UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea Triennale in Ingegneria Elettronica

Implementazione di una rete neurale convoluzionale per il riconoscimento di segnali elettrici a frequenza di rete

Relatore:
Prof. Tommaso Caldognetto
Correlatore:
Prof. Marco Stellini

Candidato:
Andrea Tonon

Anno Accademico 2023/2024

Abstract

In questa tesi verranno trattate la struttura e il funzionamento delle reti neurali convoluzionali, o CNN, e ne verrà descritto il processo di sviluppo con l'obbiettivo di costruire un riconoscitore di segnali elettrici. Per la creazione del dataset è stato utilizzato uno script Matlab per la generazione di segnali puliti e rumorosi. Successivamente i segnali sono stati normalizzati, etichettati e salvati mediante l'utilizzo di un programma Python. In seguito, è stato creato il modello della rete neurale mediante l'utilizzo della libreria Pythorch. Infine, è stato eseguito il processo di training della rete e ne è stata verificata la correttezza. Questo procedimento ha portato alla creazione di un file contenente i pesi della rete neurale, che possono poi essere utilizzati per le implementazioni su hardware.

Indice

Elenco delle figure

INTRODUZIONE

1.1 La nascita dell'intelligenza artificiale

Fin dall'antichità l'uomo si è sempre dimostrato curioso ed affascinato dalla propria mente e da tutto ciò che era in grado di immaginare e realizzare con essa. Nel corso dello scorso secolo, con l'avvento dei primi computer ed una maggiore comprensione del cervello umano, sono iniziati i primi tentativi per la creazione di intelligenze artificiali, o AI dall'inglese. Con il termine "intelligenza artificiale" si fa riferimento ad un ampio insieme di concetti, in quanto la parola "intelligenza" contiene in sé diversi significati. Sicuramente si può affermare che è una disciplina che ha come scopo quello di far ragionare i computer similmente alla mente umana. Si vuole quindi ricreare alcune capacità dell'uomo, cioè ottenere un sistema capace di percepire l'ambiente che lo circonda, ragionare, ed agire in base ad esso. Uno dei precursori di questa disciplina è stato Alan Turing. Nel 1936 ha introdotto il concetto di macchina universale: un dispositivo teorico in grado di risolvere qualsiasi problema computazionale. Successivamente nel 1950 ha pubblicato un articolo intitolato "Computing Machinery and Intelligence" nel quale descrive il famoso test di Turing, ma introduce anche alcuni concetti volti alla creazione di un'intelligenza artificiale che si avvicini ancora di più a quella umana. Per ottenere questo, l'AI non deve essere solamente capace di risolvere un problema specifico, ma deve riuscire ad imparare. C'è quindi la necessità di ricreare artificialmente non solo il risultato finale ma tutto il processo di sviluppo della mente umana.[1]

1.2 Concetti chiave

Dall'idea di Turing si è sviluppato il machine learning, o ML, una branca dell'intelligenza artificiale che si occupa di sviluppare sistemi in grado di imparare dai dati. I modelli creati da questa disciplina vengono addestrati attraverso l'analisi di grandi quantità di dati ed utilizzano l'esperienza accumulata per fare delle previsioni. Questi algoritmi sono in grado, attraverso l'addestramento, di migliorarsi analizzando gli errori commessi.

Uno strumento fondamentale per il machine learning sono le reti neurali artificiali, o ANN. Rappresentano un insieme di modelli sviluppati ispirandosi alla struttura delle reti neurali biologiche. Si compongono di nodi, o neuroni artificiali, organizzati su più livelli e connessi tra loro. Come nelle reti biologiche, ogni neurone possiede una soglia di attivazione ed ogni

collegamento sinaptico tra due nodi è caratterizzato da un peso. Le reti artificiali, che realizzano delle funzioni, riescono ad imparare dai dati che analizzano andando a modificare i valori dei pesi, ovvero cambiando la funzione che implementano. Queste strutture rappresentano un potente strumento per l'analisi di grandi moli di dati.

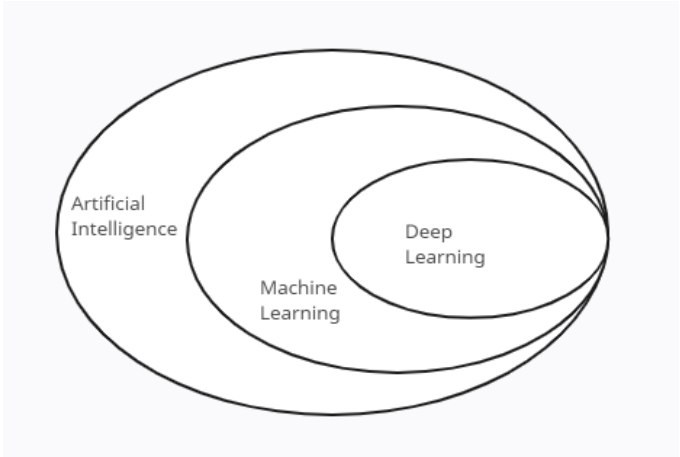

Figura 1.1: rappresentazione delle relazioni tra AI, ML e DL

Un sotto insieme del ML è il deep learning, o DL, costituito da algoritmi che fanno uso di reti neurali su molti livelli. Questo genere di strutture, molto complesse, si sono dimostrate molto efficienti come strumento di classificazione di dati quali immagini e audio.

Tutti questi modelli hanno bisogno per il loro corretto funzionamento di una grande disponibilità di dati per la fase di addestramento. Per questo hanno avuto un grande sviluppo ed una grande diffusione nell'ultimo decennio, con l'avvento dei big data. Essi rappresentano la "fonte della conoscenza" di questi algoritmi. Ad oggi l'AI, e tutte le sue sfaccettature, sono entrate a far parte della nostra quotidianità e sempre più settori fanno affidamento nell'intelligenza artificiale per risolvere problemi complessi. L'economia ed il marketing usano il machine learning per l'analisi dei dati relativi all'andamento dei mercati o alle preferenze dei consumatori. In ambito medico invece sempre più spesso sono presenti esempi di diagnosi assistita da AI, come nella lettura degli elettrocardiogrammi. Anche nella nostra quotidianità abbiamo esempi di AI come gli assistenti vocali o più semplicemente il motore di ricerca Google.

1.3 Obbiettivo della tesi

L'obbiettivo di questa tesi è quello di analizzare il funzionamento e il processo di sviluppo di una rete neurale convoluzionale. A tal scopo questo lavoro di tesi è incentrato sullo sviluppo di una rete neurale convoluzionale che permetta il riconoscimento di segnali sinusoidali alla frequenza di rete e che li classifichi in due categorie: segnali puliti e segnali rumorosi. Per riuscire in questo obbiettivo verrà prima fatto uno studio delle reti neurali, analizzandone la struttura ed il funzionamento. Successivamente verranno analizzate nello specifico le reti neurali convoluzionali, facendo chiarezza sui passi chiave nella loro realizzazione. Infine, verrà analizzata l'implementazione della rete attraverso due strumenti principali: Matlab e Pythorch.

