compilazione	19
3.3.6	Overlap	19
3.4	Streamlit	19
3.4.1	Interfaccia utente	20
3.4.2	Modalità di visualizzazione	21
3.4.3	File Upload	21
3.4.4	Avvio	22
3.5	Google Colab	23
3.5.1	Implementazione del server Colab	23
3.5.2	Problemi di accesso all'applicazione	23
3.6	Tunneling tramite Ngrok	23
3.6.1	Creazione del tunnel	23
3.6.2	Server shutdown	24
4	Caso di studio	25
4.1	Descrizione del dataset	25
4.2	Rappresentare le serie temporali	26

4.3	Data Preprocessing	26
4.3.1	Labels	27
4.4	Gestione di un numero variabile di soggetti	27
4.4.1	Soggetto singolo	27
4.4.2	Soggetti multipli	27
5	Risultati sperimentali	29
5.1	Normalizzazione	29
5.2	CNN Performance & Metrics	29
5.2.1	Introduzione alle metriche utilizzate	29
5.2.2	Visualizzazione delle Performance	30
5.2.3	Plotting	31
5.3	Output e Download dei risultati	31
5.3.1	Serie temporale originale e Finestre temporali	31
5.3.2	Performance over all the subjects	33
5.3.3	Download di risultati e immagini in formato .zip	33
5.4	Analisi di soggetti singoli	35
5.5	Analisi di soggetti multipli	37
6	Considerazioni finali	43
6.1	Da dove siamo partiti	43
6.2	L'ipotesi confermata	43
6.3	Interfaccia Human-Friendly	44
6.4	Possibili sviluppi futuri	44
A	Manuale d'uso	45
B	Grafici	55
B.1	Grafici di test su soggetti multipli per ogni valore di overlap	55
B.1.1	Overlap 25	55
B.1.2	Overlap 50	56
B.1.3	Overlap 100	57
B.1.4	Overlap 125	58

Abstract

In medicina sorge spesso la necessità di sottoporre i pazienti a diversi esami per riuscire a stabilire una diagnosi, i quali possono essere dispendiosi in termini economici, di energia e tempo. Quindi ci chiediamo: *è possibile estrarre più informazioni da un singolo esame?* e qualora lo fosse, *come si può svolgere l'interazione con l'utente finale?*

Questo progetto di tesi ha l'obiettivo di rispondere (positivamente) a entrambe le domande. Si propone dunque un metodo per l'estrazione del segnale del battito cardiaco HR (Heart Rate) a partire dall'analisi dell' Elettroencefalogramma (EEG) utilizzando una Rete Neurale Convoluzionale (CNN), e lo sviluppo di un'applicazione al servizio dei medici in fase di diagnosi. I quali potranno quindi richiedere un numero minore di esami ai diversi pazienti per ottenere le informazioni dall'analisi dell'EEG, per mezzo dello strumento sviluppato durante questo studio. Nel momento in cui questi risultati dovessero risultare (per un qualsiasi motivo) poco attendibili si creerebbe la necessità di andare a richiedere esami specifici.

Si è scelto di utilizzare questo tipo di rete neurale perché diversi studi ne hanno dimostrato l'efficacia nell'analisi di dati correlati temporalmente. L'applicazione è sviluppata tramite Google Colab, un servizio di Notebook Jupyter che offre libero accesso a risorse di calcolo, eliminando il problema di dotare i medici di dispositivi con elevata capacità computazionale, tagliando i relativi costi. Per la realizzazione dell'interfaccia viene utilizzato Streamlit, un framework per lo sviluppo di applicazioni web interattive caratterizzato da un design intuitivo e user-friendly, il quale garantisce una rapida comprensione delle interfacce. Infine Ngrok, un servizio di tunneling gratuito che ci permette di creare un link pubblico a cui gli utenti possono collegarsi. Questo permette di facilitare la connessione per esempio all'interno degli ospedali, degli ambulatori medici distaccati, o di studi privati.

La rete neurale è in grado di ricostruire il battito cardiaco dei singoli soggetti con elevata precisione e di soggetti multipli con una modesta accuratezza partendo da EEG filtrati dall'interferenza di segnali fisiologici maggiori (come il movimento degli arti) ma comunque affetti da rumore generato da piccoli movimenti (come il movimento involontario degli occhi o la respirazione). I medici dovranno fornire in input i file rappresentanti l'EEG dei pazienti assieme ai parametri che caratterizzano il tracciato stesso, come la quantità di sovrapposizione temporale e il numero di punti temporali presenti in una finestra temporale.

Capitolo 1

Introduzione

Nell'epoca attuale uno strumento molto potente che ci permette di analizzare e risolvere problemi complessi è l'AI (Artificial Intelligence), un insieme di tecnologie che permette ai calcolatori di sviluppare un'ampia varietà di funzionalità avanzate. Questo insieme include abilità come essere in grado di elaborare e riconoscere immagini, comprendere e tradurre diverse lingue, analizzare set di dati, fare raccomandazioni e molto altro.

La funzione che utilizziamo nel nostro caso è quella di **classificatore**, la quale permette a fronte di un insieme di dati in ingresso riesce ad assegnare a ogni campione analizzato una *classe*. Un esempio classico di classificazione tramite AI è quello di addestrare una rete neurale per riconoscere a che specie appartengono vari esempi di fiori date le misure dello stesso.

In questo progetto si fa uso di una *classificazione binaria*, la quale date due classi (da qui binaria) si limita a dire se un campione appartiene a una classe oppure no.

1.1 Reti Neurali Convoluzionali, un approccio vantaggioso

Si utilizza come classificatore una **Rete Neurale Convoluzionale**, la quale applica una convoluzione¹ ai dati passati in ingresso. Questo processo riesce ad attribuire un certo significato all'istante temporale corrente in base anche al valore presente in tutti gli istanti precedenti a quello [1].

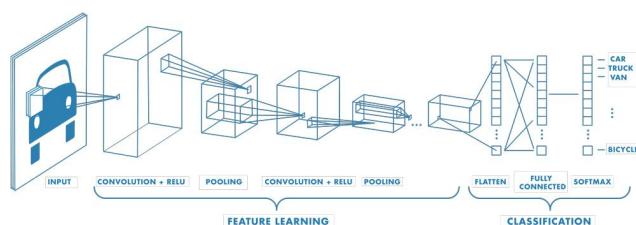


Figura 1.1: Schema generale di una rete neurale convoluzionale

¹Operazione tra due funzioni che consiste nell'integrare il prodotto tra la prima e la seconda traslata di un certo valore. Fonte: Wikipedia

Questa tecnica è molto efficace per via delle elevate prestazioni di questo tipo di reti per l'analisi di segnali temporali. Un software del genere permetterebbe di accorciare notevolmente le tempistiche e i costi delle visite mediche nonché fornire informazioni che normalmente vengono estratte da esami separati, si pensi quindi all'estensione di questa tecnica per la ricostruzione di altri segnali fisiologici, come la pressione sanguigna o la fotopletismografia ².

1.2 Impieghi nella realtà, l'IA a supporto della scienza

Parte della sfida risiede nel fornire all'utente un interfaccia con cui esso possa utilizzare i meccanismi precedentemente descritti in maniera trasparente, grazie al livello di astrazione che si interpone tra l'utente e la logica dell'applicazione [2].

Durante lo sviluppo di un'interfaccia utente per applicazioni AI per la sanità ci sono alcuni aspetti che vanno considerati:

Gli umani hanno memoria limitata

Come menzionato nell'articolo '*AI in healthcare: Improving human interface for patient safety*' (Avishek Choudhury, West Virginia University - Industrial and Management Systems Engineering) [3] un individuo nella media riesce a memorizzare al massimo 7 ± 2 set di informazioni alla volta, perciò le informazioni sanitarie dovrebbero essere suddivise in unità più piccole di questo limite. Mostrare a schermo un insieme ridotto di informazioni per pagina ridurrebbe il carico informativo che un operatore sanitario dovrebbe sopportare. Inoltre è dimostrato che l'utilizzo di colori appropriati aumenta la capacità di memorizzare le informazioni del 50%.

Percezione

Questo aspetto coinvolge due aspetti, la *teoria degli schemi*³ e la legge di Gestalt⁴. In sintesi questi principi descrivono come gli individui decodificano le informazioni e estraggono informazioni contestuali in base a cosa stanno guardando, per esempio un'immagine di un bel paesaggio porta alla mente sensazioni di calma e serenità. Le interfacce utente andrebbero modellate e strutturate tenendo di conto questo aspetto, considerando forma, colore e posizione degli elementi all'interno dello schermo.

²Un fotopletismogramma (FPG) è un plethysmogramma ottenuto per via ottica che può essere usato per rilevare variazioni del volume sanguigno all'interno di tessuti microvascolari. [Fonte:Wikipedia]

³In matematica uno schema è un concetto importante che connette i campi della geometria algebrica, dell'algebra commutativa e della teoria dei numeri. Fonte: Wikipedia

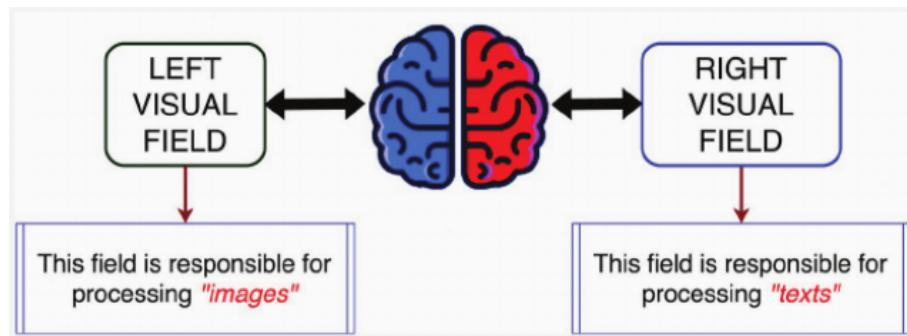
⁴È una branca della psicologia e una teoria della percezione che enfatizza la processazione di interi pattern e configurazioni, e non semplicemente componenti singole. Fonte: Wikipedia

Attenzione

Per quanto riguarda il posizionamento di testo e immagini a schermo, possiamo applicare la *left-to-right theory* secondo la quale è più efficace posizionare le informazioni critiche nell'angolo in alto a sinistra dello schermo. Questo è dovuto alla reattività degli umani, che tendono a interpretare i dati da in alto a sinistra verso in basso a destra.

Inoltre testo e immagini vanno posizionati considerando com'è organizzato il campo visivo umano, illustrato in Figura 1.2. Ciò permette di diminuire il carico cognitivo dell'utente.

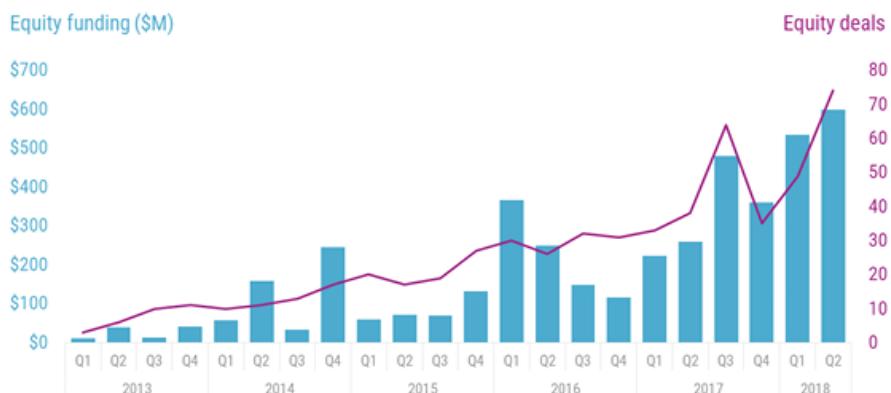
Figura 1.2: Processazione di testo e immagini nel cervello umano



Aumento dei fondi per la sanità

Si riporta un grafico⁵ che mostra come l'impiego di risorse economiche da parte di startup americane che lavorano all'IA in ambiente sanitario sia aumentato negli anni dal 2013 al 2018.

Figura 1.3: Denaro impiegato da startup americane specializzate nell'IA



⁵Fonte: cbinsights.com

Capitolo 2

Related Works

L'utilizzo delle CNN è fondamentale per una predizione accurata nel caso di serie temporali, dato che queste sono ‘comprese’ in modo molto preciso grazie appunto alla convoluzione che viene applicata, che nel caso di segnali correlati temporalmente permette di fornire una descrizione molto precisa di questi ultimi.

Dato che l'EEG è appunto un segnale temporale, si sceglie di analizzarlo utilizzando queste reti per, nel nostro caso estrarre il battito cardiaco, ma come già detto in precedenza potremmo concentrarci su qualsiasi altro segnale fisiologico.

2.1 L'analisi delle serie temporali

Come dimostrato nello studio ‘*Time-series analysis with smoothed Convolutional Neural Network*’ (*Aji Prasetya Wibawa, Agung Bella Putra Utama*) [4] l’impiego delle CNN non è sempre pensato per analizzare serie temporali, infatti nasce come approccio all’analisi delle immagini.

Le CNN necessitano di dati filtrati da rumori troppo evidenti per avere delle predizioni consistenti, quindi nasce la necessità di rendere i dati *smooth*, cioè adatti a essere dati in input alla rete.

2.1.1 Continous Wavelet Transform

Sono stati svolti esperimenti per altri tipi di classificazione, ad esempio nello studio condotto da *W-L Mao, H I K Fathurrahman, Y Lee, e T W Chang* intitolato ‘*EEG dataset classification using CNN method*’ [5] viene analizzato l’EEG di pazienti che soffrono di crisi epilettiche e messi a confronto con tracciati di pazienti sani. Il loro approccio combina la *Continous Wavelet Transform* (CWT) e l’utilizzo di una CNN. L’output prodotto risulta essere un’immagine che si classifica in 5 attributi.

2.1.2 CNN e DeepLearning

Nello studio ‘*Performance analysis of deep learning CNN in classification of depression EEG signals*’ (*Sandheep P, Meljo Poulose, Vineeth S and Suba D P*) si analizzano i tracciati EEG di 90 pazienti in totale, metà dei quali affetti da depressione e si svolge l’attività di classificazione tramite

una CNN formata da più layer nascosti, sfruttando così la tecnica del *Deep Learning*¹.

