

# Università degli Studi di Enna "Kore"

## Dipartimento di Ingegneria e Architettura

**Corso di Laurea in Ingegneria Informatica**


# TESI DI LAUREA

## Reti neurali ricorrenti per il riconoscimento di gesti dinamici


*Allievo*:
**Marc'Antonio Lopez**

*Relatore*:
**Ch.ma Prof.ssa Nicole Dalia Cilia**



**ANNO ACCADEMICO 2023 - 2024**

*Ringraziamenti*

*Vorrei esprimere la mia profonda gratitudine alla mia famiglia, in particolare a mia madre e mio padre, per il loro costante sostegno in ogni momento della mia vita. Grazie per avermi dato la forza di andare avanti nonostante le difficoltà, per avermi incoraggiato e per essere sempre stati al mio fianco.*

*Desidero ringraziare i miei nonni, che con il loro affetto e il loro sostegno costante, hanno contribuito al mio percorso. Grazie per non avermi mai fatto mancare nulla, soprattutto per l'amore incondizionato che mi avete sempre dimostrato.*

*Grazie alla mia fidanzata Giulia, che mi ha sempre sostenuto e incoraggiato lungo il mio cammino, anche nei momenti più difficili. La sua presenza costante mi ha dato la forza di andare avanti e affrontare ogni sfida con determinazione.*

*Grazie ai miei stimati colleghi, per avermi donato la vostra preziosa amicizia e compagnia durante questi tre anni. Siete persone straordinarie e sono grato di aver condiviso questo percorso con voi. I nostri momenti insieme resteranno per sempre nel mio cuore. Grazie per tutto il sostegno e l'incoraggiamento che mi avete dato lungo il cammino.*

*Un sentito ringraziamento va anche ai miei amici Filippo, Francesco e Giuseppe. Grazie per la vostra amicizia, per le risate condivise e per il sostegno che mi avete dato in ogni momento. La vostra presenza è stata fondamentale e sono grato di avervi al mio fianco in questo viaggio. Anche se non ci sentiamo spesso in quanto ognuno di noi ha preso strade diverse, la nostra amicizia ci permette di rimanere uniti e sostenerci l'un l'altro.*

*Un ringraziamento speciale va alla mia relatrice, la professoressa Nicole Dalia Cilia. Grazie per aver reso possibile la realizzazione di questa tesi, per il suo sostegno costante durante il mio percorso di ricerca e per avermi guidato non solo dal punto di vista accademico, ma anche umano. La sua dedizione e il suo supporto sono stati fondamentali e per questo le sono profondamente grato.*

*Infine, desidero ringraziare me stesso. Grazie per la determinazione e la resilienza dimostrate lungo questo percorso. Grazie per aver affrontato ogni sfida con coraggio e per non aver mai smesso di credere in te stesso. Questo traguardo è il risultato del tuo impegno, della tua passione e della tua forza di volontà. Sono orgoglioso di ciò che ho raggiunto e grato per le lezioni apprese lungo il cammino.*

*Questa tesi è dedicata a voi, che avete contribuito in modo significativo al mio percorso accademico e personale. Grazie di cuore.*

# Indice

**Sommario**

Nell'era dell'informazione, l'intelligenza artificiale (IA) sta trasformando radicalmente il modo in cui le persone vivono, lavorano e interagiscono. Una delle tecnologie fondamentali che guida questa trasformazione è il riconoscimento dei gesti, che consente ai computer di interpretare i movimenti umani come input. Questa tesi si concentra sul riconoscimento dei gesti dinamici, che presentano sfide uniche a causa della loro natura in continua evoluzione.

Viene esplorato il processo di acquisizione dei dati, che comporta la raccolta di serie temporali per catturare le caratteristiche distintive di ogni gesto. Questo processo implica la gestione di dati complessi e multivariati, poiché i gesti dinamici possono coinvolgere molteplici parametri come posizione, velocità e accelerazione. Successivamente, viene implementata una rete neurale ricorrente (RNN) per classificare questi gesti in base alle loro peculiarità.