LE RETI NEURALI

2.1 Le prime reti neurali

Il primo modello di neurone artificiale venne ideato da McCulloch e Pitts nel 1943. Il neurone riceve un input e produce un output binario in base ad una funzione di attivazione. Questa funzione determina se il neurone viene attivato, e quindi invia un segnale in uscita, oppure se rimane inattivo. Nel caso del neurone di McCulloch e Pitts la funzione di attivazione consisteva in un valore di soglia, ma pose le basi per la realizzazione dei neuroni artificiali moderni. I due ricercatori proposero anche l'assemblaggio di più neuroni, collegati tra loro con dei pesi, per la creazione di reti capaci di risolvere funzioni booleane molto complesse. Tuttavia, questa rete neurale è ancora molto lontana dall'idea di rete intelligente in quanto non riesce ad imparare ed i suoi pesi devono essere prestabiliti.

Tra gli anni '50 e '60 Frank Rosenblatt propose un modello di rete neurale chiamata Perceptron, formata da un solo neurone, il percettrone, che ha un funzionamento molto simile ai neuroni delle reti moderne. Esso riceve in input un vettore, applica pesi diversi ad ogni singolo ingresso e li somma. Il valore ottenuto viene poi confrontato con il valore di soglia della funzione di attivazione per produrre il valore di uscita. La caratteristica più importante di questa rete è la presenza di una funzione ricorsiva per la correzione dell'errore, che va a modificare i pesi degli ingressi quando in uscita viene commesso un errore di previsione. Sebbene Perceptron fosse una rete molto semplice è stata studiata ed applicata a lungo per la sua efficienza nei problemi di classificazione binaria.

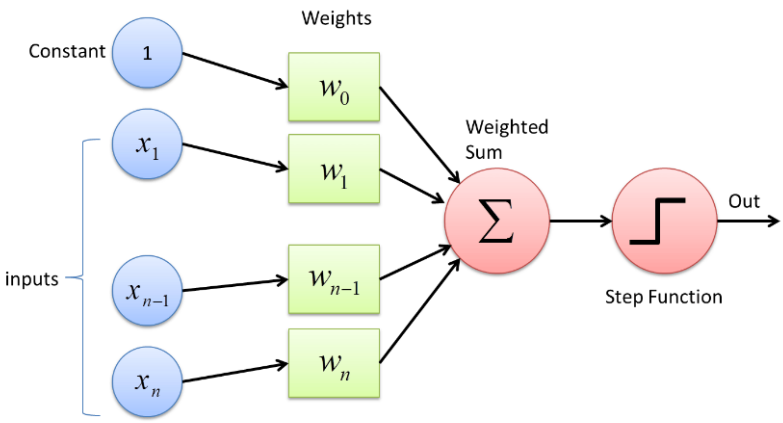

Figura 2.1: schema della struttura di Perceptron [6]

Nel 1969 Minsky e Papert, in una loro pubblicazione, dimostrarono che Perceptron, in quanto rete ad un solo strato, non era in grado di risolvere alcune funzioni logiche come la XOR. Questa pubblicazione insieme alla mancanza di metodi per l'apprendimento di modelli multistrato, distolse l'attenzione dalle reti neurali per diversi anni.

Nel 1975 Fukushima propose il primo modello di rete neurale artificiale capace di riconoscere oggetti indipendentemente dalla loro posizione o rotazione nel campo visivo della rete.

Uno dei passi più importanti nella ricerca in ambito AI è stato la diffusione, nel 1986, dell'algoritmo di Back-Propagation: una funzione ricorsiva per modificare i pesi sinaptici di una rete con un qualsiasi numero di strati e neuroni.

Grazie a questo algoritmo e alla diffusione dei personal computer, dotati di potenze di calcolo maggiori, negli anni '90 l'utilizzo e lo sviluppo di reti neurali aumentò esponenzialmente.[1][2]

2.2 I neuroni artificiali

Le reti neurali artificiali sono progettate con lo scopo di approssimare le reti neurali biologiche e, proprio come esse, sono composte da neuroni, il componente fondamentale per l'elaborazione dell'informazione. Neuroni, o nodi, di reti diverse possono differenziarsi tra loro ma hanno sempre delle caratteristiche comuni. Ogni nodo riceve in ingresso numerosi segnali, provenienti da sorgenti esterne o da altri neuroni, che vengono moltiplicati per un peso specifico prima di essere sommati. Al risultato della sommatoria viene generalmente sottratto un valore di soglia, o bias, prima di venire sottoposto alla funzione di attivazione. Il valore prodotto dalla funzione di attivazione sarà l'output del neurone.

$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right) \qquad (2.1)$$

La funzione di attivazione ha quindi un ruolo centrale nella determinazione del valore in uscita da un nodo, per questo troviamo molte funzioni che possono essere utilizzate in base al tipo di problema che vogliamo risolvere. Tra le funzioni di attivazione più comuni troviamo il gradino,

che genera un output binario, oppure funzioni lineari continue. Una funzione continua permette al neurone di trasmettere segnali con intensità variabile per trasportare maggiore informazione. Sono molto utilizzate anche funzioni di attivazione non lineari in quanto permettono alla rete di imparare a riconoscere delle caratteristiche, o features, complesse.

Nella maggior parte dei casi tutte le funzioni di attivazione vengo limitate negli intervalli [-1,1] o [0:1] con lo scopo di limitare i segnali che viaggiano all'interno della rete.

I segnali in uscita da un neurone possono essere inviati ad uno o più nodi del livello successivo. I nodi della rete sono infatti collegati tra loro formando un grafo con almeno due livelli: uno di ingresso ed uno di uscita. Per le reti neurali dedicate agli algoritmi di deep learning, in mezzo a questi due strati ne sono presenti altri, che non interagiscono con l'ambiente esterno, e per questo sono noti come hidden layers. La struttura a grafo permette l'analisi in parallelo dei dati in input, velocizzando notevolmente i tempi di calcolo.

