



**UNIVERSITÀ DEGLI STUDI DI MESSINA**

**DIPARTIMENTO DI INGEGNERIA**

CORSO DI LAUREA TRIENNALE IN: INGEGNERIA BIOMEDICA

---

**“Sviluppo di reti neurali bio-ispirate per il riconoscimento di anomalie in robot mobili”**

Tesi di laurea di:  
**Ivana Calabretta**

Relatore:  
**Chiar.mo Prof. Luca Patanè**  


---

**Anno Accademico 2024/2025**



# Indice

<b>Introduzione .....</b>	<b>3</b>
<b>Capitolo 1: Reti neurali artificiali.....</b>	<b>4</b>
1.1 Contesto biologico .....	5
1.2 Contesto storico .....	7
1.3 Reti CNN .....	10
1.4 Reti SNN.....	11
<b>Capitolo 2: Dai concetti teorici alla progettazione .....</b>	<b>13</b>
2.1 Stato dell'arte.....	13
2.2 Obiettivi .....	15
2.3 Studi preliminari.....	16
<b>Capitolo 3: Approccio metodologico: architettura, dataset e simulazioni.....</b>	<b>23</b>
3.1 Dataset utilizzato e ambiente di simulazione .....	23
3.2 Architettura della rete .....	25
3.3 Fasi di addestramento .....	27
3.4 Validazione e test sulla rete.....	29
3.5 Definizione e ottimizzazione della trashold.....	31
<b>Capitolo 4: Efficienza energetica delle reti neurali .....</b>	<b>37</b>
4.1 Confronto tra Artificial Neural Network (ANN) e Spiking Neural Network (SNN) .....	37
4.2 Confronto sperimentale .....	38
4.3 Valutazione e implicazioni del consumo energetico nelle reti neurali.....	41
<b>Conclusioni e sviluppi futuri.....</b>	<b>43</b>
<b>Bibliografia.....</b>	<b>44</b>
<b>Ringraziamenti.....</b>	<b>46</b>

## Introduzione

Negli ultimi decenni è cresciuto l'interesse nei confronti dei modelli di intelligenza artificiale. Secondo Stuart Russell e Peter Norvig (2010) – autori del celebre libro “Artificial Intelligence: A Modern Approach”:

*“L'intelligenza artificiale si occupa dello studio di agenti intelligenti: sistemi che percepiscono il loro ambiente e intraprendono azioni che massimizzano le probabilità di raggiungere i propri obiettivi.” [9]*

Alla base di molti di questi sistemi intelligenti ci sono le reti neurali artificiali, strumenti potenti che utilizzano algoritmi ispirati al cervello umano. Con l'avvento del Deep Learning si sono sviluppati modelli di rete sempre più evoluti con l'obiettivo di realizzare connessioni in grado di simulare il comportamento dei neuroni biologici. Nascono così le reti neurali spiking, costituite da neuroni che comunicano tramite spike. Questi modelli di rete trovano varie applicazioni; in ambito robotico sono in grado di fornire informazioni su come gestire la navigazione anche in presenza di ostacoli.

L'idea alla base del progetto di tesi è quella di addestrare una rete neurale con l'obiettivo di determinare il valore ottimale di un parametro dell'algoritmo di controllo (*threshold*) che consenta al robot di avvicinarsi agli ostacoli in modo ottimale, riducendo eventuali anomalie durante la navigazione. In particolare, l'ottimizzazione della *threshold* mira a bilanciare la precisione dell'avvicinamento con la stabilità del comportamento del robot, garantendo prestazioni affidabili anche in presenza di ostacoli o in condizioni non ideali.

Il secondo obiettivo che ci poniamo è quello di valutare il consumo energetico delle reti, aspetto necessario per rendere il loro utilizzo sostenibile.

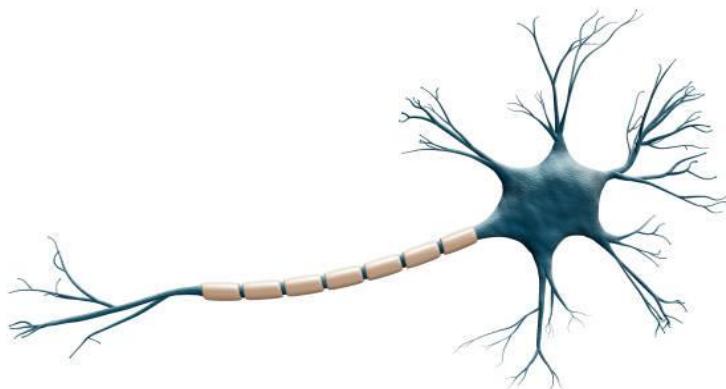
Verranno quindi confrontate implementazioni realizzate con reti neurali convoluzionali ed implementazioni basate su reti spiking.

## Capitolo 1: Reti neurali artificiali

In questo capitolo analizzeremo il funzionamento e i differenti modelli delle reti neurali artificiali. Queste reti sono costituite da una serie di neuroni interconnessi, il cui comportamento può essere simulato da un computer.

### 1.1 Contesto biologico

All'inizio del XX secolo, attraverso gli studi condotti dallo scienziato Santiago Ramón y Cajal, è stata scoperta la struttura dei neuroni, le unità funzionali del sistema nervoso. I neuroni possono essere considerati come dei blocchi in grado di elaborare le informazioni in ingresso, trasformandole in segnali d'uscita. Essi sono costituiti da un corpo neuronale, da uno o più prolungamenti denominati dendriti e dagli assoni. Il corpo neuronale è la parte del neurone che contiene il nucleo, da cui si originano i dendriti. Questi ultimi costituiscono il sistema di ricezione dei segnali, che il neurone può raccogliere da altri neuroni, o, in casi specifici, direttamente dall'ambiente esterno. L'assone prende origine da una sporgenza cronica del corpo neuronale. Il segnale elettrico si origina nel segmento iniziale dell'assone, detto cono di emergenza o monticolo assonico, ed è prodotto dal neurone in risposta a uno stimolo eccitatorio.



**Fig 1.1:** Rappresentazione di un neurone [1]

I neuroni, così come il fluido extracellulare, non sono elettricamente neutri a causa della presenza di ioni al loro interno. Gli ioni si muovono costantemente attraverso la membrana cellulare, sia dentro che fuori la cellula. Il flusso di ioni genera una corrente principalmente attribuita a tre tipi di ioni:  $\text{Na}^+$ ,  $\text{k}^+$  e  $\text{CL}^-$ . Gli ioni si muovono costantemente attraverso la membrana cellulare, sia dentro sia fuori la cellula, perché esistono gradienti di concentrazione e gradienti elettrici (forza elettrochimica) che ne determinano il movimento. Nei neuroni i segnali in ingresso provenienti da più dendriti alterano il voltaggio della membrana neuronale, poiché i neurotrasmettitori rilasciati dalle cellule presinaptiche si legano ai recettori sui dendriti e causano l'apertura o la chiusura di canali ionici. Sebbene l'afflusso di ioni sia continuo, il trasferimento di carica tende a stabilizzarsi quando la

quantità di carica in entrata e quella in uscita, per unità di tempo, si equivalgono. L'equazione di Goldman-Hodgkin-katz consente di calcolare il potenziale di membrana a riposo risultante dal contributo di tutti gli ioni che possono attraversare la membrana.

$$V_m = \frac{RT}{F} \ln \left( \frac{p_K[K^+]_o + p_{Na}[Na^+]_o + p_{Cl}[Cl^-]_i}{p_K[K^+]_i + p_{Na}[Na^+]_i + p_{Cl}[Cl^-]_o} \right)$$

[2]

R = costante universale dei gas

T = temperatura assoluta(inkelvin)

F = costante di Faraday

pK, pNa, pCl = permeabilità *relativa di potassio, sodio e cloro*

[X]o, [X]i = concentrazione extracellulare e intracellulare dello ione X

La variazione della concentrazione di uno degli ioni può modificare la permeabilità della membrana a riposo. Un aumento della permeabilità del sodio depolarizza la membrana cellulare e genera un segnale elettrico, chiamato potenziale d'azione o spike. Il potenziale d'azione è un segnale elettrico di intensità costante che viaggia lungo il neurone. Una sua caratteristica fondamentale è quella di non indebolirsi a causa della distanza, ma al contrario di rigenerarsi inducendo nuovi potenziali d'azione lungo il percorso della fibra. Tutti i potenziali d'azione presentano ampiezza costante. Dopo lo spike, si presenta un breve intervallo di tempo chiamato periodo refrattario. Esso rappresenta un intervallo durante il quale il neurone non può emettere un nuovo potenziale d'azione, indipendentemente dai segnali d'ingresso. Al periodo refrattario assoluto segue quello refrattario relativo, durante il quale è necessario un potenziale graduato più intenso del normale per generare un potenziale d'azione. Una volta raggiunta la soglia, lo spike generato viene trasmesso dal corpo cellulare alla terminazione assonica attraverso l'assone del neurone. Due parametri influenzano la velocità di conduzione: il diametro dell'assone e la resistenza della membrana assonica alla dispersione ionica verso l'esterno della cellula. Esiste una proporzionalità inversa tra i due fattori: aumentando il diametro dell'assone aumenterà la resistenza della membrana con un conseguente aumento della velocità di trasmissione del potenziale d'azione. Alcuni assoni sono rivestiti da una guaina mielinica che consente di trasportare rapidamente il potenziale d'azione su lunghe distanze senza attenuarsi. All'estremità dell'assone, nel terminale assonico, i segnali vengono trasmessi ad altre cellule nervose o a cellule muscolari. Questi punti di collegamento chimico sono chiamati sinapsi. Le sinapsi costituiscono un dispositivo che permette la trasmissione unidirezionale del segnale. Si distinguono in due categorie principali: sinapsi chimiche e sinapsi elettriche. Le sinapsi chimiche sono costituite dall'apposizione di due aree di membrana differenti, entrambe prive di rivestimento gliale. La membrana del neurone trasmittente è denominata membrana presinaptica, e permette la liberazione di un messaggero chimico, o neurotrasmettore. La membrana del neurone ricevente, detta membrana postsinaptica, è invece provvista di molecole proteiche con funzione recettoriale. Tra le due membrane rimane uno spazio, detto fessura o intervallo sinaptico, nel quale vengono rilasciati neurotrasmettitori. Questi sono immagazzinati in vescicole e rilasciati per esocitosi quando un potenziale d'azione raggiunge il terminale assonico. I neurotrasmettitori si legano ai recettori localizzati nelle cellule bersaglio. Le

informazioni relative all'intensità e alla durata dello stimolo sono correlate alla quantità di neurotrasmettore rilasciato. L'efficacia sinaptica dipende sia dalla quantità e al tipo di neurotrasmettitori, sia dal numero di canali ionici attivati. Dopo un certo periodo, le molecole di neurotrasmettitori si staccano dai loro recettori nella fessura sinaptica e vengono assorbite nel terminale dell'assone presinaptico oppure decomposte da specifici enzimi presenti nel fluido extracellulare. Le sinapsi chimiche possono essere eccitatorie o inibitorie. Le sinapsi eccitatorie depolarizzano le cellule postsinaptiche durante la trasmissione sinaptica, facilitando la generazione di potenziali d'azione. Le sinapsi inibitorie, invece, iperpolarizzano la membrana delle cellule postsinaptiche attraverso la trasmissione sinaptica e inibiscono lo sviluppo di potenziali d'azione.

Le sinapsi elettriche rivestono un ruolo importante nel cuore e nel sistema nervoso. A differenza di quelle chimiche, non utilizzano mediatori chimici, ma trasmettono direttamente le cariche della membrana da un neurone al successivo, attraverso giunzioni gap situate sulla membrana di contatto. Le sinapsi elettriche favoriscono una comunicazione rapida e sono generalmente bidirezionali. A differenza dei collegamenti tra i cavi elettrici, progettati per funzionare sempre in modo costante e ottimale, la trasmissione sinaptica può variare in efficacia. Tra un segnale e l'altro puo' presentarsi un intervallo, detto ritardo sinaptico, della durata di frazioni di millisecondo, dovuto alla propagazione del segnale tra le cellule. Inoltre, il segnale può essere trasferito con un'intensità variabile, ponderando il segnale ad ogni passo.

Ogni neurone innerva un bersaglio ed è, a sua volta, innervato da altri neuroni. Di conseguenza, ogni neurone rappresenta un elemento di una catena neuronale, ossia di un circuito nervoso. Un neurone può essere eccitato o inibito, in base alla somma degli stimoli eccitatori e inibitori che riceve in quel determinato momento. Pertanto, il neurone è un sistema modulabile capace di trasmettere un solo tipo di informazione determinato dall'attività dei neuroni che su di esso si scaricano. I neuroni che fanno parte dello stesso circuito sono distribuiti secondo una sequenza geneticamente prestabilita e contribuiscono a una sola attività nervosa. Tale attività si svolge solo nel caso in cui l'ultimo neurone del circuito è in grado di trasmettere l'informazione finale all'effettore.

## 1.2 Contesto storico

Le reti neurali biologiche ricevono dati e segnali esterni tramite gli organi di senso. Queste informazioni vengono elaborate attraverso un vasto numero di neuroni interconnessi che formano una struttura non-lineare e variabile in base agli stimoli esterni ricevuti. Le reti neurali artificiali funzionano in maniera pressoché uguale. Esse sono costituite da un insieme di neuroni in input collegati ai neuroni dello stato in output. Tra di essi possono essere presenti dei neuroni in uno stato intermedio detti interneuroni. Le reti neurali sono in grado di ricevere ed elaborare informazioni attraverso tecniche di apprendimento automatico.

Nel 1943 attraverso gli studi di McCulloch e Pitts fu sviluppato il primo modello di rete con circuiti elettrici in grado di simulare il comportamento dei neuroni. Questo modello utilizzava circuiti elettrici per simulare il comportamento dei neuroni biologici, dando origine alle cosiddette reti neurali di prima generazione. Esse sono in grado di eseguire operazioni matematiche con output booleani e di inviare segnali una volta raggiunta una determinata soglia. Questo modello ha rappresentato la base per lo sviluppo successivo delle reti neurali e ha dimostrato che, in teoria, una rete di questi neuroni era capace di calcolare qualsiasi funzione logica, ponendo così le fondamenta per l'intelligenza artificiale moderna.

Nel 1962 venne proposto il modello del percettrone, in cui l'intensità dell'impulso in ingresso viene espressa come un valore numerico continuo. Nel caso più semplice, si considerano relazioni statiche,

ovvero valide in un preciso istante di tempo. L'input può essere rappresentato con valori binari (0 o 1): 1 quando l'impulso è sufficientemente forte da generare un potenziale d'azione; 0 in caso contrario. Una variante di questo modello si basa sulla considerazione della frequenza di scarico. In questo caso, l'input viene definito come la quantità di potenziali d'azione per unità di tempo. Il neurone bersaglio riceve segnali dai neuroni sorgente attraverso contatti sinaptici la cui efficacia cambia nel tempo in base alla forza dell'attivazione. Se il potenziale della membrana dendritica supera un certo valore di riferimento si verifica un rafforzamento della sinapsi (long-term potentiation), in caso contrario la sinapsi tende a indebolirsi (long-term depression). Successivamente è stata introdotta una seconda generazione di reti neurali, nelle quali la comunicazione tra i neuroni viene modulata dai rispettivi pesi sinaptici. Il neurone somma tutti i segnali ponderati in ingresso confrontandoli poi con il proprio valore di soglia. Se l'input ponderato supera la soglia, allora il neurone si attiva; in caso contrario non si verifica alcuna risposta. Per descrivere il comportamento di un neurone, è quindi necessario considerare tre elementi fondamentali: l'intensità degli input, i pesi sinaptici e la funzione di attivazione. Un grande progresso si realizza con la scoperta e il successivo studio delle funzioni di attivazione continue, relazioni matematiche in grado di esprimere la probabilità che un neurone emetta un potenziale d'azione in funzione della forza dell'input ricevuto. Le funzioni di attivazione permettono di gestire input e output analogici e di modulare il flusso di informazione della rete.