Le RNN sono particolarmente adatte per analizzare dati basati sul tempo e rilevare pattern e relazioni temporali complesse. Viene utilizzata un'architettura di RNN bidirezionale, che consente al modello di esplorare le relazioni tra le sequenze di input in entrambe le direzioni temporali, migliorando così la capacità di discriminazione e la generalizzazione del modello. Attraverso l'implementazione di questa tecnologia, la tesi dimostra il potenziale dell'IA nel migliorare l'interazione uomo-macchina, rendendo l'input tecnologico più naturale e intuitivo. Il riconoscimento dei gesti dinamici può avere un impatto significativo su una vasta gamma di applicazioni, dall'assistenza sanitaria all'intrattenimento, facilitando la creazione di dispositivi e sistemi più adattabili e user-friendly.

In definitiva, questa ricerca fornisce un'analisi approfondita sul riconoscimento dei gesti dinamici, sottolineando l'importanza dell'acquisizione dei dati e dell'utilizzo delle reti neurali ricorrenti per la classificazione. I risultati ottenuti possono influenzare significativamente lo sviluppo di applicazioni basate sull'IA in svariati settori, contribuendo così a promuovere un'interazione più intuitiva ed efficiente tra esseri umani e tecnologia.

**Abstract**

In the information age, artificial intelligence (AI) is radically transforming the way people live, work, and interact. One of the fundamental technologies driving this transformation is gesture recognition, which enables computers to interpret human movements as input. This thesis focuses on the recognition of dynamic gestures, which present unique challenges due to their constantly evolving nature.

The data acquisition process is explored, involving the collection of time series to capture the distinctive characteristics of each gesture. This process entails managing complex and multivariate data, as dynamic gestures can involve multiple parameters such as position, speed, and acceleration. Subsequently, a recurrent neural network (RNN) is implemented to classify these gestures based on their peculiarities.

RNNs are particularly suited for analyzing time-based data and detecting complex temporal patterns and relationships. A bidirectional RNN architecture is used, allowing the model to explore the relationships between input sequences in both temporal directions, thereby improving the model's discrimination and generalization capabilities. Through the implementation of this technology, the thesis demonstrates the potential of AI in enhancing human-machine interaction, making technological input more natural and intuitive. Dynamic gesture recognition can have a significant impact on a wide range of applications, from healthcare to entertainment, facilitating the creation of more adaptable and user-friendly devices and systems.

In conclusion, this research provides an in-depth analysis of dynamic gesture recognition, highlighting the importance of data acquisition and the use of recurrent neural networks for classification. The results obtained can significantly influence the development of AI-based applications in various sectors, thus promoting more intuitive and efficient interaction between humans and technology.

# 1 Introduzione

## 1.1 Intelligenza artificiale

L'intelligenza artificiale[1] è quella disciplina che cerca di rendere i sistemi informatici in grado di simulare l'agire o il pensiero umano. L'utilizzo della parola stessa "intelligenza" implica che una macchina debba produrre per l'appunto un comportamento "intelligente", cioè la capacità di raggiungere prestazioni di livello umano in diversi compiti e, affinchè ciò sia possibile, essa deve possedere diverse capacità come:

- Elaborazione del linguaggio umano.
- Memorizzazione e rappresentazione delle informazioni.
- Ragionamento autonomo.
- Apprendimento autonomo.
- Visione artificiale per simulare processi dell'apparato visivo umano.

## 1.2 Machine learning

Il machine learning, noto anche come "apprendimento automatico", rappresenta un sottocampo dell'intelligenza artificiale dedicato alla creazione di sistemi in grado di migliorare le proprie prestazioni basandosi sui dati che elaborano. Questi algoritmi sono capaci di apprendere autonomamente una volta addestrati e trovano applicazione in una vasta gamma di settori.

Nel contesto del machine learning, tre elementi fondamentali guidano l'addestramento degli algoritmi:

- Esperienza: la quantità e la qualità dei dati forniti all'algoritmo durante l'addestramento influenzano direttamente le sue prestazioni.
- Compito: ogni algoritmo di machine learning è progettato per risolvere un compito specifico, che può variare dall'identificazione di oggetti in un'immagine al riconoscimento del linguaggio naturale.
- Misure di performance: una volta addestrato, l'algoritmo viene valutato sul suo rendimento nel compito assegnato, utilizzando metriche come l'accuratezza o l'errore.

Gli algoritmi di machine learning possono essere classificati in diverse categorie, ciascuna con caratteristiche e applicazioni specifiche:

- Apprendimento supervisionato: in questo approccio, il dataset utilizzato per l'ad-

[1] Stuart J. Russell, Peter Norvig e Francesco Amigoni. *Intelligenza artificiale. Un approccio moderno. Vol. 1.* Pearson, 2021.

destramento contiene esempi etichettati con le soluzioni desiderate. Ad esempio, nel riconoscimento di immagini, ogni immagine è associata a una classe corrispondente.