Per esempio, una rete che vuole riconoscere degli oggetti in una serie di immagini ricercherà dei pattern caratteristici di ogni oggetto, prendendo in considerazione solo piccole porzioni dell'immagine. Ogni nodo analizzerà un solo segmento dell'immagine. In questo modo l'analisi di file, anche molto pesanti come le immagini, composte da migliaia di pixel, risulterà molto veloce.[1]

2.3 Modalità di apprendimento

L'apprendimento di una rete neurale consiste nella determinazione dei pesi ottimali per le connessioni sinaptiche. Esistono diversi modi di apprendere, e tra di essi si sceglierà il più adatto in base al tipo di dati disponibili per l'addestramento e dal problema che si vuole risolvere. I principali metodi sono:

- Addestramento supervisionato: in input viene fornito un set di dati associati a delle etichette, dette labels, corrispondenti al corretto valore di output della rete. I pesi vengono modificati in base all'errore commesso dalla rete rispetto al valore di output atteso. L'addestramento si compone quindi di due fasi: forward e backward. Nella prima i pesi rimangono invariati e l'ingresso attraversa tutti i layers della rete. Nella seconda l'errore viene propagato all'indietro e, tramite opportune funzioni, va a modificare i valori dei pesi e dei bias.

- Apprendimento non supervisionato: si differenzia dal precedente in quanto non vengono fornite le labels in ingresso. La rete scopre da sola le correlazioni esistenti tra i dati. Questa modalità di apprendimento può essere essa stessa l'obbiettivo della rete nel caso in cui si stiano cercando delle relazioni complesse tra i dati.
- Apprendimento con rinforzo: la rete interagisce con un ambiente dinamico e deve raggiungere un certo obbiettivo. In base alle azioni che esegue riceve dei premi o delle punizioni in modo da indirizzarla verso la risposta giusta. Questo tipo di apprendimento è spesso usato in applicazioni di AI quali videogiochi o sistemi di controllo come la guida autonoma.

Durante l'apprendimento si possono verificare diversi problemi, tra questi c'è quello di overfitting. Questo errore implica che la rete ha imparato a memoria i dati usati per l'addestramento senza imparare a generalizzare i pattern riconosciuti. Le cause di questo problema possono essere molteplici, tra le più comuni troviamo: una rete eccessivamente complessa rispetto al problema da risolvere o un dataset di addestramento troppo piccolo.[9]

2.4 Introduzione alle reti neurali convoluzionali

Le reti neurali convoluzionali, o CNN, nascono negli anni '80 e '90. Uno dei pionieri di queste reti fu Yann LeCunn, che cercò di ricreare il funzionamento della corteccia visiva del cervello umano per riconoscere pattern visivi complessi. Nel 1998 creò LeNet-5, la prima CNN in grado di riconoscere le cifre scritte a mano. Tuttavia, è solo nell'ultimo decennio che lo sviluppo e l'impiego delle reti convoluzionali ha preso piede, specialmente grazie all'aumento della potenza di calcolo delle GPU (Graphics Processing Unit) e all'accesso a grandi quantità di dati, necessari per il funzionamento corretto delle reti dedicate al deep learning.

Questo tipo di reti si basa sulla convoluzione. Il parametro fondamentale di questa operazione è il filtro, o kernel, costituito da alcuni valori numerici che scorrono attraverso tutti i valori di ingresso.

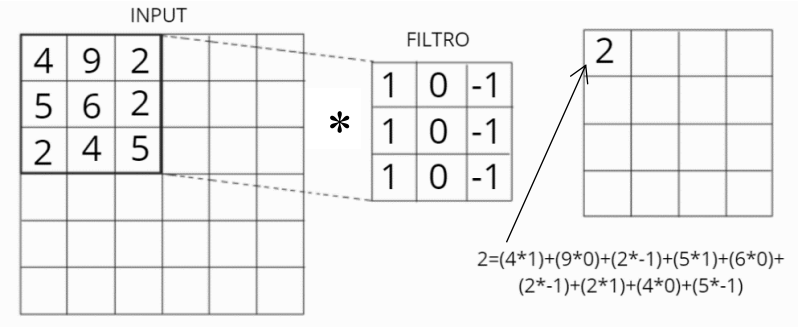

Figura 2.2: operazione di convoluzione discreta all'interno di una CNN

Come si vede in figura 2.2 i coefficienti del filtro moltiplicano i valori di un segmento dell'ingresso con la stessa dimensione del filtro, i prodotti vengono poi sommati e riportati in uscita andando a formare quella che viene chiamata "mappa delle caratteristiche". Questo tipo di operazioni permette l'estrazione di features nascoste presenti in dati complessi.[1] [2]

2.5 Funzione di attivazione

Per il riconoscimento di relazioni complesse la rete deve lavorare con una funzione di attivazione non lineare. Tra le funzioni non lineari più conosciute troviamo sicuramente la sigmoide e la tangente iperbolica, tuttavia, la funzione maggiormente utilizzata è la ReLU.

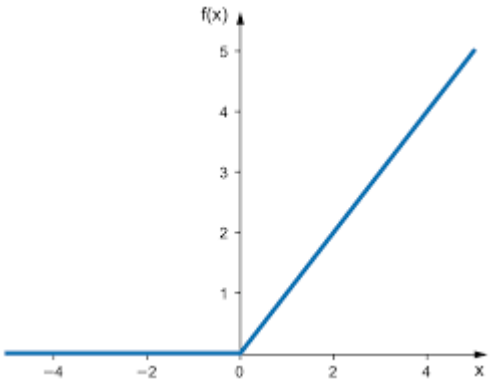

Figura 2.3: rappresentazione grafica della funzione di attivazione ReLU

Restituisce zero se riceve in ingresso un valore minore di zero, mentre restituisce l'input stesso se è maggiore di zero. Possiamo quindi scriverla come in 2.1.

$$f(x) = \max(0, x) \qquad (2.1)$$

2.6 Struttura delle CNN

Le reti neurali convoluzionali fanno parte delle reti dedicate al deep learning e proprio per rendere possibile l'esecuzione di questa categoria di algoritmi sono formate, oltre che dagli strati di ingresso ed uscita, anche da diversi strati nascosti.[2][3]

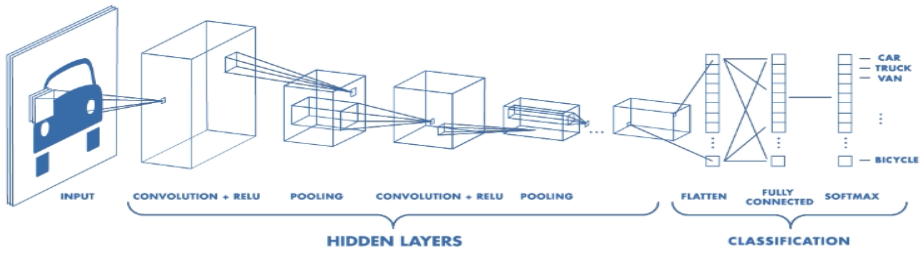

Figura 2.4: struttura di una rete neurale convoluzionale [8]

Gli strati nascosti si dividono principalmente in tre categorie:

- Layer convoluzionale: è lo strato caratteristico delle CNN, in cui avviene l'operazione di convoluzione discreta. Tipicamente sono descritti da quattro parametri: i valori dei coefficienti del filtro, la dimensione dello stesso, lo stride, che definisce come il filtro scorre lungo l'ingresso, e il padding ovvero l'aggiunta di zeri sui bordi dell'ingresso per far sì che l'uscita mantenga le dimensioni dei dati in input.
- Layer di pooling: è uno strato di sottocampionamento, in cui si riducono le dimensioni delle matrici in uscita dagli strati di convoluzione. Si dividono in due categorie: il Max Pooling, in cui viene mantenuto solamente il maggiore dell'insieme di valori considerati, e l' Average Pooling in cui viene eseguita la media dei campioni considerati. Questo strato comporta una riduzione della complessità, un aumento dell'efficienza ed una limitazione al problema di overfitting.

