Tool CNN per analisi e predizione di crisi epilettiche

Progetto di gruppo

Studenti: Manuel Cretone, Emilio Silvestri
Tutor: Prof. Marco Piangerelli

Unversità degli Studi di Camerino
Scuola di Scienze e tecnologie
Corso di laurea in Informatica L-31
Anno Accademico 2018-2019

Indice

Introduzione

Il progetto è stato realizzato nell'ambito del group progect finale previsto dal piano di studi della laurea in Informatica dell'Università di Camerino. Lo scopo principale è quello di attuare delle predizioni di crisi epilettiche su tracciati EEG che registrano le attività del cervello umano a livello neurale. Le predizioni vengono effettuate con l'ausilio di una rete neurale allenata per riconoscere le varie crisi presenti. Queste predizioni vengono eseguite su file in formato *edf* (European Data Format) i quali permettono l'archiviazione di segnali biologici e fisici multicanale. Il tool non solo permette di predirre le crisi presenti in un tracciato EEG, ma dispone di altre funzionalità come il calcolo di alcune statistiche che riguardano tutti i segnali e tutti i canali presenti nel tracciato e la visualizzazione del tracciato stesso. Viene inoltre fornita all'utente l'opportunità di creare un proprio dataset di allenamento, da utilizzare per il training di reti neurali personalizzate. Il workflow del progetto è stato suddiviso in:

- studio EEG e teoria delle crisi
- studio per la creazione del dataset (segnali ictali, preictali e postictali)
- creazione di dataset bilanciati
- studio teorico CNN
- applicazione CNN con relativo allenamento
- costruzione architettura web suddivisa in backend e frontend

1 Epilessia

L'epilessia è una malattia neurologica che colpisce circa 50 milioni di persone nel mondo. Essa è caratterizzata da specifici eventi clinici, definiti crisi epilettiche, che si ripetono in modo consecutivo e attivano simultaneamente un grande numero di neuroni generalemente posti in un'area particolare dell'encefalo, detta corteccia celebrale. Le crisi alterano la normale attività elettrica dell'encefalo, con conseguenze sul comportamento del soggetto colpito, come perdita di conoscenza, movimenti involontari, contrazioni della muscolatura che si tramutano in convulsioni. Le cause delle crisi epilettiche possono essere diverse, come patologie, traumi cranici, infenzioni SNC (meningite ecc..) e perfino alcuni farmaci.

In base all'area di estensione del cervello che è interessata da scariche elettriche anomale, possono presentarsi due tipi di crisi:

- Crisi Parziali: si manifestano convulsioni e alterazioni sensoriali. Esse si suddividono a loro volta da crisi semplici o complesse
  - Crisi parziale semplice: interessa una piccola regione del cervello, come il lobo temporale o l'ippocampo. questo tipo di crisi spesso precede una crisi peggiore, dove l'anomala attività elettrica coinvolge aree più vaste del cervello.
  - Crisi parziale complessa: Interessa aree più vaste del cervello e non è concentrata solo su alcune di esse. Questo tipo di crisi è più dannosa perchè scaturisce nel soggetto coinvolto una perdita di coscienza.
- Generalizzate: crisi più frequenti rispetto alle parziali, esse vengono generate a causa di un'alterazione dell'attività neuronale di entrambi gli emisferi.

In alcuni casi dopo uno o più episodi di crisi epilettiche il fenomeno può tendere a scomparire per cause naturali senza l'apporto di farmaci per la sua cura. In molti casi però c'è bisogno di una cura effettuata attraverso farmaci, se questi ultimi non dovrebbero essere efficaci c'è bisogno di ricorre a operazioni chirurgiche. I farmaci risultano efficaci nel 60-70 per cento dei casi; questi non curano l'epilessia ma permettono al soggetto affetto di convivere con questa patologia. La diagnosi dell'epilessia viene fatta attraverso diversi strumenti, uno su tutti è l'elettroencefalogramma (EEG). Questo strumento registra le attività elettriche dell'encefalo, poi riprodotte su dei tracciati grafici che contengono diversi canali, che mostrano delle onde corrispondenti agli elettrodi applicati sullo scalpo del soggetto. Attraverso questo strumento è possibile individuare le cause di una crisi epilettica e le alterazioni elettriche che avvengono all'interno del cervello sia in condizioni normali, di riposo, sia in situazioni di crisi. In base alle caratteristiche del tracciato risultante i medici possono dedurre di che tipo di epilessia si tratta e quale cura è meglio prescrivere.