- Apprendimento non supervisionato: qui, il dataset non contiene etichette o soluzioni predefinite. Gli algoritmi in questo caso cercano modelli o strutture intrinseche nei dati senza la guida di etichette.
- Apprendimento semi-supervisionato: questo è un approccio ibrido che combina elementi di apprendimento supervisionato e non supervisionato, utilizzando un insieme di dati etichettati e non etichettati per l'addestramento.
- Apprendimento per rinforzo: in questa categoria, l'algoritmo apprende attraverso l'interazione con un ambiente dinamico, cercando di massimizzare una ricompensa attraverso azioni specifiche. È spesso utilizzato in applicazioni come il controllo di robot o il gaming.

Gli algoritmi di machine learning più semplici, noti anche come "shallow learning" o "algoritmi superficiali", si concentrano su task meno complessi e possono avere meno strati di complessità rispetto a quelli deep learning. Queste varie forme di machine learning offrono diverse prospettive sull'addestramento degli algoritmi e trovano applicazioni in una vasta gamma di settori, dall'analisi dei dati alla robotica e molto altro ancora.

## 1.3 Deep learning

Il machine learning, una branca dell'intelligenza artificiale (AI), è un campo che comprende diverse tecniche per consentire ai sistemi di apprendere dai dati. Tra queste, il deep learning emerge come un sottocampo del machine learning, concentrato sull'utilizzo di reti neurali artificiali con più strati. Queste reti, ispirate al funzionamento del cervello umano, cercano di emulare la sua capacità di apprendimento autonomo da grandi quantità di dati.

La distinzione chiave tra il machine learning e il deep learning risiede nell'architettura delle reti. Mentre il machine learning tradizionale tende ad avere una struttura "piatta", operando direttamente sui dati in ingresso per generare un output, il deep learning utilizza reti neurali con strati multipli di neuroni artificiali. Questa stratificazione consente al modello di apprendere rappresentazioni sempre più complesse dei dati, permettendo una migliore comprensione delle loro caratteristiche.

Sebbene i computer siano estremamente efficienti nell'esecuzione di operazioni complesse come i calcoli matematici, incontrano difficoltà nel comprendere e interpretare dati non strutturati. L'uso delle reti neurali nel contesto del deep learning consente ai computer di avvicinarsi sempre di più alla capacità umana di affrontare tali sfide complesse, aprendo nuove strade per l'intelligenza artificiale e le sue applicazioni pratiche.

## 1.4 Gesture recognition

Il riconoscimento dei gesti rappresenta un'applicazione significativa della computer vision, un ramo dell'intelligenza artificiale dedicato a rendere i computer in grado di estrapolare informazioni rilevanti da immagini digitali e altri input visivi, simile al funzionamento del sistema visivo umano. Questa tecnologia offre diverse motivazioni per essere adottata:

- Interazione più naturale: consente agli utenti di interagire con la tecnologia in modo più intuitivo e naturale, simile all'interazione con il mondo fisico circostante.
- Accessibilità: può essere un'importante risorsa per persone con disabilità o difficoltà nell'utilizzo di metodi di input tradizionali come tastiere o mouse, offrendo un'alternativa più accessibile e inclusiva.
- Realtà virtuale: riveste un ruolo cruciale nell'interazione realistica con ambienti e oggetti virtuali, garantendo un'esperienza di realtà virtuale più coinvolgente e immersiva senza l'uso di controller o altri dispositivi di input. Inoltre, l'assenza di cavi o ostacoli permette una maggiore libertà di movimento.

Attraverso l'impiego di sofisticati dispositivi in grado di rilevare dettagli sulla profondità e la struttura corporea, e combinando tali informazioni con potenti reti neurali, soprattutto quelle convoluzionali, è possibile sviluppare algoritmi di deep learning altamente precisi nel riconoscimento dei gesti umani. Queste tecnologie, grazie al loro potenziale innovativo ed efficiente, possono trovare applicazione in una vasta gamma di settori e applicazioni, offrendo soluzioni all'avanguardia per diverse sfide e esigenze.