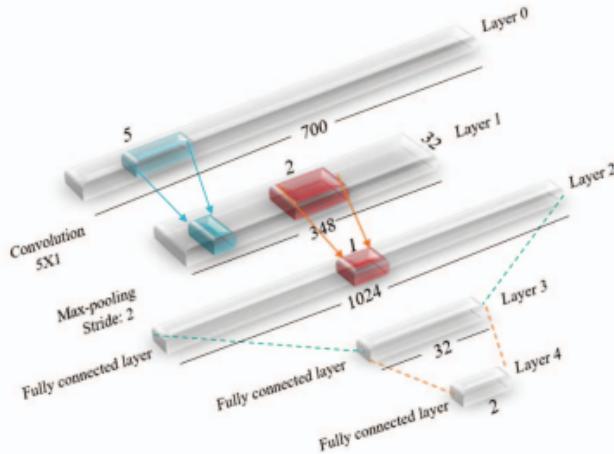


Figura 2.1: Struttura di una CNN con Deep Learning

2.1.3 CNN multi-scala

Le CNN vengono anche usate per riconoscere le intenzioni umane, come discusso nell'articolo ‘*Predicting Human Intention-Behavior Through EEG Signal Analysis Using Multi-Scale CNN*’ (*Chenxi Huang, Yutian Xiao, and Gaowei Xu*). In questo studio si cerca di predire tramite l’analisi di tracciati EEG con una rete neurale convoluzionale multi-scala l’intenzione o stato attuale del soggetto.

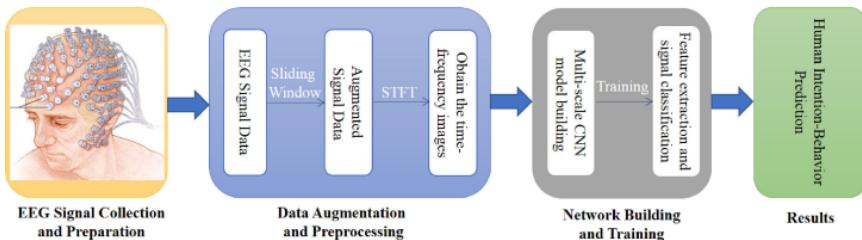


Figura 2.2: Procedura in 5 fasi

2.2 Applicazione a segnali fisiologici

I segnali elettrici che il cervello emette e che vengono quindi campionati dai vari sensori dell’apparecchio sono spesso misti a vari tipi di rumore, da quello più grande in ampiezza e facile da riconoscere e quindi rimuovere (*movimenti articolari, rumore termico, ...*) a quello lieve di movimenti impercettibili e involontari del nostro corpo (*Movimento delle palpebre, respiro, organi interni, ...*).

¹ Il deep learning è un sottoinsieme di machine learning che utilizza reti neurali multilivello, chiamate reti neurali profonde, per simulare il complesso potere decisionale del cervello umano. Fonte: IBM

La maggior parte di queste interferenze possono essere facilmente eliminabili tramite tecniche di *filtraggio*, ma come analizzato nello studio pubblicato sul Journal of Computer Science and Information Technology: ‘*Convolutional Neural Network Application in Biomedical Signals*’ (Haya, A.) [6] ci sono alcuni artefatti che non possono essere rimossi (rumore termico) o individuati facilmente (micro-movimenti), quindi è necessario far sì che la rete neurale ‘si abitui’ alla presenza di questi disturbi e li etichetti come tali.

2.3 Limitazioni

I metodi citati nelle sezioni precedenti sono efficaci ma non vengono applicati in un contesto di interazione con eventuali utenti che potrebbero trarne vantaggio. Non viene fatto alcun riferimento all’implementazione di questi metodi tramite linguaggi di programmazione o ai metodi usati nella pratica per effettuare le analisi. Ciò rende questi metodi vicini all’essere puramente teorici.

La complessità elevata delle tecniche di *Machine Learning* utilizzate richiede inoltre un hardware con elevata capacità computazionale, aspetto limitante se si considera l’idea di sviluppare del software che implementi questi approcci.

Capitolo 3

Design e Implementazione

3.1 Use Cases

Segue un elenco dettagliato dei casi d'uso individuati per ogni attore con in seguito un diagramma rappresentativo (Figura 3.1).

3.1.1 Utente

Si tratta dell'utente finale che si trova a utilizzare l'applicazione.

Addestra CNN

Dopo aver caricato i file di dati e labels l'utente addestra la CNN per ottenere un modello di predizione.

Visualizza predizione su time window

Dopo aver addestrato il modello l'utente visualizza nell'apposito grafico la presenza o meno di label nelle time window.

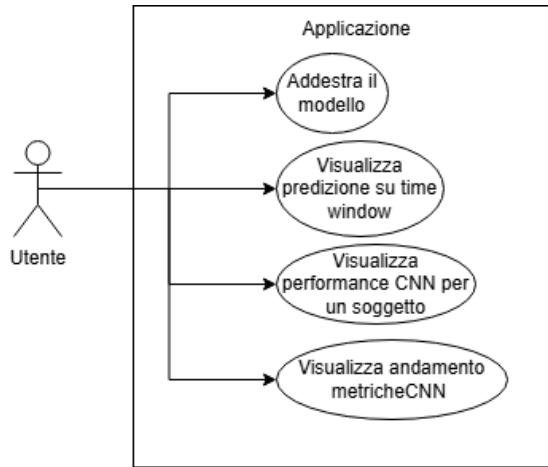
Visualizza performance CNN per un soggetto

Dopo l'addestramento della CNN l'utente visualizza le performance della CNN durante l'addestramento negli appositi grafici.

Visualizza andamento metriche CNN

Visualizzazioni delle metriche per ogni soggetto analizzato.

Figura 3.1: Diagramma degli Use Cases



3.2 Il nostro approccio

Cerchiamo rendere in primis i dati privi di ampi artefatti di rumore grazie a operazioni di *filtraggio in frequenza*, successivamente si applica una divisione tra dati di testing e training, utilizzando *un solo dataset* per la validazione e il resto come dataset di addestramento. Ciò permette di dotare la rete di una visione più ‘ampia’ del problema e generare così un modello che riesca a adattarsi a qualunque soggetto. In seguito si applica una *normalizzazione* ai dati, facendo sì che abbiano *media nulla* e *varianza unitaria* in modo da ridurre la variabilità dei dati.

Il nostro scopo è cercare di migliorare le capacità predittive della rete neurale facendo leva sul parametro di **overlap**. L’idea sta nel fatto di riuscire a fornire una quantità maggiore di informazioni temporali sovrapponendo parti di diverse finestre temporali.

3.2.1 Rappresentazione dell’output

‘Keep it simple, keep it stupid’¹

In base a questo principio alla base di molte applicazioni informatiche e ingegneristiche sceglieremo una tecnica di rappresentazione dell’output basata sui colori, in modo che possa essere colta al volo la differenza tra una predizione giusta e una sbagliata e dove si trova il picco cardiaco se presente.

¹ U.S. Navy, 1960 Fonte: Wikipedia

3.3 Costruzione del modello di rete neurale

La CNN è composta da 5 livelli (*layer*) che compongono li ‘strati’ di neuroni (*filtri*) della rete. È presente un layer convoluzionale (che da anche il nome a questo tipo di rete), due layer rispettivamente di input e di output e un ultimo layer che si occupa di ‘appiatire’ la dimensione dei dati intermedi.

3.3.1 Conv1D

Si tratta di un layer che applica la convoluzione (nel nostro caso temporale) monodimensionale tra i dati che riceve in ingresso e un *kernel*, cioè una matrice quadrata i cui valori rappresentano dei pesi e un numero di filtri pari a 64, questo altro non è che il numero di kernel utilizzati nella convoluzione.

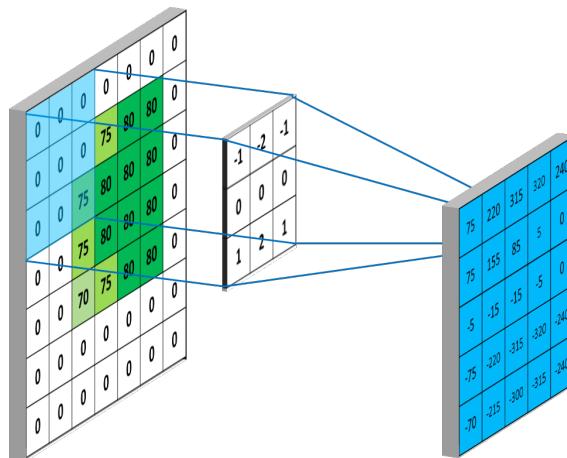


Figura 3.2: Esempio di applicazione di un kernel 3×3 a un input 7×7
Fonte: mlnotebook

3.3.2 MaxPooling1D

Questo layer si occupa di ridurre il numero di campioni a valle del layer convoluzionale prendendo quello con valore massimo in un pool di dimensione specificata (nel nostro caso 2×2). L’output di questo layer ha un nome e si indica come *Feature Map*.

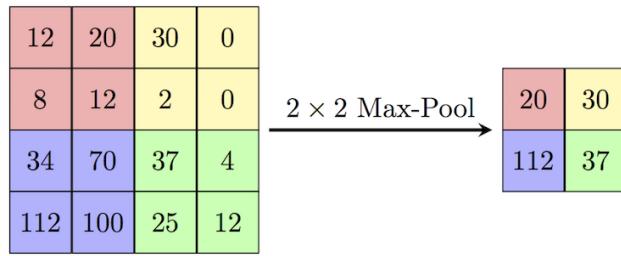


Figura 3.3: Esempio di Max Pooling con dimensione di sample 2×2
Fonte: [paperswithcode](#)

3.3.3 Flatten

Una volta estratte le feature di maggior valore tramite il precedente, il layer Flatten si occupa di appiattire la dimensione del tensore in esame, rendendolo monodimensionale. Questo ci serve per poter fornire in input ai due layer successivi dati con una sola dimensione.

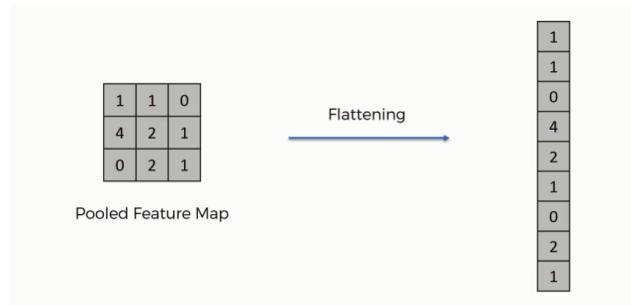


Figura 3.4: Appiattimento di una feature map di esempio
Fonte: [superdatascience.com](#)

3.3.4 Dense

Si tratta di due layer con filtri densamente connessi² con funzioni di attivazione *rettificatrice* e *sigmoida* rispettivamente con un numero di neuroni pari a 128 e 1.

²Ogni filtro è connesso a tutti i neuroni del layer precedente e successivo

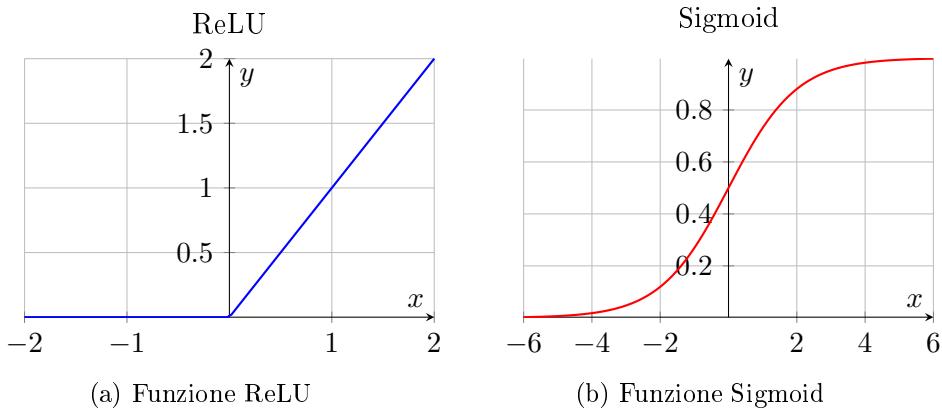


Figura 3.5: Confronto tra le funzioni di attivazione ReLU e Sigmoid.

3.3.5 Compilazione

Il modello è poi compilato con un `batch size` pari a 64 e impostato per andare avanti fino a 100 epoche.

Questo design è efficiente per il nostro caso di studio in quanto rimane un modello relativamente semplice e con un numero limitato di epoche per evitare l'overfitting (stiamo analizzando dataset piccoli, neanche vicini ai **Big Data**³), e un batch di dimensioni intermedie per analizzare una modesta quantità di dati per volta⁴.

3.3.6 Overlap

Come accennato in precedenza si aggiunge un ulteriore parametro da configurare, cioè la *sovraposizione temporale*, in inglese overlap. Dato che stiamo analizzando un segnale temporale sovrapponiamo parte di due finestre, all'inizio e alla fine di esse per fornire dati più densamente correlati tra di loro.

Dato che nei nostri dati le finestre sono tutte di lunghezza fissa a 150 punti temporali, utilizziamo dei valori di overlap inferiori a essa (25, 50, 100, 125).

3.4 Streamlit

Per interagire con l'utente si fa uso del framework **Streamlit**⁵, il quale mette a disposizione una ricca API per creare interfacce semplici e interattive all'interno di una applicazione web sviluppata utilizzando il linguaggio di programmazione **Python**.

³I quali si considerano tali per grandezze del centinaio di TB

⁴Si ricorda la regola di Pareto, per la quale l'80% dei risultati proviene dal 20% dei dati

⁵Documentazione API: streamlit.io

3.4.1 Interfaccia utente

Per programmare l’interfaccia si fa uso di funzioni di libreria che permettono l’inserimento di elementi testuali e interattivi che verranno utilizzati dall’utente per modellare i parametri in base alle proprie esigenze.

Testo

Per inserire il titolo si fa uso della funzione `st.title()`, la quale crea un header all’inizio della pagina.

Il resto del testo viene inserito tramite la funzione `st.markdown()`, che permette di processare il testo come linguaggio di markdown, utile per inserire dettagli che un semplice testo piatto non sarebbe in grado di esprimere, ad esempio gradienti di colori, emoji, testo in corsivo e in grassetto.