2 Reti Neurali

Le reti neurali sono modelli matematici basati su neuroni artificiali ispirati al funzionamento biologico del cervello umano. Inventate intorno alla metà degli anni '80, sono tornate in auge nell'ultimo decennio come strumento di risoluzione in svariati campi, dall'informatica, all'elettronica, alla simulazione. Prendendo spunto dal cervello animale, una rete neurale (ANN, Artificial Neural Network) è composta da numerosi neuroni connessi fra loro. Ogni connessione, come le sinapsi cerebrali, trasmette un segnale da un neurone all'altro. Il neurone che riceve un segnale può processarlo e inviarlo ai neuroni a cui è a sua volta connesso. Tradizionalmente, un segnale trasmesso tra due neuroni è rappresentato da un numero reale, moltiplicato per un certo peso che indica la forza della connessione. Ogni neurone esegue una somma di tutti i segnali in ingresso, moltiplicandoli per il suo set di pesi. Tale somma viene poi modellata da una funzione di attivazione, tipicamente la sigmoide $\sigma(x) = \frac{1}{1+e^{-}x}$. Un altro parametro associato ai neuroni è il *bias*, un ulteriore peso che si considera collegato ad un input fittizio con valore sempre 1, che viene sommato ai valori di input moltiplicati per i pesi. Questo peso è utile per tarare il punto di lavoro ottimale del neurone.

In una rete neurale i neuroni sono solitamente raggruppati in più layer, che applicano diverse trasformazioni agli input. I segnali in ingresso viaggiano sequenzialmente dal primo layer (Input layer) all'ultimo (Output layer), attraversando un numero n di layer intermedi (Hidden layer). Questo tipo di rete è detta feedforward, ed è senza dubbio la più diffusa. Modelli più complessi sono quelli di tipo recurrent, che prevedono connessioni di feedback verso neuroni dello stesso livello o anche all'indietro, ma non saranno trattate in questa istanza.

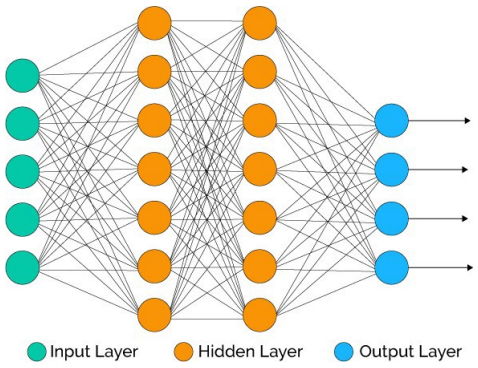

Figura 1: Tipica architettura ANN feedforward

La peculiarità delle reti neurali rispetto all'utilizzo di sistemi di risoluzione tradizionali è l'apprendimento. Le ANN vengono addestrate per capire come dovranno comportarsi nel momento in cui andranno a risolvere un problema ingegneristico. L'approccio orientato all'apprendimento si basa sui principi

del Machine Learning, la disciplina che studia i diversi metodi matematico-computazionali per apprendere informazioni dall'esperienza. Possiamo distinguere quattro principali metodologie di apprendimento:

- Supervised Learning: la rete riceve un input e il relativo risultato atteso. L'obiettivo è quello di identificare una regola generale che colleghi i dati in ingresso con quelli in uscita.
- Unsupervised Learning: la rete riceve solo set di dati in input, senza alcuna indicazione del risultato desiderato. Lo scopo è quello di risalire a schemi nascosti tra gli input, in modo da identificarne una struttura logica.
- Apprendimento per rinforzo: il comportamento del sistema è determinato da una routine di apprendimento basata su ricompensa se l'obiettivo è raggiunto, o punizione se viene commesso un errore.
- Apprendimento semi-supervised: modello ibrido basato sui primi due, in cui solo ad una parte dei dati in input è associato il rispettivo output atteso.

L'allenamento supervisionato, quello implementato in questo progetto, prevede diversi passi:

- Inizializzazione: il modello viene inizializzato con valori casuali per bias e pesi.
- Feed-forward: il modello prende i dati in input e restituisce un output, rappresentante la predizione.
- Loss function: viene calcolata la differenza tra l'output atteso e quello calcolato dalla rete. Fornisce quindi una metrica sull'accuratezza della rete. Inizialmente la precisione sarà molto bassa. L'obiettivo dell'allenamento è appunto quello di minimizzare il loss per aumentare la precisione della rete.

CrossEntropyLoss

- Differentiation: calcolando la derivata della funzione di loss, vengono identificate le modifiche da applicare ai pesi per diminuire l'errore sul risultato. Lo standard de facto per effettuare tale ottimizzazione è lo *stochastic gradient descent*. Tale metodo prevede che per il calcolo del gradiente (il vettore contenente le derivate perziali della funzione di loss) venga preso solo un sottoinsieme dei valori, in modo da ridurre la complessità del calcolo.

adam optimizer

- Backpropagation and weights update: i valori dei pesi vengono aggiornati, a partire dal layer di output fino ad arrivare al primo layer, per far sì che il loss venga minimizzato. La retropropagazione dell'errore prevede la selezione di una costante rappresentata da un numero decimale molto piccolo, che viene moltiplicato per l'incremento. Grazie a questo passaggio i pesi vengono modificati più lentamente in modo da centrare il minimo della funzione di loss.