Il riconoscimento dei gesti non solo rende l'interazione con la tecnologia più intuitiva, ma può anche migliorare l'efficienza e la sicurezza in vari contesti. Ad esempio, nell'ambito della sicurezza informatica, l'autenticazione basata su gesti può offrire un livello aggiuntivo di protezione contro l'accesso non autorizzato ai dispositivi e ai dati sensibili. Inoltre, nel settore automobilistico, i sistemi di riconoscimento dei gesti possono consentire ai conducenti di controllare le funzionalità del veicolo, come il sistema di infotainment o il condizionatore d'aria, senza distogliere lo sguardo dalla strada.

In ambito sanitario, il riconoscimento dei gesti può essere utilizzato per consentire ai chirurghi di controllare dispositivi e strumenti durante le procedure mediche senza dover interrompere l'operazione per utilizzare mouse o tastiere, migliorando così l'efficienza e la precisione delle procedure.

# 2 Tecnologie e tecniche per il riconoscimento dei gesti statici

## 2.1 Cos'è un gesto statico?

Un gesto statico è definito da una specifica configurazione (ovvero la posizione relativa delle parti del corpo) e/o posa (la posizione del corpo nello spazio) in un determinato istante. Alcuni elementi chiave nella distinzione di alcuni gesti statici potrebbero essere:

- Angolo tra le dita.
- Posizione del polso.
- Flessione delle dita.
- Curvatura della mano.

Indubbiamente, ciò che distingue i gesti statici da quelli dinamici è l'assenza di movimento: la configurazione e posa vengono mantenute invariate per un certo lasso di tempo.

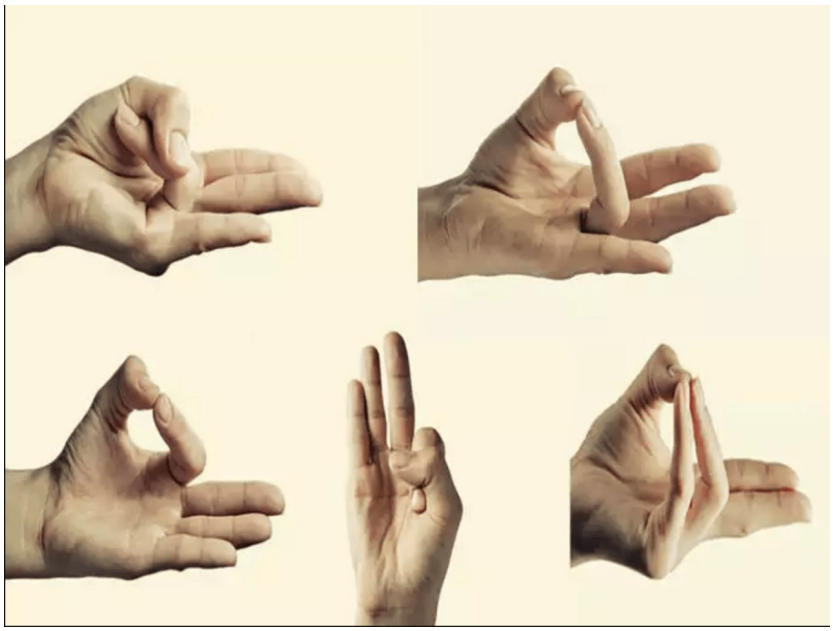


Figura 1: Alcuni gesti statici.

## 2.2 Processo di riconoscimento

Gli algoritmi di riconoscimento di gesti statici possono essere diversi tra loro, ma solitamente seguono un processo comune. Come riportato in[2], le fasi di un generico algoritmo di gesture recognition sono:

1. Hand detection: si individua la mano all'interno dell'immagine e si calcola la sua posizione.
2. Binarizzazione: si trasforma l'immagine in una immagine binaria, con soli due colori, bianco e nero.
3. Post-processing e feature extraction: si corregge il profilo della mano e si estraggono caratteristiche che descrivono la sua forma.
4. Classificazione e filtraggio: si utilizza un algoritmo di apprendimento automatico, come una rete neurale (particolarmente adoperate le reti convoluzionali), per classificare il gesto e si applica un filtro per la rimozione del rumore.

Quando si applica una rete neurale profonda, come ad esempio una rete neurale convoluzionale (CNN), al riconoscimento dei gesti, la rete può apprendere direttamente dalle immagini o dai dati di input senza bisogno di un processo separato per l'estrazione delle caratteristiche. Ciò è reso possibile dalla struttura della rete neurale, che è in grado di imparare automaticamente e in modo gerarchico le caratteristiche rilevanti per la classificazione dei gesti durante l'addestramento.