Elementi interattivi

L’utente potrà regolare i parametri dell’applicazione tramite dei *widget* di input numerico inseriti attraverso la funzione `st.number_input()`, la quale permette di inserire un valore numerico che rispetti determinati vincoli espressi attraverso i parametri attuali della funzione.

Nel nostro caso si inseriscono 3 input numerici che riguardano:

- la selezione dell’**overlap**, un valore che va da 25 a 125 con incremento di 5 e valore di default 25.
- il numero di **punti temporali** per ogni finestra, nel range [150, 250] di passo 50.
- quante **finestre** temporali campione l’utente desidera visualizzare, da un minimo di 1 a un massimo di 10.

Retrieve HR from EEG using a Convolutional Neural Network

Please upload files which file name formats as follows:

subject_[i]_MADtsROLLINGrawCLASSIFICATION_[DATA/LABELS]_[wlen]_[overlap]_BK_[grps]_[series].csv

Where:

- **i** is the subject taken in exam
- **wlen** is the length of each time window
- **overlap** is the amount of data overlap between each time window
- **grps** is the number of groups in the time series
- **series** is the number of correlated channels

Select the time window overlap

25

- +

Select the number of points per time window

150

- +

Select the number of windows extracted per subject

3

- +

Figura 3.6: User interface

3.4.2 Modalità di visualizzazione

Per far apparire i grafici a schermo creati con la libreria `matplotlib.pyplot` si utilizza la funzione `st.pyplot()` che inserisce un’immagine creata a partire dal grafico passato come input.

Questa funzione è di netta utilità per la sua trasparenza ai parametri dell’immagine.

3.4.3 File Upload

Seguono due *caricatori di file*⁶, completi di descrizione ed etichetta di aiuto per l’upload dei **dataset** rappresentanti l’EEG dei soggetti e le **label** degli stessi, dato che questo tipo di rete viene addestrata con un approccio *supervisionato*.

Vengono inseriti tramite la funzione `file_uploader()` che accetta come parametri il testo da mostrare a schermo, il tipo di file da caricare e se è possibile accettare file multipli.

⁶Max 200MB per file

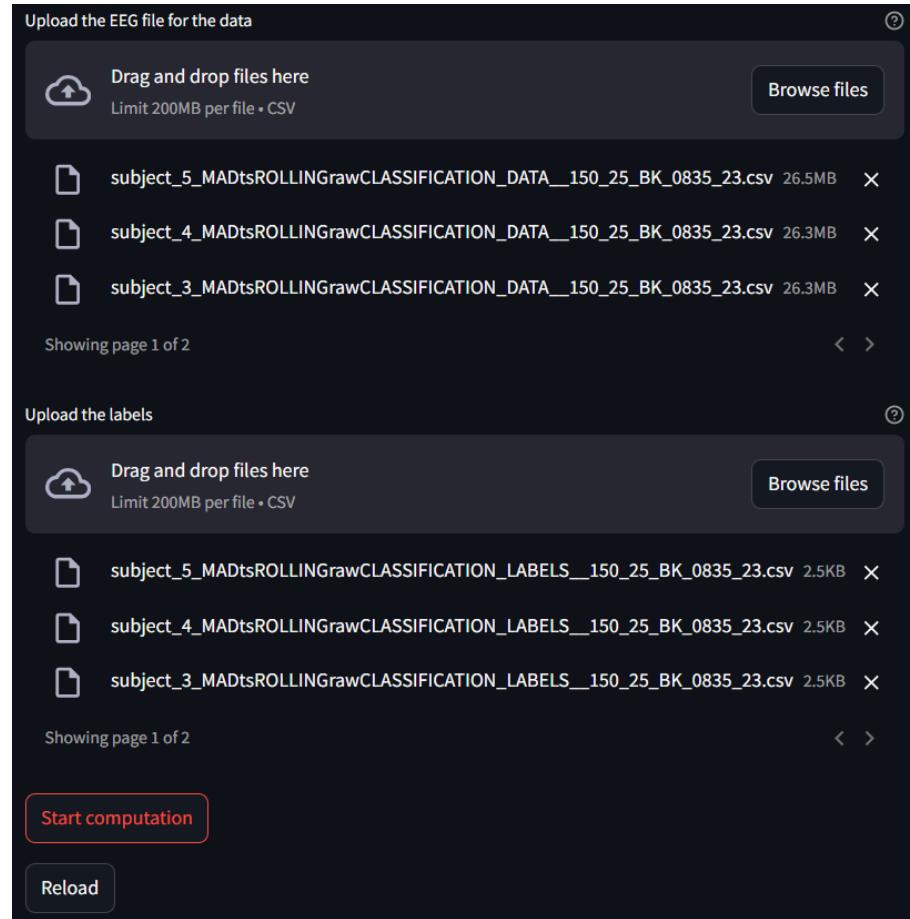


Figura 3.7: User interface

3.4.4 Avvio

Infine un apposito pulsante ‘*Start computation*’ (Figura 3.7) fa partire l’applicazione, premuto il quale l’utente è informato che l’elaborazione è in esecuzione tramite un apposito *widget animato*.

Questi due elementi sono inseriti tramite le funzioni `st.button()` e `st.spinner()`.

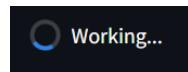


Figura 3.8: Widget di caricamento

Per lanciare l’applicazione si fa uso del comando `streamlit run applicazione.py`, il quale fornisce due indirizzi a cui collegarsi per accedere alla web app: `localhost` e un indirizzo privato interno alla rete a cui si è collegati.

3.5 Google Colab

La logica discussa nelle sezioni precedenti è implementata all'interno di un **Jupyter Notebook**⁷ all'interno di **Google Colab**, una piattaforma che permette di sviluppare *software interattivo* che verrà eseguito su delle macchine gestite da Google.

Ciò ci permette di affidarci a calcolatori gestiti da un ente certificato e con elevata potenza di calcolo, stabilendo una netta indipendenza dalla complessità dell'apparecchio che l'utente finale andrà a utilizzare.

3.5.1 Implementazione del server Colab

A livello pratico il nostro notebook farà le veci di un *srever* che fornirà agli utenti un link di accesso all'applicazione gestita attraverso **Streamlit**, il quale però non può essere acceduto direttamente dall'interno di Google Colab, perciò necessitiamo di un metodo di **tunneling**⁸ che ci permetta di accedere alla nostra applicazione in modo trasparente.

3.5.2 Problemi di accesso all'applicazione

Google Colab lavora con un suo vero e proprio FileSystem, il quale non può essere acceduto dall'utente che lo utilizza in modo diretto. Ciò crea un evidente problema di accesso, dato che Streamlit opera in *localhost* (o al massimo all'interno della propria rete privata, dunque con indirizzi privati⁹), problema che trova soluzione nell'utilizzo di **Ngrok**, il cui impiego è approfondito nella successiva sezione.

3.6 Tunneling tramite Ngrok

Nella sottosezione 3.5.2 viene introdotto il problema di connettività all'applicazione tramite Google Colab, il quale viene risolto tramite il servizio di tunneling gratuito **Ngrok**¹⁰.

3.6.1 Creazione del tunnel

Innanzitutto dobbiamo dire a Streamlit qual'è l'eseguibile che deve lanciare attraverso il tunnel, per far ciò ci serviamo di una *python magic* per creare un file all'interno del FileSystem di Colab che altro non è che l'eseguibile che ci serve, contenente il codice applicativo. Dopotiché abbiamo bisogno che sia un processo in background a occuparsi di lanciare l'applicazione, altrimenti la cella del notebook si bloccherebbe in un ciclo infinito. Inoltre ngrok necessita di un'autenticazione per garantire il servizio, la quale è soddisfatta fornendogli un token di autenticazione salvato in un file di testo su Google Drive, il quale è connesso al FileSystem di Colab.

⁷ Progetto nato per lo sviluppo di codice Python open-source, fonte: Wikipedia

⁸ Protocollo di comunicazione che permette ad un utente di fornire o accedere ad un servizio non supportato o non fornito direttamente dalla rete. Fonte: Wikipedia

⁹ Classe particolare di indirizzi IP utilizzabili all'interno di reti private. Fonte: Wikipedia

¹⁰ Sito web: ngrok.com

Ngrok a questo punto procede con la creazione del tunnel, fornendo un indirizzo IP a cui collegarsi per l'utilizzo dell'applicazione.

3.6.2 Server shutdown

Ora che abbiamo implementato il nostro server e lo stiamo facendo funzionare a dovere dobbiamo trovare un modo di spegnerlo. Per far ciò non basta altro che terminare il processo che gestisce Streamlit insieme ai processi lanciati dall'esecuzione di Ngrok.

In questo modo disconnetteremo tutti gli utenti, e impediremo nuove connessioni, dato che verrà generato un URL differente alla successiva esecuzione del server.

Capitolo 4

Caso di studio

In questo capitolo vengono esplorati i dati utilizzati durante lo sviluppo del nostro progetto.

4.1 Descrizione del dataset

I dati utilizzati in questo studio provengono da 32 soggetti analizzati durante un esperimento di stress termico sul corpo umano (cold-pressor test), mirato a studiare l’interazione funzionale fra il cervello e il cuore, ovvero l’influenza reciproca tra le attività dei due organi in una situazione di stress. Per lo svolgimento del test, ai soggetti è stato chiesto di sedersi su una sedia per garantire la stabilizzazione emodinamica. L’esperimento consiste in uno stato di riposo di 3 minuti, seguito da uno stress termico della stessa durata, in cui i volontari hanno inserito la mano in un secchio pieno d’acqua ghiacciata, e un successivo recupero, in cui hanno estratto la mano dal secchio. Durante tutta la procedura, ai soggetti è stato chiesto di tenere gli occhi chiusi per ridurre al minimo gli artefatti oculari nell’elettroencefalogramma. Ogni registrazione di dati così misurata comprende un EEG ad alta densità a 128 canali¹, ed un ECG a una derivazione, campionati a $500Hz$. I dati EEG utilizzati in questa tesi sono stati preventivamente trattati con un filtro passa-banda Butterworth tra 0.5 e $45Hz$. [7]

¹Si intende i canali periferici di un caschetto EEG

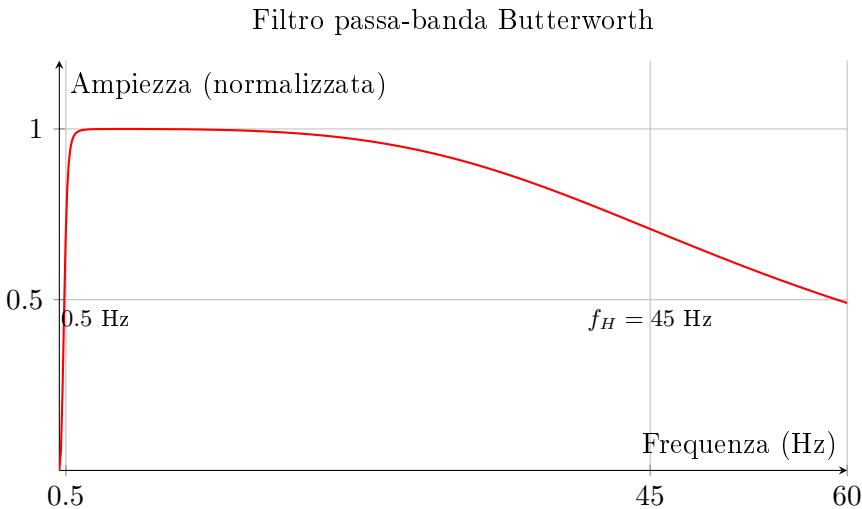


Figura 4.1: Risposta in ampiezza di un filtro passa-banda Butterworth con frequenze di taglio a 0.5 Hz e 45 Hz.

I canali contaminati da artefatti vengono rimossi o interpolati con dati vicini, a seconda della rumorosità degli stessi. Per via di questo processo rimaniamo con un pool di dataset pari a 26 elementi.

Per dividere un segnale rumoroso e composto da diverse componenti si fa uso dell’ *independent component analysis* (ICA), una tecnica applicata per ridurre ulteriormente il numero di artefatti e migliorare l’interpretabilità del segnale. Per quanto riguarda i segnali ECG, i picchi R nelle onde QRS sono stati identificati automaticamente e successivamente verificati manualmente per correggere eventuali errori, ottenendo un segnale che identifica i battiti cardiaci presenti.

4.2 Rappresentare le serie temporali

Inizialmente i file (si ricorda, in formato `csv`) sono memorizzati in un array monodimensionale, il quale rappresenta in tutto e per tutto una serie temporale, partendo dall’istante 0 di inizio misurazione all’istante finale.

4.3 Data Preprocessing

I dati subiscono poi una rimodellazione, necessaria per fornire alla rete neurale informazioni riguardanti i 23 canali periferici di un caschetto EEG.

Questa rimodellazione avviene trasformando l’array in un *tensore*² con dimensioni ($n_windows, n_points, n_channels$).

Le tre misure rappresentano rispettivamente quante finestre temporali sono presenti nella serie, quanti punti temporali vi sono all’interno di una finestra, e infine il numero di canali correlati (che rimane costante ad ogni esecuzione).

²Un oggetto matematico che generalizza tutte le strutture definite usualmente in algebra lineare a partire da un singolo spazio vettoriale. Fonte Wikipedia

4.3.1 Labels

Le label (o etichette) a differenza dei dati non subiscono alcun tipo di rimodellazione, dato che rappresentano la presenza (o meno) di un picco cardiaco all'interno di una finestra temporale, per questo sono in numero pari a queste ultime e non necessitano di alcuna trasformazione.

4.4 Gestione di un numero variabile di soggetti

I dati vengono successivamente trattati in due modi differenti a seconda del numero di soggetti che stiamo analizzando, per garantire l'analisi di un particolare soggetto oppure per cercare di creare un modello generico.

4.4.1 Soggetto singolo

In questo caso si adotta uno split del dataset stesso seguendo il Principio di Pareto. Così facendo addestriamo la rete neurale su un soggetto particolare, quindi il modello sarà creato ‘ad-hoc’ per quello specifico soggetto.