Tutti questi passi vengono ripetuti sequenzialmente per tutti i dati di input diverse volte. Una singola iterazione prende il nome di *epoca*. Per ridurre il fenomeno dell'*overfitting* (sovradattamento: la rete impara a riconoscere il dataset ma non è in grado di generalizzare il problema a dati sconosciuti) durante l'allenamento viene scelto un sistema di validazione. Il dataset viene suddiviso in tre parti: trainset, validation set, test set. I dati del validation set vengono passati alla rete solo alla fine di ogni epoca, e questo passaggio non prevede i passi di backpropagation. In questo modo si può tenere conto dell'adattamento della rete ai dati del training set. Il test set viene invece utilizzato solo al termine dell'allenamento per ottenere una misura della precisione della rete. Per dataset di piccole dimensioni il validation set è solitamente un sottoinsieme del training set, isolato da esso all'inizio di ogni epoca. Nel progetto sono stati implementati due metodi di individuazione dei dati di validazione: k-fold training e k-window training. Il loro funzionamento è descritto nella sezione API.

3 Reti Neurali Convoluzionali

Il funzionamento delle reti neurali tradizonali è poco efficiente quando è necessario analizzare un numero molto grande di dati. Ogni neurone prevede una connessione con ogni neurone del layer successivo (per questo sono chiamati fully-connected layers). Ciò implica che al crescere della dimensione della rete, cresce inesorabilmente il numero di parametri di cui tener traccia, portando a casi si sovradattamento della rete.

Già dalla fine degli anni '90 i progettisti di reti neurali hanno iniziato ad introdurre dei modelli di reti convoluzionali, che permettono di ridurre di molto la grandezza della rete. Queste sono molto simili alle reti tradizionali: sono anch'esse formate da neuroni e tengono traccia di parametri come funzioni di attivazione e pesi che vengono modificati durante l'apprendimento. La caratteristica differente risiede nei layer convoluzionali di queste reti. Essi sono particolarmente indicati per il riconoscimento di proprietà nell'architettura dei dati in ingresso, che sono trattati come fossero immagini. La convoluzione prende infatti spunto dal funzionamento della corteccia visiva animale. Intuitivamente, un layer convoluzionale ha il compito di riconoscere alcune caratteristiche visive dell'immagine, come contorni, linee, colori ecc.. concentrandosi su piccole porzioni dell'immagine, e non prendendola nel suo complesso come accade nelle reti fully-connected. Una volta individuata una caratteristica in una certa sezione dell'immagine, la rete sarà in grado di riconoscerla se dovesse presentarsi in altri punti. In una successione di layer convoluzionali, inoltre, un layer più imparare a riconoscere combinazioni di caratteristiche base individuate nei precedenti strati. Ciò le rende particolarmente adatte alla comprensione di pattern complessi.

Un convolutional layer è generalmente formato da un set di filtri (kernel) della stessa dimensione. Durante la computazione tali filtri vengono applicati ai dati in ingresso, attraverso una moltiplicazione element-wise. I valori della matrice di ogni moltiplicazione element-wise vengono sommati fra loro e restituiti

in output. La *Figura 1* mostra graficamente un'operazione di convoluzione. In verde, la matrice di output.

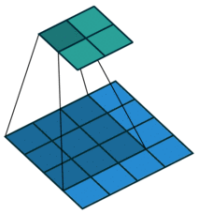

Figura 2: Convoluzione con kernel 3x3 applicato a dati 4x4

Un layer convoluzionale è governato da tre parametri:

- profondità (K): numero di filtri da applicare durante la convoluzione. Questo determinerà infatti la profondità dell'output. Per avere un'idea visuale, l'output di una convoluzione sarà il risultato dell'operazione in *Figura 1* per ogni kernel. Il risultato saranno più matrici di colore verde impilate, una per ogni filtro.

- stride (S): indica il numero di segnali per cui si vuole traslare il filtro ad ogni spostamento. Uno stride piccolo genera molti più spostamenti, aumentando la dimensione dell'output.

- zero-padding (P): segnali settati a zero, aggiunti ai bordi dei segnali di input. Serve spesso per adattare la dimensione dell'input con quella dell'output. Uno strato aggiuntivo di zeri infatti aumenterà la dimensione del vettore di output.

Dato un set di dati in ingresso (I) la dimensione del volume di output (O) di un livello convoluzionale è calcolata come

$$O = \frac{(I - K - 2P)}{S} + 1 \tag{1}$$

Tipicamente tra un layer convoluzionale e l'altro viene inserito uno strato di Pooling, che ha la funzione di diminuire la dimensione dell'input, riducendo il numero di parametri e controllando il sovradattamento. A differenza dei convolutional layer, il Pooling Layer non tiene traccia di alcun peso, dato che applica agli input una certa operazione deterministica, solitamente il massimo o la media. Anche il layer di pooling è caratterizzato dai parametri di dimensione del kernel, stride e padding. La dimensione del suo output è pertanto calcolabile come l'output del layer convoluzionale.

