Università degli Studi di Ferrara

UNIVERSITÀ DEGLI STUDI DI FERRARA

DIPARTIMENTO DI INGEGNERIA

Corso di Laurea in
Ingegneria Elettronica e Informatica

DNN-Tuner e Dash framework

Relatore:
Ch.ma Prof. Evelina Lamma

Correlatore:
Dott. Michele Fraccaroli

Laureando:
Andrea Benà
Matr:133793

Anno Accademico 2019/2020

Indice

Elenco delle figure

Introduzione

Questa tesi vuole illustrare il lavoro svolto durante il periodo di internato in collaborazione con Ch.ma Prof. Evelina Lamma ed il Dott. Michele Fraccaroli.
Il soggetto principale, protagonista di questa esperienza, è la dashboard realizzata per *Probabilistic DNN-Tuner*, un software in grado di settare automaticamente i parametri ottimali per il training delle *Deep Neural Networks* (DNNs). Attraverso la dashboard è possibile, mediante l'utilizzo di grafici e indicatori, monitorare i risultati ottenuti durante l'apprendimento. Per la sua realizzazione si è scelto di utilizzare i componenti offerti dal framework *Dash*, un progetto open source che sfrutta la semplicità del linguaggio Python nella realizzazione di applicazioni web.

In questo elaborato si presentano alcuni componenti che hanno permesso di migliorare l'applicazione dal punto di vista della espandibilità dando la possibilità di aggiungere nuove funzioni in modo semplice e veloce. Il tutto è reso possibile grazie ad un altro componente open source: *Dash Bootstrap Components*. Bootstrap offre componenti pronti all'uso e responsive che non richiedono conoscenze approfondite di *CSS* o *JavaScript*.

Il primo capitolo descrive il concetto di rete neurale artificiale: com'è strutturata, i principali modelli, le sue applicazioni e limiti. Nel secondo capitolo, viene descritto il software che si occupa di migliorare il tuning dei parametri che permettono di allenare la DNN. Dopo questa breve introduzione, prosegue il terzo capitolo dove vengono descritti i componenti contenuti all'interno della dashboard. Nello specifico andremo ad analizzare com'è stata strutturata ed il contenuto di ogni tab. Il quarto capitolo rappresenta una raccolta di tutti gli strumenti utilizzati per realizzare l'applicazione web: una dashboard per Probabilistic DNN-Tuner. Particolare attenzione verrà data al framework Dash che, grazie ai suoi componenti, garantisce un'ottima soluzione per creare applicazioni rivolte al mondo del *Machine Learning*.

Capitolo 1

Deep Neural Networks

Nel 1956, con l'avvento dei primi computer, fu coniato il termine *Intelligenza Artificiale* [1]. L'idea alla base è quella di realizzare programmi in grado di effettuare alcuni ragionamenti logici, in modo da riprodurre le capacità intuitive e di ragionamento, tipiche degli esseri umani.
Con il passare degli anni si affermò una nuova branca dell'intelligenza artificiale, il *Machine Learning* [2], noto anche come apprendimento automatico. Esso sfrutta diversi metodi sviluppati in varie comunità scientifiche, quali *statistica computazionale, riconoscimento dei pattern, reti neurali artificiali, elaborazione delle immagini, data mining,* ecc. Attraverso metodi statistici è possibile migliorare le performance di un algoritmo nell'identificare pattern nei dati, senza dover realizzare programmi che lo facciano esplicitamente. Un'ulteriore branca dell'intelligenza artificiale e del Machine Learning è il Deep Learning.

Il *Deep Learning* [3] è l'insieme dei metodi di machine learning che fanno uso di un modello computazionale chiamato *Rete Neurale Artificiale* [4] (o Artificial Neural Network, ANN). Una *Deep Neural Network* (DNN) è una rete neurale artificiale formata da uno o più livelli nascosti, detti *hidden layers*. Questa si ispira alle reti neurali biologiche degli animali e si occupa di portare il sistema a migliorare progressivamente le proprie abilità, andando a considerare degli esempi e producendo risultati per i quali non è stata specificatamente programmata.

1.1 Struttura delle DNNs

Una DNN è composta da neuroni che possono appartenere ad uno dei seguenti strati:

- ***Input layer***: riceve i dati che si vogliono analizzare
- ***Hidden layer***: può essere formato da più strati interconnessi tra loro dove ciascun neurone, sulla base dei dati in ingresso, esegue una funzione di attivazione
- ***Output layer***: produce il risultato finale sulla base dei dati in ingresso.

Una rete neurale può avere diversi tipi di layout a seconda del tipo di impiego per cui verrà utilizzata [5], pertanto ciascun neurone può essere collegato in modo completo o parziale con altri. Un neurone artificiale è costituito da un'unità che riceve in ingresso un valore numerico il quale, nel caso in cui venga o meno superata la soglia di attivazione, lo rende attivo o inattivo. Questo valore numerico è influenzato da un peso (*weight*) che può variare durante la fase di apprendimento (*training*) e che fa aumentare o diminuire l'intensità del segnale che si sta inviando.

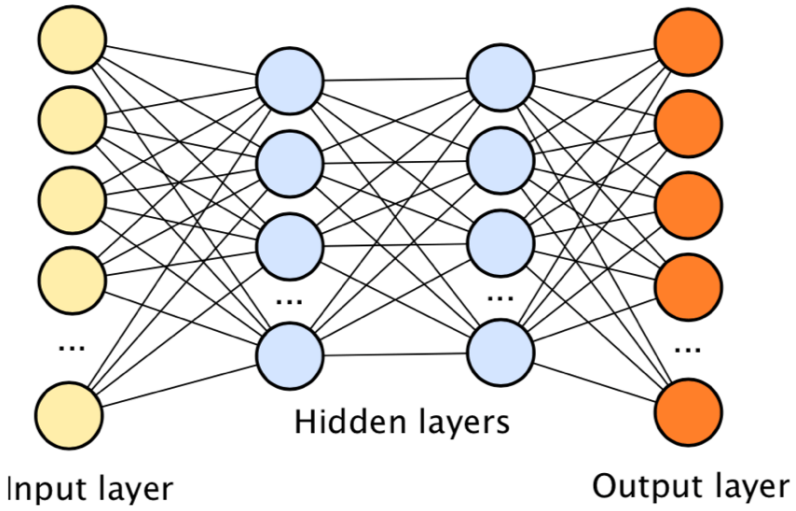

Figura 1.1: Struttura delle Deep Neural Networks

source by: `https://bdtechtalks.com/2018/02/27/limits-challenges-deep-learning-gary-marcus/deep-neural-networks/`

1.2 Paradigmi di apprendimento

Vi sono tre grandi paradigmi di apprendimento:

- Apprendimento ***supervisionato***: viene utilizzato un insieme di dati per l'addestramento (o *training set*) che contiene esempi di ingressi con i corrispondenti valori in uscita. Per esempio, se si vuole far riconoscere un'immagine, vengono definiti gli oggetti che devono essere identificati. Dopo aver confrontato i valori calcolati dalla rete con i valori corretti, si ottiene il livello di errore attraverso il quale è possibile aggiustare i pesi in modo da ottenere i risultati attesi
- Apprendimento ***non supervisionato***: vengono utilizzati degli algoritmi per modificare i pesi utilizzando come punto di riferimento i soli valori in ingresso. Ciò che si vuole ottenere è una classificazione ed organizzazione dei dati in input sulla base di caratteristiche comuni in modo da effettuare previsioni e ragionamenti sugli input successivi
- Apprendimento per ***rinforzo***: questo modus operandi realizza *agenti autonomi* che sono in grado di prendere decisioni future basandosi sullo stato attuale e sull'interazione con l'ambiente in cui essi sono immersi. Per ogni azione che rappresenta un miglioramento dell'agente, viene assegnata una ricompensa che si ispira al concetto di rinforzo.

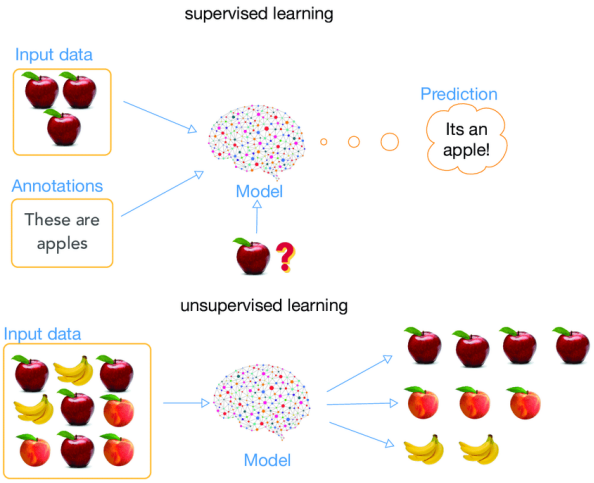

Figura 1.2: Apprendimento supervisionato e non supervisionato

source by: `https://www.researchgate.net/figure/Supervised-learning-and-unsupervised-learning-Supervised-learning-uses-annotation_fig1_329533120`

1.3 Vantaggi, limiti ed impiego delle DNNs

Tra i principali vantaggi che comporta l'utilizzo di una rete neurale abbiamo:

- ***Elevato parallelismo***: è possibile eseguire più operazioni contemporaneamente in modo da ridurre i tempi necessari a processare grandi moli di dati
- ***Tolleranza ai guasti***: garantita dal forte parallelismo realizzato mediante l'uso di più neuroni
- ***Tolleranza al rumore***: gestisce in maniera efficace l'output anche qualora l'input dovesse essere incompleto
- ***Evoluzione adattativa***: una rete neurale è in grado di adattarsi all'evoluzione dell'ambiente in cui opera.

Purtroppo sono presenti anche alcuni limiti:

- ***Periodo di learning più o meno lungo***: la durata necessaria per realizzare l'apprendimento è influenzata non solo dal numero di input e dalla loro complessità, ma anche dall'algoritmo utilizzato e da altri fattori
- ***Incertezza sul risultato***: non è possibile stabilire a priori se il training della rete andrà a buon fine o meno
- ***Funzionamento a black box***: la computazione eseguita non è interamente analizzabile. Anche se i risultati ottenuti sono corretti, non è possibile esaminare i singoli stadi di elaborazione che li determinano.

Attualmente le reti neurali vengono utilizzate per svariate applicazioni, come: *riconoscimento ed elaborazione di immagini, diagnosi mediche, data mining, finanza* (per effettuare previsioni sull'andamento di mercato), *riconoscimento vocale* ecc. [6]. Tutto questo è reso possibile grazie alla grande disponibilità di dati raccolti in questi ultimi anni e all'innovazione riguardante le schede grafiche di ultima generazione [7].

Capitolo 2

DNN-Tuner

Con l'aumentare della complessità e delle dimensioni delle DNNs, risulta essere sempre più impegnativa la gestione degli *iperparametri* [§ 3.2].
Settare questi valori manualmente può portare a commettere molti errori, per questo si utilizzano delle tecniche come:

- ***Grid search***: si basa su un approccio *brute-force* che va a testare l'algoritmo di apprendimento con tutte le possibili combinazioni degli iperparametri contenute all'interno di una griglia di valori. Questo genera una grande mole di iperparametri da settare, nonché un grande spreco in termini di tempo
- ***Random search***: seleziona in modo casuale i possibili valori da attribuire agli iperparametri prendendoli dalla stessa griglia utilizzata nel Grid search. Il numero di iterazioni della ricerca è definito in base al tempo/costo di utilizzo della risorsa
- ***Bayesian optimization***: è senz'altro una delle tecniche più utilizzate poiché è applicabile a qualsiasi forma di funzione. Sfrutta un approccio probabilistico grazie al quale è in grado di ottimizzare anche funzioni black-box [8].

Sebbene il tuning di una DNN sia molto oneroso, molti esperti continuano a settare gli iperparametri manualmente.

DNN-Tuner [9] è in grado di rilevare i principali problemi che una DNN può incontrare, come ad esempio: *overfitting* [§ 3.4.1], *underfitting* [§ 3.4.2], valori incorretti di *learning rate* [§ 3.2.1] e *batch size* [§ 3.4.3]. Per farlo, sfrutta un'analisi delle metriche delle reti eseguita successivamente le fasi di *training* e *validation* [§ 3.3]. In base a queste analisi, viene generata una diagnosi dei problemi. Ogni problema che DNN-Tuner è in grado di rilevare, ha una *Tuning Rules* associata (vedi [Fig.2.1]).
Le Tuning Rules sono regole/azioni che vengono applicate allo spazio di ricerca degli iperparametri e/o alla struttura della rete neurale in risposta ai problemi identificati nella fase di diagnosi.

Issue	Tuning Rules
Overfitting	Regularization
	Batch Normalization
Underfitting	More neurons
Fluctuating loss	Increase of batch size
Increasing loss trend	Decrease of learning rate

Figura 2.1: Azioni automatiche eseguite dall'algoritmo di DNN-Tuner

2.1 Probabilistic DNN-Tuner

Visto l'ottimo risultato ottenuto con DNN-Tuner, è stata realizzata una versione estesa che utilizza un *linguaggio simbolico* per imparare quale regola è meglio applicare data una diagnosi (in base alle esperienze maturate durante le precedenti iterazioni dell'algoritmo), offrendo una migliore interpretazione e garantendo maggiore modularità.
Nella figura 2.2 si può notare come, rispetto la precedente versione, siano state introdotte per ciascuna tipologia di problema, ulteriori *Tuning Action*.