In breve, quando si utilizzano approcci basati sul deep learning per il riconoscimento dei gesti, la fase di estrazione delle caratteristiche potrebbe non essere necessaria o potrebbe essere integrata direttamente all'interno della rete neurale stessa, a differenza degli approcci di shallow learning che richiedono una fase preliminare di definizione delle caratteristiche.

### 2.2.1 Hand detection

Il processo di hand detection è un'operazione chiave nel riconoscimento della mano in un'immagine. Consiste nell'individuare e isolare la mano, determinandone la posizione esatta all'interno dell'immagine stessa. Una volta identificata, viene creata una regione di interesse (ROI) che include esclusivamente la mano, escludendo altre parti dell'immagine che non sono rilevanti per l'analisi successiva.

Tuttavia, la complessità del processo di localizzazione può variare a seconda dell'immagine. In alcuni casi, potrebbe essere sufficiente utilizzare un algoritmo di rimozione dello sfondo per separare chiaramente la mano dal resto dell'immagine. Tut-

[2]Vito Gentile et al. "Gesture recognition using low-cost devices: Techniques, applications, perspectives". In: *Mondo Digitale* 15.63 (2016), pp. 161–169.

tavia, questo approccio potrebbe essere limitato se nell'immagine sono presenti altre parti del corpo oltre alla mano, come braccia o volti.

In queste situazioni più complesse, potrebbe essere necessario ricorrere a tecniche più sofisticate, come l'analisi dei contorni e delle forme, o l'utilizzo di algoritmi di apprendimento automatico, come le CNN, per individuare e distinguere con precisione la mano dalle altre parti del corpo presenti nell'immagine.

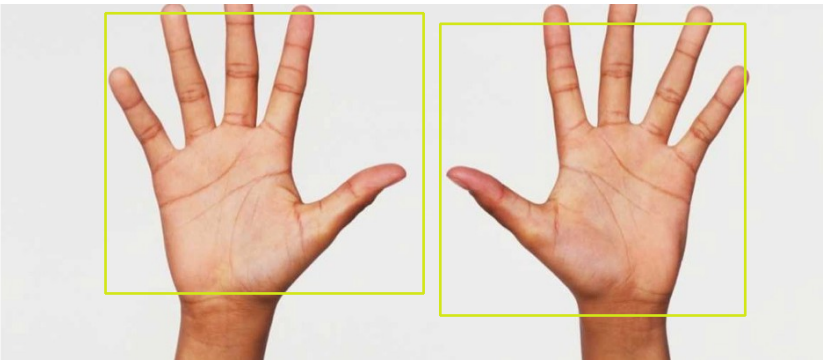


Figura 2: Tracciamento della ROI.

### 2.2.2 Binarizzazione

Una volta individuata la ROI su cui operare per l'estrazione delle caratteristiche (ipotizzando l'utilizzo di un algoritmo shallow), si procede con l'operazione di binarizzazione. In generale, si possono utilizzare approcci diversi, basati sul colore della pelle, sui dati di profondità, o su entrambe queste informazioni. Alcuni dei metodi più adoperati sono:

- Binarizzazione di Otsu: calcola una soglia ottimale per separare le regioni di interesse dall'immagine di sfondo, ed è basata sulla varianza intra-classe, che funziona bene per immagini bimodali (con due picchi nell'istogramma)[3].
- Binarizzazione tramite soglia fissa: qui si seleziona manualmente una soglia fissa per convertire l'immagine in binario, ed è molto semplice da implementare anche se richiede una scelta accurata della soglia.
- Binarizzazione adattiva: questo metodo adatta la soglia localmente, in base alle caratteristiche dell'immagine; è utile quando l'illuminazione varia in diverse parti dell'immagine.

La binarizzazione di un'immagine può essere migliorata utilizzando dati di profondità acquisiti attraverso sensori come telecamere a profondità, LiDAR (light detec-

[3]Fabio Nelli. "OpenCV & Python – La binarizzazione di Otsu". In: *Meccanismo Complesso* (2018). URL: `https://www.meccanismocomplesso.org/opencv-python-otsu-binarization-thresholding/`.

tion and ranging) o sensori stereoscopici. L'aggiunta di informazioni sulla profondità può migliorare indubbiamente l'accuratezza e l'affidabilità del processo di binarizzazione, consentendo di distinguere meglio le mani da altri oggetti o lo sfondo dell'immagine. Come illustrato in[4], alcune soluzioni di binarizzazione che facciano uso dei dati di profondità potrebbero essere:

- Binarizzazione a soglia basata sulla profondità: questo metodo coinvolge l'applicazione di una soglia alla mappa di profondità dell'immagine per separare le regioni che corrispondono alle mani da quelle che non lo sono.
- Segmentazione basata sulla profondità: invece di utilizzare una soglia fissa per binarizzare l'immagine, si segmenta l'immagine di profondità in diverse regioni che corrispondono a oggetti diversi nella scena; le mani possono essere identificate come regioni distinte e quindi estratte.
- Fusion di dati RGB e profondità: combina le informazioni provenienti dalle immagini RGB e dalle mappe di profondità per migliorare la precisione della binarizzazione, sfruttando sia le informazioni di colore che di profondità per distinguere le mani dagli altri oggetti.
- Binarizzazione basata su reti neurali: si sfruttano reti neurali profonde, in particolare le reti neurali convoluzionali, che possono essere addestrate per eseguire la binarizzazione in base ai dati di profondità. Un esempio è YOLO (You Only Look Once), un popolare algoritmo di object detection utilizzato nell'ambito della visione artificiale e dell'intelligenza artificiale, noto per la sua capacità di rilevare oggetti in tempo reale con prestazioni eccellenti.

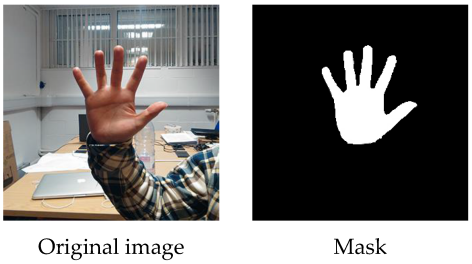



Figura 3: Un esempio di binarizzazione.

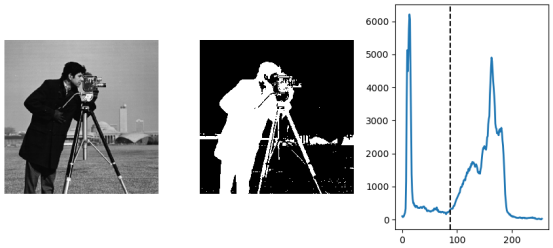



Figura 4: Binarizzazione di Otsu.

### 2.2.3 Post-processing e feature extraction

Dopo la fase di binarizzazione, è fondamentale eseguire un processo di correzione del profilo della forma ottenuta per garantire una rappresentazione accurata e dettagliata

[4]Duong Hai Nguyen et al. "Hand segmentation and fingertip tracking from depth camera images using deep convolutional neural network and multi-task segnet". In: *arXiv preprint arXiv:1901.03465* (2019).

delle caratteristiche dell'oggetto in esame. Questo processo è essenziale per eliminare eventuali imperfezioni e rumorosità presenti nell'immagine binaria, che potrebbero compromettere la precisione del riconoscimento e della classificazione.

Una delle tecniche comuni utilizzate durante il post-processing è l'applicazione di filtri per lo smussamento dei contorni, come ad esempio il filtro mediano. Questo tipo di filtro è efficace nel ridurre il rumore e nell'ottenere contorni più uniformi ed evidenti, migliorando così la qualità complessiva dell'immagine. L'obiettivo finale è ottenere un'immagine binaria ottimizzata, che metta in risalto le caratteristiche essenziali dell'oggetto da riconoscere e che sia al contempo adatta per il successivo processo di classificazione. È importante notare che, durante questo processo, si cerca di mantenere un alto grado di invarianza rispetto a fattori come la scala e la rotazione, in modo da garantire una robustezza del sistema di riconoscimento anche di fronte a variazioni nelle condizioni di acquisizione dell'immagine.

Come illustrato in[5], una strategia plausibile per questa fase del processo potrebbe consistere nella trasformazione dell'immagine originale in scala di grigi. Questo passaggio è cruciale poiché consente di lavorare su un'unica dimensione di intensità anziché su tre canali di colore, semplificando l'elaborazione successiva. Una volta ottenuta l'immagine in scala di grigi, si può procedere con l'applicazione dell'algoritmo di rilevamento dei bordi di Canny. L'algoritmo di Canny[6] è un metodo ben noto e ampiamente utilizzato per il rilevamento dei bordi nelle immagini. Funziona attraverso una serie di fasi, tra cui il calcolo del gradiente dell'immagine, la soppressione dei non massimi e la sogliatura dei bordi. Questo processo produce un'immagine che evidenzia in modo accurato e nitido i bordi presenti nell'immagine originale, riducendo al contempo il rumore di fondo. Alternativamente, se l'immagine è già binarizzata, questa fase può essere omessa e si può passare direttamente all'estrazione delle caratteristiche.