L'architettura CNN può anche prevedere uno o più strati Fully-Connected, solitamente aggiunti alla fine della struttura. Il loro compito è quello di raggruppare le informazioni degli strati precedenti, esprimendole attraverso un numero da utilizzare nei calcoli successivi per la classificazione finale.

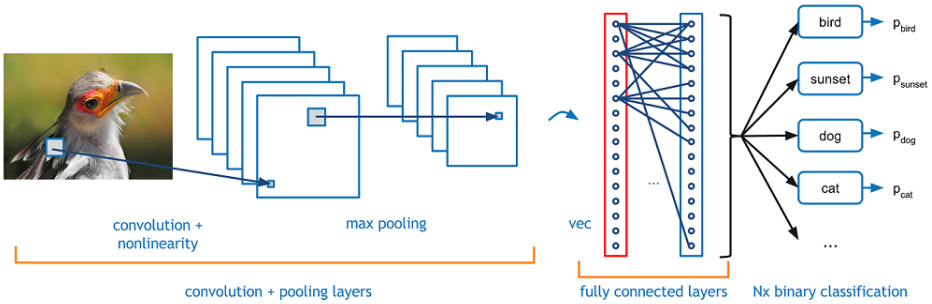

Figura 3: Tipica architettura CNN

3.1 Convoluzione 1D

Le reti di convoluzione più diffuse sono senza dubbio quelle bidimensionali. Esse sono particolarmente adatte per l'analisi e il riconoscimento di immagini. Le convoluzioni scelte per il nostro progetto sono invece monodimensionali. Queste sono utili quando ci si aspetta di derivare le caratteristiche di un segmento dei dati totali, e quando la posizione di tale segmento all'interno dell'insieme di input non è rilevante. Sono quindi utili per l'analisi di segnali e per il natural language processing (NLP), in cui il segmento analizzato corrisponde ad un insieme di parole contigue di una frase o, nel nostro caso, ad una finestra di segnali.

Tutte le CNN condividono le stesse caratteristiche e seguono lo stesso approccio. Ciò che distingue una convoluzione 1D, 2D o 3D è la dimensione dell'input e come i filtri (feature detector) scorrono lungo i dati. La *Figura 2* mostra graficamente la differenza tra una convoluzione monodimensionale e una bidimensionale. Nella rete monodimensionale, l'input è in due dimensioni C, W. Nel caso di un elettroencefalogramma, W rappresenta il numero di segnali per una finestra e C il numero di canali campionati. Possiamo notare come il filtro ricopra completamente la prima dimensione e scorra lungo l'altra. Sempre nel nostro caso, un filtro verrà applicato ad un certo numero di segnali di tutti i canali, per poi essere traslato lungo la dimensione W fino a considerare tutta la finestra.
Graficamente possiamo invece notare come la convoluzione 2D si applichi facilmente alle immagini. I dati di input sono infatti in tre dimensioni C, H, W, dove H e W rappresentano rispettivamente il numero di pixel in altezza e larghezza, mentre C indica il numero dei canali di colore, solitamente i 3 canali RGB. La figura mostra come il filtro prenda una porzione dell'immagine considerando tutta la dimensione C (quindi tutti i canali di cui è composta) e scorra lungo le altre due dimensioni.
Lo stesso procedimento, con una dimensione in più, si applica alle convoluzioni 3D.