Problem	Tuning Action (TA)
Overfitting	Regularization and Batch Normalization
	Increase dropout
	Data augmentation
Underfitting	Decrease the learning rate
	Increase the number of neurons
	Addition of fully connected layers
	Addition of convolutional blocks
Increasing loss	Decrease the learning rate
Fluctuating loss	Increase the batch size
	Decrease the learning rate
Management of the learning rate	Increase the learning rate
	Decrease the learning rate

Figura 2.2: Azioni automatiche eseguite dall'algoritmo di Probabilistic DNN-Tuner

La dashboard di cui andremo a parlare nel capitolo successivo, è nata per raccogliere i dati elaborati da Probabilistic DNN-Tuner.

Capitolo 3

Dashboard per Probabilistic DNN-Tuner

Per poter tenere sotto controllo la fase di training della rete neurale, è stata realizzata una dashboard per *Probabilistic DNN-Tuner*. Essa offre la possibilità di rappresentare i dati attraverso dei tool che trasformano i risultati in grafici [10] e indicatori [11]. E' molto importante verificare che l'apprendimento avvenga correttamente per non rischiare di perdere tempo e di ritrovarsi con una rete neurale che non produce l'output desiderato.

Una delle caratteristiche fondamentali di una dashboard è l'espandibilità. Per ottenere questo risultato si è scelto di inserire i componenti in più tab, raggruppandoli per categorie. Qualora fosse necessario inserire una nuova categoria, basterà semplicemente aggiungere una tab.

La dashboard è suddivisa in 4 tab:

1. *Network architecture*
2. *Hyper-parameters*
3. *Metrics*
4. *Probabilistic weights*

3.1 Network architecture

I dati archiviati, relativi all'architettura della rete neurale, sono stati salvati nel formato .h5 che rappresenta un file HDF, ovvero *Hierarchical Data Format* [12], un formato standard di archiviazione. Questo tipo di formato offre la possibilità di sottolineare le dipendenze nonché le relazioni che sussistono tra i dati raccolti. Esistono diversi software per leggere i file HDF. Nell'applicativo realizzato si è scelto di utilizzare Netron [13], un visualizzatore creato specificatamente per reti neurali.

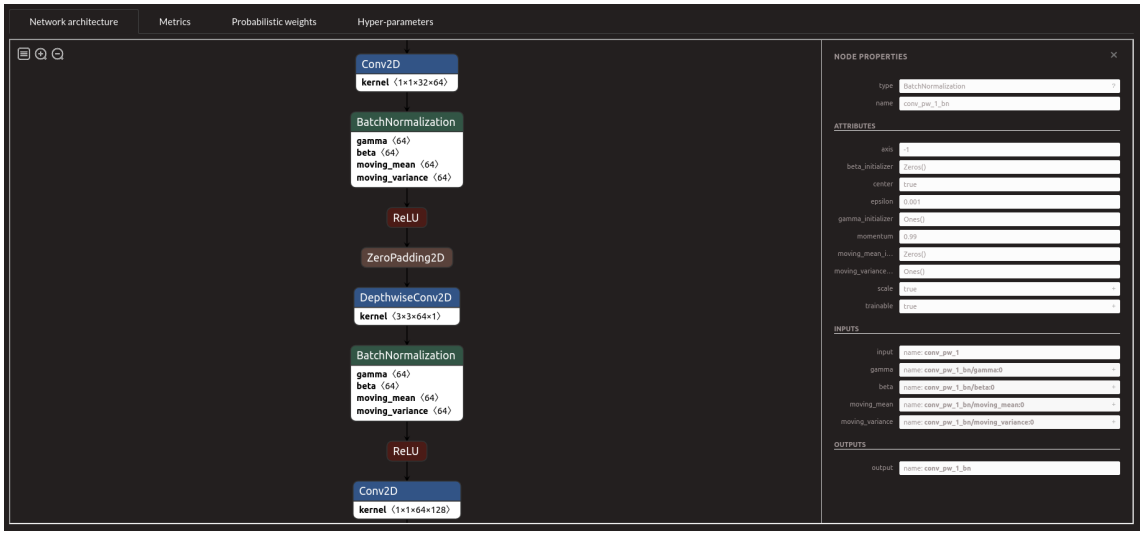

Figura 3.1: Implementazione del software Netron all'interno della dashboard

3.2 Hyper-parameters

Gli *hyper-parameters* (in italiano: iperparametri) rappresentano le variabili che determinano la struttura della DNN e il modo in cui questa sta apprendendo (ad esempio: *Learning rate*).
Per ogni iterazione, questi iperparametri, vengono accuratamente scelti. E' possibile utilizzare diverse tecniche e, se la rete è molto piccola, si può pensare di applicare manualmente le modifiche necessarie. Nel nostro caso, gli iperparametri vengono settati con un approccio automatico utilizzando la *Bayesian optimization* implementata all'interno del software Probabilistic DNN-Tuner. Nel realizzare l'interfaccia si è pensato di raccogliere, per ogni iterazione, i dati rappresentanti gli iperparametri. Per farlo, si è utilizzato *Cytoscape* [14], ovvero un programma open source impie-

gato nella bioinformatica per visualizzare le reti di interazione molecolare. Grazie a questo approccio si può consultare velocemente lo storico dei parametri utilizzati di volta in volta, senza doverli ricercare all'interno del database.