4.4.2 Soggetti multipli

Analizzando più soggetti alla volta consideriamo il fatto di addestrare la rete su tutti i soggetti meno uno per ogni soggetto, utilizzando quest’ultimo come soggetto di test. Così facendo andiamo a creare un tensore che avrà come prima dimensione il numero di finestre temporali moltiplicato per il numero dei soggetti analizzati meno uno.

In questo modo creiamo una serie temporale lunga a sufficienza da permettere alla rete neurale di capire il comportamento generale dei soggetti.

Anche le label subiscono un trattamento simile, semplicemente vengono concatenate le label dei vari soggetti senza considerare quello preso come test.

Capitolo 5

Risultati sperimentali

Questo capitolo intende accompagnare il lettore nell’analisi svolta, dalle metriche utilizzate per misurare l’efficienza dell’applicazione fino ai tipi di grafici impiegati nella visualizzazione dei risultati delle elaborazioni.

5.1 Normalizzazione

Successivamente allo split viene applicata una normalizzazione ai dataset tramite un ridimensionamento standard:

$$z = \frac{x_i - \mu}{\sigma}$$

5.2 CNN Performance & Metrics

5.2.1 Introduzione alle metriche utilizzate

La compilazione del modello avviene tenendo conto di 4 metriche:

1. **Accuracy**: metrca standard, la quale misura l’accuratezza del modello
2. **F1 score**¹: metrca implementata all’interno del codice che misura anch’essa l’accuratezza del modello ma è particolarmente rilevante in casi come il nostro in cui abbiamo dataset *sbilanciati* ed è espressa con la formula
$$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$
3. **Precision**²: misura la proporzione di campioni classificati come positivi che sono effettivamente positivi, rispondendo alla domanda ‘Quanto possiamo fidarci della classificazione positiva del modello’?

$$\frac{TP}{TP + TN}$$

¹F1 Score on Wikipedia

²Precision on Wikipedia

4. **Recall**³: misura la proporzione di campioni effettivamente positivi che sono stati classificati come tali, rispondendo alla domanda ‘*Quanto è abile il modello a catturare campioni effettivamente positivi?*’

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

5.2.2 Visualizzazione delle Performance

Durante l’addestramento della nostra rete neurale è possibile visualizzare il suo comportamento attraverso le epoches (che ricordiamo essere 100) e durante la validazione.

Per fare ciò vengono mostrati a schermo 3 grafici, i quali riportano l’andamento delle metriche descritte all’inizio di questa sezione durante l’addestramento e la validazione, con l’aggiunta di una quinta metrica (standard): **loss**. Essa indica quanto bene (o male) il modello sta predicendo, basandosi appunto sulle label fornite dall’utente.

E’ oltretutto la funzione che il modello cerca di minimizzare per mezzo di metodi basati per esempio sulla *discesa del gradiente*, noi utilizziamo la funzione di ottimizzazione **Adam**⁴

Nel nostro caso utilizziamo la funzione di loss **Binary Cross-Entropy**, la quale è specifica per problemi di classificazione binaria e definita per mezzo dell’espressione:

$$Loss = -\frac{1}{N} \sum_{i=1}^N y_i \log(\hat{y}_i) + (1 - y) \log(1 - \hat{y}_i)$$

Dove y_i è una label che vale 0 o 1 e \hat{y}_i è la probabilità predetta dal modello.

I nostri grafici sono così composti: i primi due indicano rispettivamente l’andamento di accuracy, f1, precision e recall durante l’addestramento, il secondo invece durante la validazione. Il terzo grafico riporta invece l’andamento del valore della loss sia durante l’addestramento che durante la validazione, in modo da avere un confronto diretto tra i due.

Si è scelto di rappresentare la loss in un grafico separato per la grande differenza tra i valori di quest’ultima e le altre metriche, che nei casi più estremi differiscono infatti di *almeno un paio di ordini di grandezza*.

³Recall on Wikipedia

⁴Adaptive Moment Estimation (Keras)

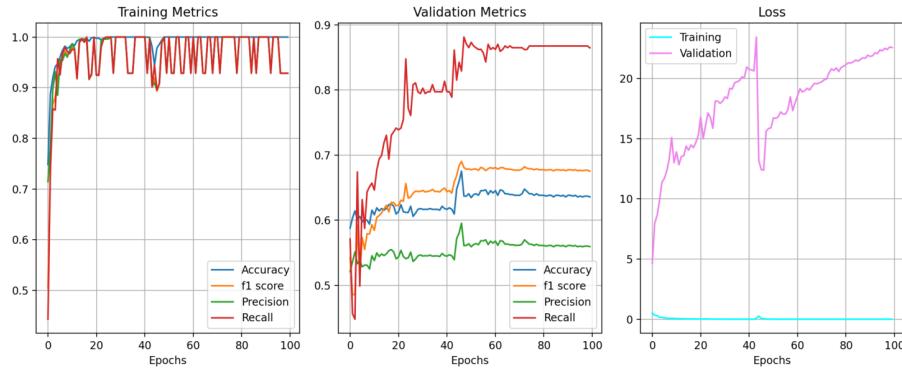


Figura 5.1: Rappresentazione delle metriche

5.2.3 Plotting

La visualizzazione a schermo dei grafici discussi precedentemente avviene per mezzo del pacchetto `matplotlib.pyplot` per creare le figure e del pacchetto `streamlit` per visualizzarle a schermo.

Viene scelta una colorazione apposita per rappresentare a schermo le linee dei vari grafici, la quale è riproposta successivamente in fase di stampa dei risultati.

5.3 Output e Download dei risultati

Durante l'operazione di testing vengono memorizzate le metriche di validazione, che vengono poi mostrate numericamente all'utente utilizzando gli stessi colori dei grafici descritti nella sottosezione 5.2.3.

Nel caso in cui vi siano più soggetti da analizzare l'applicazione memorizza la migliore e peggiore accuracy tra quelle misurate nel corso dell'elaborazione, ricordandosi anche a quale soggetto appartengono.

Questo può aiutarci a capire quali soggetti evidenzino un'atipicità rispetto agli altri, riconoscendo possibili anomalie nel tracciato dell'EEG o casi particolari che richiedono particolare cura nel trattamento.

Oltre a questo viene memorizzato anche il tempo di training per ogni soggetto, per poi calcolare e mostrare all'utente il tempo medio di addestramento alla fine di tutto il processo.

5.3.1 Serie temporale originale e Finestre temporali

Per avere un'idea di come si è comportata la rete neurale e soprattutto per cercare una *spiegazione* al fenomeno analizzato si plotta l'intera serie temporale del soggetto in questione (Figura ??) e in seguito un numero scelto dall'utente (come spiegato nella sottosezione 3.4.2) di finestre temporali da visualizzare.

Queste ultime sono delimitate da una riga verticale tratteggiata per rendere evidente la separazione tra esse, viene inoltre delimitato l'overlap con una riga verticale più sottile, a indicare quanti punti temporali fanno parte di una finestra e della precedente/successiva.

Le finestre temporali che vengono mostrate vengono scelte in modo randomico, estraendo un numero estratto con una distribuzione uniforme discreta di probabilità⁵ a cui poi viene sommato il numero di finestre scelto dall'utente secondo la formula

$$\begin{aligned} start &= n_points \cdot r \\ stop &= n_points \cdot (r + n_finestre_utente) \end{aligned}$$

Dove r è il numero estratto, $n_finestre_utente$ è il numero di finestre temporali che l'utente desidera visualizzare e n_points il numero di punti temporali per finestra.

La linea continua rappresentata nel grafico può assumere il colore **rosso** nel caso in cui la rete abbia generato una previsione errata, nel caso contrario sarà colorata in **verde**.

Se nella finestra temporale è inoltre presente una label positiva (che sta a indicare che è presente un picco cardiaco) viene indicata con un quadratino **blu** in corrispondenza del valore del picco più alto presente all'interno della finestra.

I grafici sopra descritti sono accompagnati da una legenda che spiega all'utente i colori associati alle varie linee nel grafico.

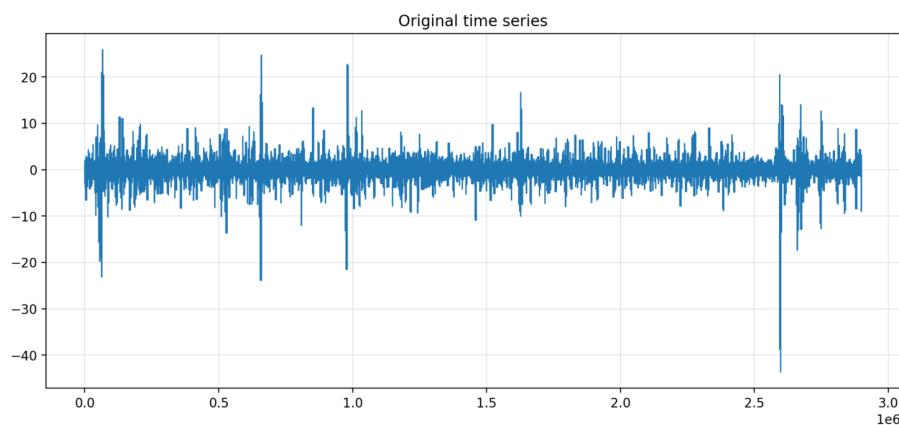


Figura 5.2: Rappresentazione della serie temporale originale

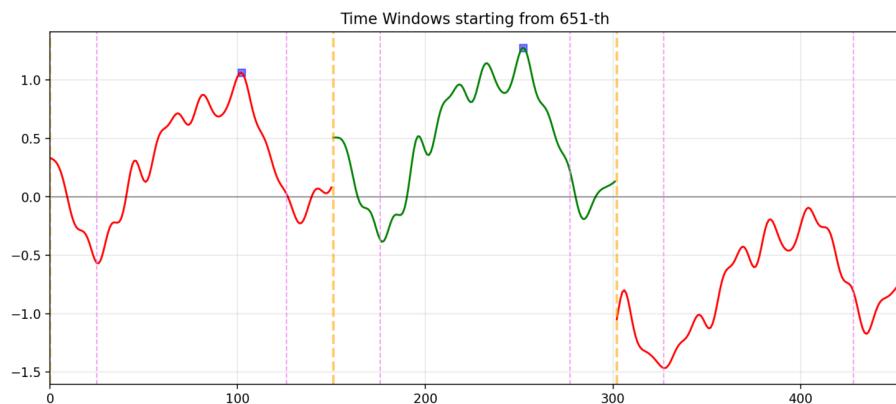


Figura 5.3: Rappresentazione delle finestre temporali

⁵Fonte: numpy.org

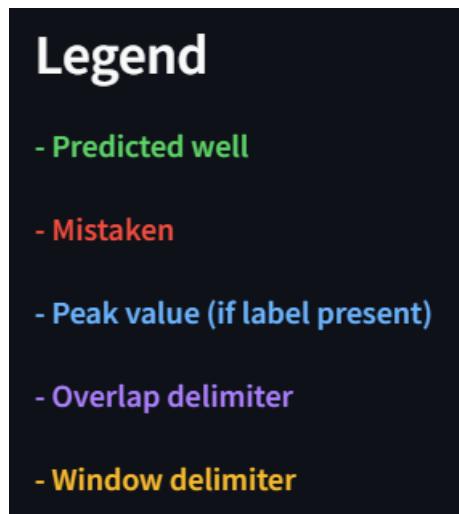


Figura 5.4: Legenda

5.3.2 Performance over all the subjects

Se abbiamo analizzato dati relativi a soggetti multipli, vengono plottate le varie metriche discusse nella sottosezione 5.2.1 (inclusa la *loss*) in due grafici separati, in funzione di ogni soggetto analizzato (Figura 5.5).

5.3.3 Download di risultati e immagini in formato .zip

Nel caso l'utente voglia salvare nel proprio filesystem i risultati dell'analisi appena conclusa può farlo tramite due appositi pulsanti (Figura 5.5).

File di testo

Un pulsante è dedicato al salvataggio dei risultati numerici visualizzati a schermo, viene creato un archivio **.zip** contenente un file per ogni soggetto contenente le performance della CNN per quel soggetto.

File immagini

Un altro pulsante invece si occupa del salvataggio delle immagini contenenti i grafici discussi nella sottosezione 5.3.1, nello stesso modo descritto precedentemente.

Figura 5.5: Risultati, metriche per ogni soggetto e pulsanti di download



5.4 Analisi di soggetti singoli

Segue un breve report sperimentale dell'analisi di singoli soggetti. Viene testata l'applicazione su file con overlap crescente, appartenenti al soggetto 6.