Le funzioni di attivazioni maggiormente utilizzate sono:

- i. La funzione sigmoide che limiterà l'uscita ad un valore tra 0 e 1, e che viene spesso usata nella classificazione binaria. La funzione è la seguente:

$$\sigma(x) = 1/(1 + e^{(-x)})$$

- ii. La funzione SoftMax in cui ogni input viene associato ad una probabilità compresa tra 0 e 1, e che viene spesso usata per la classificazione multclasse.

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K k e^{z_j}}$$

- iii. La funzione Leaky Relu che permette di restituire un valore tra 0 e infinito.

$$\text{LeakyReLU}(x) = x \quad \text{se } x \geq 0$$

$$\text{LeakyReLU}(x) = \alpha x \quad \text{se } x < 0$$

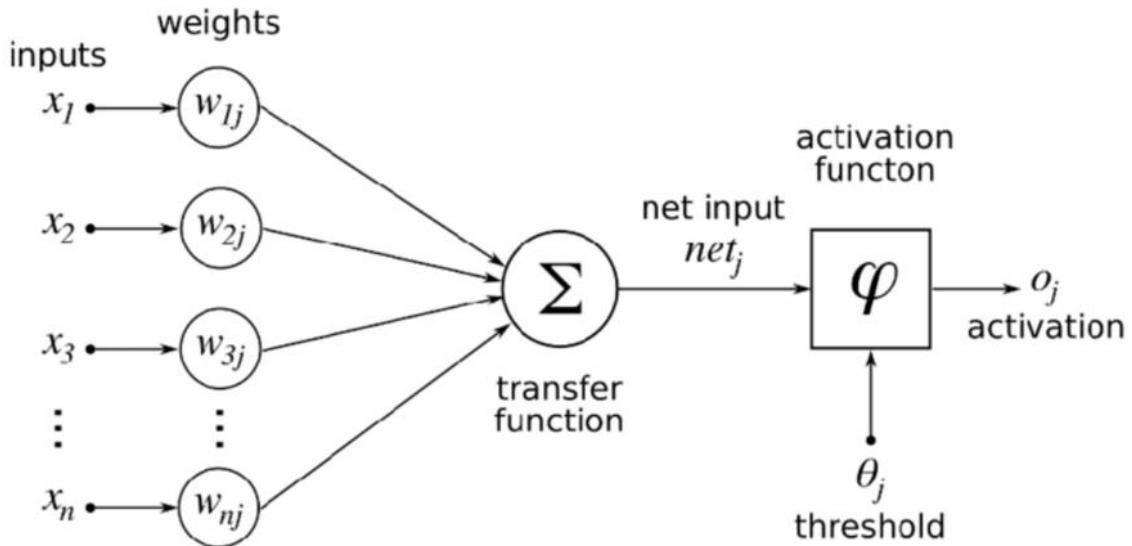
Dove  $\alpha$  è il più piccolo valore positivo che permette di evitare il problema dei neuroni morti, ovvero quando la rete produce sempre zero, portando all'inattivazione dei neuroni.

L'input viene in questo modo ponderato in base alla forza della connessione sinaptica, motivo per cui si parla di peso sinaptico. Se il peso è pari a 1, l'input viene trasmesso per intero al neurone successivo. Per valori compresi tra 0 e 1, l'input viene trasferito in modo proporzionalmente indebolito. Un peso sinaptico pari a 0 corrisponde alla totale assenza di effetto da parte dell'input sul neurone successivo.

Una volta ricevuti tutti gli input dai neuroni connessi, al risultato viene aggiunto un bias, un valore costante sommato al calcolo che coinvolge i pesi.

L'output sarà dato da:

$$\text{output} = \text{inputs} \times \text{weights} + \text{bias}.$$



**Fig. 1.2:** Rappresentazione struttura rete neurale di seconda generazione [3]

L'architettura delle reti neurali implica una variante di questo strato per creare reti neurali profonde dette DNN. Questo tipo di rete è costituita da uno strato di ingresso, uno strato nascosto (o convoluzionale) e uno strato d'uscita. I neuroni dello strato intermedio non rappresentano né l'input né l'output, ma costituiscono una sorta di generalizzazione dell'input, assumendo in parte il significato di entrambi. Ogni volta che viene attivato questo tipo di contenuto, il neurone di output viene inibito dal neurone dello strato intermedio, impedendone l'attivazione. A questa categoria appartengono le reti convoluzionali, denominate CNN, spesso utilizzate per il riconoscimento di immagini e la classificazione di oggetti.

Seguendo il modello biologico, è stato introdotta una terza generazione di reti neurali che comunicano direttamente attraverso una sequenza di spike. Queste reti, dette Spiking Neural Networks (SNN), impiegano meccanismi di codifica a impulsi che permettono di integrare informazioni spazio-temporali che andrebbero altrimenti perse. Nel modello SNN il neurone artificiale è implementato come una macchina a stati finiti, dove le transizioni di stato dipendono da una variabile che rappresenta il potenziale di membrana della cellula.

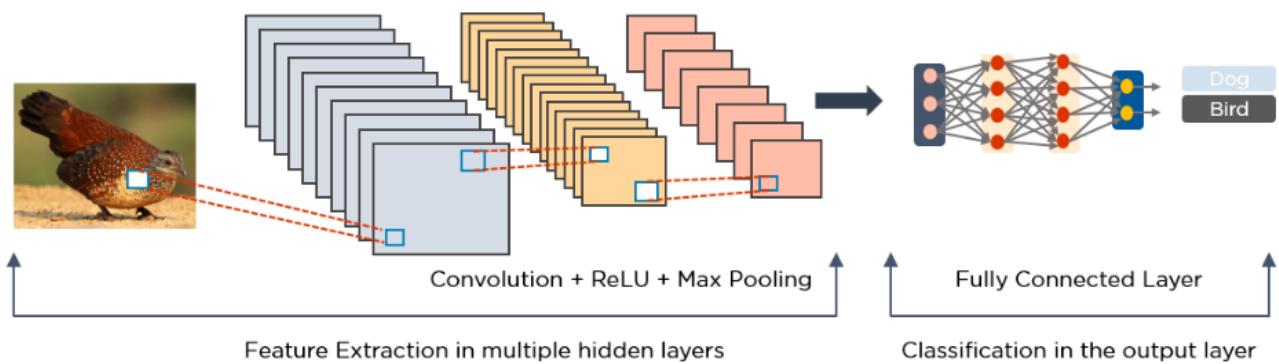
Le reti possono essere di due tipologie:

- Reti Feed-Forward: le informazioni viaggiano da reti in input, attraverso eventuali nodi nascosti fino ai nodi di output, senza mai tornare indietro. E' il modello più semplice, poiché non prevede cicli.
- Reti Ricorrenti: trasmettono le informazioni attraverso cicli diretti. Queste reti sono utilizzate per elaborare sequenze arbitrarie di input sfruttando una memoria interna.

Una volta stabilite tutte le caratteristiche di una rete neurale occorre determinare i pesi delle connessioni avendo a disposizione il training set. Questa operazione prende il nome di addestramento della rete neurale.

### 1.3 Reti CNN

Nel mio progetto viene utilizzato un modello di rete CNN per l'elaborazione delle immagini ricavate durante la simulazione. Le reti CNN si ispirano ai processi biologici della corteccia cerebrale, coinvolte nel riconoscimento dell'immagine. Questo modello di rete si articola su più livelli, in cui la complessità e la capacità di apprendimento aumenta progressivamente. L'obiettivo finale della rete è il riconoscimento dell'immagine.



**Fig 1.3:** Schema di rete neurale convoluzionale. [4]

I livelli della rete sono:

- livello computazionale: rappresenta il livello centrale della rete e permette di ricavarne le dimensioni. In questo livello è presente il kernel, un rilevatore capace di riconoscere curve e particolari caratteristiche dell'immagine attraverso un processo denominato convoluzione. Durante la convoluzione, il kernel analizza l'immagine esaminando una regione alla volta e restituisce un valore numerico in output per ciascuna di esse. Al termine del processo, il kernel ha percorso l'intera immagine. Possono presentarsi più strati di convoluzione, detti strati convoluzionali aggiuntivi, che consentono di ripetere l'operazione di convoluzione più volte per estrarre nuovi parametri.
- livello di pooling: esegue processi di riduzione della dimensionalità e dei parametri in input. Questo livello aumenta la dimensione del campo recettivo per gli strati successivi e agisce come un filtro restituendo l'immagine grezza.
- Il livello ReLU (Rectified Linear Unit) ha l'obiettivo di eliminare i valori non significativi prodotti dai livelli precedenti, mantenendo solo quelli positivi. Viene solitamente applicato dopo i livelli convoluzionali per introdurre non linearità nel modello.

- livello completamente connesso: solitamente è l'ultimo livello della rete. Utilizza funzioni di attivazione, come la Softmax, che restituiscono il valore finale di output, spesso interpretato come una probabilità associata a ciascuna classe.

## 1.4 Reti SNN

Le reti SNN rappresentano le reti di ultima generazione. Questo modello, ispirato ai processi biologici, è in grado di elaborare più impulsi in input per generare un unico valore in output. Nel mio lavoro di tesi, questo tipo di rete viene utilizzato per analizzare i dati registrati dai sensori di movimento durante la simulazione. L'attività neuronale in questo modello è rappresentata dalla frequenza di attivazione del neurone. Per realizzare una rete neurale SNN è necessario definire la struttura generale della rete, il modello del neurone e il modello di sinapsi. Per poter rappresentare i neuroni ai vari livelli esistono più modelli che, attraverso l'utilizzo di equazioni differenziali, permettono di simulare la struttura e il comportamento dei neuroni biologici.

I modelli di neuroni più utilizzati nelle SNN sono:

- Il modello di Hodgkin-Huxley descrive la dinamica del neurone attraverso un sistema di quattro equazioni differenziali. La prima equazione rappresenta la variazione del potenziale di membrana in funzione della permeabilità, mentre le altre tre modellano il comportamento dei canali ionici (sodio, potassio e perdite). Il potenziale d'azione viene generato quando il potenziale di membrana supera una soglia critica. Nonostante la sua accuratezza nel rappresentare i meccanismi elettrofisiologici, il modello presenta alcune limitazioni. Dal punto di vista biologico, non considera adeguatamente il periodo di refrattarietà del neurone. Inoltre, sul piano computazionale, risulta piuttosto oneroso in termini di risorse, rendendone complessa l'applicazione su larga scala o in simulazioni estese.
- Il modello Integrate and Fire: questo modello considera il timing dello spike, ovvero l'istante di tempo esatto in cui il neurone emette un potenziale d'azione. È il tipo di neurone maggiormente utilizzato nel contesto delle SNN. I segnali d'ingresso generano una corrente postsinaptica, che può essere rappresentata dalla seguente equazione:

$$I_{\text{syn}}(t) = g_{(\text{syn}(t))} (V_m - E_{\text{syn}})$$

$g_{\text{syn}}(t)$ : conduttanza sinaptica variabile nel tempo

$V_m$ : potenziale di membrana del neurone

$E_{\{\text{syn}\}}$ : potenziale di inversione della sinapsi

Questa corrente carica il neurone aumentano il potenziale di membrana, il cui valore può essere espresso da:

$$Cm * dVm/dt = -(Vm - V_{\text{rest}})/Rm + I(t)$$

$C_m$ : capacità della membrana

$V_m$ : potenziale di membrana

$V_{rest}$ : potenziale di riposo

$R_m$ : resistenza della membrana

Quando il neurone raggiunge una determinata soglia viene generato il potenziale d'azione, seguito dal periodo refrattario.

- Modello basato sugli spike: prende in considerazione la tempistica dei singoli spike, ossia l'ordine in cui i spike si verificano. Nel collegamento tra due neuroni possono verificarsi due situazioni differenti e opposte tra loro: la prima, quando uno spike presinaptico precede uno spike postsinaptico, provoca un potenziamento del segnale, aumentando la forza sinaptica; la seconda, quando un spike postsinaptico precede uno presinaptico, provoca una depressione, indebolendo il segnale. Questo fenomeno è noto come Spike-Timing Dependent Plasticity STDP ed è espresso attraverso il seguente modello matematico:

$$\Delta w = A^+ \cdot \exp(-\Delta t / \tau^+) \text{ se } \Delta t > 0$$

$$\Delta w = -A^- \cdot \exp(\Delta t / \tau^-) \text{ se } \Delta t < 0$$

$A^+$  e  $-A^-$  sono il massimo valore di potenziamento e depressione

$\tau^+$  e  $\tau^-$  sono le costanti di tempo per il decadimento

$\Delta t / \tau$  è la differenza temporale tra spike post- e pre-sinaptici

Esistono vari vantaggi nell'utilizzo delle SNN:

- Capacità di integrare informazioni spazio-temporali;
- Metodo di adattamento più semplice rispetto ai modelli precedenti;
- Maggiori plausibilità biologica;
- Velocità ed efficienza energetica;
- Miglioramento nelle capacità computazionali e di elaborazione delle informazioni.

## Capitolo 2: Dai concetti teorici alla progettazione

Negli ultimi anni, l'applicazione delle tecniche di Machine learning e, in particolare, del Deep learning, ha rivoluzionato il campo della robotica. Questo progetto di tesi si propone di studiare e simulare il comportamento motorio di un robot mediante l'utilizzo di reti neurali artificiali. Si parte dalla definizione degli obiettivi, per poi proseguire con una fase preliminare di sperimentazione, durante la quale sono sviluppate due reti neurali distinte: la prima, addestrata su un dataset dedicato al riconoscimento delle diverse tipologie di superficie, e la seconda, basata su un dataset specifico per l'individuazione di ostacoli lungo il percorso.

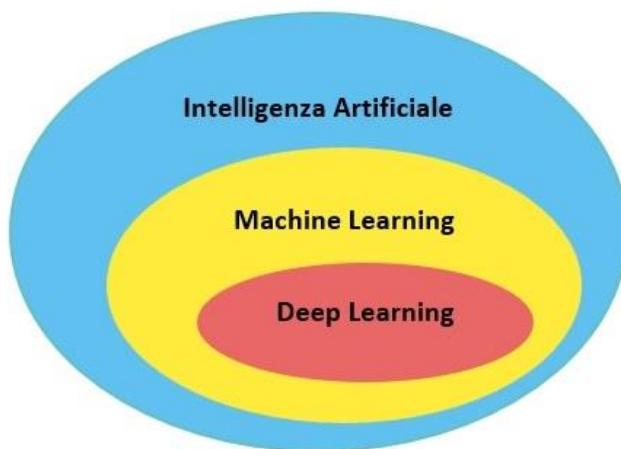
### 2.1 Stato dell'arte

L'impiego di algoritmi classici per l'analisi di segnali e immagini risale agli anni '80, periodo in cui furono sviluppati i primi modelli in grado di evitare ostacoli e riconoscere elementi chiave dell'ambiente, ad esempio i segnali stradali. Tuttavia, questi approcci basati su algoritmi simbolici e visione computazionale classica hanno mostrato limiti significativi, soprattutto in ambienti complessi e non strutturati.

I principali limiti riscontrati includono:

- Scarsa risoluzione delle telecamere utilizzate;
- Approccio deterministico e rigido;
- Algoritmi basati su feature semplici;
- Limitata capacità di individuare ed evitare ostacoli.

Per superare tali limiti, si è resa necessaria l'introduzione di tecniche più evolute, in particolare di quelle basate sul Machine Learning.