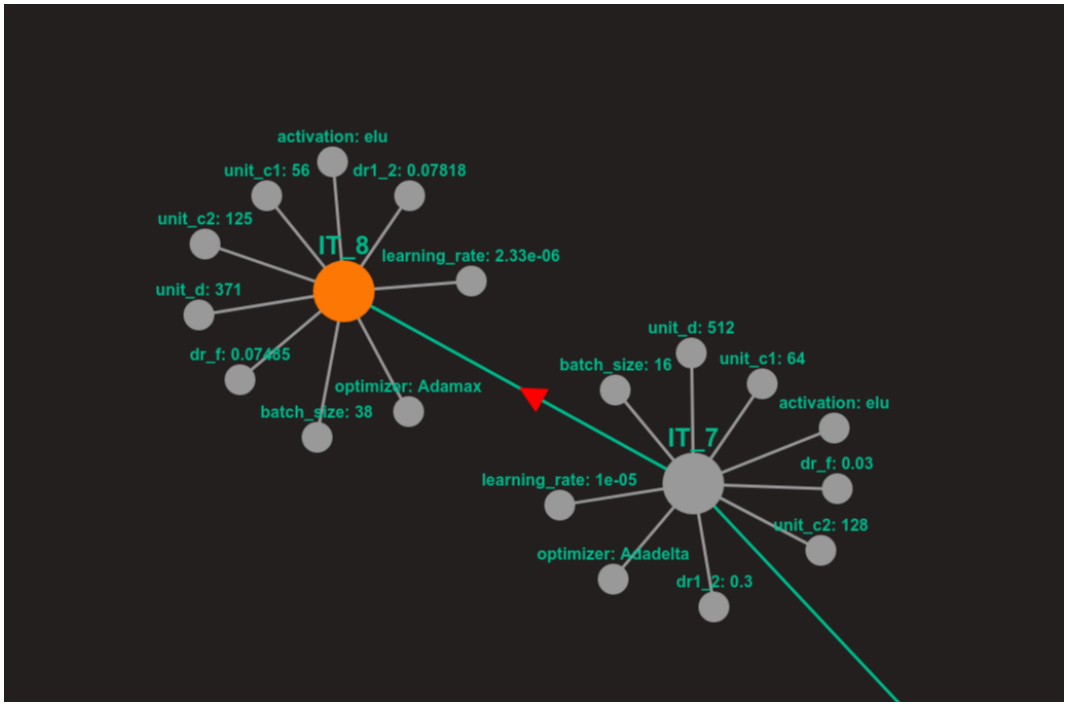

Figura 3.2: Cytoscape utilizzato per rappresentare gli Hyper-parameters

3.2.1 Learning rate

Il parametro più importante è sicuramente la *Learning rate*. Il suo valore va ad influire sulla variazione dei weights e rappresenta il tasso con il quale verranno modificati.
Generalmente viene rappresentata con valori compresi tra 0.0 e 1.0. La velocità di adattamento del modello dipende dalla learning rate. Più il suo valore è basso, maggiore sarà il numero di iterazioni necessarie. Questo può portare la fase di training ad una situazione di stallo in cui non avviene alcun apprendimento. Al contrario, se si aumenta eccessivamente il suo valore, questo fa aumentare la velocità di apprendimento ma si rischia di non ottenere la convergenza alla soluzione del problema.

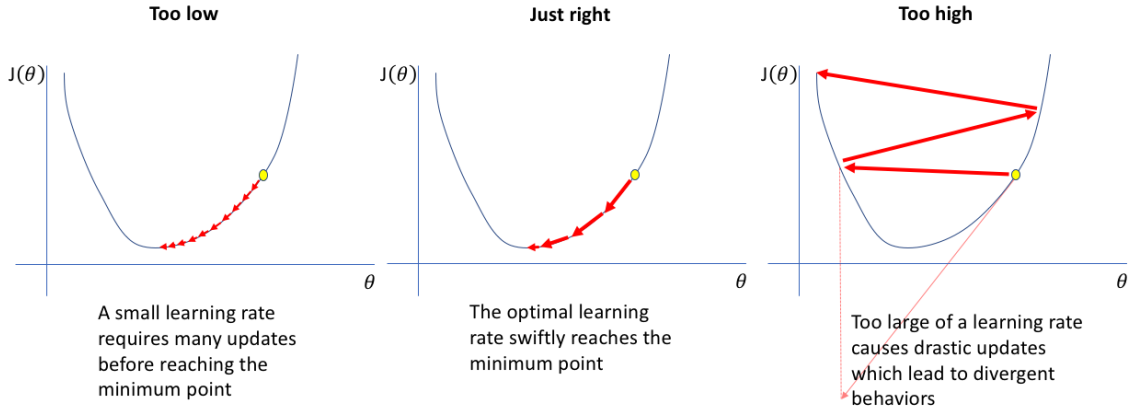

Figura 3.3: Influenza learning rate sulla velocità di apprendimento

source by: `https://www.jeremyjordan.me/nn-learning-rate/`

3.3 Metrics

All'interno della tab riferita alle metriche vengono presi in considerazione i valori raccolti nelle fasi di *training e validation*.

Durante la fase di training sono analizzati ripetutamente i dati contenuti nel training set. Questi dati rappresentano gli input sul quale la rete neurale verrà addestrata. Ovviamente questo potrebbe richiedere del tempo a causa di fattori quali il tipo di rete, il numero di hidden layers, la dimensione ed il numero di input utilizzati.

Nella fase di validation la rete neurale verifica il suo livello di allenamento. Per farlo utilizza dei campioni appositamente selezionati che sono diversi da quelli elaborati nella fase di training. Dai risultati che si ottengono è possibile comprendere se i parametri settati forniscono correttamente l'output desiderato oppure necessitano di un ulteriore tuning.
Durante il training e la validation possiamo ricavare i valori prodotti dalla *Accuracy function* e dalla *Loss function*.

3.3.1 Accuracy function

La accuracy function viene calcolata sotto forma di percentuale e rappresenta l'accuratezza, ovvero il grado di attendibilità, prodotta dall'algoritmo di apprendimento automatico che è stato utilizzato.

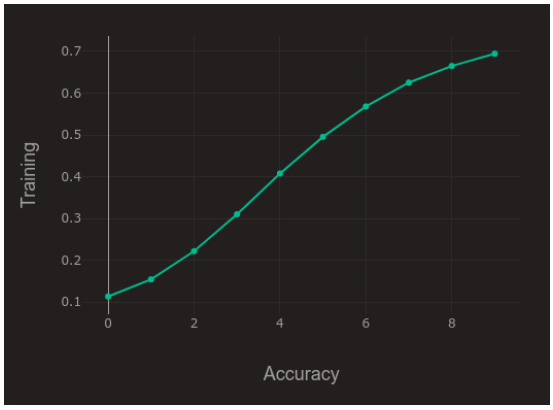

Figura 3.4: Grafico Training-Accuracy

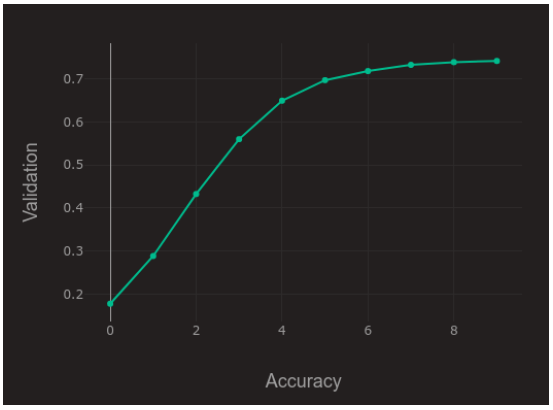

Figura 3.5: Grafico Validation-Accuracy

3.3.2 Loss function

La loss function, a differenza dell'accuracy, non è espressa in percentuale. Viene calcolata in fase di training e validation. Il risultato ottenuto viene interpretato come la distanza tra il valore reale ed il valore ottenuto dalla rete.
Grazie alla loss function si è in grado di comprendere il comportamento del modello ad ogni iterazione compiuta dalla rete neurale. Sulla base dei dati raccolti, l'algoritmo di Probabilistic DNN-Tuner, utilizza delle specifiche *Tuning Actions* il cui obiettivo è quello di azzerare la perdita (loss).