Per l'estrazione delle caratteristiche, una tecnica comune è l'utilizzo dell'algoritmo ORB (Oriented Fast and Rotated Brief). ORB[7] è un metodo efficiente che combina due tipi di descrittori: FAST (Features from Accelerated and Segments Test) e BRIEF (Binary Robust Independent Elementary Features). FAST viene utilizzato per individuare punti chiave nell'immagine, mentre BRIEF genera descrittori binari per ciascun punto chiave. In alternativa all'ORB, esistono altri algoritmi di rilevamento e descrizione delle caratteristiche, come ad esempio SIFT [1] (Scale-Invariant Feature Transform) e SURF (Speeded-Up Robust Features). Questi algoritmi offrono vantaggi specifici in termini di robustezza alla scala e rotazione, per cui la scelta dipenderà dalle esigenze

[5]Ashish Sharma et al. "Hand gesture recognition using image processing and feature extraction techniques". In: *Procedia Computer Science* 173 (2020), pp. 181–190.

[6]Renjie Song, Ziqi Zhang e Haiyang Liu. "Edge connection based Canny edge detection algorithm". In: *Pattern Recognition and Image Analysis* 27 (2017), pp. 740–747.

[7]Monika Bansal, Munish Kumar e Manish Kumar. "2D object recognition: a comparative analysis of SIFT, SURF and ORB feature descriptors". In: *Multimedia Tools and Applications* 80.12 (2021), pp. 18839–18857.

specifiche dell'applicazione.

### 2.2.4 Classificazione e filtraggio

Nel contesto della gesture recognition, l'accuratezza della classificazione dei gesti umani è di vitale importanza per il funzionamento efficace dei sistemi. Le due tecniche più comunemente utilizzate per questo scopo sono basate sulle reti neurali e sulle macchine a vettori di supporto (SVM), entrambe capaci di apprendere dai dati e di fare predizioni precise.

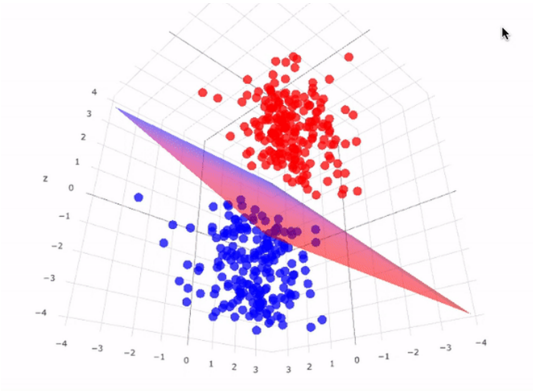


Figura 5: Rappresentazione spaziale di una SVM.

Le reti neurali sono modelli computazionali ispirati al cervello umano, composti da strati di neuroni artificiali che trasformano i dati di input in previsioni di output. Queste reti sono particolarmente adatte per l'apprendimento dai dati non strutturati, come le immagini dei gesti umani, e possono catturare relazioni complesse tra le caratteristiche dei gesti e le loro etichette di classe. D'altra parte, le macchine a vettori di supporto (SVM) sono algoritmi di classificazione che cercano di trovare l'iperpiano di separazione ottimale tra i punti di dati delle diverse classi. Questi algoritmi sono noti per la loro capacità di generalizzazione e per la loro robustezza rispetto a spazi di grandi dimensioni, rendendoli adatti per la classificazione dei gesti umani in contesti complessi. Tuttavia, alcuni ricercatori hanno esplorato approcci alternativi per la classificazione dei gesti, ad esempio utilizzando una metrica di "distanza" tra una rappresentazione della mano e prototipi caratteristici della classe. Questo approccio si basa sulla definizione di prototipi per ciascuna classe di gesti e sulla valutazione della distanza tra la mano in input e i prototipi. La classe associata al prototipo con la distanza minore viene quindi assegnata al gesto in ingresso.

Infine, per migliorare ulteriormente la qualità delle previsioni, è comune applicare filtri per rimuovere il rumore dai risultati della classificazione. Ad esempio, l'applicazione di un gaussian blur può aiutare a ridurre il rumore nelle immagini dei gesti, migliorando così l'accuratezza complessiva della classificazione.