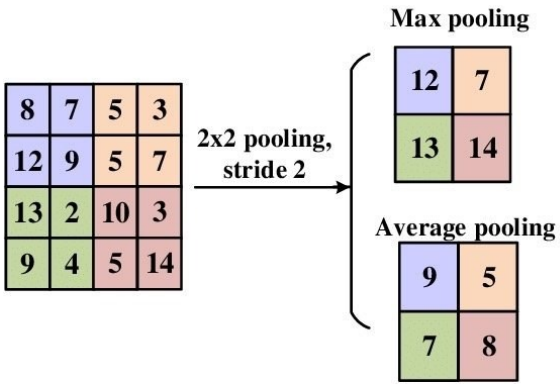

Figura 2.5: operazioni di max ed average pooling [7]

- Layer completamente connesso: generalmente è posizionato alla fine della rete, quando i dati sono già stati elaborati e semplificati. Ogni neurone di questo layer riceve in ingresso tutti i valori prodotti dallo strato precedente. Esso, sfruttando le caratteristiche estratte dagli strati precedenti, risulta fondamentale per la classificazione ed il riconoscimento di relazioni complesse tra i dati.

2.7 Applicazioni

Le CNN sono nate come reti adatte a lavorare su dati complessi come immagini, video o suoni, perché la loro struttura riesce a suddividere questi dati in parti più piccole, che possono essere analizzate separatamente lavorando in parallelo, portando ad una riduzione dei tempi di analisi. Il caso più intuitivo, che storicamente ha rappresentato anche il primo problema in cui queste caratteristiche furono messe in uso, è la ricerca di un oggetto all'interno di un'immagine. Questo è un problema molto complesso in quanto si cerca all'interno di un'immagine un insieme di caratteristiche. Per esempio, possiamo ricercare dei bordi con una forma ben precisa, ma allo stesso tempo dei colori specifici. Le CNN utilizzando filtri diversi, riescono a suddividere l'immagine principale in segmenti che vengono analizzati uno alla volta facendo scorrere i filtri dello strato di convoluzione. Utilizzando più filtri in parallelo, riusciamo a dividere il problema principale in problemi più semplici e veloci da risolvere.

Successivamente le CNN hanno trovato sempre più spazio nell'analisi di segnali audio. I suoni infatti possono essere analizzati attraverso il calcolo del loro spettrogramma, ovvero una rappresentazione bidimensionale che mostra come la potenza del segnale varia in funzione del tempo e della frequenza. Questo tipo di rappresentazione permette di analizzare i suoni come

se fossero delle immagini. Le CNN non erano in origine adatte a lavorare con dati monodimensionali, e si rendevano necessarie delle tecniche di conversione da 1D a 2D per poter utilizzare le strutture di analisi delle immagini. Tuttavia, queste procedure portavano ad alcuni svantaggi: erano richieste le complessità hardware di una rete bidimensionale anche per lavorare con dati più semplici, inoltre si rendeva necessario un dataset molto ampio per l'addestramento e non erano adatte per applicazioni real-time o low-power.

Dal 2015 le reti neurali convoluzionali 1D hanno raggiunto livelli di efficienza paragonabili a quelli delle reti 2D, venendo sempre più utilizzate. Tra i vantaggi che portano troviamo sicuramente una minor complessità computazionale: infatti in una rete bidimensionale la convoluzione viene calcolata tra due matrici quadrate di dimensioni N e K, portando quindi ad una complessità $O(N^2 \cdot K^2)$, mentre in una rete 1D la convoluzione viene eseguita tra vettori portando ad una complessità computazionale $O(N \cdot K)$. L'analisi di dati più semplici permette inoltre l'implementazione di reti meno complesse e con meno hidden layers, rendendo possibile l'addestramento attraverso una semplice CPU, senza dover ricorrere a GPU o ad acceleratori hardware. La semplicità strutturale e la minor complessità computazionale permettono inoltre l'uso delle CNN per utilizzi real-time anche su dispositivi wearables.[2][3]

2.8 Acceleratori hardware

Gli acceleratori hardware sono dispositivi nati per velocizzare le operazioni di calcolo richieste dalle reti neurali profonde, tra cui le CNN. Questi strumenti hardware risultano più performanti rispetto alle CPU dei processori general purpose, rendendo possibile l'utilizzo di reti neurali anche su dispositivi "on edge". Esistono diverse categorie di acceleratori:

- GPU: sono unità di elaborazione grafica caratterizzate da un alto parallelismo. Grazie al modello di programmazione CUDA (computer unified device architecture) di NVIDIA, possono essere utilizzate per l'addestramento e l'esecuzione di reti neurali.
- FPGA: circuiti hardware programmabili. Con l'aumento della densità di integrazione dei processi tecnologici, è stato possibile creare dispositivi di dimensioni ridotte capaci di implementare strutture di reti neurali molto complesse. La velocità di calcolo, il basso consumo e la riprogrammabilità hanno reso questi dispositivi molto richiesti, specialmente per le applicazioni on-edge.

- ASIC: circuiti hardware specifici (application specific integrated circuit). Sono disponibili diverse architetture, ognuna delle quali presenta caratteristiche specifiche adatte ad un utilizzo specifico.

DESIGN DEL PROGETTO

Il problema che vogliamo risolvere è il riconoscimento, e quindi classificazione, di segnali sinusoidali. I segnali verranno creati e suddivisi in due gruppi in base alla presenza o meno di un disturbo e dovranno essere classificati dalla rete in due categorie: segnali rumorosi e segnali puliti. Come abbiamo visto i problemi di classificazione sono il maggior ambito d'applicazione delle reti neurali convoluzionali, sarà quindi necessario sviluppare una CNN, addestrarla con un set di dati ed infine eseguire dei test per verificarne il funzionamento.