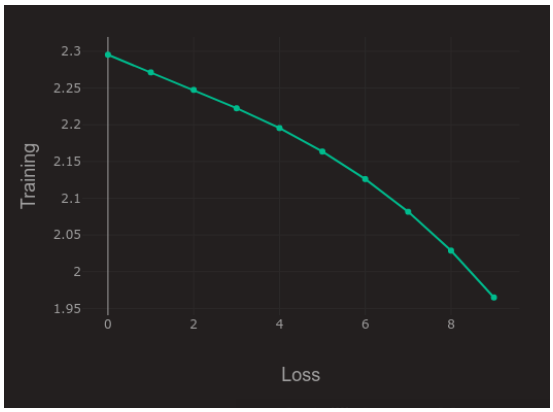

Figura 3.6: Grafico Training-Loss

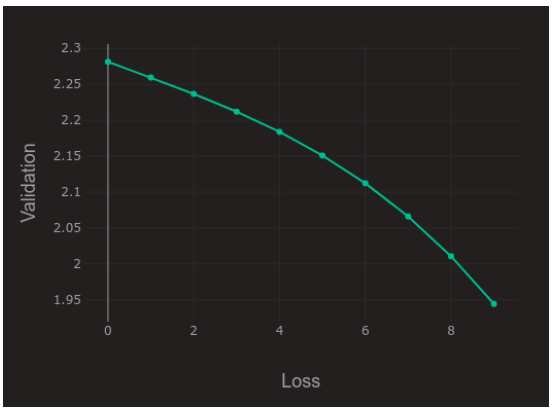

Figura 3.7: Grafico Validation-Loss

3.4 Probabilistic weights

Nella tab riguardante i probabilistic weights, sono stati inseriti 8 indicatori che descrivono i pesi probabilistici che Probabilistic DNN-Tuner ha associato alle *Tuning Rules*. Questi pesi vengono appresi e modificati da Probabilistic DNN-Tuner per dare più o meno peso alle regole che hanno portato a un miglioramento della rete neurale. In questo modo, le regole che hanno portato dei miglioramenti alla rete vengono premiate e hanno maggiore possibilità di essere riapplicate.

Gli indicatori che andremo ad analizzare sono stati opportunamente divisi in base alla tipologia di problema a cui si riferiscono:

- *Overfitting*
- *Underfitting*
- *Floating Loss*

Nella figura 3.8 sono rappresentati graficamente i fenomeni di underfitting ed overfitting:

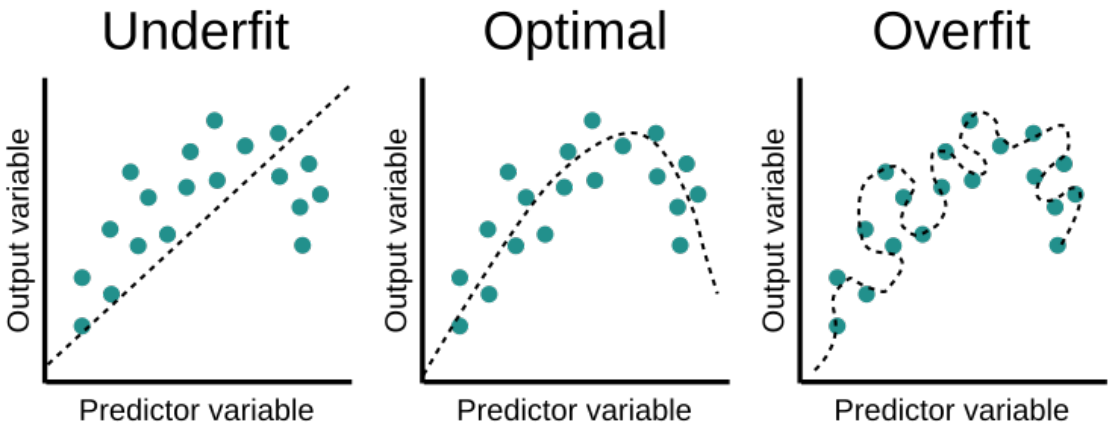

Figura 3.8: Underfitting, Optimal e Overfitting a confronto

source by: `https://www.educative.io/edpresso/overfitting-and-underfitting`

3.4.1 Overfitting

Si parla di *overfitting* (in italiano: adattamento eccessivo) quando il modello statistico si adatta molto bene ai dati contenuti nel training set ma non risponde allo stesso modo in riferimento ai dati utilizzati nella fase di validation.
Ciò avviene qualora l'apprendimento sia durato troppo a lungo oppure il numero di campioni analizzati è troppo basso. In questo caso, il modello sarà incapace di adattarsi ed effettuare previsioni su dati non ancora visionati.

Per cercare di ridurre l'overfitting, è possibile agire su due parametri:

- ***Increment dropout***: in maniera randomica, si eliminano temporaneamente alcuni neuroni nella rete neurale
- ***Data augmentation***: se non sono disponibili altri campioni, è possibile crearne di nuovi a partire da quelli di partenza. Ad esempio, se si stanno confrontando delle immagini, basterà ritagliarle o semplicemente ruotarle per ottenerne di nuove.

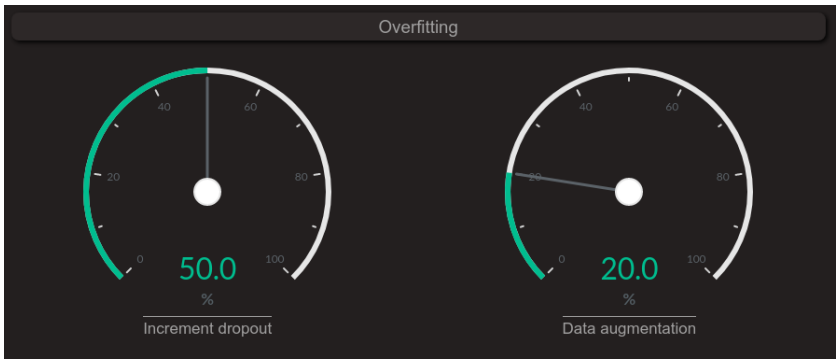

Figura 3.9: Indicatori dei parametri riferiti all'Overfitting

3.4.2 Underfitting

La condizione di *underfitting* si verifica quando un modello non è in grado di acquisire il pattern sottostante dei dati. Ciò accade qualora la quantità di dati per costruire il modello sia insufficiente oppure nel caso in cui si stia cercando di costruire un modello lineare utilizzando dati non lineari.

Quando l'underfitting viene rilevato da Probabilistic DNN-Tuner, l'algoritmo interviene effettuando il tuning dei seguenti parametri:

- *Decrement the Learning rate*
- *Increment Neurons*
- *Addition of fully connected layers*
- *Addition of convolutional layers*

Figura 3.10: Indicatori dei parametri riferiti all'Underfitting

3.4.3 Floating Loss

La maggior parte delle reti neurali viene addestrata basandosi sulla fluttuazione del gradiente in discesa. Un fattore che influisce sull'accuratezza della stima del gradiente è senz'altro il *batch size*. Esso rappresenta il numero di campioni che verranno propagati attraverso la rete neurale.