Per completezza, è bene specificare che a tutte le grandezze di input so-

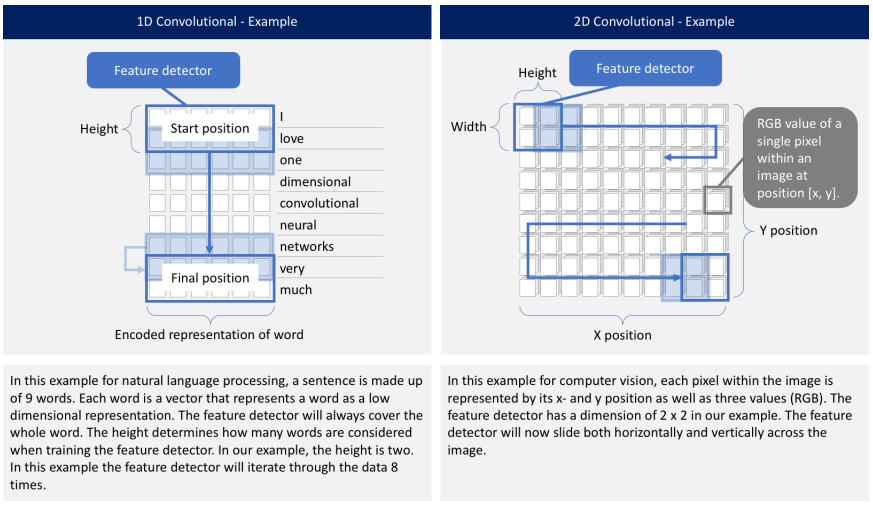

Figura 4: Differenza convoluzione 1D e 2D

pra indicate, va aggiunta un'ulteriore dimensione N. Essa rappresenta la *batch size*, cioè il numero di campioni (sample) di input passati alla rete in fase di training. Con dataset molto grandi, infatti, è possibile passare ad una rete un certo numero di input contemporaneamente, in modo da velocizzare l'allenamento e contenere l'utilizzo di memoria. Le operazioni di calcolo del gradiente e backpropagation verranno effettuate a partire da ogni batch e non sul singolo input.

3.2 Modello CNN proposto

Una funzionalità offerta dal programma è quella di predizione delle crisi su un file di segnali attraverso una rete pre allenata. Il dataset su cui è stato effettuato il training è descritto nel successivo capitolo. In questo paragrafo verranno illustrati i vari layer della rete.

La *Figura 3* mostra l'architettura della CNN proposta. I primi due layer sono sequenziali, cioè sono un gruppo di layer che processa i dati in maniera progressiva. Essi comprendono un primo livello convoluzionale, una livello ReLu e uno di MaxPooling 1D. Il livello ReLu (Rectified Linear Unit) viene utilizzato come funzione di attivazione, in sostituzione della classica sigmoide. Consiste nell'applicazione della funzioe $f(x) = max(0, x)$ che permette di scartare tutti i valori negativi in output da un livello. A seguire questi livelli sequenziali troviamo un livello di Dropout, che entra in funzione solo in fase di training. Dropout setta come nulli una percentuale p di valori scelti in modo casuale ad ogni operazione di forward. Questa tecnica è utilizzata per prevenire il fenomeno di sovradattamento della rete ai dati di training. In coda la rete prevede due livelli lineari (o fully-connected) per il raggruppamento dei dati e un livello finale di Softmax per normalizzare l'output nell'intervallo $[0, 1]$.

La rete è capace di analizzare file suddividendoli in finestre temporali di 30 secondi ciascuna. Dato che il numero di segnali per ogni finestra deve essere fisso, la frequenza di campionamento dei file deve coincidere con quella dei file utilizzati per il training, quindi 256 Hz.

```
<bound method Module.state_dict of ConvNet(
  (layer1): Sequential(
    (0): Conv1d(23, 10, kernel_size=(10,), stride=(5,), padding=(9,))
    (1): ReLU()
    (2): MaxPool1d(kernel_size=15, stride=10, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv1d(10, 5, kernel_size=(5,), stride=(4,), padding=(4,))
    (1): ReLU()
    (2): MaxPool1d(kernel_size=10, stride=5, padding=0, dilation=1, ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5)
  (fc1): Linear(in_features=35, out_features=15, bias=True)
  (fc3): Linear(in_features=15, out_features=2, bias=True)
  (soft): Softmax()
)>
```

Figura 5: Modello CNN proposto

3.3 CNN modulare

Il programma offre all'utente anche la possibilità di costruire una propria rete, per allenarla con un dataset e successivamente utilizzarla nelle predizioni di crisi epilettiche. Benché il numero di livelli e i parametri di ogni livello sono a discrezione dell'utente, alcune scelte architetturali sono fissate a priori. In particolare l'utente può scegliere un numero arbitrario di livelli convoluzionali 1D, per ognuno dei quali verrà automaticamente creato un livello ReLu e uno di MaxPool1D. La dimensione del kernel, dello stride il numero di filtri e i parametri del livello di pooling sono scelti dall'utente, verificando che i parametri scelti abbiano consistenza. Il modello prevede un layer di Dropout e un layer fully connected finale, che generi un output di dimensione 2 normalizzato dalla funzione di Softmax. Al layer lineare è possibile far precedere un ulteriore layer fully connected, di dimensione scelta dall'utente.

Un possibile sviluppo futuro del progetto potrebbe prevedere ulteriori libertà di scelta dell'utente, per offrire una maggiore personalizzazione del modello.

4 Dataset

Come descritto nelle sezioni precedenti, l'allenamento delle reti neurali prevede l'utilizzo di un dataset. Si è scelto un allenamento di tipo supervisionato, pertanto il dataset necessita di valori di input e relativi output attesi, da confrontare con i risultati della rete. Il dataset utilizzato è quello del Children's

Hospital Boston, che comprende EEG provenienti da 22 diversi pazienti pediatrici. Le misurazioni sono state effettuate con una frequenza di campionamento di 256 Hz e la maggior parte dei files contiene 23 canali. Per l'allenamento della rete è stata utilizzata solo una parte di questi files, in particolare sono stati selezionati files con almeno una crisi e con esattamente 23 segnali EEG provenienti dai pazienti 1 ,2, 3, 5, 7 e 8.