Overlap 25

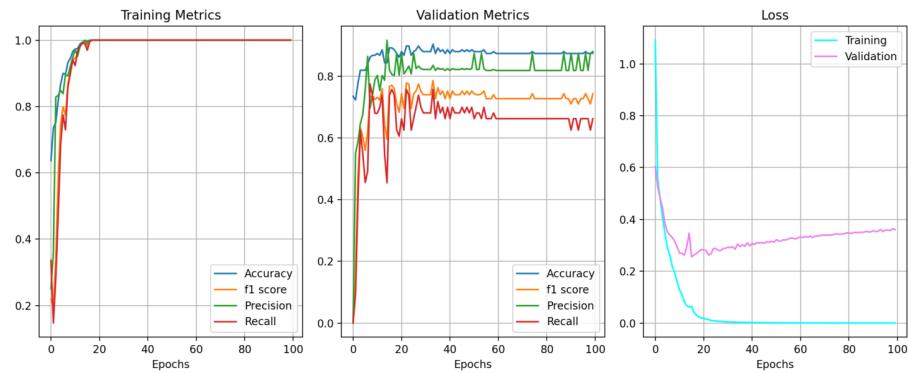


Figura 5.6: Metriche in addestramento e testing



Figura 5.7: Risultati numerici

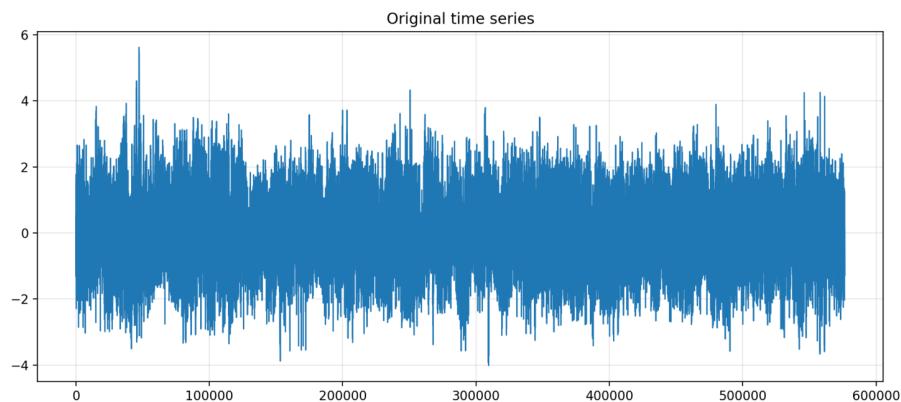


Figura 5.8: Serie temporale con applicato l'overlap indicato

Overlap 125

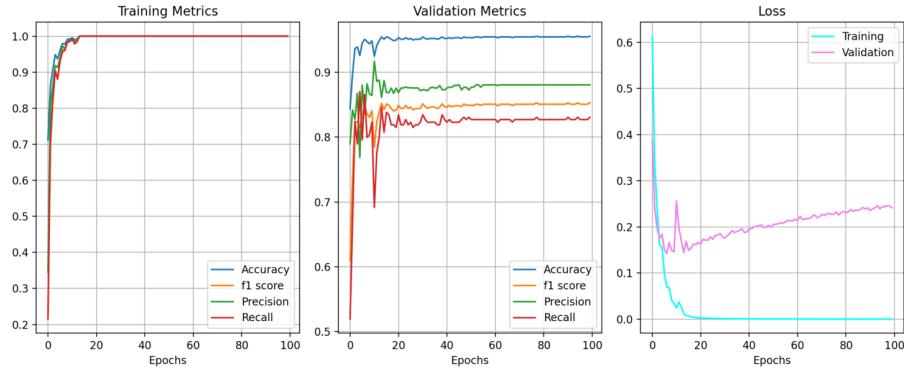


Figura 5.9: Metriche in addestramento e testing

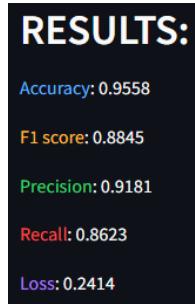


Figura 5.10: Risultati numerici

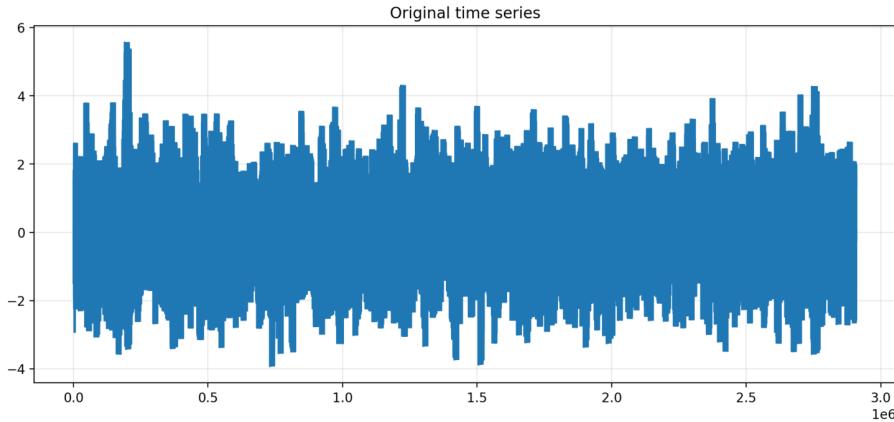


Figura 5.11: Serie temporale con applicato l'overlap indicato

Si vede chiaramente una crescita dell'accuracy ma soprattutto del punteggio F1 con l'aumentare della sovrapposizione temporale applicata, il che indica che per soggetti singoli un maggiore overlap è indice di maggiore precisione della rete neurale.

Il che ha senso considerando il fatto che si hanno più istanti temporali ‘condensati’ assieme grazie alla sovrapposizione, cos che lavora molto bene con la convoluzione applicata dalla rete.

5.5 Analisi di soggetti multipli

Segue un breve report sull'analisi dei risultati di soggetti multipli, nello specifico sono stati analizzati 5 soggetti variando il parametro di sovrapposizione da 25 fino a 125.

Per avere un'idea generale delle analisi sono riportati solo i risultati dei casi migliori e peggiori in termini di *accuracy*

Overlap 25

Best case: soggetto 3

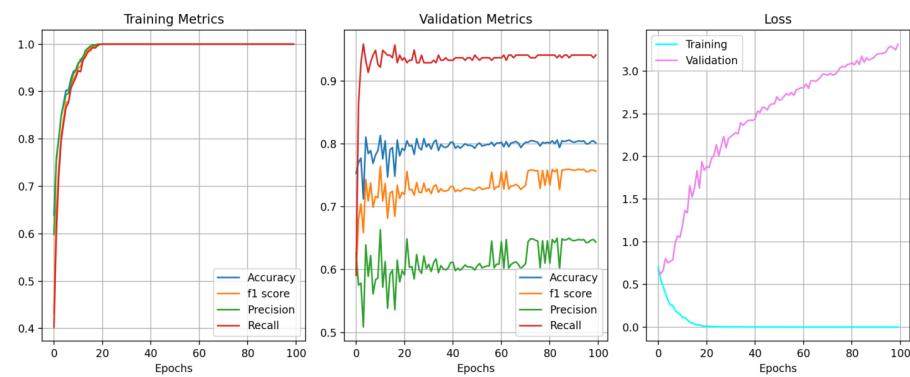


Figura 5.12: Metriche in addestramento e testing

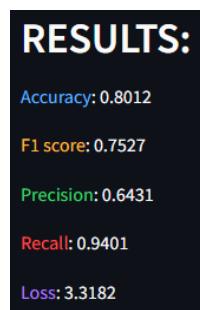


Figura 5.13: Risultati numerici

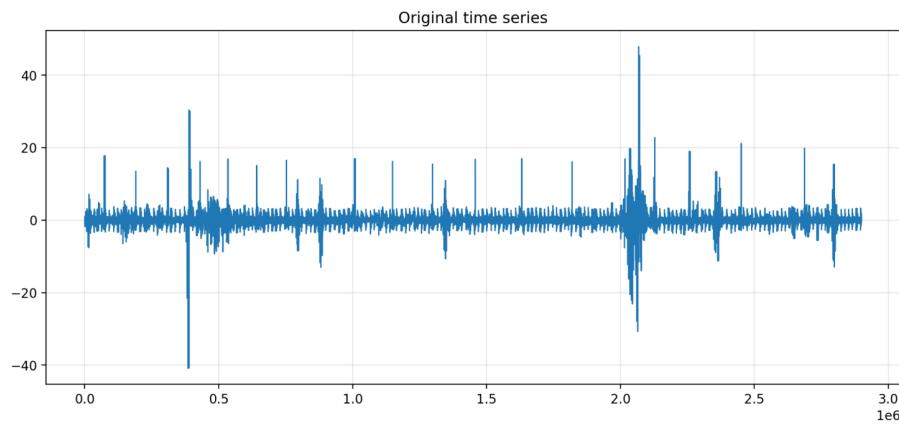


Figura 5.14: Serie temporale con applicato l'overlap indicato

Worst case: soggetto 5

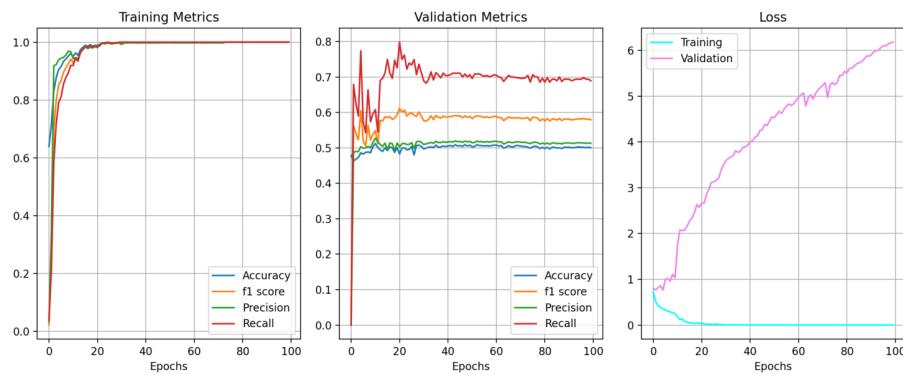


Figura 5.15: Metriche in addestramento e testing

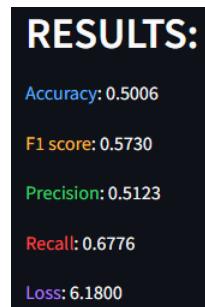


Figura 5.16: Risultati numerici

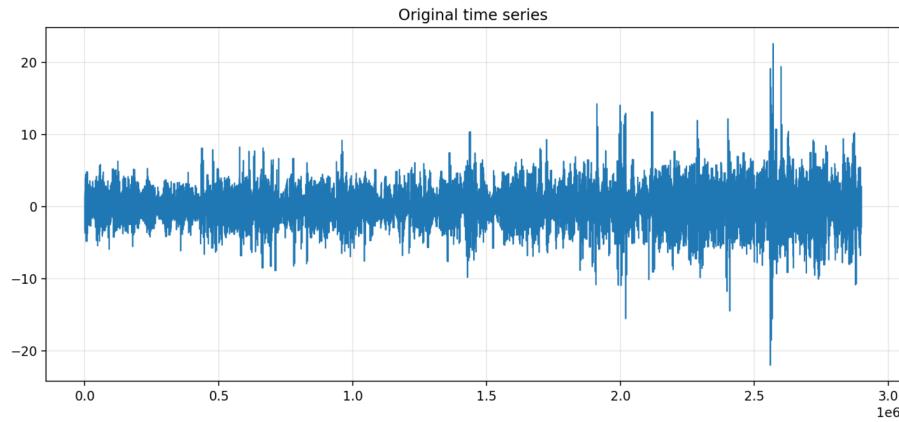


Figura 5.17: Serie temporale con applicato l'overlap indicato

Overlap 125

Best case: soggetto 3

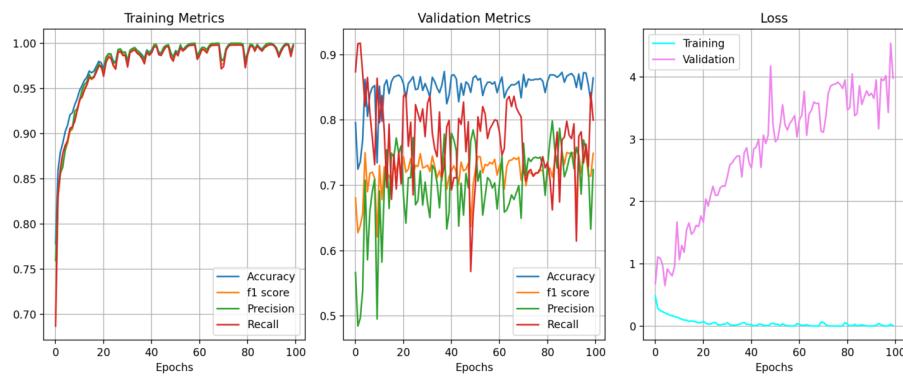


Figura 5.18: Metriche in addestramento e testing

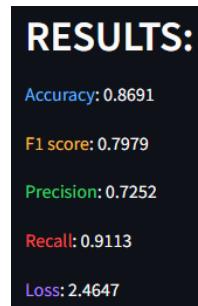


Figura 5.19: Risultati numerici

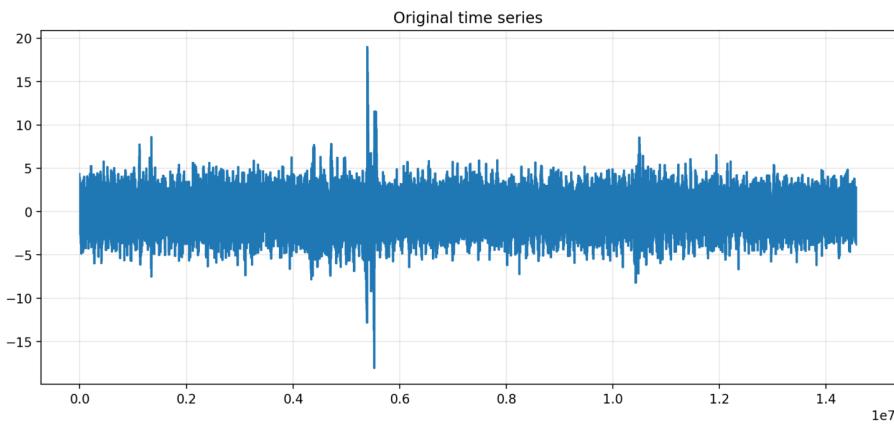


Figura 5.20: Serie temporale con applicato l'overlap indicato

Worst case: soggetto 5

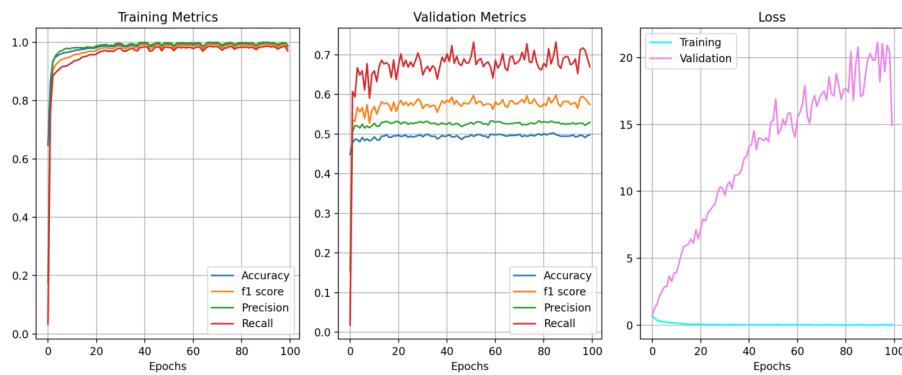


Figura 5.21: Metriche in addestramento e testing

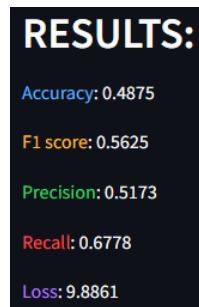


Figura 5.22: Risultati numerici

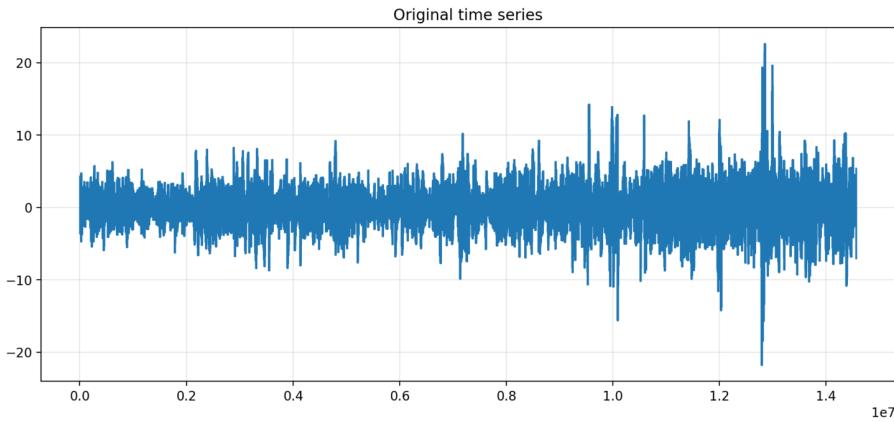


Figura 5.23: Serie temporale con applicato l'overlap indicato

Considerazioni

Da questa analisi si può concludere che il modello cattura meglio i dettagli in caso di minore variabilità dei dati lungo tutta la serie temporale.