Immaginiamo di avere a disposizione 550 *training samples* e di impostare un batch size pari a 100. L'algoritmo userà i primi 100 campioni del training set per allenare la rete, dopodiché andrà a considerare i successivi 100 campioni e così via fino ad arrivare a 500. Siccome nell'esempio abbiamo 550 campioni, questi non sono divisibili per il batch size di 100. La soluzione è addestrare la rete utilizzando gli ultimi 50 campioni che rimangono.

Nel caso in cui il batch size sia minore del numero totale di campioni, si otterranno i seguenti vantaggi:

- ***Richiesta meno memoria***: questo è importante specialmente qualora non fosse possibile caricare il dataset completo in memoria
- ***La fase di addestramento è più veloce***: ciò è dovuto al fatto che in seguito ad ogni propagazione, i pesi vengono aggiornati. Nel nostro esempio sono stati propagati 6 batches (5 composti da 100 esempi ed uno da 50) e per ciascuno di loro, sono stati aggiornati i parametri della rete.
  Se si fossero utilizzati tutti i campioni, ci sarebbe stato solo un aggiornamento.

Purtroppo l'utilizzo di un batch size inferiore al numero di campioni comporta anche uno svantaggio non indifferente, ovvero una stima poco accurata. Infatti l'utilizzo di un valore piccolo del batch size comporta la fluttuazione del gradiente, il cui valore si discosta molto dal risultato che si otterrebbe utilizzando un batch size delle dimensioni dell'intero training set. Da queste osservazioni si può intuire la necessità di trovare un trade-off tra velocità ed accuratezza.

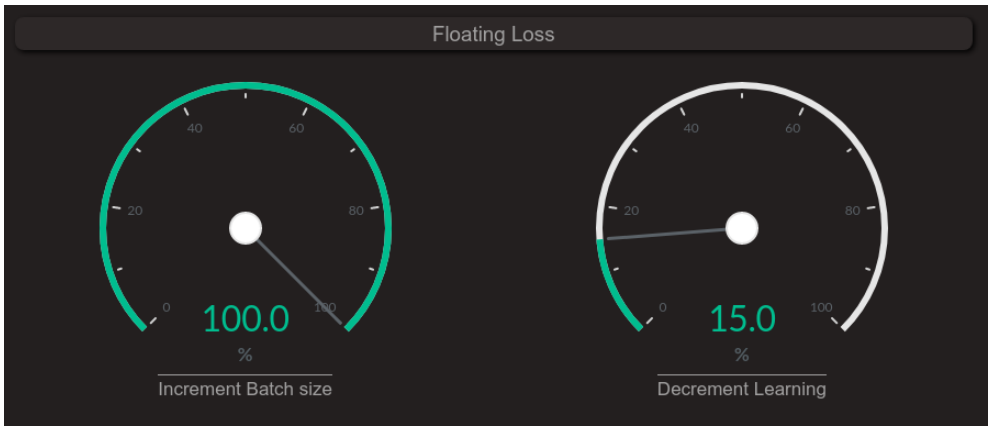

Figura 3.11: Indicatori dei parametri riferiti al Floating Loss

Capitolo 4

Strumenti utilizzati

4.1 Dash Framework

La dashboard è stata realizzata con *Dash* [15], un framework particolarmente utilizzato nel campo del Machine Learning e nel Data science. Si tratta di un framework sviluppato in *Python* [§ 4.4] basato su *Flask [16], Plotly.js [17]* e *React.js [18].*
Dash fornisce molti componenti necessari alla realizzazione di applicazioni per la visualizzazione di dati, come grafici, indicatori o controlli UI. Essendo un progetto open source, viene migliorato continuamente in modo da garantire stabilità ed affidabilità nel tempo. Grazie al linguaggio Python offre anche agli utenti con meno esperienza la possibilità di sviluppare la propria applicazione web. Dash è *multipiattaforma.* Infatti, per la sua esecuzione, utilizza il web browser che è presente anche nei dispositivi mobile.

La dashboard contiene, all'interno della directory *assets*, tutte le risorse necessarie per il caricamento dell'applicazione web. In particolare al suo interno troviamo sia i file .CSS che il logo di Probabilistic DNN-Tuner, compreso la favicon che compare sulla tab del browser. Non è necessario specificare il percorso della directory in quanto, di default, assets rappresenta la cartella predefinita per questo genere di contenuti.
Per creare i grafici [10] e gli indicatori [11] presenti nella dashboard sono stati utilizzati dei componenti rilasciati da *Plotly* che necessitano, per il loro corretto funzionamento, di particolari funzioni dette *callback functions* [19]. Le callback functions sono delle funzioni Python che vengono eseguite automaticamente da Dash qualora la proprietà di input di un componente vari.

Ad esempio, ad ogni grafico ed indicatore utilizzato, è stato assegnato un altro componente (*dcc.Interval* [20]) in grado di richiedere, trascorso un determinato intervallo di tempo, l'aggiornamento dei dati, senza dover ricaricare la pagina.
Se non si vuole attendere il refresh, è presente un pulsante per forzare l'operazione in qualsiasi momento.

4.1.1 Dash Bootstrap Components

Bootstrap è un framework appositamente rilasciato per aiutare chi sviluppa la parte *front-end* delle applicazioni web. Esso fornisce una grande raccolta di strumenti già implementati e testati che sono basati su HTML [§ 4.2], CSS [§ 4.3] e JavaScript.
Garantisce compatibilità con le ultime versioni di tutti i principali browser e supporta il *responsive web design*. Infatti, Bootstrap permette alle pagine web di adattarsi dinamicamente a seconda del dispositivo utilizzato. La compatibilità tra Dash e Bootstrap è ottenuta grazie al *Dash Bootstrap Components* [21]. Per realizzare la dashboard si è scelto uno dei temi di Bootstrap disponibili online offrendo un notevole risparmio di tempo [22]. Un componente molto importante, che garantisce l'espandibilità della dashboard, sono le tab [23]. In pochi passi è possibile aggiungere nuove schede ed aggiungere ulteriori contenuti al loro interno.
Per quanto riguarda la parte "responsive", si sfrutta il *grid system* [24] di Bootstrap. Ad esempio, come si può vedere in figura [Fig. 4.1], gli indicatori sono stati disposti su due righe (in rosso) le quali a loro volta sono state suddivise in colonne (in verde).