**Fig. 2.1:** Relazione tra Machine Learning, Deep Learning e Intelligenza artificiale [5]

Nell'immagine viene illustrato il legame tra Intelligenza Artificiale, Machine Learning e Deep Learning, dove ciascuno rappresenta un sottoinsieme dell'altro. Il Machine Learning (ML), ramo dell'intelligenza artificiale, consente ai sistemi di analizzare insiemi di dati e di apprendere schemi e relazioni utili per prendere decisioni in modo autonomo e migliorare con l'esperienza. I dati sono

l'elemento centrale del Machine Learning. Questi dati sono contenuti all'interno di dataset, tabelle costituite da feature e target. Le feature sono valori in input presentati in forma di vettori e possono contenere valori numerici o categorici. I target sono i valori che il modello deve imparare a predire. L'abilità dell'algoritmo viene valutata in termini di performance. Uno dei problemi principali del Machine Learning è la classificazione, il cui scopo è associare ogni dato in input al corrispondente valore in output. Per valutare i processi di classificazione si utilizza come metrica l'accuratezza, che rappresenta la percentuale di valori correttamente predetti. Le relazioni tra i dati possono essere ricavate attraverso varie tecniche di apprendimento che includono:

- Apprendimento supervisionato: il modello apprende da dati etichettati;
- Apprendimento non supervisionato: il modello identifica autonomamente schemi nei dati;
- Apprendimento per rinforzo: il modello migliora attraverso un sistema di ricompense e penalità.

All'interno del Machine Learning si collocano le architetture di Deep Learning, che sfruttano reti neurali composte da molti strati (layer), capaci di apprendere rappresentazioni complesse dei dati. Il Deep Learning comprende un insieme di tecniche di apprendimento dei dati basati sulle neuroscienze. L'anno dello sviluppo del Deep Learning è sicuramente il 2012, anno in cui furono presentati gli studi del professore Geoffrey Hinton dell'Università di Toronto. Questi studi erano basati sulla capacità di riconoscimento visivo dell'uomo e dell'animale. Questa grande scoperta ha posto le basi per gli studi e le scoperte successive, tra cui quella condotta da Microsoft, che ha proposto un modello di Deep Learning con 152 livelli di astrazione. Il Deep Learning utilizza le reti neurali artificiali, già introdotte nel capitolo precedente, per elaborare una serie di dati. L'idea di utilizzare reti neurali per il controllo del movimento di un robot risale alla realizzazione di ALVINN, una rete neurale progettata per controllare un veicolo. Questa rete è stata addestrata con l'utilizzo di un generatore di strade simulate, per guidare autonomamente un veicolo terrestre. Numerosi sono gli studi successivi condotti per applicare il Deep Learning al riconoscimento degli ostacoli, alla pianificazione dei percorsi e al controllo dei movimenti robotici. Oggi, grazie alla potenza di calcolo dei moderni processori e alle grandi quantità di dati disponibili, le reti neurali profonde vengono utilizzate con successo in robotica mobile, veicoli autonomi, droni e in molte altre applicazioni. Recentemente si stanno concentrando sulla sensor fusion, ovvero sulla capacità di combinare dati provenienti da diversi sensori. L'uso combinato di più sensori diversi permette di restituire valori più accurati e di generare mappe più dettagliate dell'ambiente circostante.

Esistono vari tipi di sensori:

-Sensori Lidar: generano nuvole di punti 3D misurando la distanza dagli oggetti attraverso impulsi laser. Essi sono molto precisi ma molto costosi.

-Sensori radar: utilizzano onde radio per rilevare gli ostacoli.

-Sensori ultrasonici: sono spesso impiegati a breve distanza, ad esempio nei robot o nei veicoli per manovre di parcheggio, grazie alla loro capacità di rilevare ostacoli vicini con costi contenuti

Per la percezione dell'ambiente circostante vengono utilizzate anche telecamere, che permettono di catturare immagini dell'ambiente utili a rilevare ostacoli o segnali presentati lungo il percorso. Uno dei primi modelli di robot dotati di una telecamera integrata risale agli anni '70, con il robot Shakey, il primo robot mobile dotato di una telecamera per la mappatura dell'ambiente circostante. Oltre la

telecamera sono presenti anche dei sensori sonori. Questo robot attualmente esposto al Computer History Museum di Mountain View in California ha apportato numerose novità sia in termini di struttura sia in termini di capacità. È infatti in grado di scomporre i dati e prendere decisioni autonome.

## 2.1 Obiettivi

Negli ultimi decenni l'utilizzo dei robot in ambito industriale è diventato sempre più diffuso. In questo contesto, è fondamentale sviluppare sistemi in grado di fornire ai robot la capacità di generare mappe dettagliate in tempo reale, garantendo così un elevato livello di sicurezza nell'ambiente operativo.

Questa abilità di mappatura dinamica consente al robot di adattare i propri movimenti, riconoscendo ed evitando gli ostacoli lungo il percorso. Il controllo del movimento è reso possibile grazie all'impiego di sensori e telecamere integrate, i cui dati vengono elaborati mediante reti neurali artificiali.

Negli ultimi anni, numerosi studi si sono concentrati sullo sviluppo di reti neurali sempre più avanzate, ispirate al funzionamento del cervello umano, sia per quanto riguarda l'apprendimento sia per l'adattabilità. L'obiettivo principale del nostro progetto è utilizzare queste reti neurali per il decision making, ovvero per permettere al robot di prendere decisioni autonome in base agli input ricevuti dall'ambiente circostante. Tale capacità si fonda su reti opportunamente addestrate e valutate attraverso metriche e algoritmi di Deep Learning.

Uno degli obiettivi centrali del progetto è il rilevamento degli ostacoli, l'analisi della traiettoria e la definizione della distanza minima di sicurezza.

Per queste analisi si fa uso di algoritmi di classificazione, una tecnica tipica dell'apprendimento supervisionato. In particolare, il sistema deve identificare se lungo il percorso del robot è presente o meno un ostacolo, classificando ogni input secondo due categorie:

- Anomalia (0): presenza di un ostacolo;
- Normale (1): assenza di ostacoli.

Durante la fase di addestramento, la rete neurale riesce a distinguere le due situazioni sulla base dei dati raccolti dai sensori.

Per valutare le prestazioni del modello di classificazione utilizziamo la metrica dell'accuratezza, definita dalla formula:

$$\text{Accuratezza} = \frac{TP + TN}{TP + TN + FP + FN}$$

dove:

- TP (True Positives): ostacoli correttamente rilevati;
- TN (True Negatives): assenza di ostacoli correttamente identificata;
- FP (False Positives): falsi allarmi (ostacolo rilevato ma non presente);

- FN (False Negatives): ostacoli non rilevati.

Per analizzare la traiettoria si va a definire graficamente un diagramma che riporta con un colore diverso i valori considerati anomali, in cui il robot si avvicina in maniera esagerata all'ostacolo. In questo contesto si colloca lo studio del valore della threshold, ovvero la soglia critica di distanza tra il robot e un potenziale ostacolo. Quando questa soglia viene superata, si genera una condizione di *anomalia* che può compromettere il funzionamento o la sicurezza del robot. La threshold rappresenta quindi il limite oltre il quale si attivano azioni correttive o di emergenza.

I risultati ottenuti da questi studi sono fondamentali per poter garantire un elevato grado di sicurezza nell'utilizzo di un robot. La programmazione del movimento di un robot è infatti un lavoro di grande responsabilità. Il robot deve infatti garantire:

- precisione e affidabilità: la corretta programmazione delle reti deve far in modo che il robot sia in grado di riconoscere e prendere decisioni in maniera appropriata.
- sicurezza: il robot non deve compromettere né la sicurezza degli operatori con cui lavora, né danneggiare l'ambiente circostante.
- accuratezza: la programmazione deve essere sufficientemente accurata affinché il robot sia in grado di saper evitare e riconoscere ostacoli.

L'utilizzo delle reti neurali in questo contesto ha permesso al robot, come dimostrano i risultati ottenuti dal nostro studio, di riconoscere l'ambiente circostante e di prendere decisioni autonome in base ai dati. Tuttavia, se da un lato l'impiego delle reti neurali rappresenta un grande progresso in ambito robotico, dall'altro è fondamentale fornire i giusti dati e addestrare correttamente le reti per evitare eventuali danni derivanti dal loro utilizzo.

## 2.2 Studi preliminari

Cominciamo le prime sperimentazioni realizzando una rete neurale SNN connessa a un dataset denominato “Robot\_data” [6], contenente informazioni sul movimento di un robot nello spazio. Utilizzando un semplice algoritmo di apprendimento supervisionato di base, la rete è in grado di eseguire una classificazione, identificando il tipo di superficie su cui il robot si sta muovendo. Il modello sarà in grado di associare ad ogni input il valore target corrispondente.

Il dataset contiene due file differenti.

Il primo, denominato “data”, presenta 276 feature, contiene informazioni sulla posizione e sull'orientamento assunto dal robot, oltre ai dati registrati dai sensori.

orientation_X_mean	orientation_X_median	orientation_X_max	orientation_X_min	orientation_X_std	orientation_X_range
-0.75866578125	-0.75853	-0.75822	-0.75953	0.0003626990018525	0.00131
-0.95860640625	-0.958595	-0.95837	-0.95896	0.0001513494906073	0.0005899999999999
-0.51205734375	-0.512035	-0.50944	-0.51434	0.0013774714278157	0.0049
-0.9391690625	-0.93917	-0.93884	-0.93968	0.0002273241727743	0.0008399999999999
-0.891300546875	-0.89094	-0.88673	-0.89689	0.0029553171908565	0.0101599999999999

**Fig. 2.1:** Estratto del dataset Robot\_data: visualizzazione delle prime 5 righe e delle prime 6 colonne.

Il secondo, chiamato “target”, presenta tre colonne:

- series\_id: identifica una sequenza specifica da classificare.
- group\_id: raggruppa diverse series\_id in uno stesso soggetto, sessione o condizione comune.
- surface: contiene i target. Sono presenti dieci classi di output per la classificazione. Ogni classe, individuata da un numero da 0 a 9, rappresenta una superficie diversa.

Come architettura, realizziamo un modello di rete neurale spiking (SNN) a due layer, composto da due strati di neuroni. Ogni neurone accumula il segnale in ingresso e, quando supera una certa soglia, genera uno spike. Il tipo di neurone utilizzato è il modello Leaky Integrate-and-Fire (LIF), capace di integrare il segnale nel tempo, perdendo però parte della carica accumulata a causa del fenomeno di "leakage". La rete presenta una dinamica temporale su 25 step, che permette di simulare l'evoluzione del segnale e dell'attività dei neuroni nel tempo, migliorando la capacità di elaborare informazioni temporali e sequenziali.

Per addestrare la rete utilizziamo il metodo del Backpropagation Through Time (BPTT), una tecnica che permette ai neuroni di apprendere attraverso diverse fasi:

- Elaborazione dell'input da parte della rete.
- Calcolo dell'output corrispondente.
- Calcolo dell'errore, definito come la differenza tra l'output prodotto dalla rete e il valore target.
- Retro propagazione dell'errore per aggiornare i pesi della rete

Per il training utilizziamo un numero di epoche pari a 20, ossia l'intero dataset viene passato 20 volte alla rete. Il dataset, a ogni epoca, viene suddiviso in mini-batch, ed ogni batch viene elaborato separatamente.

Per ciascuna epoca calcoliamo l'accuratezza tramite la funzione print\_batch\_accuracy, che sfrutta il conteggio degli spike per determinare il numero di previsioni corrette. Poi calcoliamo la perdita utilizzando nn.CrossEntropyLoss, la quale si basa sul potenziale di membrana per valutare le predizioni errate del modello. Come ottimizzatore, impieghiamo Adam.

Nel corso delle iterazioni, osserviamo che la perdita diminuisce progressivamente, mentre l'accuratezza tende ad aumentare. I valori ottenuti al termine dell'ultima epoca sono:

- Training loss: 77.95
- Training set accuracy: 6.90%.

A questo punto, ci concentriamo sul processo di ottimizzazione della rete. Per migliorare le prestazioni, applichiamo delle modifiche che includono:

1. Normalizzazione: durante la fase del pre-processing del dataset, normalizziamo i dati mediante MinMaxScaler. Questo processo permette di trasformare i dati adottando un range comune che va da 0 a 1 per tutti i valori, rendendo il modello più preciso.
2. Aumento del numero delle epoche di training a 30: ciò consente di passare più volte l'intero dataset all'interno della rete neurale, migliorando le capacità di apprendimento della rete.

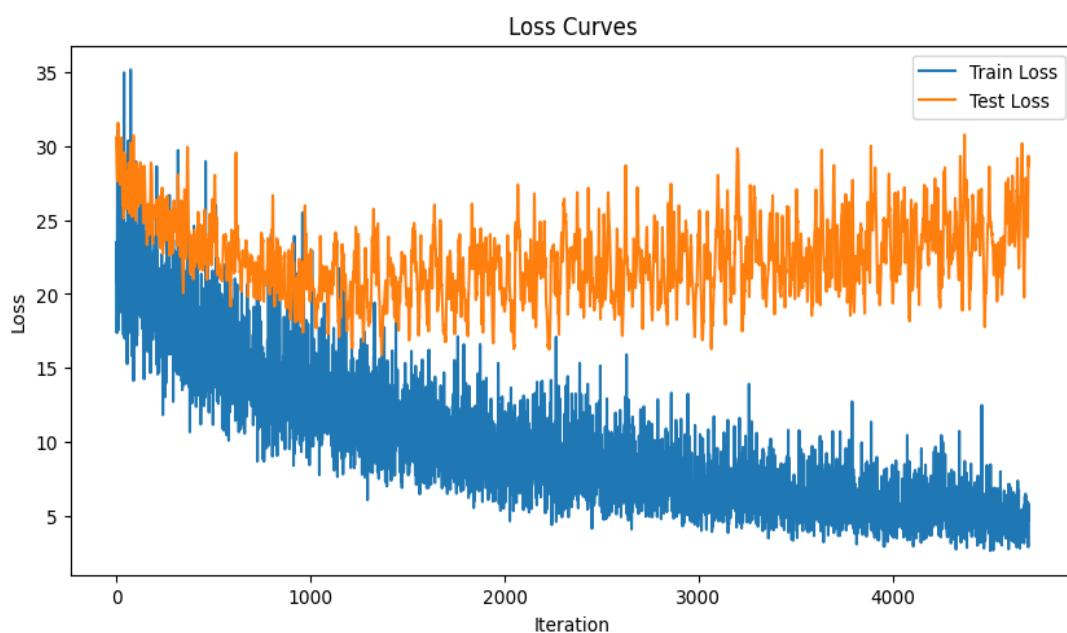
- Aumento del numero di layer, il quale permette di incrementare il numero di neuroni nascosti (hidden neurons). Nel caso in cui non siano presenti hidden neurons il contributo dell'errore nel processo di Backpropagation Through Time di un singolo neurone dipende solo dalla differenza tra target e output. Nel caso di hidden neurons, invece, dipende invece dalla somma di tutti gli errori del layer successivo a cui il neurone è collegato. Questo permette alla rete di generalizzare meglio su dati mai visti e di migliorare la performance. Aggiungiamo alla nostra rete 5 hidden layer.

Dopo aver ottimizzato la rete, ricalcoliamo i valori di accuratezza nelle prime fasi dell'addestramento, osserviamo infatti un miglioramento delle prestazioni, sia in termini di loss che di accuratezza:

- Training loss: 57,416
- Training set accuracy: 10,94%

Per analizzare l'andamento dell'ottimizzazione, realizziamo un grafico della funzione di perdita durante l'addestramento, ottenendo due curve distintive:

- Curva blu (Training Loss): rappresenta l'andamento della perdita sul set di addestramento. La funzione di perdita utilizzata è la Cross-Entropy Loss. Essa misura la differenza tra la distribuzione di probabilità prevista dal modello e la distribuzione reale. Nel grafico osserviamo una progressiva diminuzione della perdita nel tempo, segnale che il modello sta migliorando le proprie previsioni man mano che apprende.
- Curva arancione (Test Loss): rappresenta la perdita calcolata sul set di test, ovvero su dati mai visti dalla rete. Anche questa curva tende a decrescere nel tempo, indicando che il modello è in grado di generalizzare efficacemente. Tuttavia, i valori del test loss risultano più alti rispetto a quelli della training loss, come atteso, poiché il modello non è stato addestrato direttamente su questi dati.