L'allenamento della rete è stato effettuato su finestre di 30 secondi di tempo. Per la creazione del dataset si sono seguiti i seguenti passi per ogni file:

- Isolamento segnali di crisi epilettica, in base ai secondi di inizio e fine crisi riportati nel file *chbNN-summary.txt* di ogni paziente
- Suddivisione dei segnali di crisi in finestre da 30 secondi, applicando uno stride di 1 secondo (overlapping 29 secondi tra una finestra e l'altra)
- Eliminazione dei segnali pre e post ictali, scelti convenzionalmente 5 minuti prima e dopo la crisi. Questo per evitare di influenzare il dataset con segnali aventi parziali caratteristiche di crisi
- Selezione di finestre non di crisi, in numero pari alle finestre di crisi calcolate in precedenza: in questo modo il dataset risulta perfettamente bilanciato.

5 Structure Application

L'applicazione creata è suddivisa in due grandi componenti, da una parte il backend che offre servizi e modella la struttura dei dati, dall'altra il frontend che richiede i servizi e ne implementa l'interfaccia grafica, in modo da garantire un'interazione user friendly con l'utente. Questo garantisce anche una più facile manutenzione del codice, visto che ognuna delle due parti è divisa in diversi componenti. L' aplicazione è una web app, essa infatti è capace di girare sia nell'ambito web che mobile grazie ai vari framework utilizzati.

5.1 Funzionalità offerte

L'applicazione costruita consente all'utente diverse esperienze e interazioni all'interno della web app. Le funzionalità offerte sono le seguenti:

- inserimento file edf (contenente tracciato EEG) con relativa ispezione di uno o tutti i canali settando la porzione temporale desiderata
- conseguenti statistiche riguardanti tutti i canali del file relativi a tutta la durata dello stesso
- sezione con relativo grafico che mostra il numero totale di valori registrati all'interno del tracciato, suddivisi per intervalli.
- predizione sull'intero file selezionando la rete desiderata da usare con relativo grafico che mostra le finestre di crisi e statistiche del caso

- possibilità di creazione della propria rete neurale che avviene per passi successivi:
  - possibilità di inserire una lista di file, aventi caratteristiche simili, come frequenza, numero canali ecc. Questa lista di file servirà poi per creare il proprio dataset di dati di allenamento
  - creazione del dataset inserendo la lunghezza delle finestre con relativo stride
  - setting della rete con tutti i relativi parametri, possibilità di creare piu livelli convoluzionali
  - setting delle impostazioni di allenamento della rete, attraverso l'inserimento del numero di epoche e la modalità di validazione

6 Backend

La parte di backend è utilizzabile indipendentemente dall'interfaccia fornita. Di seguito le librerie e i framework utilizzati e una descrizione delle API fornite.

6.1 Django Framework

Django è un server-side web framework Python estremamente popolare. Un web framework è un insieme di componenti che rendono lo sviluppo di siti web più facile e veloce. Django aiuta a eliminare tutte quelle attività che vengono ripetute durante lo sviluppo dell'applicazione, rendendo il processo di sviluppo un'attività facile e rapida. Si tratta di un web framework di alto livello scritto in linguaggio Python, sviluppato dalla Django Software Foundation e distribuito open source. Essendo un framework di sviluppo web, quindi con un interfaccia web, si parla di un pattern alla base chiamato MVC, che corrisponde a tre componenti: Model, View e Controller. Anche Django usa questo pattern, però nel suo caso il pattern è chiamato MVT, acronimo che indica i tre componenti di cui dispone: Model, View e Template.

- Model: corrisponde a un insieme di classi python che descrivono il modello dei dati. Esse forniscono una rappresentazione delle tabelle del database, consentendo di sfruttare gli oggetti per effettuare operazioni CRUD sui dati. Le informazioni delle classi sono contenute nel file *models.py*.
- Template: consiste in una serie di documenti per la generazione di codice HTML, combinando l'utilizzo di parti statiche e tag dinamici. Essa descrive quindi tutti quei componenti visuali facenti parte della GUI con cui l'utente andrà poi ad interagire. In questo progetto non è stato tuttavia utilizzato alcun template, dato che la parte visuale è gestita esternamente dal framework Ionic.
- View: funzioni python che gestiscono il flusso di esecuzione dell'applicazione, implementando la parte del controller del pattern MVC. Garantisce

la possibilità di creare delle pagine e di descriverne il comportamento che esse avranno in funzione dell'iterazione con l'utente. La View descrive e modella il comportamento dell'intera applicazione, rispondendo a eventi scaturiti dall'interazione con l'utente e modellando i dati facente parti del modello.