In questo contesto multisoggetto la rete fatica a generare un modello che riesca a ‘comprendere’ il problema a fondo, fatto che si evince dai valori particolarmente alti del parametro di loss.

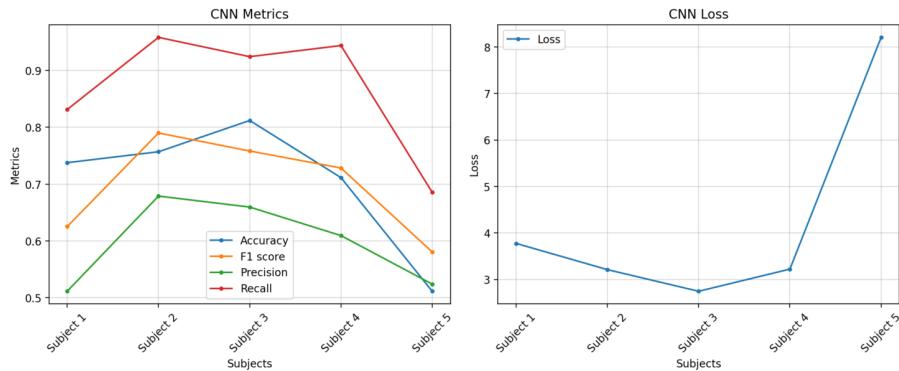


Figura 5.24: Riassunto dell'esperimento

Ciononostante riesce a mantenersi nei parametri accettabili di accuracy nei casi non affetti da anomalie.

Si riporta inoltre una tabella che riassume il valore dell'**Accuracy** e del **F1 score** (in punti percentuali) prodotto dalla CNN in fase di validazione in base al tipo di analisi e al valore di overlap.

overlap	25	50	100	125
monosoggetto				
S6	87.95	89.9	93.78	95.58
multisoggetto				
S1	69.4	75.8	85.6	86.4
S2	75.5	73.4	75.9	78.5
S3	80.2	82.5	85.1	83.4
S4	67.5	64.7	65.8	62.3
S5	50.2	47.9	49.5	49.9
μ	68.56	68.9	72.38	72.1
σ	10.22	11.93	13.53	13.87

Tabella 5.1: Accuracy

overlap	25	50	100	125
monosoggetto				
S6	78.83	79.52	82.73	88.45
multisoggetto				
S1	62.5	64.5	70.1	72.8
S2	78.2	76.9	77.2	79.6
S3	76.1	76.8	79.3	76.2
S4	72.3	67.8	69.7	66.7
S5	67.5	54.9	57.3	56.7
μ	71.32	68.18	70.72	70.4
σ	5.72	8.25	7.71	8.07

Tabella 5.2: F1 score

Si nota come le parametrizzazioni migliori siano con overlap di 100 e 125 punti temporali, la maggiore incertezza si ha nel caso multisoggetto, mentre nell'altro le performance migliorano sempre.

Questi risultati e queste osservazioni ci portano a comprendere che nell'impiego di queste reti neurali per l'analisi di segnali correlati nel tempo, la sovrapposizione temporale gioca un ruolo *fondamentale* nell'incremento delle performance, come avevamo ipotizzato sin dall'inizio.

Capitolo 6

Considerazioni finali

6.1 Da dove siamo partiti

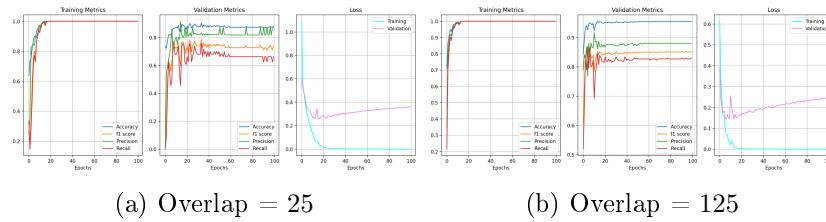
Riassumiamo quindi il nostro operato, siamo partiti da una serie temporale rappresentante il segnale dell'EEG poi trasfrmato e rimodellato come un tensore, suddiviso in finestre temporali affette da sovrapposizione e abbiamo testato il sistema con valori crescenti di questo parametro.

Ci siamo aspettati dunque un miglioramento della CNN con l'aumentare di quest'ultimo, fornendo dati temporali più ‘densi’ alla convoluzione.

6.2 L'ipotesi confermata

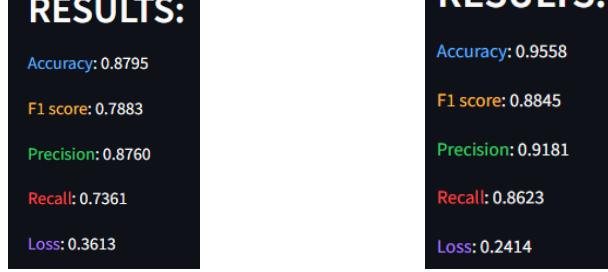
Abbiamo quindi notato un'incidenza netta dell'overlap nella predizione: nel caso di analisi monosoggetto un valore maggiore ha indicato un'accuracy e soprattutto un punteggio F1 maggiori, massimizzati con sovrapposizione pari a 125 unità.

Si riportano per comodità le immagini dell'analisi del soggetto 6 con overlap pari a 25 e 125



(a) Overlap = 25

(b) Overlap = 125



(a) Overlap = 25

(b) Overlap = 125

6.3 Interfaccia Human-Friendly

Abbiamo sviluppato un'applicazione che riesce a predire in maniera consistente l'ECG dei soggetti partendo dal relativo EEG, facendo uso di una interfaccia utente interattiva e semplice, per alleggerire il carico cognitivo dell'operatore, il quale può osservare a fine elaborazione i risultati della CNN in manierachiara tramite gli appositi grafici. Quelli intermedi servono ad avere un approfondimento per ciascun soggetto qualora fosse necessario.

Siamo stati in grado di rendere l'applicazione indipendente dalla capacità computazionale della macchina dell'utente, sfruttando il servizio di Google Colab.

6.4 Possibili sviluppi futuri

Studi futuri potranno quindi approfondire ulteriormente l'argomento analizzando quale sia la quantità giusta di overlap da inserire all'interno dei dati, magari infunzione di caratteristiche dei dati stessi o derivate dal problema.

Inoltre potrebbe essere un obiettivo riuscire a localizzare con precisione il qual'è la parte del segnale EEG interna alla finestra temporale che corrisponde effettivamente al segnale di battito cardiaco, in modo da restituire in output il tracciato completo di quest'ultimo.

Una volta sviluppato un modello di CNN opportunamente addestrato, che mantiene un'accuracy e un F1 score sopra a una certa soglia si potrebbe pensare di renderlo *unsupervised*, rimuovendo l'esigenza di fornire in input alla CNN anche le label di ogni soggetto.

Appendice A

Manuale d'uso

Il software prodotto è pensato in maniera procedurale, senza una visione orientata agli oggetti. Di conseguenza non sono state definite classi o tipi di dati aggregati.

Funzioni definite

`recall_m(y_true, y_pred)`: implementa il calcolo della metrica recall, prendendo in input gli array di label effettive e predette

`precision_m(y_true, y_pred)`: implementa il calcolo della metrica precision, prendendo in input gli array di label effettive e predette

`f1_m(y_true, y_pred)`: implementa il calcolo della metrica f1 score, prendendo in input gli array di label effettive e predette

`LOG(msg, lvl='INFO')`: funzione di utilità che stampa in console un messaggio, assieme al livello di importanza di esso, impostabile a piacere con valore di default 'INFO'.

`create_zip(strings, filename)`: funzione che permette la creazione dell'archivio contenente i file di testo in cui sono salvati i risultati dell'elaborazione. Utilizza dei buffer per la memorizzazione dei file temporanei che verranno inseriti nell'archivio

`create_image_zip(images, filename)`: funzione che permette la creazione dell'archivio contenente le immagini in cui sono salvati i grafici dell'elaborazione. Utilizza dei buffer per la memorizzazione dei file temporanei che verranno inseriti nell'archivio, con un metodo apposito per la scansione di immagini.

Codice sorgente

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import warnings
5 import streamlit as st
6 import time
7 import io
8 import zipfile
9 from tensorflow.keras.models import Sequential
10 from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
11 from sklearn.preprocessing import StandardScaler
12 # New import rule from tensorflow
13 from tensorflow.keras import backend as K
14
15 # Disable all warnings
16 warnings.filterwarnings("ignore")
17
18
19 st.set_page_config(
20     page_title="Convolutional Heart Rate Discovery",
21     page_icon="icons/heart-rate.png"
22 )
23
24 def recall_m(y_true, y_pred):
25     y_true = K.cast(y_true, "float32")
26     true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
27     possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
28     recall = true_positives / (possible_positives + K.epsilon())
29     return recall
30
31 def precision_m(y_true, y_pred):
32     y_true = K.cast(y_true, "float32")
33     true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
34     predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
35     precision = true_positives / (predicted_positives + K.epsilon())
36     return precision
37
38 def f1_m(y_true, y_pred):
39     y_true = K.cast(y_true, "float32")
40     precision = precision_m(y_true, y_pred)
41     recall = recall_m(y_true, y_pred)
42     return 2 * ((precision * recall) / (precision + recall + K.epsilon()))
43
44 def LOG(msg, lvl='INFO'):
45     print(f"[{lvl}]: {str(msg)}")
46
47 def create_zip(strings, filenames):
48     zip_buffer = io.BytesIO()
49
50     with zipfile.ZipFile(zip_buffer, "w", zipfile.ZIP_DEFLATED) as zip_file:
51         for string, filename in zip(strings, filenames):
52             zip_file.writestr(filename, string)
53
54     zip_buffer.seek(0)
55     return zip_buffer
56
57 def create_image_zip(images, filenames):
```

```

58     zip_buffer = io.BytesIO()
59
60     with zipfile.ZipFile(zip_buffer, "w", zipfile.ZIP_DEFLATED) as zip_file:
61         for image, filename in zip(images, filenames):
62             zip_file.writestr(filename, image.read())
63
64     zip_buffer.seek(0)
65     return zip_buffer
66 # ===== Data structures =====#
67
68 if "png_buffers" not in st.session_state:
69     st.session_state.png_buffers = []
70
71 if "txt_buffers" not in st.session_state:
72     st.session_state.txt_buffers = []
73
74 cnn_stats = {
75     'acc': [],
76     'f1': [],
77     'precision': [],
78     'recall': [],
79     'loss': []
80 }
81
82 # best, id, worst, id
83 cnn_best_worst_acc_case = [0, -1, np.inf, -1]
84
85 base_path = "C:/Users/Utente/Drake/UniPi/Tesi/RH_from_EEG_with_XAI/"
86
87 # ===== UI display =====#
88 st.title("Retrieve HR from EEG using a Convolutional Neural Network")
89 st.markdown('',
90             'Please upload files **which file name formats as follows**:  

91             subject\_*[i]*\_MADtsROLLINGrawCLASSIFICATION\_[DATA/LABELS]\_*[wlen]*\_  

92             *[overlap]*\_BK\_*[grps]*\_*[series]*.csv\n  

93             Where:\n
94             - **i** is the subject taken in exam
95             - **wlen** is the length of each time window
96             - **overlap** is the amount of data overlap between each time window
97             - **grps** is the number of groups in the time series
98             - **series** is the number of correlated channels
99             ')
100
101 overlap = st.number_input("Select the time window overlap", 25, 125, "min", 5,
102                           "%d")
103 n_points = st.number_input("Select the number of points per time window", 150,
104                           250, "min", 50, "%d") + 1
105 window_printed = st.number_input("Select the number of windows extracted per
106                                   subject", 1, 10, "min", 1, "%d")
107
108 data = []
109 data_files = st.file_uploader("Upload the EEG file for the data", type="csv",
110                             accept_multiple_files=True, help='Upload CSV files representing the EEG of
111                             the subjects')
112 for f in data_files:
113     data.append(pd.read_csv(f))
114
115 labels = []
116 label_files = st.file_uploader("Upload the labels", type="csv",
117                             accept_multiple_files=True, help='Upload CSV files representing the labels')

```

```

        for the subjects')
112 for f in label_files:
113     labels.append(pd.read_csv(f))