**Fig. 2.2:** Andamento della funzione di loss nel corso della simulazione.

Il modello ha raggiunto un'accuratezza finale dell'82,39%, riuscendo a classificare correttamente 580 campioni su un totale di 704 nel set di test. Questo risultato indica che la rete è in grado di riconoscere e classificare correttamente la maggior parte delle immagini.

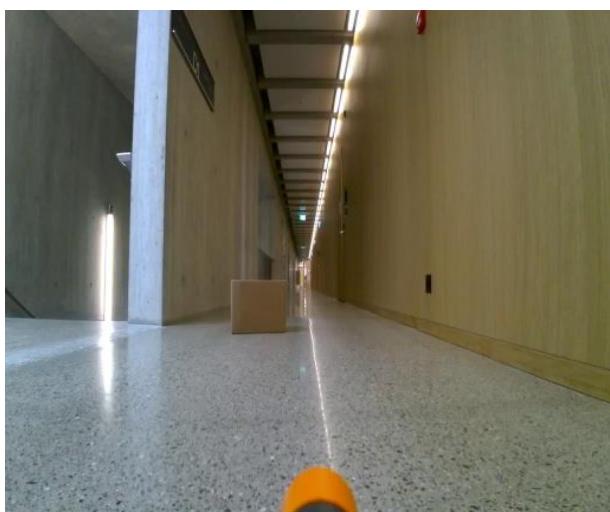
Per la seconda sperimentazione invece analizziamo un modello di rete neurale spiking convoluzionale (CSNN). Per l'addestramento utilizziamo il dataset “Subset”, che contiene 17.775 immagini di un robot che si muove lungo un corridoio. Lo scopo è quello di classificare le immagini in due categorie distinte, basandosi sulla presenza o assenza di uno scatolone nel corridoio. A ogni immagine viene assegnata un'etichetta:

- 0 → “Normal”: immagini in cui lo scatolone non è presente;



**Fig. 2.3:** Esempio di immagine appartenente al subset del dataset, etichettata

- 1 → “Anormal”: immagini in cui lo scatolone è presente.



**Fig. 2.4:** Esempio di immagine appartenente al subset del dataset, etichettata come “Anormal”

L'architettura di rete convoluzionale da utilizzare è: 12C5-MP2-64C5-MP2-1024FC10

- 12C5 è un  $5 \times 5$  kernel convoluzionali con 12 filtri;
- MP2 è un  $2 \times 2$  funzione di max-pooling;
- 1024FC10 è uno strato completamente connesso.

Anche in questo caso utilizziamo l'algoritmo di backpropagation per addestrare il modello. Nel contesto dell'elaborazione di immagini, non ci riferiamo più a connessioni dense tra tutti i neuroni, come nelle reti completamente connesse, bensì l'input, ovvero l'immagine, viene elaborato mediante kernel. Tali filtri hanno il compito di estrarre caratteristiche salienti dall'immagine, generando le feature maps, ovvero tensori che rappresentano specifiche informazioni visive apprese dal modello.

Successivamente, queste feature maps vengono sottoposte a un'operazione detta pooling, per poi essere trasferiti ai fully connected layers che costituiscono la parte finale della rete neurale, strutturata come un Multi-Layer Perceptron (MLP). Tuttavia, mentre le feature maps hanno una struttura tridimensionale, i fully connected layers operano su vettori monodimensionali.

Per consentire questo passaggio, è necessario eseguire un'operazione di flattening, che converte il tensore tridimensionale prodotto dagli ultimi layers convoluzionali in un vettore monodimensionale. In questo modo, ciascun neurone del layer fully connected riceve in ingresso tutti i valori precedentemente calcolati.

Nel nostro caso specifico, il numero di neuroni presenti nell'ultimo layer fully connected corrisponde al numero di classi da riconoscere nel task di classificazione.

L'addestramento tramite backpropagation inizia con la forward pass, ossia la propagazione dell'input attraverso tutti i layers della rete fino all'output. Durante questa fase, la rete calcola le proprie predizioni.

Dopo la forward pass, la nostra rete neurale elabora ciascuna immagine attraverso più passi temporali, durante i quali i neuroni nei layers convoluzionali e fully connected accumulano potenziali di membrana e generano spike discreti (0 o 1). Al termine della simulazione, l'output dell'ultimo layer fully connected è rappresentato da un vettore di conteggi di spike, in cui ogni elemento indica quante volte il neurone corrispondente a una certa classe ha emesso spike. Questi conteggi costituiscono le predizioni della rete: infatti, maggiore è il numero di spike emessi da un neurone, maggiore è la confidenza della rete che l'immagine appartenga a quella classe. La classe predetta viene quindi determinata scegliendo l'indice del neurone con il conteggio di spike più alto. Gli stessi conteggi vengono inoltre utilizzati come input per calcolare la loss tramite la funzione CrossEntropyLoss.

Viene usata la CrossEntropyLoss, poiché si tratta di un problema di classificazione a più classi. Un esempio di valore iniziale della loss calcolata è pari a 2.303, corrispondente al logaritmo naturale del numero di classi, valore tipico quando la rete parte da pesi casuali e le predizioni sono ancora casuali.

Dopo il calcolo della loss, si procede con la backward pass, durante la quale l'errore viene propagato all'indietro nella rete. Tuttavia, in queste reti sorge una difficoltà: la funzione spike (che determina la generazione o meno di uno spike) non è derivabile, poiché è una funzione a soglia dura (step function). Per superare questo ostacolo, viene impiegato il metodo del Surrogate Gradient. In pratica, durante la backward pass, si sostituisce la derivata non esistente dello spike function con una funzione derivabile, utilizziamo la derivata di una funzione sigmoide, consentendo di calcolare i gradienti tramite il metodo della discesa del gradiente.

Infine, scegliamo l'ottimizzatore Adam per aggiornare i pesi della rete, esattamente come avviene nelle reti neurali tradizionali. L'obiettivo è ridurre progressivamente il valore della loss e migliorare l'accuratezza del modello nella classificazione.

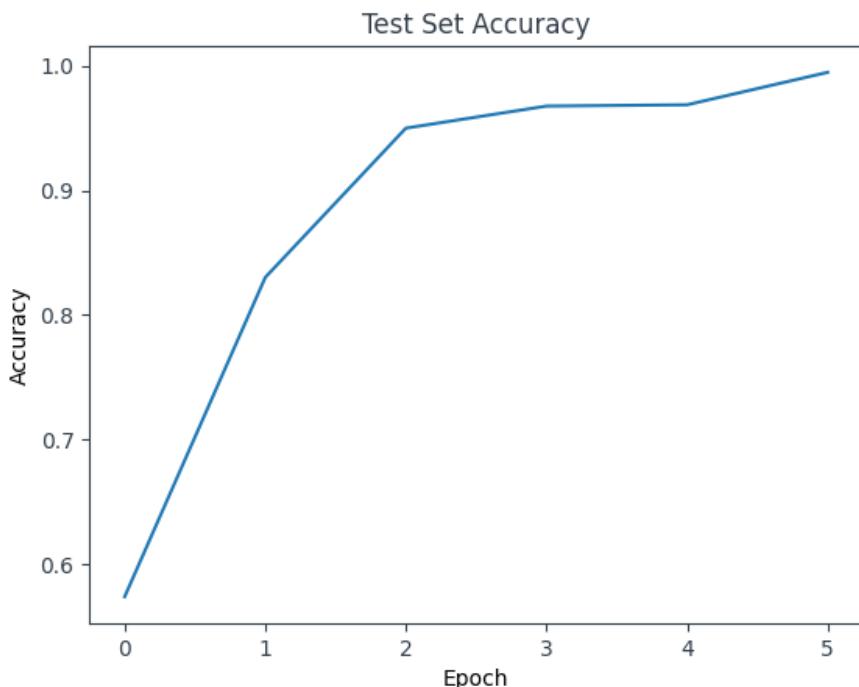
Valutiamo inizialmente l'accuratezza su un singolo batch, ottenendo un valore pari al 59,375%. Estendendo la valutazione all'intero DataLoader, otteniamo un'accuratezza complessiva del 60,94%.

Per monitorare il comportamento della rete durante l'addestramento, implementiamo un ciclo di validazione: ogni 50 batch, il modello viene valutato su un set di validazione separato, al fine di misurare metriche come l'accuratezza e verificare la corretta convergenza del modello, senza però influenzare il processo di training stesso.

Ottengo i seguenti risultati:

- Iteration 0, Val Acc: 57.40%
- Iteration 50, Val Acc: 88.02%
- Iteration 100, Val Acc: 92.08%

Realizziamo un grafico per monitorare l'accuratezza del modello sul test set. Il grafico presenta nell'asse delle x le epoche di allenamento, mentre in quella delle y il valore dell'accuratezza. Il grafico permette di visualizzare il valore dell'accuratezza, che tende ad aumentare progressivamente nel tempo.



**Fig. 2.5:** Grafico che mostra l'andamento dell'accuratezza sul set di validazione per ogni epoca.

**Tabella riassuntiva**

	Primo modello	Secondo modello
<b>Dataset</b>	Robot_Data	Subset
<b>Output Classes</b>	10	2
<b>Loss Function</b>	CrossEntropyLoss	CrossEntropyLoss
<b>Optimizer</b>	Adam	Adam
<b>Batch Size</b>	128	128
<b>Training Loss</b>	5.003	2.303
<b>Training Accuracy</b>	82.39%	92.08%

**Fig. 2.6:** Tabella contenente il confronto dei parametri tra i due modelli

Questa fase iniziale di sperimentazione è fondamentale per analizzare il problema e considerare separatamente le diverse problematiche legate al processo di classificazione. Queste reti rappresentano un prototipo di studio per testare ipotesi, architetture e fasi di apprendimento. Tale fase preliminare, oltre a facilitare l'individuazione di eventuali criticità, consente di ottimizzare le soluzioni adottate e di riutilizzare conoscenze, configurazioni e moduli funzionali nella rete finale. In questo modo, si pongono basi per lo sviluppo di un modello completo, più efficiente e robusto.

## Capitolo 3: Approccio metodologico: architettura, dataset e simulazioni

In questo capitolo viene presentato un nuovo caso di studio nel quale un robot su due ruote (Pioneer P3AT) naviga in un ambiente simulato per raggiungere una posizione target, evitando gli ostacoli presenti. Il robot è dotato di un sensore di visione e di 16 sensori di distanza di tipo sonor collocati sulla superficie del robot. Questi sensori permettono al robot di acquisire informazioni dall'ambiente circostante. Il controllo della navigazione è basato su un algoritmo di tipo Potential Field [7]. Questo algoritmo permette di assegnare dei valori di potenziale a ciascun punto nell'ambiente di navigazione del robot. In questo modo il robot può muoversi verso la meta, cercando di evitare gli ostacoli.

### 3.1 Dataset utilizzato e ambiente di simulazione

Per raccogliere entrambe le tipologie di dati, è stata utilizzata una simulazione condotta tramite CoppeliaSim, una piattaforma di simulazione robotica altamente versatile. L'utilizzo di un ambiente simulato ha consentito sia la riproduzione di scenari di navigazione realistici per il robot, sia la registrazione controllata dalle immagini acquisite dai sensori visivi e dai dati numerici relativi al movimento.

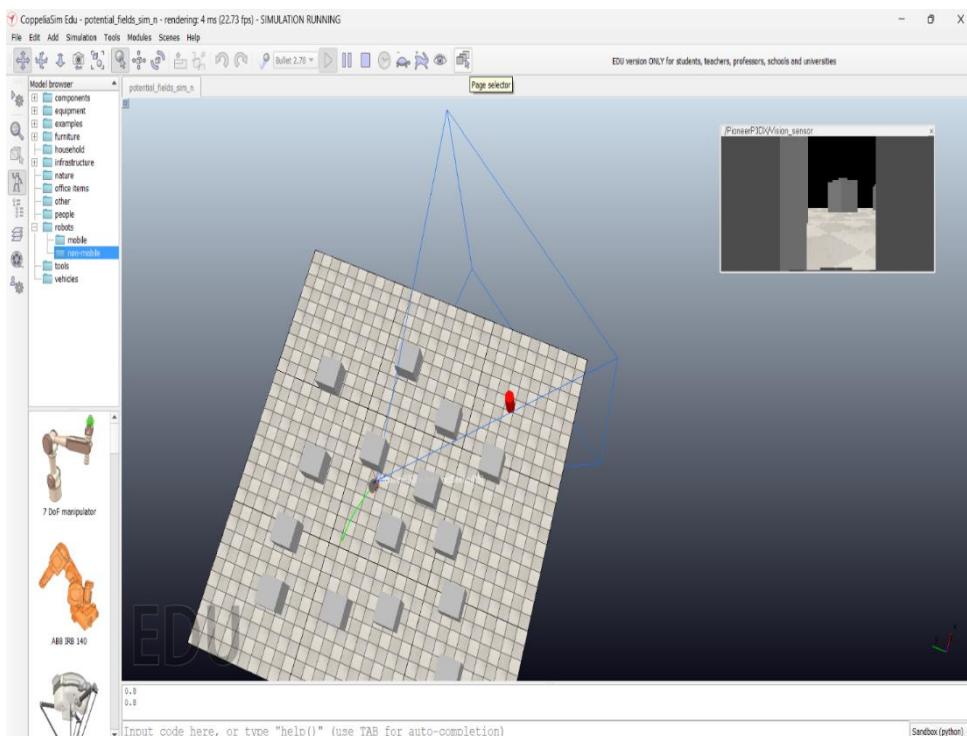


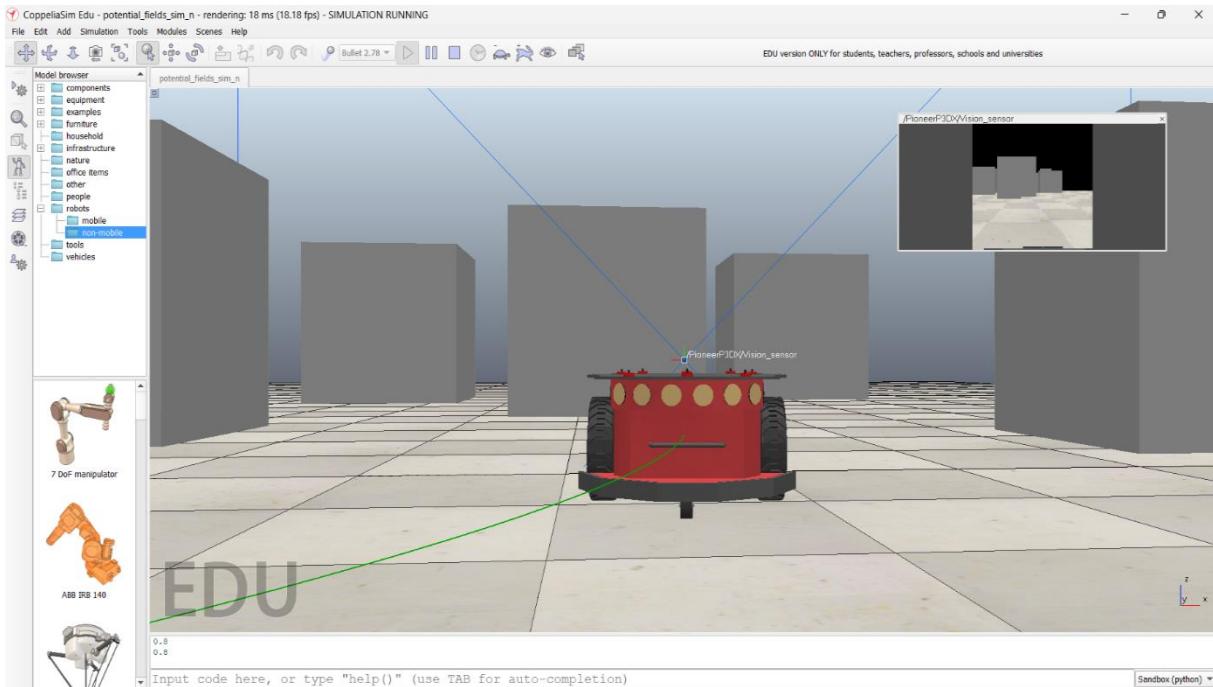
Fig. 3.1– Vista dell'ambiente virtuale utilizzato per la simulazione all'interno di CoppeliaSim

Impostiamo la nostra scena di simulazione con:

-Robot Pioner: un tipo di manipolatore mobile dotato di due ruote motrici indipendenti e due ruote per l'equilibrio. È equipaggiato con una telecamera visiva e sedici sensori per il movimento, come mostrato nella Figura 3.2, evidenziati dai cerchi gialli disposti a cerchio intorno al robot. Questi sensori sono utilizzati per evitare ostacoli e percepire l'ambiente circostante.