3.1 Creazione ed ottimizzazione dei dati

Una CNN si basa sull'apprendimento attraverso l'analisi di un dataset, per questo la prima operazione da eseguire è la creazione di un set di segnali. Questi segnali dovranno essere delle sinusoidi e appartenere al range di frequenze considerato quindi comprese tra i 10 Hz e i 20kHz. I segnali creati dovranno essere divisi in due categorie: puliti, ovvero privi di rumore, e rumorosi. Come disturbo verrà utilizzato un rumore bianco gaussiano. Sarà necessario fare attenzione alla potenza del rumore, essa infatti dovrà essere superiore al rumore che, in una applicazione pratica, può essere associato per esempio al quantizzatore di un ADC, che non sarà mai evitabile.

Per la creazione dei segnali verrà utilizzato Matlab che, oltre a permetterci la creazione di segnali sinusoidali, mette a disposizione una funzione apposita per la creazione di un disturbo gaussiano. Inoltre, permette il salvataggio dei segnali mediante un numero costante di campioni. Questa caratteristica è importante in quanto i dati usati per l'addestramento della rete devono avere tutti la stessa dimensione.

Successivamente i segnali creati dovranno essere divisi in due gruppi: quelli dedicati all'addestramento, che comprenderanno circa l'80% di tutto il dataset, e quelli di test. Questa divisione è necessaria per riuscire a valutare correttamente le prestazioni della rete una volta finito l'addestramento.

Per poter utilizzare questi segnali nella rete neurale sarà necessario eseguire una normalizzazione di tutti i campioni. I segnali normalizzati saranno poi salvati all'interno di un vettore, e contestualmente verrà creato un altro vettore contenente le rispettive etichette: uno 0 per i segnali puliti ed un 1 per i segnali rumorosi.

3.2 Struttura della rete

Per la realizzazione di una rete neurale uno dei linguaggi di programmazione più utilizzati è Python in quanto fornisce diversi vantaggi. Tra questi troviamo: la presenza di molte librerie specifiche per il machine learning, come Pythorch o TensorFlow, ed un'alta ottimizzazione dei calcoli grazie all'uso di librerie apposite e alla possibilità di attivare l'accelerazione GPU. Per la creazione della CNN verrà utilizzata la libreria Pythorch in quanto garantisce velocità e semplicità nella costruzione della rete. Per l'implementazione di tutti gli strati della rete basterà richiamare delle funzioni di libreria assegnando i valori desiderati ai vari parametri della rete.

Il problema che vogliamo risolvere richiede l'analisi, sul dominio del tempo, dei campioni di un segnale. La rete neurale sarà quindi caratterizzata da una struttura monodimensionale. Gli ingressi saranno dei vettori, per questo anche i filtri convoluzionali dovranno essere vettoriali, la cui dimensione sarà inizialmente impostata a tre e i valori numerici saranno casuali.

Avendo un set di dati semplici e non complessi come potrebbero essere delle immagini, la CNN avrà bisogno di un numero di layers ridotto: oltre agli strati di ingresso e uscita sarà presente un solo layer convoluzionale, seguito da uno di pooling ed infine un layer completamente connesso.

Il sottocampionamento sarà di tipo MaxPooling e prenderà in considerazione coppie di valori, in modo da dimezzare la dimensione dei dati.

Come funzione di attivazione useremo la ReLU che come anticipato precedentemente è la funzione più utilizzata e grazie alla sua non linearità permette il riconoscimento di pattern complessi.

3.3 Fase di addestramento

Vista la disponibilità di dati già etichettati e l'obbiettivo della rete, la metodologia più adatta per ottenere i valori migliori dei pesi è l'addestramento supervisionato.

L'addestramento della rete si sviluppa in diverse fasi chiamate epoche. In ognuna di esse viene analizzato l'intero set di dati destinati all'addestramento. Maggiore è il numero di epoche maggiore è la precisione del modello, tuttavia, in presenza di dataset molto grandi può richiedere un tempo molto lungo e causare problemi di overfitting.

Durante ogni epoca i dati di ingresso vengono suddivisi in gruppi e fatti passare attraverso la rete nella fase chiamata forward propagation. Solo alla fine della propagazione di ogni insieme di segnali si avrà la fase di back propagation dell'errore in cui si modificano i valori dei pesi.

Gli altri parametri importanti per l'addestramento sono la funzione di loss e l'ottimizzatore. La funzione di loss serve a quantificare quanto la predizione del modello si discosta dalla realtà. La Cross-Entropy Loss Function è una delle più usate nelle reti dedicate ai problemi di classificazione. La funzione di ottimizzazione invece, serve a regolare le modifiche dei pesi della rete per ridurre il valore di perdita. Anche in questo caso esistono numerose funzioni, ma quella che verrà usata nel nostro progetto è la funzione Adam. È un algoritmo che modifica dinamicamente il tasso di apprendimento, un iperparametro che indica quanto velocemente la rete apprende. Questa caratteristica garantisce all'algoritmo velocità e robustezza.[4]

3.4 Fase di test

Una volta addestrata la rete dovrà essere testata attraverso i segnali che non sono ancora stati utilizzati. I segnali verranno dati in ingresso alla rete, i cui pesi non verranno più modificati. I risultati verranno confrontati con la classificazione corretta e verrà calcolata la percentuale di precisione della rete. Se l'efficacia della rete non è soddisfacente si deve procedere con un nuovo addestramento andando però a modificare alcuni dei parametri utilizzati. Per esempio, si può utilizzare un dataset più ampio, oppure una struttura della rete più complessa e delle funzioni di loss ed ottimizzazione diverse. La fase di test deve essere svolta con cura per assicurarsi che i risultati rispecchino la vera precisione della rete.

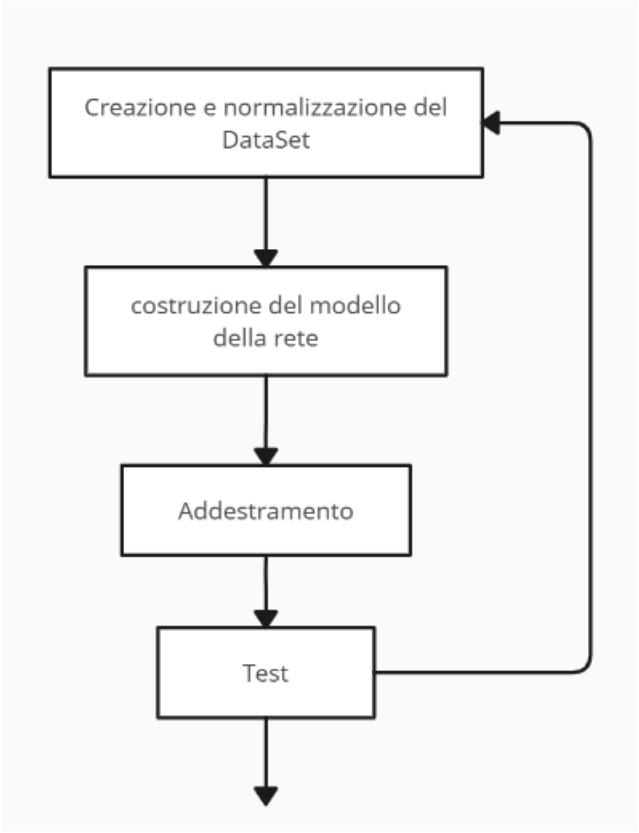

Figura 3.1: processo di costruzione di una CNN

Alla fine dei test, se il risultato è soddisfacente, è possibile salvare i pesi in un file. Sarà il file che servirà qualora si volesse caricare la rete su un dispositivo per delle applicazioni pratiche.