114
115 s = st.button('Start computation')
116
117 if st.button("Reload"):
118     st.rerun()
119
120 n_soggetti = len(data)
121 n_series = 23 # Numero di canali correlati
122
123 multiple_subj = True
124
125 totalTime = 0
126
127 if (len(data) == len(labels)) and data and labels and s:
128     scaler = StandardScaler()
129     with st.spinner('Working...'):
130         n_samples = int(data_files[0].name[-11:-7])
131         for i in range(n_soggetti):
132             if n_soggetti < 2:
133                 LOG("Loading data and labels")
134
135             multiple_subj = False
136
137         X = data[0].values.reshape(n_samples, n_points, n_series)
138         y = labels[0].values
139
140         test_idx_end = int((n_samples // 2) + int(n_samples * 0.1))
141         test_idx_begin = int((n_samples // 2) - int(n_samples * 0.1))
142
143
144         # Create a boolean mask to select elements to keep for
145         # training and testing sets
146         mask_train = np.ones(X.shape[0], dtype=bool)
147         mask_train[test_idx_begin:test_idx_end] = False
148         X_train = X[mask_train, :, :]
149         y_train = y[mask_train]
150
151         mask_test = np.zeros(X.shape[0], dtype=bool)
152         mask_test[test_idx_begin:test_idx_end] = True
153         X_test = X[mask_test, :, :]
154         y_test = y[mask_test]
155
156         X_train_2D = X_train.reshape(X_train.shape[0], -1)
157         X_test_2D = X_test.reshape(X_test.shape[0], -1)
158
159         # Apply StandardScaler
160         X_train = scaler.fit_transform(X_train_2D)
161         X_test = scaler.transform(X_test_2D)
162
163         # Reshape back to 3D for CNN
164         X_train = X_train.reshape(X_train.shape[0], n_points, n_series
165                                 )
166         X_test = X_test.reshape(X_test.shape[0], n_points, n_series)
167
168         LOG("Data and labels loaded!")
169         LOG(f'Train data: {X_train.shape}, test data: {X_test.shape},
170             train labels: {y_train.shape} test labels: {y_test.shape}',
```

```

168     else:
169         #----- DATA GENERATION -----
170         if n_soggetti > 1:
171             n_samples_overall = 0
172             X_train = []
173             X_test = []
174             LOG(f"Loading data of subject {i+1}...")
175             for j in range(n_soggetti):
176                 # Dataset di test
177                 if j == i:
178                     X_test = data[i].values
179                     X_test = scaler.fit_transform(X_test)
180                     X_test = X_test.reshape(n_samples, n_points,
181                                         n_series)
182                     continue
183
184                 # Dataset di training
185                 # Estende la serie
186                 X_train.extend(data[j].values)
187                 n_samples_overall += int(data_files[j].name[-11:-7])
188
189                 X_train = scaler.fit_transform(X_train)
190                 X_train = X_train.reshape(n_samples_overall, n_points,
191                                         n_series)
192
193             LOG("Data loaded!")
194             LOG(f'Train data: {X_train.shape}')
195             LOG(f'Test data: {X_test.shape}')
196
197             #----- LABEL GENERATION -----
198
199             y_train = []
200             y_test = []
201             first_iter = True
202             LOG(f"Loading labels of subject {i+1}...")
203
204             for j in range(n_soggetti):
205                 # Label di testing
206                 if j == i:
207                     y_test = labels[i].values
208                     continue
209                 # Label di training
210                 if first_iter:
211                     y_train = labels[j].values
212                     first_iter = False
213                 else:
214                     y_train = np.append(y_train, labels[j].values, axis=0)
215
216             LOG("Labels loaded!")
217             LOG(f'Training labels: {y_train.shape}')
218             LOG(f'Testing labels: {y_test.shape}')
219
220             #----- CNN -----
221             model = Sequential()
222             model.add(Conv1D(filters=64, kernel_size=5, activation='relu',
223                             input_shape=(n_points, n_series)))
224             model.add(MaxPooling1D(pool_size=2))
225             model.add(Flatten())
226             model.add(Dense(128, activation='relu'))
227             model.add(Dense(1, activation='sigmoid'))

```

```

226     # Compile the model
227     model.compile(loss='binary_crossentropy', optimizer='adam',
228                     metrics=['accuracy', f1_m, precision_m, recall_m])
229
230     startTime = time.time()
231     # Training
232     history = model.fit(X_train, y_train,
233                           epochs=100, batch_size=64,
234                           validation_data=(X_test, y_test),
235                           verbose=0)
236
237     accuracy = history.history['accuracy']
238     val_accuracy = history.history['val_accuracy']
239
240     f1 = history.history['f1_m']
241     val_f1 = history.history['val_f1_m']
242
243     precision = history.history['precision_m']
244     val_precision = history.history['val_precision_m']
245
246     recall = history.history['recall_m']
247     val_recall = history.history['val_recall_m']
248
249     loss = history.history['loss']
250     val_loss = history.history['val_loss']
251
252     st.header(f'Subject {i+1}:')
253
254     # training
255     plt.figure(figsize=(12, 5))
256     plt.subplot(1, 3, 1)
257     plt.plot(accuracy, label='Accuracy')
258     plt.plot(f1, label='f1 score')
259     plt.plot(precision, label='Precision')
260     plt.plot(recall, label='Recall')
261     plt.xlabel('Epochs')
262     plt.title('Training Metrics')
263     plt.grid(True)
264     plt.legend()
265
266     # validation
267     plt.subplot(1, 3, 2)
268     plt.plot(val_accuracy, label='Accuracy')
269     plt.plot(val_f1, label='f1 score')
270     plt.plot(val_precision, label='Precision')
271     plt.plot(val_recall, label='Recall')
272     plt.xlabel('Epochs')
273     plt.title('Validation Metrics')
274     plt.grid(True)
275     plt.legend()
276
277     # loss
278     plt.subplot(1, 3, 3)
279     plt.plot(loss, label='Training', color='cyan')
280     plt.plot(val_loss, label='Validation', color='violet')
281     plt.xlabel('Epochs')
282     plt.title('Loss')
283     plt.grid(True)
284     plt.legend()
285
286     plt.tight_layout()

```

```

287     st.pyplot(plt)
288
289     # Testing
290     loss, accuracy, f1, precision, recall = model.evaluate(X_test,
291                 y_test, verbose=0)
292
293     if multiple_subj:
294         if accuracy > cnn_best_worst_acc_case[0]:
295             cnn_best_worst_acc_case[0] = accuracy
296             cnn_best_worst_acc_case[1] = i+1
297         if accuracy < cnn_best_worst_acc_case[2]:
298             cnn_best_worst_acc_case[2] = accuracy
299             cnn_best_worst_acc_case[3] = i+1
300
301     cnn_stats['acc'].append(accuracy)
302     cnn_stats['f1'].append(f1)
303     cnn_stats['loss'].append(loss)
304     cnn_stats['precision'].append(precision)
305     cnn_stats['recall'].append(recall)
306
307     # model predictions
308     predictions = model.predict(X_test)
309     y_pred = (predictions > 0.5).astype(int)
310
311     res = f'''blue[Accuracy]: {cnn_stats["acc"][i]:.4f}\n
312 :orange[F1 score]: {cnn_stats["f1"][i]:.4f}\n
313 :green[Precision]: {cnn_stats["precision"][i]:.4f}\n
314 :red[Recall]: {cnn_stats["recall"][i]:.4f}\n
315 :violet[Loss]: {cnn_stats["loss"][i]:.4f}
316
317     endTime = time.time()
318     totalTime += endTime - startTime
319
320     st.divider()
321     st.header(f'RESULTS :')
322     st.markdown(res)
323
324     if len(st.session_state.txt_buffers) <= i:
325         st.session_state.txt_buffers.append(f'{i+1}) {time.strftime("%
326 d-%m @ %H:%M")}\n' + res)
327
328     r = np.random.randint(0, len(y_test))
329     LOG(f'Starting label: {r}')
330
331     start = n_points * r
332     end = n_points * (r + window_printed)
333     t_window = X_test.flatten()[start:end]
334
335     plt.figure(figsize=(12, 5))
336     plt.plot(X_test.flatten(), linewidth=1)
337     plt.grid(alpha=0.3)
338     plt.title('Original time series')
339     st.pyplot(plt)
340
341     plt.figure(figsize=(12, 5))
342     plt.plot(t_window, linewidth=1)
343     plt.grid(alpha=0.3)
344     plt.title('Original time windows')
345     st.pyplot(plt)
346
347     plt.figure(figsize=(12, 5))

```

```

346         for k in range(window_printed):
347             to_print = t_window[k*n_points:(k+1)*n_points]
348
349             if y_test[k]:
350                 plt.scatter(np.argmax(to_print)+(k*n_points), np.max(
351                     to_print), color='blue', marker='s', alpha=0.5)
352             color = 'green' if y_pred[k] and y_test[k] else 'red'
353             plt.plot(range(n_points*k, n_points*(k+1)), to_print, color=
354                     color)
355
356             plt.axvline(x=(k*n_points), color='orange', linestyle='--',
357                         linewidth=2, alpha=0.6)
358             plt.axvline(x=len(to_print)*(k+1)-overlap, color='violet',
359                         linestyle='--', linewidth=1, alpha=0.8)
360             plt.axvline(x=overlap+k*n_points, color='violet', linestyle='
361                         --', linewidth=1, alpha=0.8)
362             plt.axhline(y=0, color='black', alpha=0.4, linewidth=1)
363             plt.xlim([0, len(t_window)])
364             plt.title(f'Time Windows starting from {r}-th')
365             plt.grid(alpha=0.3)
366
367             st.pyplot(plt)
368
369             st.subheader('Legend')
370             st.markdown('',
371                         **:green[- Predicted well]**\n
372                         **:red[- Mistaken]**\n
373                         **:blue[- Peak value (if label present)]**\n
374                         **:violet[- Overlap delimiter]**\n
375                         **:orange[- Window delimiter]**\n
376                         '')
377
378             img_buf = io.BytesIO()
379             plt.savefig(img_buf, format='png')
380             img_buf.seek(0)
381
382             if len(st.session_state.png_buffers) <= window_printed:
383                 st.session_state.png_buffers.append(img_buf)
384
385             st.divider()
386
387             if multiple_subj:
388                 st.write(f'Best Accuracy score: {cnn_best_worst_acc_case[0]:.6f} with
389                         subject {cnn_best_worst_acc_case[1]}')
390                 st.write(f'Worst Accuracy score: {cnn_best_worst_acc_case[2]:.6f} with
391                         subject {cnn_best_worst_acc_case[3]}')
392
393                 st.write(f'Avg computation time: {totalTime/n_soggetti:.2f}')
394
395                 sub_label = [f'Subject {j+1}' for j in range(n_soggetti)]
396
397                 plt.figure(figsize=(12, 5))
398                 plt.subplot(1, 2, 1)
399                 plt.plot(cnn_stats['acc'], label='Accuracy', marker='.', linestyle='-
400                         ')
401                 plt.plot(cnn_stats['f1'], label='F1 score', marker='.', linestyle='-
402                         ')
403                 plt.plot(cnn_stats['precision'], label='Precision', marker='.', linestyle='-
404                         ')
405                 plt.plot(cnn_stats['recall'], label='Recall', marker='.', linestyle='-
406                         ')
407
408                 plt.xlabel('Subjects')

```

```

397     plt.ylabel('Metrics')
398     plt.xticks(np.array(range(n_soggetti)), sub_label, rotation=45)
399     plt.title('CNN Metrics')
400     plt.grid(alpha=0.5)
401     plt.legend()
402
403     plt.subplot(1, 2, 2)
404     plt.plot(cnn_stats['loss'], label='Loss', marker='.', linestyle='--')
405     plt.xlabel('Subjects')
406     plt.ylabel('Loss')
407     plt.xticks(np.array(range(n_soggetti)), sub_label, rotation=45)
408     plt.title('CNN Loss')
409     plt.grid(alpha=0.5)
410     plt.legend()
411
412     plt.tight_layout()
413     st.pyplot(plt)
414
415     filenames = [f"Subject_{i+1}.txt" for i in range(n_soggetti)]
416     txt_zip = create_zip(st.session_state.txt_buffers, filenames)
417     st.download_button(
418         label="Download Results",
419         data = txt_zip,
420         file_name='archivio.zip',
421         mime = "application/zip",
422         key = 'Text'
423     )
424
425     filenames = [f"img_{i+1}.png" for i in range(n_soggetti*window_printed)]
426     png_zip = create_image_zip(st.session_state.png_buffers, filenames)
427     st.download_button(
428         label="Download Images",
429         data = png_zip,
430         file_name= "archivio_immagini.zip",
431         mime = "application/archive",
432         key = 'Images'
433     )