-14 ostacoli posizionati all'interno dell'ambiente simulato.

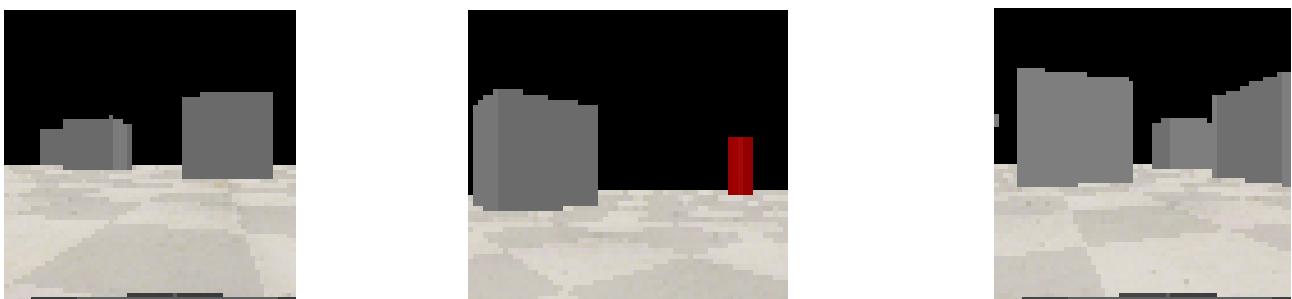
-1 goal, di colore rosso, rappresenta la posizione finale da raggiungere per l'end-effector.



**Fig. 3.2** – Foto ravvicinata del robot utilizzato durante la simulazione in CoppeliaSim.

In questo ambiente, vengono eseguite cinque simulazioni distinte, i cui risultati sono registrati in due dataset separati: uno contenente immagini e l’altro dati tabellari. Entrambi i dataset vengono successivamente utilizzati per addestrare la rete neurale.

Il primo dataset è composto da immagini che mostrano la percezione visiva dell’ambiente circostante. Tramite la telecamera presente sul robot vengono acquisite delle immagini dell’ambiente circostante. La simulazione carica i dati in una cartella denominata “camera frames” contenente quattro sotto cartelle.



**Fig. 3.3** Esempi di immagini registrate nel dataframe

Il secondo dataset, invece, è costituito da dati numerici relativi al movimento del robot Pioneer. Tali dati includono:

- la posizione lungo l’asse x, y, z del Piooner;
- le coordinate che rappresentano la posizione del goal;
- i valori registrati dai 16 sensori ultrasuoni.

run	time	pioneer_x	pioneer_y	pioneer_z	goal_x	goal_y	goal_z	sensor_0	sensor_1
1.0	0.0	-2.798277	-4.988876	0.486514	-1.475299	3.754067	0.5	-1.0	-1.0
1.0	0.9	-2.775448	-4.99003	0.138652	-1.475299	3.754067	0.5	-1.0	-1.0
1.0	1.11	-2.731863	-4.989362	0.138683	-1.475299	3.754067	0.5	-1.0	-1.0
1.0	1.29	-2.683708	-4.982145	0.138692	-1.475299	3.754067	0.5	-1.0	-1.0
1.0	1.48	-2.622885	-4.965482	0.138683	-1.475299	3.754067	0.5	-1.0	-1.0

**Fig. 3.4** – Estratto del dataset contenente le prime 10 colonne e 5 righe.

I due dataset vengono caricati nella rete neurale per l'addestramento, dove, i dati sono ordinati in base al valore del “ran”, prima colonna del dataset, per avere uno split stabile.

### 3.2 Architettura della rete

La prima rete utilizzata è una CNN che processa le immagini acquisite dal robot durante la navigazione. Questa rete prende in ingresso immagini a un solo canale e applica:

- una convoluzione con 12 filtri, che permette di ricavare le caratteristiche principali delle immagini grazie all'utilizzo di un kernel;
- una funzione di attivazione ReLU, utile per individuare anche eventuali relazioni non lineari;
- un'operazione di flattening, che trasforma l'immagine in un vettore unidimensionale (1D);
- un layer lineare, che riduce il vettore a 128 valori, producendo una rappresentazione compatta dell'immagine.

La seconda rete è una SNN che ho utilizzato per analizzare i parametri dei sensori. Questa rete ha un'architettura feedforward a due strati e ogni strato presenta:

- un layer lineare per trasformare l'input numerico;
- un neurone LIF.

La rete presenta:

- num\_inputs = 24      neuroni in input;
- num\_hidden = 1000      numero di neuroni nel layer nascosto;
- num\_outputs = 2      valori in uscita, che rappresentano due classi 1 (Classe normale), o 0 (Anomalia)

Di seguito il codice utilizzato per definire l'architettura delle due reti:

```

# Parametri SNN
spike_grad = surrogate.fast_sigmoid(slope=25)
beta = 0.5
num_steps = 50
# CNN
class ImageCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(1, 12, 5),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(12*24*24, 128)
        )

    def forward(self, x, y=None):
        return self.net(x)

num_inputs = 24
num_hidden = 1000
num_outputs = 2
#SNN
class SensorSNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, dropout_rate=0.3):
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.lif1 = snn.Leaky(beta=beta)
        self.fc2 = nn.Linear(hidden_size, output_size)
        self.lif2 = snn.Leaky(beta=beta)

    def forward(self, x):
        batch_size = x.size(0)
        mem1 = self.lif1.init_leaky().to(x.device)
        mem2 = self.lif2.init_leaky().to(x.device)
        spk2_rec = []

        for step in range(num_steps):
            cur1 = self.fc1(x)
            spk1, mem1 = self.lif1(cur1, mem1)
            cur2 = self.fc2(spk1)
            spk2, mem2 = self.lif2(cur2, mem2)
            spk2_rec.append(spk2)
        return torch.stack(spk2_rec, dim=0), mem2

```

**Algoritmo 3.1:** Codice contenete la definizione dei due modelli

Infine, uniamo le due reti in un unico modello: SimpleModel, un modello ibrido che combina immagini e dati tabellari.

```

class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()

        self.cnn = nn.Sequential(
            nn.Conv2d(1, 8, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(8, 16, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Flatten()
        )
        self.fc_img = nn.Linear(16*7*7, 64)
        self.fc_tabular = nn.Linear(23, 64)
        self.fc_final = nn.Linear(128, 1)

    def forward(self, img, tabular):
        img_feat = self.fc_img(self.cnn(img))
        tab_feat = self.fc_tabular(tabular)
        combined = torch.cat([img_feat, tab_feat], dim=1)
        out = self.fc_final(combined)
        return out

    def extract_features(self, img, tabular):
        with torch.no_grad():
            img_feat = self.fc_img(self.cnn(img))
            tab_feat = self.fc_tabular(tabular)
            combined = torch.cat([img_feat, tab_feat], dim=1)
        return combined

```

**Algoritmo 3.2:** Codice contenente l’architettura della rete neurale ibrida *SimpleModel*

### 3.3 Fasi di addestramento

Dopo aver normalizzato i dati e allineato le dimensioni dei dati tabellari e delle immagini in modo che corrispondano, a tutti gli esempi viene assegnato lo stesso target, ovvero il valore 1. Questa scelta è dettata dall’obiettivo di implementare una classificazione in cui il modello deve imparare a riconoscere una singola classe positiva (target 1). Successivamente, il dataset viene suddiviso in due insiemi: training set (80%) per l’addestramento e test set (20%) per la valutazione. Infine, creiamo i DataLoader, strumenti che in PyTorch permettono di gestire il caricamento dei dati suddividendoli in batch.

```

# ISTANZA MODELLO, LOSS, OPTIMIZER
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = SimpleModel().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters())

# TRAINING LOOP
num_epochs = 10

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

    for img_batch, tab_batch, target_batch in train_loader:
        img_batch = img_batch.to(device)
        tab_batch = tab_batch.to(device)
        target_batch = target_batch.to(device).float()

        optimizer.zero_grad()
        outputs = model(img_batch, tab_batch)

        loss = criterion(outputs.squeeze(), target_batch)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * img_batch.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss:.4f}")

```

**Algoritmo 3.3:** Codice per il training loop in PyTorch, con calcolo della loss e aggiornamento dei pesi per un modello misto su immagini e dati tabellari.

Il codice di addestramento implementa il ciclo di apprendimento, descritto in precedenza, di un modello di rete neurale eseguendo iterativamente un numero pari a 10 di epoche. In ogni epoca, il modello riceve in input batch di dati costituiti da immagini e da dati tabellari. Per ogni batch, il modello genera delle predizioni che vengono confrontate con i valori target tramite la funzione di perdita MSELoss, che misura l'errore quadratico medio. Successivamente, si calcola il gradiente dell'errore rispetto ai parametri del modello, e i pesi vengono aggiornati utilizzando l'ottimizzatore Adam. Durante l'epoca, si accumula la somma delle perdite dei vari batch, per poi calcolare la perdita media sull'intero set di training, consentendo di monitorare il progresso dell'apprendimento del modello. Questa procedura permette al modello di migliorare progressivamente la capacità di predire i target corretti a partire dagli input combinati di immagini e dati tabellari. Si ottengono i seguenti risultati:

- Epoch 1, Loss: 0.3874
- Epoch 2, Loss: 0.0979
- Epoch 3, Loss: 0.0706
- Epoch 4, Loss: 0.0210
- Epoch 5, Loss: 0.0260
- Epoch 6, Loss: 0.0127
- Epoch 7, Loss: 0.0119
- Epoch 8, Loss: 0.0101
- Epoch 9, Loss: 0.0074
- Epoch 10, Loss: 0.0072

Questi risultati mostrano l'andamento della funzione di perdita (loss) durante le 10 epoche di addestramento del modello. Si osserva una diminuzione significativa e costante della perdita, che passa da un valore iniziale di circa 0.39 alla prima epoca a circa 0.007 alla decima epoca. Questa tendenza indica che il modello sta migliorando progressivamente la sua capacità di predire correttamente i target, riducendo l'errore medio tra le sue previsioni e i valori reali. La rapida discesa della loss nelle prime epoche suggerisce che il modello è in grado di apprendere rapidamente le caratteristiche principali dai dati. Nelle epoche successive, la perdita tende a stabilizzarsi, indicando che l'ottimizzazione sta convergendo verso una soluzione stabile e sufficientemente accurata.

### 3.4 Validazione e test sulla rete

Il modello viene testato su un insieme di dati di test che rappresentano il 20% del totale e che non sono mai stati utilizzati durante la fase di addestramento. Questi dati consentono di valutare la capacità del modello di generalizzare a esempi nuovi e inediti. L'obiettivo del test è anche quello di individuare eventuali anomalie nel comportamento del modello. Come previsto, il modello classifica correttamente tutti gli esempi nella classe 1. Sono stati calcolati la funzione di perdita e l'accuratezza sul test set. I risultati ottenuti confermano l'efficacia del modello: la loss sul test è pari a 0.0685, mentre l'accuratezza raggiunge il 100%, indicando che il modello ha appreso in modo stabile e generalizza correttamente su dati mai visti prima. Successivamente, per testare ulteriormente la rete in condizioni diverse, si interviene sullo script di simulazione in CoppeliaSim modificando la soglia di avvicinamento all'ostacolo (threshold) portandola da un valore predefinito di 0.5 a un valore più basso pari a 0.3.

Questa modifica ha lo scopo di generare una nuova simulazione in cui:

- i punti con valori prossimi allo 0.3 vengono considerati potenzialmente anomali, in quanto vicino agli ostacoli;
- i punti lontani da 0.3 vengono considerati normali, e classificati nella classe 1.

Di conseguenza, il modello verrà testato su questi nuovi dati per valutare la sua capacità di riconoscere comportamenti anomali, classificando correttamente tali punti all'interno della classe 1. Questo approccio consente di simulare condizioni limite e di verificare la sensibilità del modello rispetto a variazioni nei parametri critici.

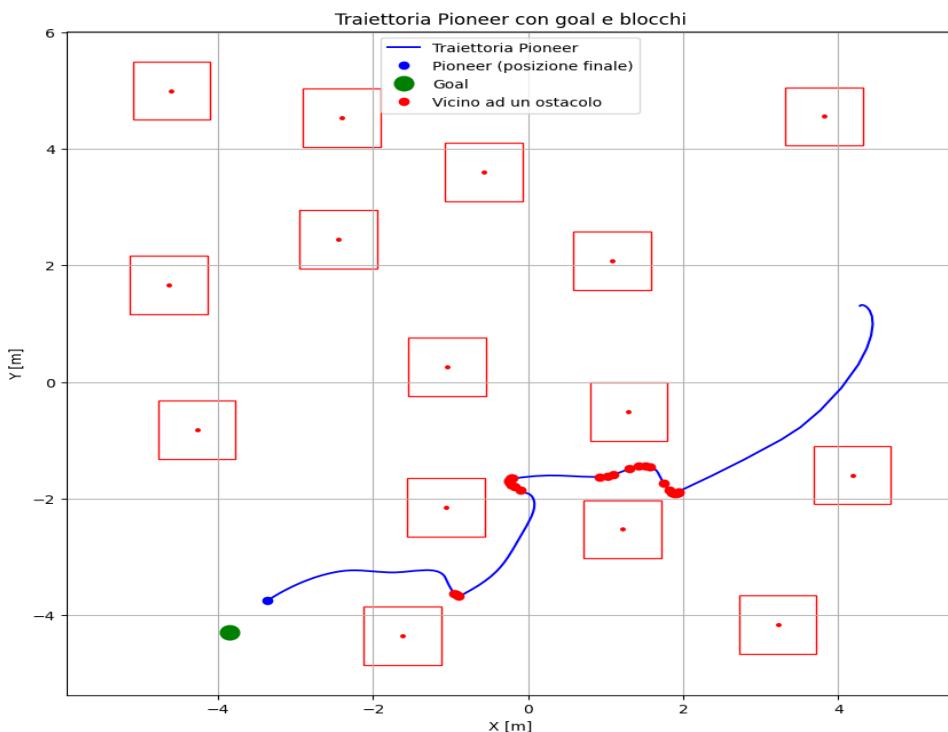
Il modello viene impostato in modalità di valutazione (`eval()`), disattivando così operazioni specifiche del training come il dropout, una tecnica utilizzata durante l'addestramento per prevenire

l'overfitting. All'interno di un blocco `torch.no_grad()`, che disabilita il calcolo dei gradienti per risparmiare memoria e velocizzare l'inferenza, il modello produce delle previsioni che vengono trasformate in probabilità attraverso la funzione di attivazione sigmoid. Il risultato rappresenta la probabilità che ciascun campione appartenga alla classe 1, cioè quella associata ai comportamenti normali. Successivamente, le etichette reali (`y_true`) vengono confrontate per individuare gli esempi appartenenti alla classe 0 (anomali) e quelli appartenenti alla classe 1 (normali). I risultati ottenuti sono i seguenti:

- Numero esempi classe 0: 102;
- Numero esempi classe 1: 213.