REALIZZAZIONE DELLA RETE

4.1 Creazione del dataset

Come descritto precedentemente, il primo passaggio fondamentale per la realizzazione della CNN è la creazione dei dati da utilizzare per l'addestramento e il test della rete. Per la realizzazione dei segnali verrà utilizzato un codice Matlab.

```
f=10;
fs=44*10^6;
L=1001;
for i=1:1999
    T=1/f;
    nome=f+"";
    nome_n=f+"n";
    t=(0:(T/1000):T);
    y=0.5*sin(2*pi*f*t);
    plot(t,y)
    save (nome+'.mat' , "t","y")
    load(nome+'.mat')
    filename='C:\Users\Utente\Desktop\segnali\puliti\'+nome+'.wav';
    audiowrite(filename,y,fs);
    w_noise = wgn(1,1001,-20);
    y1=y+w_noise;
    hold on
    plot(t,y1)
    save (nome_n+'.mat' , "t","y1")
    load(nome_n+'.mat')
    filename='C:\Users\Utente\Desktop\segnali\rumorosi\'+nome_n+'.wav';
    audiowrite(filename,y1,fs);
    f=f+10;
    hold off
end
```

Figura 4.1: codice Matlab per la creazione del dataset

In questo codice vengono inizialmente impostate tre variabili: la frequenza di partenza per la creazione dei segnali, la frequenza di campionamento e la lunghezza del segnale in numero di campioni. Successivamente tramite un ciclo for, che incrementa la frequenza del segnale di 10 Hz ad ogni iterazione, vengono creati sia i segnali puliti sia quelli rumorosi. I segnali vengono salvati in due cartelle diverse.

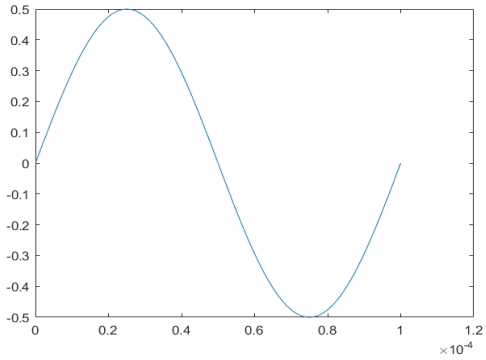

Figura 4.2: grafico del segnale privo di disturbi

Per la generazione del rumore da sommare al segnale sinusoidale di base, viene utilizzata la funzione "wgn" che genera un rumore bianco gaussiano con le specifiche dettate dai tre parametri presenti. I primi due servono a specificare la lunghezza del vettore contenente i campioni del disturbo, che sarà uguale a quella del segnale impostata all'inizio del codice. Il terzo parametro indica la potenza del rumore in dBW. Ovvero viene specificata la potenza rispetto ad 1 Watt. Nella realtà i segnali non saranno mai completamente privi di disturbi; infatti, anche nel caso di un segnale estremamente pulito, in fase di lettura di un convertitore analogico digitale verrà inserito un certo errore dovuto alla quantizzazione. Perciò sarà necessario considerare come segnali rumorosi quelli con una potenza di rumore superiore ad una certa soglia. Il confronto verrà fatto tra i valori del rapporto segnale rumore dovuto alla quantizzazione di un ADC a 10 bit e del segnale rumoroso considerato. Per prima cosa va calcolata la potenza del segnale sinusoidale privo di rumore:

$$P = \left(\frac{A_{\mathrm{RMS}}}{\sqrt{2}}\right)^2 = \left(\frac{0.5}{\sqrt{2}}\right)^2 = 125\,\mathrm{mW} \qquad (4.1)$$

Sapendo che il terzo parametro della funzione wgn indica una potenza di rumore pari a 1 centesimo di Watt, possiamo calcolare l'SNR del segnale rumoroso e l'SNR del quantizzatore, ricordando che stiamo considerando un convertitore a 10 bit:

$$\mathrm{SRN}_q = 6.02 \times n + 1.76 \approx 62\,\mathrm{dB} \qquad (4.2)$$

$$\mathrm{SNR}_{\mathrm{dB}} = 10\log_{10}\left(\frac{P_{\mathrm{segnale}}}{P_{\mathrm{rumore}}}\right) = 10\log_{10}\left(\frac{125\,\mathrm{mW}}{10\,\mathrm{mW}}\right) \approx 11.96\,\mathrm{dB} \qquad (4.3)$$

Vediamo che il rapporto segnale rumore della sinusoide rumorosa è peggiore di quello calcolato con l'errore di quantizzazione. Questo significa che, come desiderato, in un'applicazione pratica, i segnali caratterizzati da un disturbo simile a quello di quantizzazione verranno considerati come dei segnali puliti.

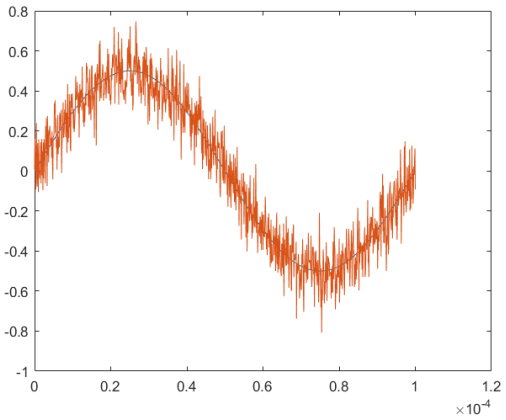

Figura 4.3: grafico del segnale con un disturbo di potenza 10 mW

I segnali vengono convertiti e salvati come file audio, nel formato .wav. In questo tipo di file sarà presente un'intestazione, contenente alcune informazioni come la frequenza di campionamento e la lunghezza del file, seguita dai campioni del segnale. Nel formato .wav i dati non vengono compressi, quindi non si ha perdita di informazione e si mantengono tutti i campioni prodotti da Matlab. Inoltre, è un formato facilmente leggibile in ambiente Python.

Una volta creato, tutto il dataset verrà ulteriormente diviso creando quattro gruppi di segnali distinti. Saranno presenti: due cartelle contenenti l'80% dei segnali puliti e di quelli rumorosi, che verranno dedicate per l'addestramento, e due cartelle con i restanti segnali dedicate alla fase di test.