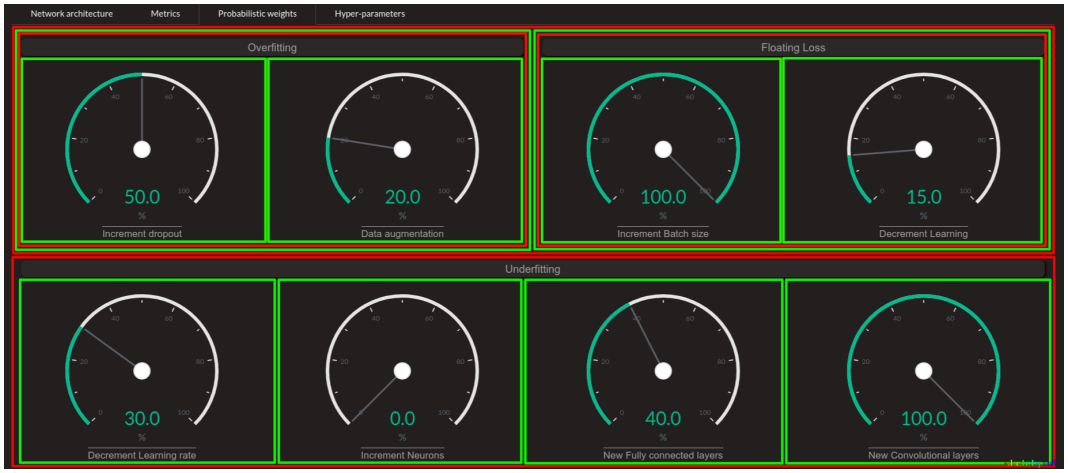

Figura 4.1: Suddivisione componenti
In rosso, le righe (rows), in verde, le colonne (cols)

Utilizzando il Grid system offerto da Bootstrap, possiamo osservare come il layout si adatta alle dimensioni dello schermo del dispositivo in uso:

Figura 4.2: Layout standard a 1880px

Figura 4.3: Layout responsive a 1070px

Figura 4.4: Layout responsive a 1070px

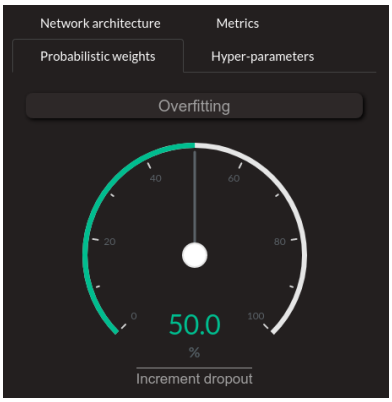

Figura 4.5: Layout responsive a 495px

4.1.2 Dash Cytoscape

Cytoscape è un software open source utilizzato in bioinformatica per la visualizzazione delle reti di interazione molecolare [14]. Per la realizzazione della dashboard si è pensato di integrare le potenzialità offerte da Cytoscape in modo da rappresentare i valori degli iperparametri. I nodi principali, che rappresentano le iterazioni, sono uniti tra loro da una freccia che indica l'ordine di progressione. A ciascun nodo principale, sono collegati altri nodi che contengono il valore degli iperparametri utilizzati per quella specifica iterazione.

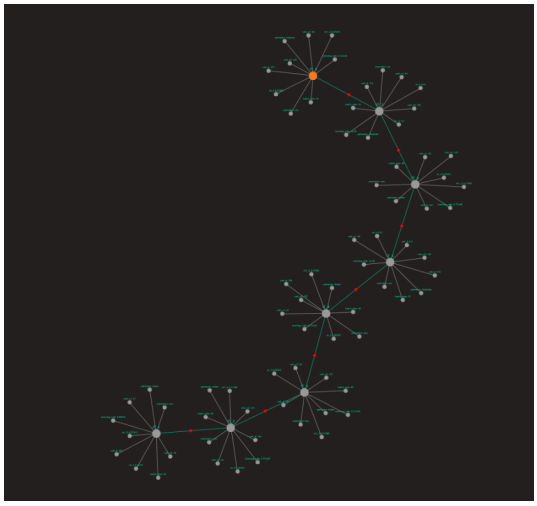

Figura 4.6: Cytoscape contenente gli iperparametri di ogni iterazione

4.2 HTML

HyperText Markup Language (HTML) [25] è un linguaggio di *markup* realizzato per la formattazione e l'impaginazione di documenti *ipertestuali*. La sintassi è stata stabilita dal *World Wide Web Consortium* (W3C) [26] ed è utilizzata come standard per garantire compatibilità tra i diversi browser.
Il framework Dash crea un documento in linguaggio HTML che, attraverso l'uso di tag, definisce la struttura dell'intera pagina. Uno tra questi, rivelatosi particolarmente utile, è il tag *iframe* (inline frame). Esso permette di visualizzare ed interagire con una pagina HTML contenuta dentro un'altra pagina.
Specificando, in maniera opportuna, l'indirizzo della pagina da visualizzare all'interno dell'iframe, è stato possibile integrare *Netron* all'interno della dashboard realizzata.

4.3 CSS

Per personalizzare la formattazione della dashboard è stato utilizzato un linguaggio apposito: *Cascading Style Sheets* (CSS) [27].
CSS è uno strumento che permette di separare il contenuto della pagina dalla sua formattazione. Nella dashboard si è scelto di non scrivere i file CSS partendo da zero ma di utilizzare uno dei temi rilasciati da Bootstrap e disponibili online [22]. Questa scelta ha permesso di risparmiare molto tempo, ottenendo una applicazione responsive che è in grado di adattarsi a schermi di diverse dimensioni.

4.4 Python

È un linguaggio di programmazione di "alto livello", orientato a oggetti, che vuole essere *semplice, flessibile e dinamico* [28]. E' stato rilasciato da *Guido van Rossum* nel 1991. Possiede numerose librerie utili per chi sta sviluppando software rivolti al mondo del Machine Learning come ad esempio *TensorFlow, Keras, Pytorch* e *Scikit-learn.*
Tra le caratteristiche che lo distinguono da altri linguaggi, ci sono le variabili non tipizzate e la mancanza di parentesi o segni di punteggiatura. Attraverso il framework Dash è possibile sfruttare tutta la sua potenza per andare a realizzare applicazioni web con poche righe di codice.

4.5 SQLite3

Per raccogliere i dati ottenuti per ciascuna iterazione si è scelto di utilizzare un database relazionale che fosse compatto, veloce e semplice [29].
SQLite3, non essendo un processo stand alone, può essere integrato facilmente all'interno dell'applicazione realizzata. La libreria è stata scritta in linguaggio C ed in alcuni casi si rivela più veloce di altri RDBMS [30] rinomati come *MySQL e PostgreSQL*. Non è in grado di gestire le problematiche riferite alla concorrenza, ma nel nostro caso questo non è necessario. In questi ultimi anni ha avuto molto successo e viene utilizzato in software rinomati come *Chrome, Mozilla Firefox, Skype* ed *Adobe Photoshop Lightroom*.