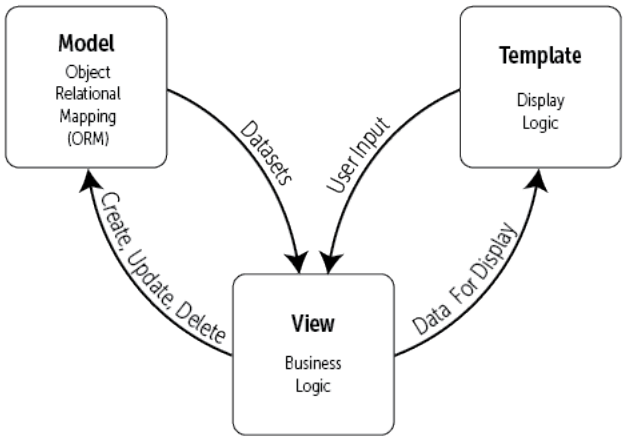

Figura 6: Relazioni modello MTV

6.2 Python

La sezione di backend del programma è stata scritta in Python, linguaggio di programmazione interpretato, di alto livello e generico. Esso supporta diversi paradgmi di programmazione, come quella procedurale, orientata agli oggetti e funzionale. Di seguito verranno descritte le più importanti librerie utilizzate nel progetto.

6.3 PyEDFlib

Il funzionamento base del sistema prevede la lettura di tracciati EEG. Questi vengono comunemente distribuiti nel formato EDF (European Data Format), ideato per lo scambio di segnali multicanali fisici e biologici. La libreria che si occupa dell'interazione con tali file è PyEDFlib, basata a sua volta su edflib e NumPy. Essa mette a disposizione funzioni per la lettura dei segnali del file e dei suoi metadati, tra cui il numero di canali, la lunghezza e la frequenza del campionamento, l'inizio e la fine della rilevazione.

6.4 PyTorch

La parte di neural network e deep learning è stata realizzata mediante l'utilizzo del framework PyTorch. Si tratta di è una libreria del linguaggio Python basata sul framework torch. Essa contiene diverse funzioni e metodi scientifici per le applicazioni dedicate al machine learning, al deep learning e al natural processing language. PyTorch offre due funzionalità di alto livello:

- Tensor Computing (come NumPy) con forte accelerazione attraverso unità di elaborazione grafica(GPU)
- Piattaforma di ricerca del deep learning che offre la massima flessibilità e velocità

6.5 API

L'utente ha la possibilità di interagire con il sistema attraverso richieste HTTP, ricevendo in risposta dei file in formato JSON. Di seguito sono descritte le principali Views messe a disposizione. Nelle successive sezioni verranno invece discusse le scelte riguardanti l'interfaccia grafica, che sfrutta queste views per fornire un'esperienza più facile e immediata anche agli utenti meno esperti.

Single File analysis

- */* [myfile]: upload del file mediante richiesta POST. Restituisce i parametri del file EDF caricato, come numero di canali, lunghezza, frequenza di campionamento. Solo dopo sver caricato il file è possibile fare richieste GET alle successive views.
- */values* [channel, start, len]: ottiene i valori registrati in un determinato intervallo di tempo, provenienti da uno specifico canale. Restituisce la scala temporale in cui tali valori sono misurati.
- */complete*[start, len]: restituisce una finestra completa dell'EEG (quindi comprendente segnali di ogni canale) in uno specifico intervallo di tempo.
- */statistic*[channel]:relativamente ad uno specifico canale, calcola le statistiche dei segnali misurati, quali valore massimo e minimo, media, varianza e deviazione standard.
- */distribution*[channel]: relativamente ad uno specifico canale, restituisce due liste: una con intervalli di valori, e un'altra con il numero di segnali registrati in ogni intervallo. Con queste misure è possibile costruire un'istogramma che mostri la distribuzione dei valori.
- */predict*[model_id]: effettua la predizione di crisi epilettiche sul file caricato. L'utente ha la possibilità di scegliere il modello da utilizzare, fornendone l'id. I segnali del file vengono suddivisi in finestre della dimensione

accettata dal modello scelto. Per ogni finestra la rete restituirà la predizione sotto forma di 1 e 0 rispettivamente per segnale di crisi e non. Vengono inoltre restituiti il numero di finestre con crisi e il numero di finestre totali.