Come descritto nel capitolo precedente i dati dovranno anche essere normalizzati, etichettati e salvati in un vettore. Queste operazioni vengono fatte attraverso un programma Python.

```python
def normalize_signal(signal):
    #max=np.max(np.abs(signal))
    max=np.max(signal)
    min=np.min(signal)
    #normalized=signal/max if max>0 else signal
    normalized=((signal-min)/(max-min))
    return normalized

for filename in os.listdir(cartella_train_puliti):
    if filename.endswith('.wav'):
        audio_path=os.path.join(cartella_train_puliti, filename)
        audio_data, sampling_freq=librosa.load(audio_path, sr=None)
        normalized=normalize_signal(audio_data)
        #print(len(normalized))
        train_signals.extend([normalized])
        #print(normalized)
        train_labels.append(0) # etichetta segnali puliti: 0
```

Figura 4.4: codice Python dedicato alla normalizzazione ed etichettatura dei segnali

Il codice in figura 4.4 esegue un ciclo for nel quale vengono analizzati tutti i file presenti nella cartella contenente i segnali puliti dedicati all'addestramento della rete. Ogni segnale viene letto attraverso una funzione di libreria e normalizzato attraverso la funzione "normalize_signal".

$$\text{normalized} = \frac{\text{signal} - \min}{\max - \min} \qquad (4.4)$$

La normalizzazione utilizzata, rappresentata dalla formula 4.4, normalizza i campioni nell'intervallo [0,1]. Una volta normalizzato il segnale viene concatenato al vettore dedicato ai segnali di training. Viene aggiornato anche il vettore contenente le etichette, dove verrà aggiunta l'etichetta corrispondente al segnale analizzato, ovvero 0 per i segnali puliti e 1 per quelli rumorosi.

Questo procedimento viene applicato a tutti i segnali, puliti e rumorosi, sia a quelli dedicati all'addestramento sia a quelli riservati alla fase di test.

I vettori così creati vengono poi convertiti in array numpy. Questi vettori speciali, appartenenti ad una libreria di Python, permettono alcune funzionalità avanzate, tra cui una velocità superiore nelle operazioni algebriche tra essi e come vedremo anche la loro trasformazione in Tensori, l'oggetto matematico principe dello sviluppo delle reti neurali convoluzionali con la libreria Pythorch.

4.2 Costruzione del modello

Per la costruzione della rete neurale come anticipato useremo la libreria Pythorch e più precisamente il suo modulo torch.nn che fornisce non solo i layers di cui abbiamo bisogno ma anche le funzioni di attivazione, loss e ottimizzazione.

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool1d(kernel_size=2, stride=2)
        self.fc = nn.Linear(16 * dimension_after_pooling, 2)  # Output layer with 2 classes

    def forward(self, x):
        x = x.unsqueeze(1)  # Add channel dimension
        x = self.conv1(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = x.view(x.size(0), -1)  # Flatten
        x = self.fc(x)
        return x
```

Figura 4.5: codice Python per la costruzione del modello della CNN

Come riportato nel codice di figura 4.5 la CNN viene dichiarata come una classe, al cui interno vengono definiti due metodi.

All'interno del metodo "__init__" vengono istanziati i layers che formano la rete neurale. Questa operazione avviene attraverso la chiamata a funzioni di libreria alle quali vengono passati i valori dei parametri desiderati.

Il layer convoluzionale è caratterizzato da cinque parametri:

1. "in_channel=1": indica che gli ingressi alla rete sono monodimensionali;
2. "out_channel=16": dichiara che in uscita avremo 16 mappe di caratteristiche diverse, ovvero che ad ogni input verranno applicati 16 filtri diversi;
3. "kernel=3": i filtri applicati saranno vettori di lunghezza pari a 3;
4. "stride=1": il filtro durante la convoluzione dell'ingresso si sposta di una posizione alla volta;
5. "padding=1": aggiunge uno zero all'inizio ed alla fine del vettore di ingresso per avere in uscita un vettore della stessa lunghezza;

Come funzione di attivazione viene scelta la ReLU, che, come detto precedentemente, è la più comune per i problemi di classificazione.

Per il sottocampionamento è stato scelto un layer di MaxPooling, che, grazie ad un kernel di dimensione 2 e ad un padding anch'esso di lunghezza 2, dimezza il numero di campioni.

Infine, lo strato completamente connesso è caratterizzato da un numero di neuroni pari al numero di campioni in uscita dal layer di pooling. Il suo output, che coincide con l'uscita della rete, ha dimensione 2, come il numero di classi in cui sono divisi i segnali da classificare.

Nel metodo "forward" viene definito il passaggio dei dati da un layer al successivo. Vengono inoltre utilizzate le funzioni unsqeeze e view che servono ad assicurarsi che i tensori di ingresso alla rete e allo strato completamente connesso siano delle dimensioni corrette. La correttezza delle dimensioni dei dati è fondamentale per il corretto funzionamento della rete.

4.3 Addestramento

Il primo passo per l'addestramento è la conversione dei vettori contenenti i segnali e le etichette in tensori. Questi sono dei vettori multidimensionali su cui si basano le reti neurali implementate con la libreria pythorch.

```
train_features=np.load('trainDataset_features.npy')
train_labels=np.load('trainLabels.npy')
X=torch.tensor(train_features, dtype=torch.float32)
Y=torch.tensor(train_labels, dtype=torch.long)
test_features=np.load('testDataset_features.npy')
test_labels=np.load('testLabels.npy')
X1=torch.tensor(test_features, dtype=torch.float32)
Y1=torch.tensor(test_labels, dtype=torch.long)
```

Figura 4.6: trasformazione dei vettori numpy in tensori

Con questa operazione otteniamo quattro tensori: uno contenente tutti i segnali dedicati all'addestramento, uno con i segnali dedicati alla fase di test e due contenenti le rispettive etichette.

Prima di procedere con l'addestramento della rete è necessario definire ancora alcuni parametri fondamentali. Come funzione di loss viene scelta la CrossEntropyLoss Function, che viene istanziata sempre tramite una funzione del modulo di libreria torch.nn. Come ottimizzatore

viene scelta la funzione Adam, che abbiamo visto essere la più adatta ai problemi di classificazione.