4.5.1 Struttura database

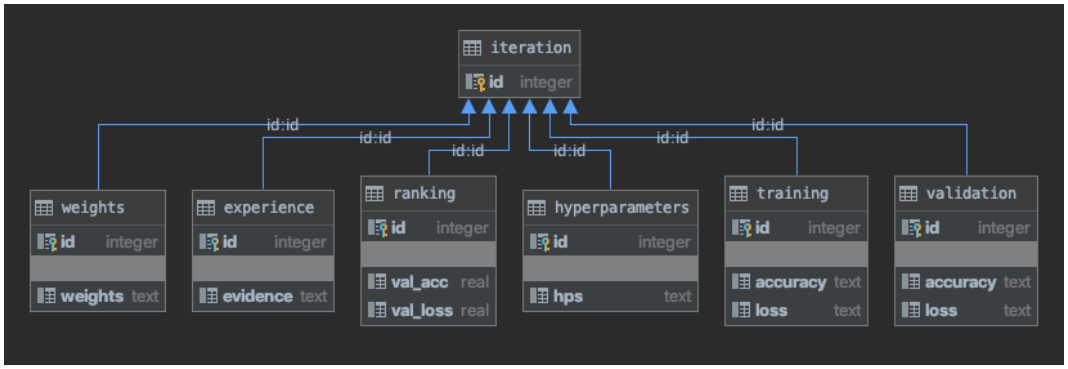

Figura 4.7: Struttura del database realizzato con SQLite3

Come si può notare in figura [4.7], abbiamo un'entità iteration da cui discendono altre 6 entità. Queste ultime sono collegate mediante una chiave esterna (l'attributo id), alla chiave primaria id contenuta all'interno di iteration.

Per ogni iterazione che avviene durante la fase di training, viene aggiunta una nuova tupla in ciascuna relazione presente all'interno del database. In questo modo, vengono registrati tutti i parametri utilizzati per allenare la DNN. Viene di fatto creato uno storico con il quale è possibile tenere traccia delle scelte attuate dall'algoritmo contenuto in Probabilistic DNN-Tuner. La dashboard è stata realizzata con l'intento di presentare i dati riferiti all'ultima iterazione, ad eccezione della tab contenente gli hyper-parameters, dove è possibile consultare lo storico completo attraverso il cytoscape appositamente implementato.

Capitolo 5

Conclusioni

Con il lavoro svolto, durante il periodo di internato, è stato possibile sfruttare le risorse messe a disposizione dal framework Dash per realizzare una dashboard compatibile con il software *Probabilistic DNN-Tuner*.

In particolar modo, l'uso del framework *Dash Bootstrap Components*, ha reso possibile ottenere un'applicazione web responsive ed espandibile. Il *Grid system*, implementato da Bootstrap, definisce delle regole che permettono di organizzare velocemente e al meglio il layout della pagina. Raccogliendo i componenti in più tab si è potuto aumentare la loro dimensione, migliorando l'esperienza di utilizzo. La dashboard, così realizzata, può essere utilizzata su qualsiasi piattaforma, a prescindere dalle dimensioni dello schermo del dispositivo.

Anche se il risultato finale rispetta le caratteristiche desiderate, sarebbe interessante introdurre ulteriori funzioni che possano migliorare l'interazione con l'utente. Tra queste, potrebbe rivelarsi utile, dare la possibilità all'utente di inserire manualmente i dati elaborati dalla dashboard, direttamente dalla pagina principale.

Bibliografia

[1] Artificial Intelligence. [Online]. Disponibile: https://en.wikipedia.org/wiki/Artificial_intelligence

[2] Machine Learning. [Online]. Disponibile: https://en.wikipedia.org/wiki/Machine_learning

[3] I. Goodfellow, Y. Bengio, e A. Courville, *Deep learning*. MIT press, 2016.

[4] Artificial Neural Network. [Online]. Disponibile: https://en.wikipedia.org/wiki/Artificial_neural_network

[5] Chart of neural networks. [Online]. Disponibile: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

[6] Main types of neural networks and its applications. [Online]. Disponibile: https://medium.com/towards-artificial-intelligence/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e

[7] Artificial neural network. [Online]. Disponibile: https://developer.nvidia.com/discover/artificial-neural-network

[8] Bayesian optimization. [Online]. Disponibile: https://en.wikipedia.org/wiki/Bayesian_optimization

[9] Michele Fraccaroli, Evelina Lamma, Fabrizio Riguzzi, "*Automatic Setting of DNN Hyper-Parameters by Mixing Bayesian Optimization and Tuning Rules*."

[10] Plotly line charts. [Online]. Disponibile: https://plotly.com/python/line-charts/

[11] Dash daq gauge. [Online]. Disponibile: https://dash.plotly.com/dash-daq/gauge

[12] Leah A. Wasser. *Hierarchical Data Formats - What is HDF5?* [Online]. Disponibile: https://www.neonscience.org/about-hdf5

[13] Netron. [Online]. Disponibile: https://github.com/lutzroeder/netron

[14] Cytoscape. [Online]. Disponibile: https://js.cytoscape.org/

[15] Dash framework by plotly. [Online]. Disponibile: https://dash.plotly.com/

[16] Flask. [Online]. Disponibile: https://palletsprojects.com/p/flask/

[17] Plotly. [Online]. Disponibile: https://plotly.com/

[18] React. [Online]. Disponibile: https://it.reactjs.org/

[19] Basic dash callbacks. [Online]. Disponibile: https://dash.plotly.com/basic-callbacks

[20] Dash Core Components Interval. [Online]. Disponibile: https://dash.plotly.com/dash-core-components/interval

[21] Dash Bootstrap Components. [Online]. Disponibile: https://dash-bootstrap-components.opensource.faculty.ai/

[22] Bootswatch darkly theme. [Online]. Disponibile: https://bootswatch.com/darkly/

[23] Dash Bootstrap Components Tabs. [Online]. Disponibile: https://dash-bootstrap-components.opensource.faculty.ai/docs/components/tabs/

[24] Grid system by Bootstrap 4. [Online]. Disponibile: https://getbootstrap.com/docs/4.0/layout/grid/

[25] HyperText Markup Language. [Online]. Disponibile: https://en.wikipedia.org/wiki/HTML

[26] World Wide Web Consortium. [Online]. Disponibile: https://en.wikipedia.org/wiki/World_Wide_Web_Consortium

[27] Cascading Style Sheets. [Online]. Disponibile: https://en.wikipedia.org/wiki/CSS

[28] Python. [Online]. Disponibile: https://www.python.org/

[29] Sqlite. [Online]. Disponibile: https://www.sqlite.org/index.html

[30] Relational Database Management System. [Online]. Disponibile: https://it.wikipedia.org/wiki/Relational_database_management_system/