Training

- */uptraining* [myfile][seizureStart, seizureEnd]: caricamento dei files da utilizzare per l'allenamento della rete. Per ogni file l'utente deve specificare in quale intervallo di tempo si presenta la crisi. Vengono restituite le info sul file caricato e una lista di tutti i file finora caricati.
- */cleanfiles*: cancella tutti i file di training finora caricati.
- */convert* [windowSize, stride]: converte tutti i file di training finora caricati in un dataset, con finestre della dimensione scelta. L'utente può inoltre scegliere un valore di stride da applicare alle finestre della crisi, in modo da ottenerne in numero maggiore. Questa view è necessaria prima di chiamare la view del training. La chiamata a questa view cancella tutti i files di training.
- */addconv*[output, kernel, stride, padding, pool_kernel, pool_stride]: permette l'aggiunta di un livello convoluzionale alla rete da creare. I parametri scelti dall'utente vengono salvati come record del database, dopo averne controllato la consistenza (evitare che l'output sia nullo, o che il kernel sia di dimensioni maggiori della finestra di segnali). Notare che insieme al layer convoluzionale l'utente specifica anche i parametri di un livello di MaxPool aggiunto automaticamente.
- */cleanlayers*: elimina tutti i livelli finora impostati.
- *initializenet*[linear]: crea un'istanza di rete pronta per essere allenata. I parametri vengono ripresi dal database in ordine di inserimento. Viene automaticamente aggiunto un livello lineare finale che standardizzi l'output a due valori; l'utente ha inoltre la possibilità di aggiungere un ulteriore livello lineare impostando il parametro *linear*.
- */train* [epochs, train_method, name]: crea una nuova rete e la allena sul dataset precendentemente creato. L'utente deve indicare il numero di epoche e quale metodo di validazione utilizzare. La validazione permette di prevenire l'overfitting dei dati, cioè che l'allenamento permetta alla rete di generalizzare il suo funzionamento. A tale scopo, vengono scelti dei dati da passare alla rete senza effettuare backpropagation. É possibile sccegliere il validation seti con diversi metodi. Ne sono stati implementati due:
  - k-fold training: all'interno di ogni epoca i segnali di un file vengono isolati, e passati alla rete dopo l'allenamento senza effettuare backpropagation. Nell'epoca successiva questi segnali faranno parte del

dataset di allenamento, e verrà selezionato il file successivo per la validazione.

- k-window training: all'inizio di ogni epoca vengono selezionate randomicamente il 20% delle finestre da utilizzare per la validazione.

Al termine dell'allenamento la rete viene salvata nel sistema, e sarà possibile utilizzarla per le predizioni.

- */usermodels]*: restituisce id e nome dei modelli creati attraverso il training.
- */cleanmodels*: elimina tutti i modelli in memoria.

7 Frontend

La parte di frontend è stata implementata per garantire la gestione dell'interfaccia grafica con cui l'utente andrà ad interagire; essa rappresenta il confine del sistema.

7.1 Angular

Angular è un framework javascript per la scrittura di applicazioni web client. Per raggiungere questo obiettivo, viene usato l'approccio dichiarativo dell'Html per quanto riguarda l'interfaccia grafica, e vengono forniti strumenti per costruire un'architettura modulare avente logica applicativa. Un concetto fondamentale di angular sono i componenti: l'applicazione è formata da un insieme di componenti che interagiscono tra loro ed hanno il controllo di una porzione dello schermo implementando una view. Caratteristiche peculiari di Angular sono:

- binding bidirezionale (two-way binding)
- dependency injection
- supporto al pattern MVC
- supporto ai moduli
- separazione delle competenze
- testabilità del codice
- riusabilità dei componenti

7.2 Ionic

Ionic framework è un toolkit di interfaccia utente per la creazione di applicazioni mobili e desktop performanti che utilizzano tecnologie web. Ionic si concentra sull'esperienza utente di un'app (controlli, interazioni, gesti, animazioni). Esso si integra in modo facile a tecnologie come Angular. Il framework Ionic consente di sviluppare app per tutte le piattaforme più comuni, come Android, iOs e

Windows. Con Ionic risulta molto facile creare la parte di interfaccia grafica, che è molto vicina a quella nativa del dispositivo dove verrà eseguita l'app.

7.3 HIghChart

Highchart è una libreria grafica che permette di disegnare grafici a lasciando allo sviluppatore un alto grado di personalizzazione. Con Highchart è possibile rappresentare grafici anche con grandi moli di dati.

Riferimenti bibliografici

[1] Nils Ackermann. *Introduction to 1D Convolutional Neural Networks in Keras for Time Sequences*. URL: https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf.

[2] Nicoletta Boldrini. *Reti neurali: cosa sono e a cosa servono*. 2019. URL: https://www.ai4business.it/intelligenza-artificiale/deep-learning/reti-neurali/.

[3] Assaad Moawad. *Neural networks and back-propagation explained in a simple way*. URL: https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e.

[4] Jordi Torres. *Convolutional neural networks for beginners*. URL: https://towardsdatascience.com/convolutional-neural-networks-for-beginners-practical-guide-with-python-and-keras-dc688ea90dca.

[5] Wikipedia. *Artificial neural network*. URL: https://en.wikipedia.org/wiki/Artificial_neural_network.

[6] Wikipedia. *Convolutional neural network*. URL: https://en.wikipedia.org/wiki/Convolutional_neural_network.