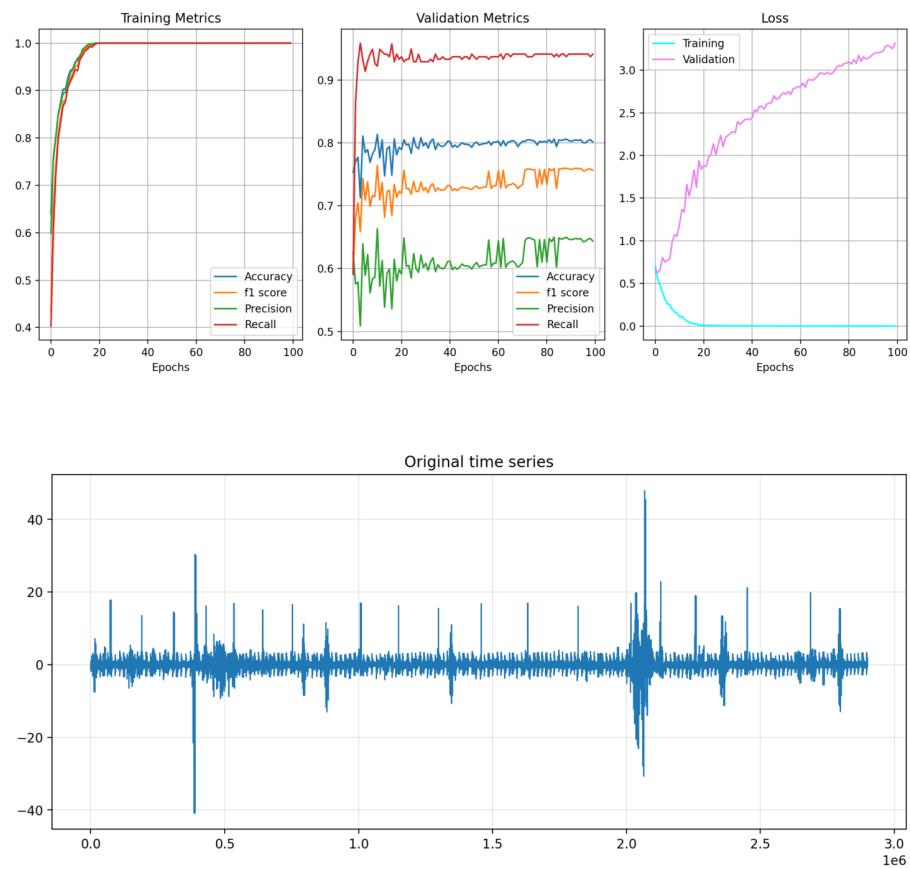
```

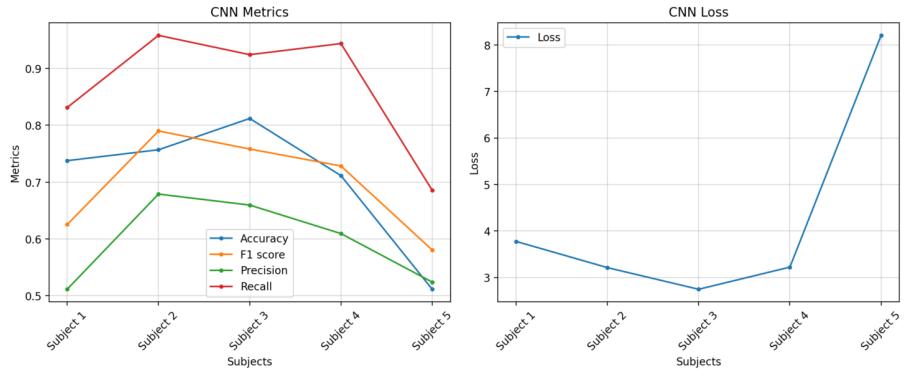

Appendice B

Grafici

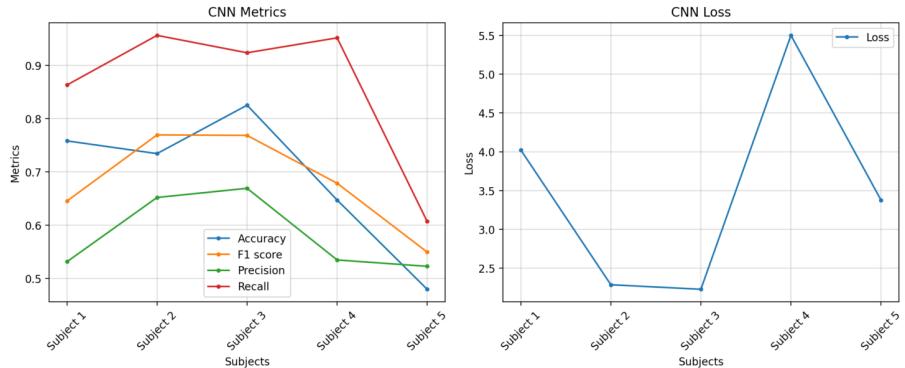
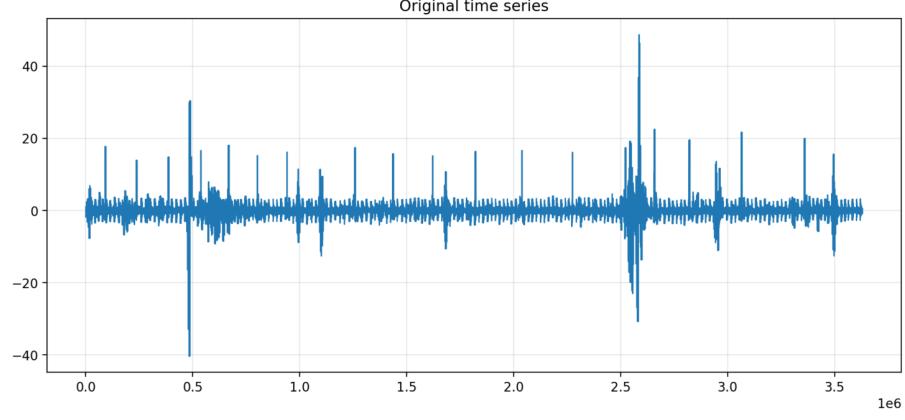
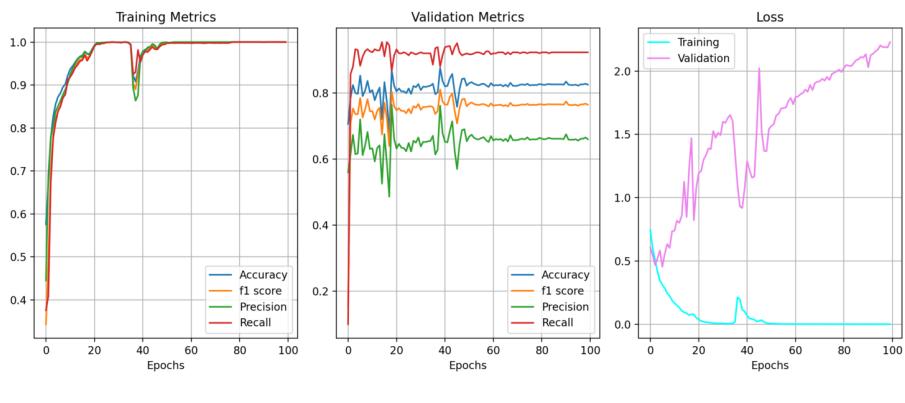
B.1 Grafici di test su soggetti multipli per ogni valore di overlap

B.1.1 Overlap 25

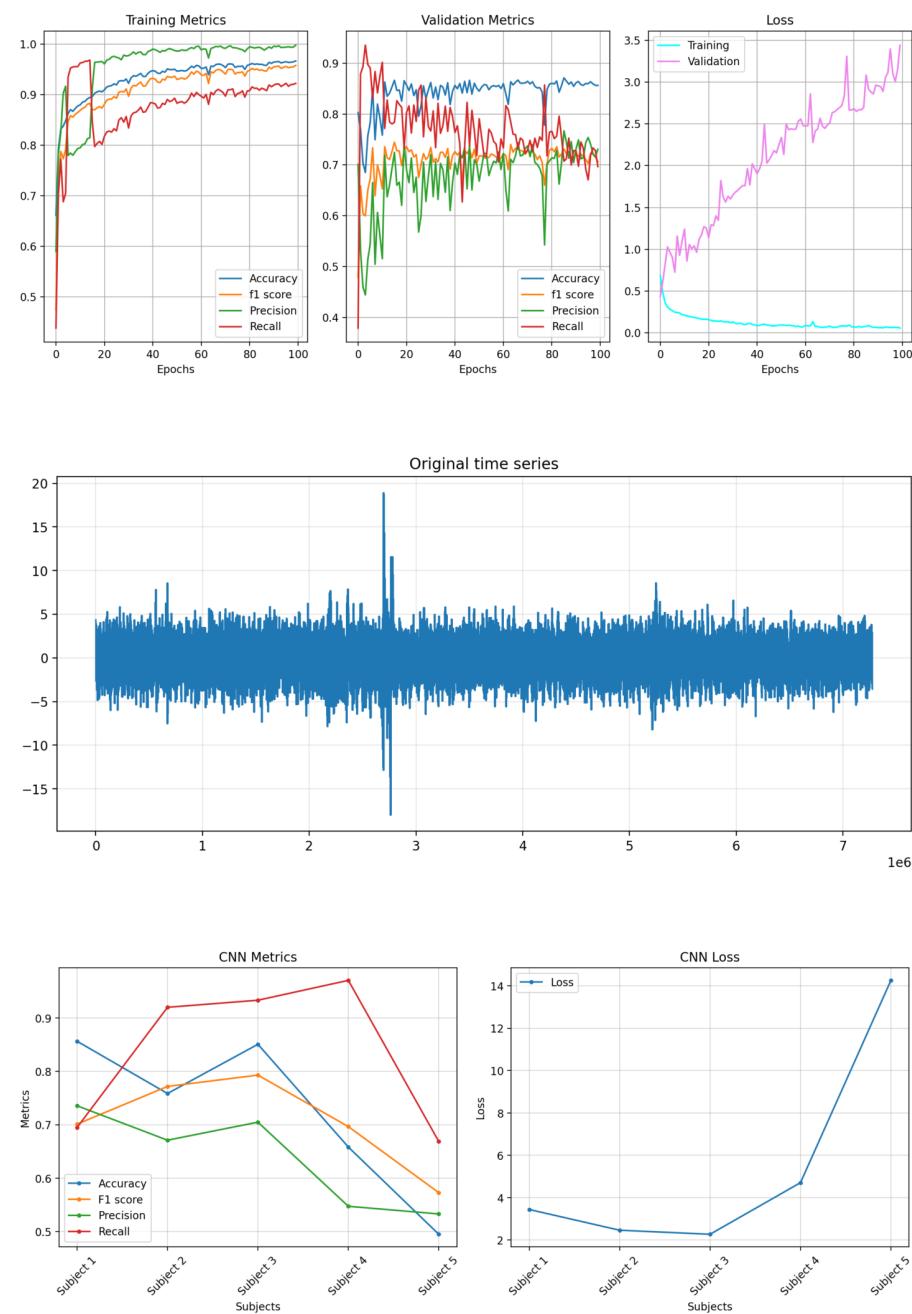




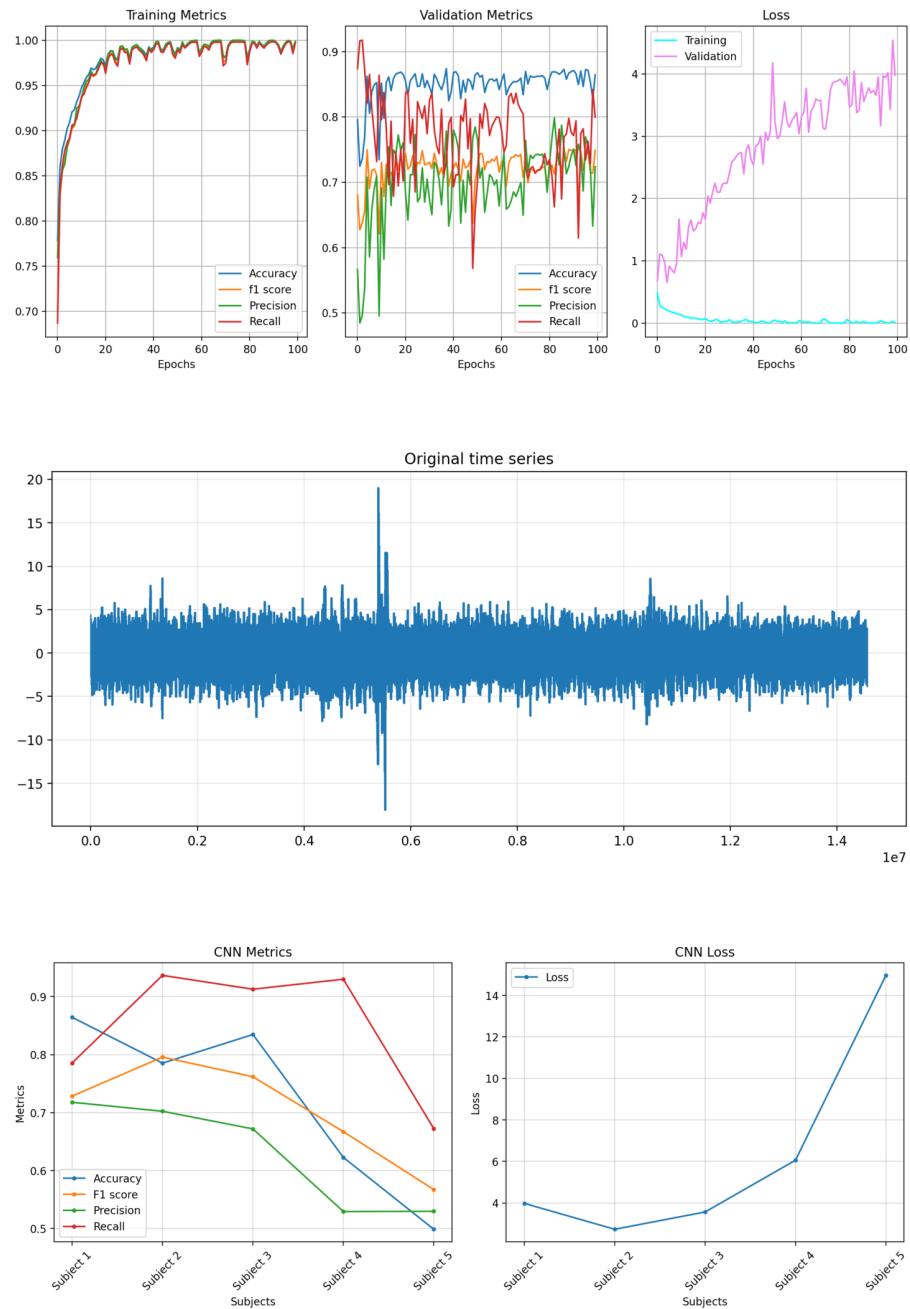
B.1.2 Overlap 50



B.1.3 Overlap 100



B.1.4 Overlap 125



*A tutti i miei amici, ai miei
colleghi, e a te...*

Bibliografia

- [1] A. Meini, “Sperimentazione e testing di una rete neurale convoluzionale per la ricostruzione del battito cardiaco partendo dall’elettroencefalogramma,” 2024.
- [2] J. Souza and C. K. Leung, “Explainable artificial intelligence for predictive analytics on customer turnover: A user-friendly interface for non-expert users,” *Explainable AI within the digital transformation and cyber physical systems*, 2021.
- [3] A. Choudhury, “AI in healthcare Improving human interface for patient safety,” *Explainable AI within the digital transformation and cyber physical systems*, 2020.
- [4] A. P. Wibawa and A. B. P. Utama, “Time series analysis with smoothed Convolutional Neural Network,” *Journal of Big Data*, 2022.
- [5] Y. L. W L Mao, H I K Fathurrahman and T. W. Chang, “Eeg dataset classification using CNN method,” *Journal of Physics: Conference Series*, 2020.
- [6] A. Haya, “Convolutional Neural Network Application in Biomedical Signals,” *ResearchGate*, 2018.
- [7] R. D. Luca, “Realizzazione di una rete neurale per la rilevazione dei battiti cardiaci dalle componenti indipendenti dell’elettroencefalogramma,” 2024.