Il valore dell'accuratezza sui dati della seconda simulazione è pari a 0.6762.

Per studiare in modo approfondito la traiettoria seguita dal robot, eseguiamo una nuova simulazione durante la quale vengono registrati i dati relativi al movimento del robot all'interno dell'ambiente. Questi dati vengono successivamente utilizzati per costruire un grafico che rappresenta visivamente il percorso compiuto dal robot evidenziando in particolare i punti in cui esso si avvicina maggiormente agli ostacoli presenti nello scenario simulato. All'interno di questo grafico, i punti considerati anomali, ovvero quelli in cui il robot si avvicina eccessivamente a ostacoli, vengono evidenziati in rosso, così da essere immediatamente riconoscibili. Questa rappresentazione visiva risulta estremamente utile per analizzare il comportamento del robot, in quanto consente di individuare con facilità le aree critiche del tragitto, valutando quanto il sistema sia in grado di reagire correttamente a situazioni potenzialmente rischiose. Inoltre, l'evidenziazione dei punti anomali fornisce una conferma dell'efficacia del modello di rilevamento adottato: la capacità di identificare con precisione questi punti dimostra che il modello sviluppato riesce a interpretare correttamente il contesto ambientale e a segnalare condizioni anomale nel comportamento del robot.

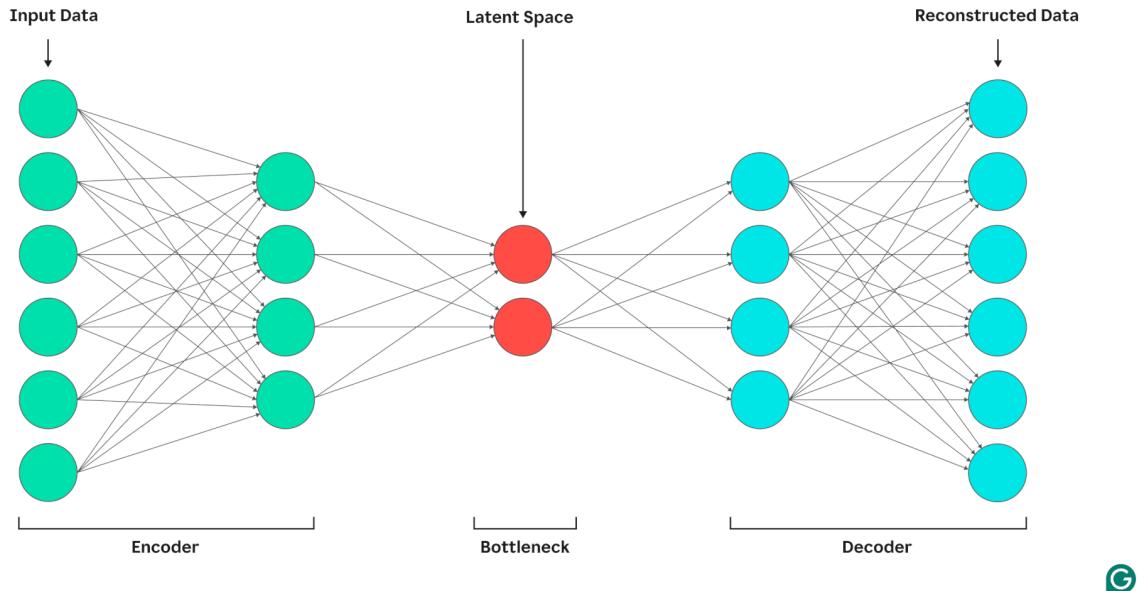


**Fig. 3.5 – Traiettoria seguita dal Robot durante il movimento**

### 3.5 Definizione e ottimizzazione della threshold

La soglia (threshold) rappresenta la distanza minima che consente al robot di avvicinarsi agli ostacoli senza generare segnalazioni di anomalia. Nei primi esperimenti abbiamo utilizzato l'architettura della rete descritta in precedenza, ma questo modello ha mostrato diversi limiti: sono state rilevate numerose anomalie, anche in condizioni normali di movimento, e in particolare si sono verificati frequenti falsi allarmi in prossimità del goal. È stato necessario realizzare un modello autoencoder.

Gli autoencoder sono una tecnica di Deep Learning non supervisionata che, attraverso appositi algoritmi, permette di estrarre pattern significativi a partire da dati grezzi, creando rappresentazioni più semplici e più facili da comprendere e processare.



**Fig.3.6:** Rappresentazione di un autoencoder [16]

Un autoencoder è composto da due parti principali:

- Un encoder, che prende i dati in input e li trasforma in un codice
- Un decoder, che restituisce l'output originale a partire dal codice generato dall'encoder.

Tra encoder e decoder è presente il bottleneck, ovvero uno strato nascosto con dimensionalità inferiore rispetto allo strato di input. Questo strato obbliga la rete a comprimere le informazioni e a conservare solo quelle più rilevanti.

Nel nostro caso, l'autoencoder è implementato come una rete neurale spiking. Lo strato di output presenta lo stesso numero di neuroni dell'input, per poter riprodurre l'input stesso in fase di ricostruzione.

L'encoder comprime i dati di input. Nel nostro modello, l'input è costituito da 16 sensori e dai pixel dell'immagine, elaborati senza convoluzione. Questi vengono processati come un array flatten, che elimina la dimensione spaziale dell'immagine trasformandola in un vettore monodimensionale, diminuendo la complessità del modello e semplificando l'integrazione tra i dati dei sensori e quelli visivi.

Di seguito viene riportata l'architettura della rete.

```

class SAE(nn.Module):
    def __init__(self, num_inputs, num_hidden, num_steps=25, beta=0.95):
        super().__init__()

        self.num_inputs = num_inputs
        self.num_hidden = num_hidden
        self.num_steps = num_steps
        self.beta = beta

        self.fc1 = nn.Linear(self.num_inputs, self.num_hidden)
        self.lif1 = snn.Leaky(beta=self.beta)
        self.fc2 = nn.Linear(self.num_hidden, self.num_inputs)
        self.lif2 = snn.Leaky(beta=self.beta)

    def forward(self, x):

        mem1 = self.lif1.init_leaky()
        mem2 = self.lif2.init_leaky()

        spk2_rec = []
        mem2_rec = []

        for step in range(self.num_steps):
            cur1 = self.fc1(x)
            spk1, mem1 = self.lif1(cur1, mem1)
            cur2 = self.fc2(spk1)
            spk2, mem2 = self.lif2(cur2, mem2)
            spk2_rec.append(spk2)
            mem2_rec.append(mem2)

        return torch.stack(spk2_rec, dim=0), torch.stack(mem2_rec, dim=0)

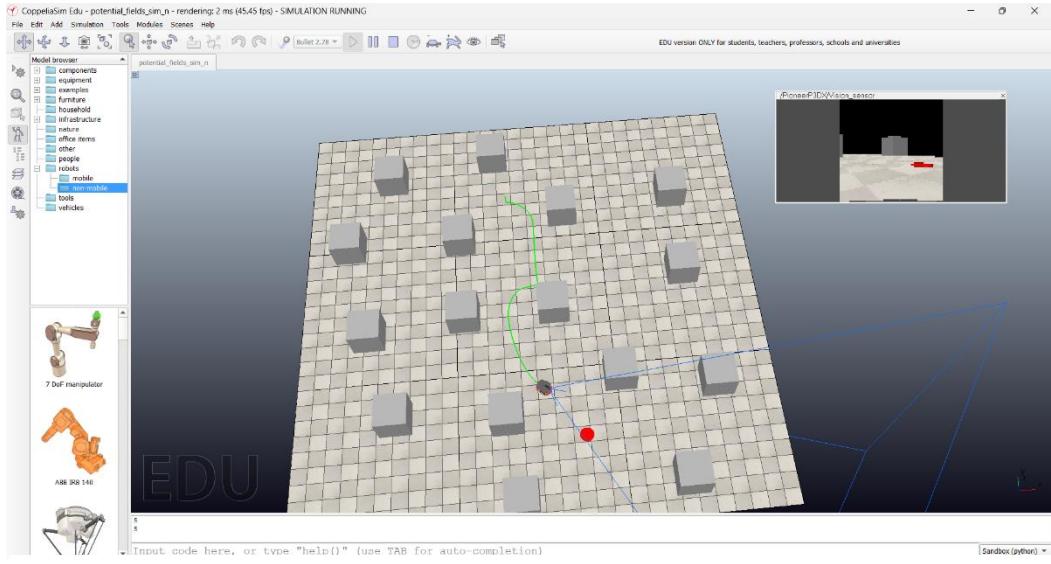
```

**Algoritmo 3.4:** Architettura di un modello Spiking Autoencoder

La rilevazione delle anomalie si basa sulla soglia definita dall'errore medio di ricostruzione, un indice statistico che misura quanto, in condizioni normali, il modello sbaglia nel riprodurre l'input. Nel nostro caso, l'errore medio di ricostruzione calcolato sui dati normali è pari a 0,1.

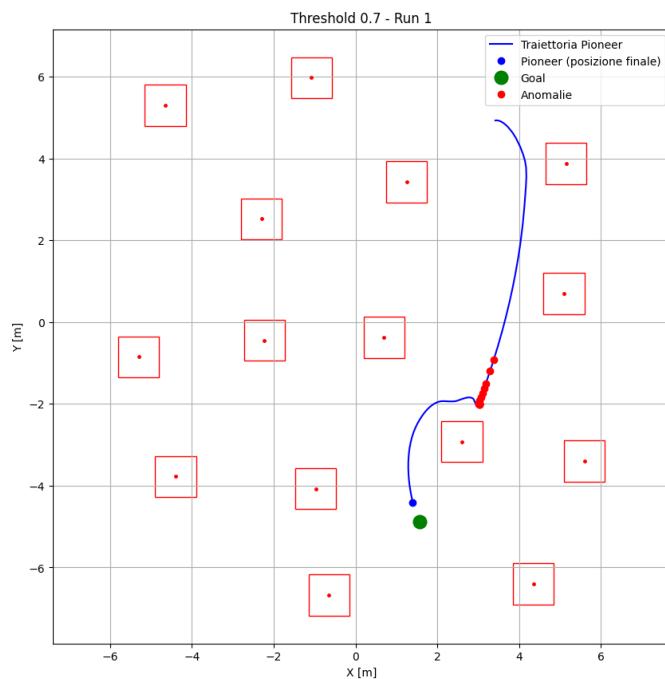
Per stabilire la soglia moltiplichiamo questo valore per due, ottenendo una soglia iniziale fissata a 0,2. Tale soglia rappresenta il limite oltre il quale un input viene considerato anomalo.

In fase di test, forniamo al modello gli input e calcoliamo l'errore medio di ricostruzione rispetto all'input stesso: se l'errore supera la soglia stabilita, quell'input viene classificato come anomalo.

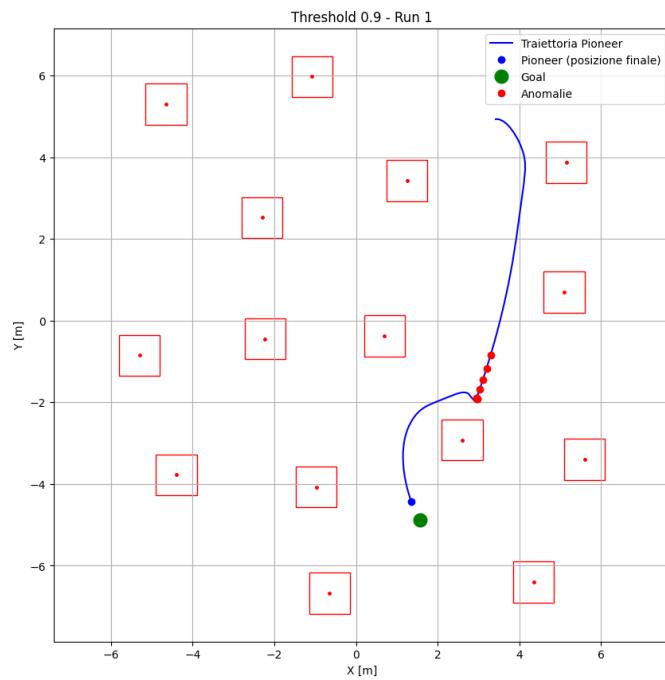


**Fig. 3.7 – Rappresentazione del seed fisso scelto in CoppeliaSim**

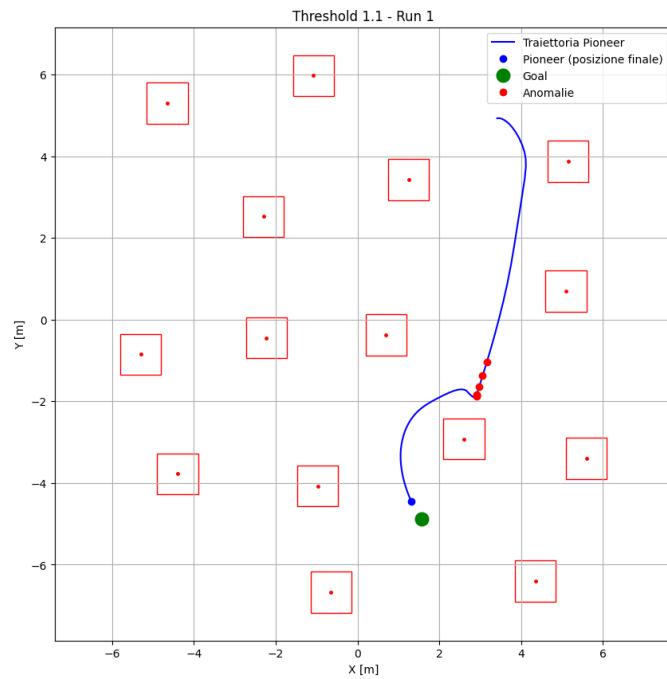
Per eseguire i test, impostiamo un seed fisso nell'ambiente di simulazione. L'utilizzo di un seed fisso permette di replicare la stessa scena, riproducendo la stessa traiettoria, la stessa posizione del goal e degli ostacoli, così da permettere di replicare le simulazioni nelle stesse condizioni. In questo modo possiamo confrontare in modo rigoroso i risultati ottenuti sotto condizioni identiche. L'unico parametro che varia da una simulazione all'altra è il valore della threshold. L'obiettivo è individuare le anomalie, partendo da una soglia pari a 1.0 fino ad arrivare a un valore di 3.0. Per ciascun valore di soglia, calcoliamo il numero di anomalie rilevate e otteniamo i seguenti risultati:



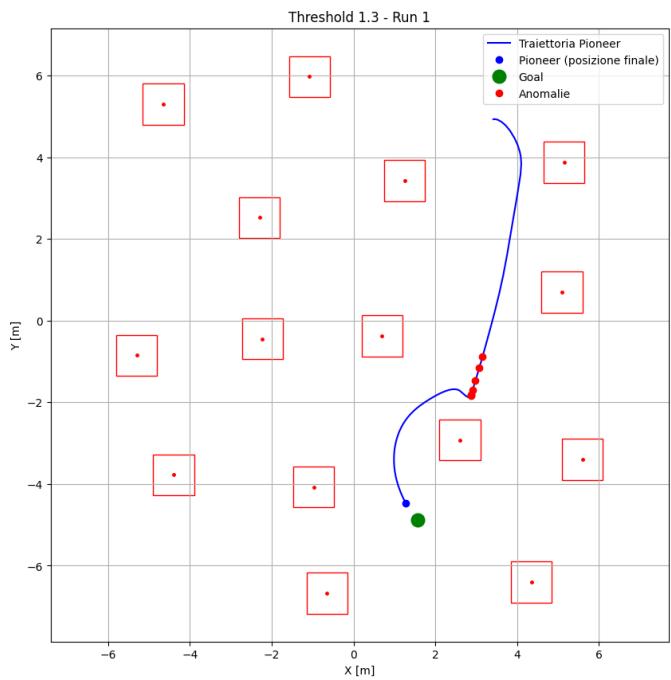
**Fig. 3.8: Rappresentazione delle 19 anomalie rilevate con una soglia pari a 0.7.**



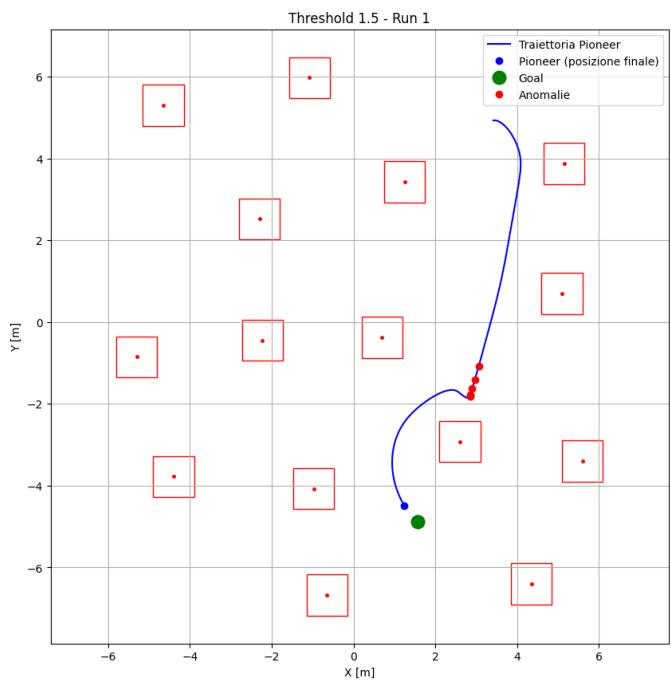
**Fig. 3.9:** Rappresentazione delle 9 anomalie rilevate con una soglia pari a 0.9.