```
#definisco le funzioni di loss e di ottimizzazione
loss_function=nn.CrossEntropyLoss()
optimizer=optim.Adam(model.parameters(), lr=0.001)
#creo i dataloader
train_dataset=TensorDataset(X, Y)
train_loader=DataLoader(train_dataset, batch_size=32, shuffle=True)

test_dataset=TensorDataset(X1,Y1)
test_loader=DataLoader(test_dataset, batch_size=32, shuffle=False)

# Addestramento della CNN
epochs = 3
```

Figura 4.7: definizioni necessarie per l'addestramento della rete

In figura 4.7 vediamo anche le definizioni di altri 3 parametri: il dataset, il dataLoader e le epochs. Il TensorDataset è un oggetto messo a disposizione da pytorch per unire i tensori dei dati e delle etichette, in modo da facilitare l'estrazione delle coppie di dati necessarie per l'addestramento supervisionato. Il DataLoader è invece lo strumento con il quale viene caricato nella rete il dataset. Tra i suoi parametri vediamo che viene definito il batch_size, che corrisponde al numero di coppie segnale-etichetta che vengono analizzate dalla CNN prima di applicare le modifiche ai pesi. Infine, viene definito il numero di epoche dell'addestramento.

A questo punto avviene l'addestramento vero e proprio che si svolge attraverso due cicli for: il primo serve a passare da un'epoca a quella successiva. Al suo interno viene azzerata la variabile che contiene l'errore commesso dalla rete e viene impostato il modello in modalità train.

```
for epoch in range(epochs):
    running_loss = 0.0
    model.train()

    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss=loss_function(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print(f'Epoch [{epoch + 1}/{epochs}], Loss: {running_loss / len(train_loader)}')
```

Figura 4.8: addestramento della rete

Il secondo ciclo for, interno al primo, serve a caricare nella rete tutti i dati presenti nel dataset tramite l'utilizzo del dataloader. Si riconoscono inoltre le varie fasi dell'addestramento

supervisionato: inizialmente vengono caricati i dati in ingresso e vengono calcolate le uscite, che vengono confrontate con le etichette dei segnali e viene calcolato l'errore. A questo punto, nella fase di backward, l'errore si propaga all'indietro e viene effettuata la chiamata alla funzione di ottimizzazione che andrà a modificare i pesi della rete.

4.4 Test

Per valutare se l'addestramento ha portato la rete a riconoscere i segnali con una precisione adeguata vanno eseguiti dei test sfruttando i segnali che abbiamo salvato negli appositi tensori, e che non sono stati utilizzati nelle fasi precedenti.

```
model.eval()
correct=0
total=0

with torch.no_grad():
    for inputs, labels in test_loader:
        outputs=model(inputs)
        _, predicted=torch.max(outputs, 1)
        total+=labels.size(0)
        correct+=(predicted==labels).sum().item()

accuracy=correct/total
print(f'Accuracy:{accuracy*100:.2f}%')
```

Figura 4.9: verifica dell'accuratezza del modello

Il modello viene impostato in modalità valutazione per non calcolare una serie di parametri utilizzati per il calcolo dell'errore. Vengono poi inizializzati a zero due contatori: il primo conta tutti i segnali analizzati dalla rete mentre il secondo conta solamente quelli che vengono classificati correttamente. Anche in questo caso tramite il dataloader vengono caricati ed analizzati tutti i segnali presenti nel dataset di test, ed il risultato della classificazione viene confrontato con l'etichetta del segnale. I contatori vengono modificati ad ogni iterazione del ciclo ed alla fine viene calcolata la percentuale di classificazioni corrette su tutte quelle eseguite.

CONCLUSIONI

Eseguendo il codice mostrato nel capitolo precedente possiamo verificare il corretto funzionamento della rete neurale convoluzionale costruita.

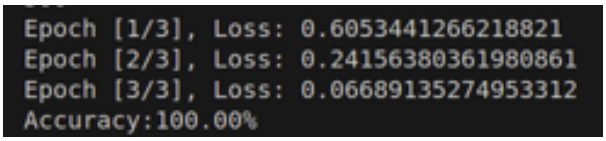

Figura 5.1: risultati dell'esecuzione del codice

Come si vede in figura 5.1 durante la fase di addestramento, il valore in uscita dalla funzione di loss alla fine di ogni epoca si riduce notevolmente, fino a risultare un ordine di grandezza inferiore rispetto a quello iniziale. Possiamo quindi essere soddisfatti delle prestazioni della funzione di ottimizzazione. Inoltre, una riduzione veloce dell'errore implica anche una struttura della CNN adeguata al problema che deve risolvere.

Per quanto riguarda la fase di test, i dati in ingresso vengono classificati correttamente nel 100% dei casi. Il risultato lascia sicuramente alcuni dubbi sulla sua veridicità in quanto, per qualsiasi rete neurale, una risoluzione del problema nel 100% dei casi risulta molto difficile da ottenere. A questo risultato possiamo dare diverse spiegazioni. Sicuramente il problema di classificazione che ci siamo posti è abbastanza semplice per una CNN visto la grossa differenza tra i segnali puliti e quelli rumorosi. Più complessa sarebbe potuta essere la classificazione di segnali con potenze di rumore più simili.

In generale siamo comunque riusciti nella realizzazione di tutti i passaggi necessari per la creazione di una rete neurale convoluzionale, partendo dalla creazione del dataset fino ad arrivare alla fase di test. Sono state inoltre analizzate le maggiori caratteristiche di funzionamento delle reti convoluzionali, studiando tutte le operazioni a cui vengono sottoposti i dati in ingresso.

In questa tesi non è stata realizzata un'implementazione su hardware, tuttavia, grazie alla fase di addestramento è possibile salvare in un file tutti i valori dei pesi dei filtri dello strato convoluzionale. Questo è il file necessario per quella che viene chiamata inferenza, ovvero l'applicazione della rete su nuovi dati.

BIBLIOGRAFIA

[1] Dario Floreano, Claudio Mattiussi. Manuale sulle reti neurali. 2002. Il Mulino.

[2] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, Daniel J. Inman. 1D convolutional neural networks and applications: A survey. Mechanical Systems and Signal Processing 151(2021)

[3] Itamar Arel, Derek C.Rose e Thomas P-Karnowski. “Deep Machine Learning—A New Frontier in Artificial Intelligence Research”. In: (2010).

[4] R. Shimizu, S. Yanagawa, T. Shimizu, M. Hamada and T. Kuroda, "Convolutional neural network for industrial egg classification," 2017 International SoC Design Conference (ISOCC), Seoul, Korea (South), 2017, pp. 67-68, doi: 10.1109/ISOCC.2017.8368830.

[5] Vishnu Subramanian. Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch. Packt Publishing Ltd, 2018. 1788626079, 9781788626071

[6] https://medium.com/@cprasenjit32/perceptron-a-simple-yet-mighty-machine-learning-algorithm-9ff6b7d86a71

[7] Yingge, Huo & Ali, Imran & Lee, Kang-Yoon. (2020). Deep Neural Networks on Chip - A Survey. 589-592. 10.1109/BigComp48618.2020.00016.

[8] MathWorks. https://it.mathworks.com/discovery/convolutional-neural-network.html Accesso il 14 febbraio 2024

[9] https://www.ai4business.it/intelligenza-artificiale/neuroni-artificiali-cosa-sono-come-possono-essere-impiegati/