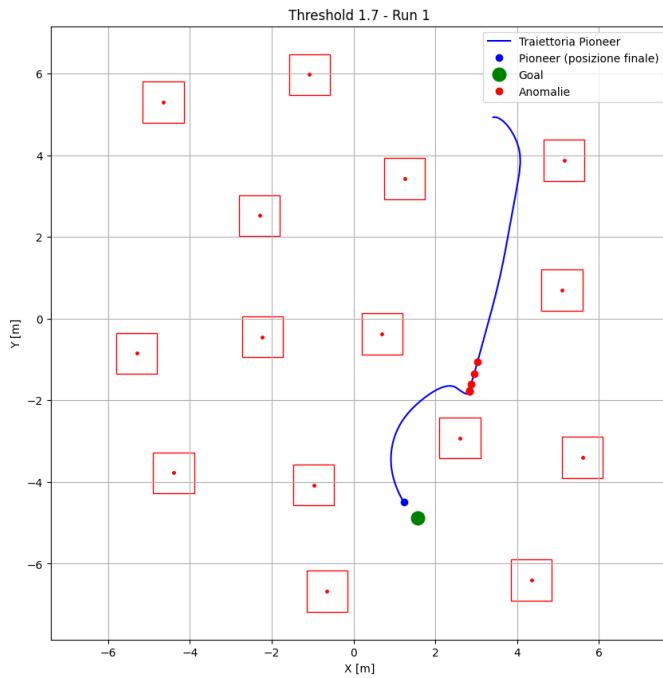
**Fig. 3.10:** Rappresentazione delle 7 anomalie rilevate con una soglia pari a 1.1.



**Fig. 3.11:** Rappresentazione delle 7 anomalie rilevate con una soglia pari a 1.3.

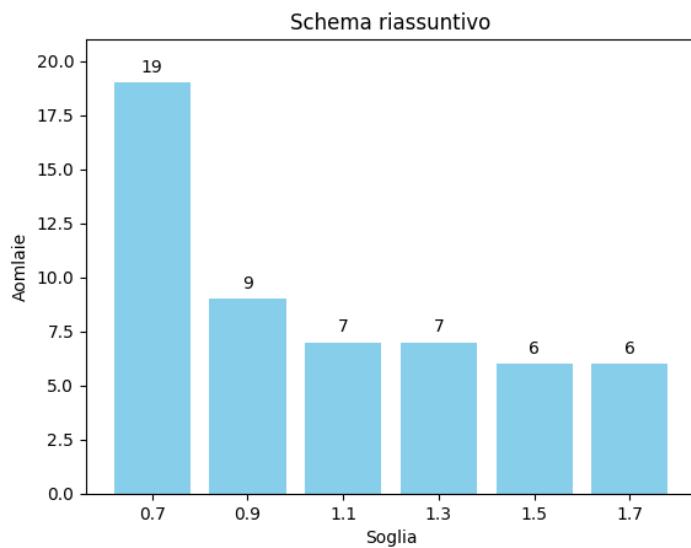


**Fig. 3.12:** Rappresentazione delle 6 anomalie rilevate con una soglia pari a 1.5.



**Fig. 3.13:** Rappresentazione delle 6 anomalie rilevate con una soglia pari a 1.7.

Possiamo osservare che, al variare del valore di soglia (threshold), cambia la distanza minima a cui il robot si avvicina agli ostacoli. Con una soglia bassa, il robot tende ad avvicinarsi molto all'ostacolo, assumendo un comportamento più “audace” ma potenzialmente rischioso, infatti possiamo notare che rileva più anomalie. Al contrario, impostando una soglia più alta, il robot mantiene una distanza maggiore dagli ostacoli, adottando un comportamento più prudente e sensibile, riducendo così il rischio di situazioni anomale.



**Fig. 3.14:** Grafico che mostra il numero delle anomalie al variare del valore della soglia.

## Capitolo 4: Efficienza energetica delle reti neurali

In questo capitolo confronteremo diversi modelli di reti neurali studiando gli aspetti che differenziano le ANN (reti neurali basati su algoritmi di machine learning classico) e le SNN (reti neurali spiking). L'obiettivo è quello di valutare i consumi energetici e le prestazioni in termini di efficienza delle due diverse reti.

### 4.1 Confronto tra Artificial Neural Networks (ANN) e Spiking Neural Networks (SNN)

Le Artificial Neural Networks (ANN) sono reti costituite da neuroni organizzati in uno o più strati layers. Le informazioni viaggiano attraverso i neuroni in modo continua e in maniera sequenziale. Ogni neurone riceve in input valori continui che vengono pesati e sommati tramite una funzione lineare, restituendo un valore in output. Le Spiking Neural Networks (SNN) invece, si ispirano al funzionamento del cervello umano ed elaborano le informazioni in modo discreto, sotto forma di spike. Questa netta differenza ha portato a un'innovazione nei processi computazionali: se nelle ANN il segnale fluisce in modo continuo, nelle SNN il segnale viene inviato attraverso spike solo quando si raggiunge una determinata soglia. Lo spike è un evento binario che assume valore 1 quando il segnale viene inviato, e 0 in caso contrario. In questo modo se da un lato le SNN possono trasmettere un'informazione immediatamente senza attendere l'intero ciclo di addestramento; dall'altro, necessitano di tecniche di apprendimento più complesse a causa della non differenziabilità degli spike. Negli SNN, non è possibile applicare direttamente la backpropagation, perché il calcolo dei gradienti richiede la derivata della funzione di attivazione. Tuttavia, nei modelli SNN, il segnale è discreto (gli spike sono eventi binari) e quindi non derivabile. Per risolvere questo problema, si ricorre a tecniche specifiche come il Surrogate Gradient, che consistono nel sostituire la derivata degli spike con funzioni continue e differenziabili durante il calcolo del gradiente. La complessità dell'addestramento delle SNN deriva non solo dalla natura discreta del segnale, ma anche dal fatto che molte di queste tecniche sono ancora oggetto di ricerca e sperimentazione. Negli ultimi anni, numerosi studi si sono concentrati sul meta-learning, ossia la capacità del modello di imparare ad adattarsi velocemente a nuovi compiti a partire da un numero molto ridotto di esempi (few-shot learning). Un interessante studio è quello condotto da Kenneth Stewart ed Emre Neftci (2022), che hanno applicato il meta-learning alle SNN combinando il surrogate gradient con algoritmi come MAML (Model-Agnostic Meta-Learning). In questo lavoro, i gradienti surrogati vengono resi due volte derivabili, rendendo possibile l'impiego di tecniche di ottimizzazione avanzate e permettendo alle SNN di adattarsi rapidamente a nuovi compiti. I risultati hanno dimostrato che le SNN meta-addestrate possono raggiungere prestazioni paragonabili o superiori ai modelli ANN tradizionali in scenari di few-shot learning. La complessità computazionale dell'addestramento delle SNN è però bilanciata da due principali vantaggi. Il primo è rappresentato dal fatto che tendono a codificare l'informazione nel tempo oltre che nell'intensità del segnale. Ogni neurone, a differenza di quanto avviene nelle ANN, accumula input e genera uno spike solo al raggiungimento di una soglia. Il secondo vantaggio delle SNN è legato alla scalabilità ovvero la capacità della rete di mantenere buone prestazioni anche aumentando la dimensione della rete o del compito. Le SNN, infatti, possono beneficiare di un miglioramento dei tempi di risposta con l'aumentare dei dati in input, un fenomeno che non si verifica nelle ANN. Inizialmente, un fattore determinante nel confronto tra le due reti era rappresentato dal numero di layer. I primi modelli di ANN erano caratterizzati da un numero minore di layer, solitamente compreso tra 1 o 2 layer. L'introduzione del deep learning ha segnato un cambiamento significativo, rendendo possibile la progettazione di reti profonde con decine, centinaia o addirittura migliaia di layer. L'aumento del numero di layer, infatti, permette di estrarre relazioni e rappresentazioni tra i dati sempre più complesse. Proprio per questa ragione, numerosi sono stati gli studi condotti al fine di migliorare questa caratteristica delle ANN. I modelli più recenti possono

presentare moltissimi layer, anche oltre 1000, superando di gran lunga le SNN sia in termini di profondità che di capacità di apprendimento. Tuttavia, questa maggiore complessità comporta anche un aumento del fabbisogno computazionale e della quantità di dati necessari per un addestramento efficace. Dal punto di vista algoritmico, le ANN utilizzano metodi di apprendimento più semplici da interpretare rispetto alle SNN. Le SNN, al contrario, sono spesso considerate delle black box: risulta difficile, se non impossibile, ricostruire in modo dettagliato i processi che conducono a una specifica decisione o output. Gli algoritmi utilizzati nelle reti neurali artificiali (ANN) sono il risultato di decenni di ricerca e sviluppo, che hanno permesso di perfezionare metodi di addestramento robusti e ben consolidati, come la retropropagazione e le sue varianti. Al contrario, le reti neurali spiking (SNN) rappresentano una tecnologia più recente e ancora in fase di studio, per cui gli algoritmi di apprendimento sono spesso più sperimentali e meno maturi. Questo rende le SNN un campo in rapida evoluzione, con ampi margini di miglioramento e di innovazione futura.

Differenze	ANN	SNN
Informazioni	Continue	Discrete (Spike)
Tecniche di addestramento	Più semplici	Più complesse
Codifica	Sequenziale	Temporale
Scalabilità	Non presente	Presente
Layer	Più layer	Meno layer
Algoritmi	Algoritmi consolidati	Algoritmi sperimentali

**Fig.4.1** Tabella riassuntiva che mette a confronto le caratteristiche e differenze principali tra ANN e SNN

Un ultimo aspetto fondamentale da considerare riguarda la velocità di esecuzione delle reti neurali, siano esse ANN o SNN. Questo parametro è strettamente legato all'hardware utilizzato e può mettere in evidenza differenze significative anche in termini di consumo energetico. Tale dimensione sarà oggetto del presente studio, attraverso il confronto tra due reti neurali: una implementata con Keras e l'altra con PyTorch, con l'obiettivo di analizzarne il comportamento energetico su due differenti piattaforme hardware, Loihi e CPU.

## 4.2 Confronto sperimentale

Il presente studio si propone di confrontare, in termini di efficienza energetica, le reti neurali spiking (SNN), implementate in PyTorch, e le reti neurali artificiali (ANN), sviluppate in Keras, mantenendo architetture analoghe. Il confronto si basa sui risultati ottenuti in relazione al consumo energetico misurato su due diverse piattaforme hardware.

Il framework Keras, basato su TensorFlow, è uno dei più diffusi per la progettazione di reti neurali artificiali. Le ANN sviluppate con Keras si avvalgono di neuroni artificiali che operano mediante funzioni di attivazioni continue, come la funzione ReLu (Rectified Liner Unit). Questi modelli aggiornano il proprio stato in modo sincrono e risultano ampiamente impiegati in numerosi ambiti applicativi, dalla robotica alla medicina. Tuttavia, presentano anche alcuni limiti: operando in modalità sincrona e aggiornando i pesi a ogni step, richiedono un elevato numero di floating-point e processi computazionali complessi.

SNNtorch è una libreria sviluppata sulla base di PyTorch, progettata per facilitare la progettazione e l’addestramento di reti SNN. Essa è pensata principalmente per l’apprendimento supervisionato, facendo uso di gradienti surrogati al fine di superare il problema della non derivabilità degli spike. Ispirate ai modelli biologici, le SNN cercano di riprodurre la comunicazione tra neuroni tramite una codifica temporale: non tutti i neuroni si attivano in modo sincrono, ma generano uno *spike* solo al raggiungimento di un determinato valore. Nonostante siano molto utilizzate, le SNN pongono nuove sfide dal punto di vista dell’addestramento, richiedendo un diverso paradigma nell’elaborazione e interpretazione dei dati. In questo contesto, SNNtorch rappresenta attualmente uno dei framework più avanzati e diffusi per lo sviluppo di reti spiking.

Dal punto di vista degli hardware è necessario ancora svolgere studi e ricerche. La sfida è quella di cercare un punto di equilibrio tra efficienza, velocità di esecuzione e consumo energetico.

In questo studio valutiamo il consumo energetico sia con CPU che con Loihi.

Loihi è un processore neuromorfico progettato per offrire un calcolo altamente parallelo ed efficiente dal punto di vista energetico in modalità asincrona. Il chip si basa su un’architettura neuromorfica composta da 128 neurocore, la cui gestione è affidata a tre processori embedded x86. Una rete on-chip asincrona (NoC – Network-on-Chip) connette i neurocore, permettendo la comunicazione tra neuroni artificiali. Quando viene prodotto uno spike questi vengono passati ad altri neuroni attraverso la Noc. La rete Noc impiega router a doppio canale, i quali consentono la trasmissione di un solo messaggio di spike per canale alla volta. Al termine dell’invio degli spike, un neurocore trasmette un messaggio di barriera ai core adiacenti, segnalando la fine della trasmissione. Ogni core destinatario è connesso al core mittente tramite connessioni assonali (axon), ciascuna identificata da un ID univoco. Tali connessioni sono esclusive tra le coppie di core e permettono l’interazione diretta tra le uscite e le entrate dei neurocore.

CPU (Central Processing Unit) è un processore general-purpose progettato per eseguire istruzioni in modo sequenziale o con un grado limitato di parallelismo tramite core multipli. Ogni core della CPU esegue le operazioni di calcolo, controllo e gestione della memoria in modalità sincrona, regolata da un clock globale che coordina tutte le unità di elaborazione. La comunicazione tra i core e le unità periferiche avviene tramite bus di sistema e controller di memoria, con tempi di latenza e sincronizzazione gestiti centralmente. A differenza delle reti neuromorfiche, la CPU elabora dati e istruzioni seguendo un flusso deterministico, senza meccanismi nativi per la trasmissione di eventi asincroni come gli spike. Ogni core accede alle risorse di memoria condivisa per leggere e scrivere dati, e le istruzioni vengono elaborate una alla volta o in piccoli gruppi attraverso tecniche di pipeline e multithreading, che però comportano un maggiore consumo energetico rispetto ai processori neuromorfici specializzati.

Per confrontare il consumo energetico tra modelli artificiali (ANN) e neuromorfici (SNN), abbiamo adottato approcci distinti per le rispettive implementazioni in Keras (TensorFlow) e PyTorch (utilizzando la libreria snnTorch).

Nel caso dei modelli realizzati in Keras, abbiamo utilizzato la funzione ModelEnergy fornita dalla libreria keras\_spiking. Questa funzione stima il consumo energetico del modello in base a costi energetici noti per diverse piattaforme hardware, in particolare:

- Loihi: 0.25 nJ per spike
- CPU: 15 nJ per spike

La funzione si basa sulla struttura del modello e tiene conto dei tipi di layer e delle attivazioni, fornendo una stima energetica realistica su ciascuna piattaforma.

Per quanto riguarda i modelli neuromorfici realizzati in PyTorch con snnTorch, non è disponibile una funzione diretta equivalente a ModelEnergy. Pertanto, abbiamo adottato un approccio manuale per la stima dell'energia:

1. Conteggio degli Spike: Durante l'inferenza, tracciamo il numero di spike generati in ciascun layer spiking.
2. Stima dell'Energia: Il consumo energetico viene stimato moltiplicando il numero totale di spike per il costo energetico medio per spike:
  - Su Loihi: 0.25 nJ/spike
  - Su CPU: 15 nJ/spike

Questo metodo ci consente di ottenere una stima comparabile con quella fornita da keras `_spiking`, benché basata su un'implementazione diversa.

Per garantire un confronto equo, abbiamo progettato architetture equivalenti in Keras e in PyTorch, mantenendo la stessa struttura (numero di layer, neuroni, funzioni di attivazione equivalenti) in entrambi gli ambienti.

I risultati ottenuti sul modello ANN sono:

Dispositivo	Energia stimata [Joule]
Loihi	2.83e-02 J
CPU	4.14e-07 J

I risultati ottenuti per un modello SNN sono:

Dispositivo	Energia stimata [Joule]
Loihi	1.42e-08 J
CPU	8.55e-07 J

Quando si utilizza keras `_spiking`.ModelEnergy, il metodo di analisi del consumo energetico varia a seconda del modello di rete considerato:

- Nel caso in cui il modello sia una rete spiking, ovvero una rete che include layer specifici per le reti neurali spiking come SpikingActivation o SurrogateLIF, l'analisi dell'energia tiene conto della natura evento-driven degli spike. Ciò significa che il consumo energetico viene stimato basandosi sul fatto che gli spike si generano solo in corrispondenza di eventi specifici.

Distribuendo il segnale nel tempo, questa caratteristica consente all'hardware neuromorfico, come Loihi, di sfruttare efficacemente la sparsità degli spike per ridurre drasticamente il consumo energetico rispetto a una CPU tradizionale, che invece elabora continuamente tutti i dati in modo sincrono

- Al contrario, se il modello è una rete classica non spiking implementata in Keras, l'energia stimata per l'esecuzione su Loihi risulterà spesso più alta o al massimo paragonabile a quella consumata su CPU. Questo accade perché in assenza della natura sparsa tipica delle reti spiking, non vi è alcun meccanismo di risparmio energetico legato agli spike, e la rete non è progettata per sfruttare l'architettura neuromorfica. Di conseguenza, Loihi non può beneficiare delle sue caratteristiche di efficienza, e il consumo energetico riflette l'elaborazione continua e sincrona tipica delle reti artificiali tradizionali.

#### 4.3 Valutazione e implicazioni del consumo energetico nelle reti neurali

La valutazione energetica dei modelli rappresenta un elemento fondamentale nello studio delle reti neurali. Quando si progettano reti neurali, oltre a valutarne l'efficacia e le prestazioni, è necessario stabilirne il consumo. Le reti neurali stanno trovando, nel corso degli anni, un impiego sempre più ampio, e studiarne i consumi energetici permette di determinare la sostenibilità e l'efficienza della rete sviluppata. È proprio sui valori di consumo energetico che si basa il confronto tra le diverse strutture hardware.

Oltre a Loihi, sono stati sviluppati vari hardware in grado di eseguire reti neurali in modo più efficiente dal punto di vista energetico. Questi sistemi sono progettati per ridurre il dispendio energetico rispetto alle architetture tradizionali basate su CPU o GPU, ottimizzando l'elaborazione dei calcoli neurali e consentendo di utilizzare le reti neurali anche in contesti con risorse limitate, come dispositivi mobili, robotica o applicazioni edge. I primi hardware sviluppati per le reti neurali sono gli ASIC, che presentano come obiettivo quello di ridurre la complessità dei calcoli. Poi sono stati realizzati i FDDI, chip programmabili caratterizzati da ottime capacità di ottimizzazione hardware e costi contenuti. Solo recentemente sono stati sviluppati i chip neuromorfici come Loihi, che presentano un'elaborazione dei dati basata su spike. Sono state introdotte varie versioni di Loihi. Il primo chip, che è stato lanciato nel 2018, riscontrando un enorme successo, ha portato alla creazione del primo computer capace di sentire e percepire gli odori. Nel 2025 è stata proposta la seconda versione che presenta una elevata velocità di elaborazione delle reti neurali con un consumo molto più basso rispetto a tutti gli altri chip sviluppati in precedenza. Esistono varie differenze tra questi chip neuromorfici e quelli tradizionali come CPU. Nei modelli neuromorfici le informazioni viaggiano attraverso neuroni programmati per lavorare in maniera asincrona, ogni neurone decide quando inviare il segnale agli altri. Un'altra differenza risiede nella memoria. Se negli hardware tradizionali è presente una singola memoria centrale, nei chip neuromorfici sono presenti delle piccole memorie per ogni neurone. Questo aspetto è evidente nell'ultima versione di Loihi, dove ogni singolo neurone può essere programmato per elaborare e lavorare con due diversi programmi contemporaneamente. L'utilizzo di chip come Loihi, la cui elaborazione event-driven è basata su spike, rappresentano un grande passo avanti in termini di consumo energetico. Studiare e confrontare questi hardware è quindi essenziale per scegliere le soluzioni più adatte in termini di prestazioni e di consumo energetico.

Inoltre, il miglioramento dell'efficienza energetica è fondamentale per ottimizzare l'esecuzione delle reti neurali nell'ambito dell'edge computing, in cui le risorse sono limitate e il basso consumo energetico è essenziale. Con edge computing ci riferiamo ad un paradigma in cui l'elaborazione dei

dati avviene vicino alla fonte di produzione. Portare queste reti all'edge significa ridurre l'energia, mantenendo però le prestazioni della rete. Per questo motivo negli ultimi anni sono stati sviluppati modelli leggeri di reti neurali, detti anche TinyLM, progettati per essere eseguiti su dispositivi con risorse limitate, come droni o microcontrollori, dove memoria ed energia disponibili sono fortemente vincolate. Queste reti vengono implementate con specifici accorgimenti, riducendo il numero di parametri e la complessità dei calcoli, in modo da garantire un funzionamento rapido e a basso consumo energetico.

Un esempio di rete TinyLM è Tiny-YOLO, un modello utilizzato per il rilevamento di ostacoli che riesce a mantenere buone prestazioni pur operando in contesti con risorse limitate.

Nella progettazione di reti neurali, è sempre necessario valutare attentamente il giusto compromesso tra accuratezza e consumo energetico: reti più leggere tendono infatti a ridurre l'energia richiesta, ma spesso a discapito di una minore precisione rispetto a modelli più complessi. Trovare il corretto equilibrio tra questi due aspetti è cruciale, l'efficienza energetica è tanto importante quanto le prestazioni del modello.

È fondamentale porre particolare attenzione all'impatto ambientale delle reti neurali in termini di consumo energetico. Attualmente, una parte consistente dell'energia elettrica utilizzata proviene ancora da fonti fossili, che rilasciano anidride carbonica ( $\text{CO}_2$ ) nell'atmosfera. Questo gas è uno dei principali responsabili dell'effetto serra, contribuendo in modo significativo al riscaldamento globale e ai fenomeni connessi al cambiamento climatico. Alla luce di ciò, diventa di primaria importanza progettare modelli di rete neurale sempre più efficienti dal punto di vista energetico, promuovendo al contempo l'adozione di fonti rinnovabili per alimentare le infrastrutture computazionali. L'utilizzo di fonti rinnovabili, come quella solare, per alimentare i data center e le infrastrutture dove vengono addestrate le reti neurali può portare ad un significativo risparmio energetico. Per valutare e ottimizzare il consumo energetico delle reti neurali, sono disponibili numerosi strumenti software di profilazione energetica, che stimano i consumi in base al codice eseguito, alla tipologia di hardware impiegato e alla configurazione del modello. Inoltre, molti processori moderni, in particolare quelli di tipo CPU, includono funzionalità di monitoraggio integrate, come Intel RAPL (Running Average Power Limit), che consentono di misurare e limitare dinamicamente il consumo energetico durante l'esecuzione. Questi metodi consentono di monitorare e ottimizzare il consumo, favorendo un utilizzo più responsabile delle risorse. Solo attraverso un approccio integrato che combina innovazione tecnologica e sostenibilità ambientale sarà possibile contenere l'impatto ecologico delle reti neurali nel lungo termine.

## Conclusioni

Nella parte iniziale della tesi sono stati introdotti i concetti base delle reti neurali, ponendo particolare attenzione sulle reti neurali Spiking( SNN), il cui funzionamento è ispirato alle interazioni tra i neuroni nel sistema nervoso. Per questa ragione è stato necessario introdurre un paragrafo sui processi biologici che stanno alla base della comunicazione neuronale. Successivamente ci siamo soffermati anche su un altro modello di reti, le CNN, utilizzate per il riconoscimento delle immagini. Queste due reti SNN e CNN rappresentano i modelli utilizzati nel nostro studio che si pone l'obiettivo l'individuare anomalie nella traiettoria compiuta da robot mobili.

Dopo aver fornito le informazioni principali sul contesto scientifico di riferimento, che dal Machine learning ha condotto alla nascita del Deep Learning e al successivo sviluppo delle reti neurali, ci siamo concentrati su una prima fase di studio preliminare. In questa fase abbiamo studiato e allenato due diversi tipi di reti. Una SNN per il riconoscimento della superficie in cui si sta muovendo il robot e una CNN che permette di classificare le immagini del dataset in base alla presenza o meno di uno scatolone lungo il tragitto. Questo iniziale studio ha posto le basi per gli esperimenti successivi.

Ma è possibile utilizzare le reti neurali per il riconoscimento di anomalie riscontrate da un robot mobile durante il suo percorso?

Proprio su questa domanda abbiamo impostato il nostro studio. Abbiamo utilizzato CoppeliaSim come ambiente di simulazione per ricavare sia dati che immagini, registrate dai sensori posti sul robot. Il robot si muove in un ambiente in cui sono presenti degli ostacoli. Attraverso varie simulazioni siamo riusciti a far riconoscere al robot eventuali anomalie riscontrate durante il suo percorso. Le anomalie sono i punti in cui il robot si avvicina troppo all'ostacolo. Proprio su questa soglia di avvicinamento, in grado di determinare eventuali anomalie, abbiamo incentrato il nostro studio. Abbiamo svolto delle simulazioni che ci hanno permesso di rilevare dei grafici con le corrispondenti anomalie riscontrate variando il valore della task. Questo studio ci ha permesso di definire il valore minimo della theshold che permette al robot di avvicinarsi, riscontrando un numero contenuto di anomalie.

Gli studi condotti mostrano che le reti neurali possono essere utilizzate per il riconoscimento di anomalie durante il percorso e hanno tutte le potenzialità per diventare uno degli strumenti più potenti per il controllo del movimento robotico. Lo studio dei metodi di apprendimento e degli algoritmi che regolano il corretto funzionamento delle reti neurali è infatti alla base di numerosi studi. Recentemente le reti neurali hanno infatti trovato grande impiego in diversi domini, da quello industriale a quello medico. Riuscire a realizzare reti in grado di sfruttare i processi biologici è l'obiettivo che si desidera raggiungere. Secondo me, questo è un obiettivo che poterà essere conseguito in breve tempo, considerando i rapidi progressi compiuti finora. Essi hanno portato alla scoperta delle reti neurali SNN basati su algoritmi di Deep Learning a partire dalla conoscenza dei modelli di rete Ann basati su algoritmi di machine learning classici.

L'ultima parte della tesi si concentra proprio sulla differenza tra questi due modelli di rete, ponendo particolare attenzione sul loro consumo energetico. Abbiamo effettuato un confronto tra reti Snn in pythorch e ANN in keras, misurando il consumo energetico in Lohi e Cpu. Loihi hardware neutrofico e Cpu hardware tradizionale. Nell'ultimo paragrafo della tesi affrontiamo la tematica del consumo energetico nelle reti neurali e dell'impatto ambientale scaturito dal loro utilizzo.

È auspicabile che, in futuro, il miglioramento delle prestazioni delle reti neurali sia accompagnato anche da progressi sul fronte del risparmio energetico, così da poterle rendere più sostenibili.

## Bibliografia

- [1] iStock (n.d.). *Neuroni* [Immagine]. iStock. <https://www.istockphoto.com/it/immagine/neuroni>
- [2] Hodgkin, A.L., Katz, B. (1949). The effect of sodium ions on the electrical activity of giant axon of the squid. *Journal of Physiology*, 108(1), 37–77.  
Goldman, D.E. (1943). Potential, Impedance, and Rectification in Membranes. *Journal of General Physiology*, 27(1), 37–60
- [3] Da *Artificial neuron* (figura) in Wikipedia. Estratta da [https://en.wikipedia.org/wiki/Artificial\\_neuron](https://en.wikipedia.org/wiki/Artificial_neuron) i
- [4] Deep Learning Italia. (n.d.). *Schema LeNet* [Immagine]. Deep Learning Italia. <https://deeplearningitalia.com/comprendere-le-reti-convoluzionali-lenet/>
- [5] Weturtle. (n.d.). *Intelligenza artificiale, machine learning, deep learning: Facciamo chiarezza.* Weturtle. <https://www.weturtle.org/dettaglio-articolo/25/intelligenza-artificiale-machine-learning-deep-learning-facciamo-chiarezza.html>
- [6] Bache, K., & Lichman, M. (2013). *Robot Data Set*. UCI Machine Learning Repository. <https://archive.ics.uci.edu/ml/datasets/Robot+Data>
- [7] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1), 90–98. <https://doi.org/10.1177/027836498600500106>
- [8] Yamazaki, K. (2023). Spiking neural networks and their applications: A review. *Journal of Neural Computing*, 123–145. <https://doi.org/10.1016/j.jnc.2023.01.005>
- [9] Spitzer, Manfred. *Intelligenza artificiale. Come cambia la nostra vita*. Corbaccio, 2021.
- [10] Arena, Paolo, Luigi Fortuna, Mattia Frasca, e Luca Patanè. 2009. “Learning Anticipation via Spiking Networks: Application to Navigation Control.” *IEEE Transactions on Neural Networks* 20 (2): 202–216. <https://doi.org/10.1109/TNN.2008.2005134>

[11] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, Pearson, 2021, p. 35.

[12] Loaiza, Francisco L., et al. *Utility of Artificial Intelligence and Machine Learning in Cybersecurity*. Institute for Defense Analyses, 2019. JSTOR,  
<http://www.jstor.org/stable/resrep22692>.

[13] Anastasi, G. (2012). *Anatomia umana*. Edi-Ermes.

[14] Silverthorn, Dee Unglaub. *Fisiologia umana*. 8<sup>a</sup> ed., Pearson, 2020.

[15] “Guida rapida alle funzioni di attivazione nel Deep Learning”, 23 Marzo 2021 -  
<https://netai.it/guida-rapida-alle-funzioni-di-attivazione-nel-deep-learning/#page-content>

[16] Speciale, M. (12 gennaio 2025). *Introduzione agli autoencoder*. *diariodiunanalista.it*. Estratto da [diariodiunanalista.it/posts/introduzione-agli-autoencoder](https://diariodiunanalista.it/posts/introduzione-agli-autoencoder)