# 3 Tecnologie e tecniche per il riconoscimento dei gesti dinamici

## 3.1 Cos'è un gesto dinamico?

I gesti dinamici possono essere descritti come movimenti che coinvolgono una sequenza di azioni fluide e in evoluzione nel tempo. Questi movimenti possono includere gesti semplici, come agitare una mano o muovere il braccio, o complessi. Rispetto ai gesti statici, che coinvolgono pose fisse e immobili, i gesti dinamici implicano un cambiamento continuo nella posizione e nell'orientamento del corpo o delle parti del corpo coinvolte.

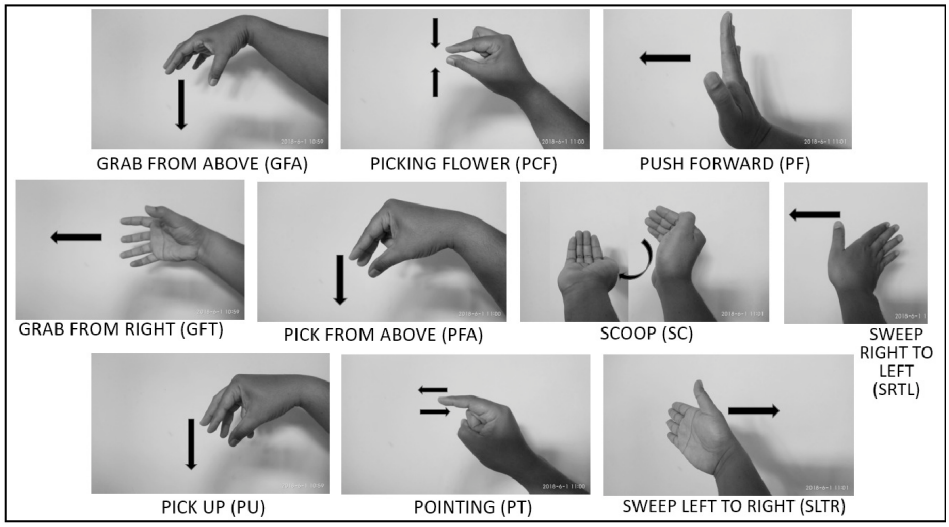



Figura 6: Una sequenza di gesti dinamici.

Il riconoscimento dei gesti dinamici richiede l'analisi di più frame o immagini consecutive catturate nel corso del tempo. Questo processo di analisi implica l'osservazione delle variazioni nella posizione, nella direzione e nella velocità dei movimenti per identificare e interpretare correttamente il gesto complessivo. Poiché coinvolge un'elaborazione di dati temporali e spaziali più complessa rispetto ai gesti statici, il riconoscimento dei gesti dinamici comporta generalmente un carico computazionale maggiore.

Inoltre, i gesti dinamici possono essere influenzati da fattori ambientali come l'illuminazione, il rumore e le interferenze, che possono complicare ulteriormente il processo di riconoscimento. Pertanto, sviluppare algoritmi e sistemi efficaci per il rilevamento e l'interpretazione dei gesti dinamici rappresenta una sfida significativa nell'ambito della visione artificiale e dell'interazione uomo-macchina.

## 3.2 Dati per il riconoscimento dei gesti dinamici

Alcuni autori[8] mostrano come i dati sui gesti dinamici possono essere acquisiti mediante dispositivi indossabili o tecniche di computer vision: i primi offrono dati più precisi anche se meno pratici per un uso quotidiano; al contrario, la visione artificiale mediante una telecamera è più conveniente e ha visto un aumento d'interesse grazie al miglioramento della potenza di calcolo e delle prestazioni degli algoritmi di riconoscimento.

Come già detto, il riconoscimento dei gesti statici avviene tramite un'immagine catturata in un momento specifico, mentre quello dei gesti dinamici utilizza una sequenza di immagini catturate nel corso del tempo: questi ultimi non solo considerano l'aspetto della mano nelle immagini come la posizione, il contorno e le texture, ma anche caratteristiche temporali che descrivono la traiettoria della mano nella sequenza, come ad esempio la velocità del movimento, accelerazione e direzione del gesto nel corso della sequenza. Le informazioni adoperate nel riconoscimento di questo tipo di gesti sono le più disparate, ed ognuna di queste può essere considerata come una forma di informazione separata, ciascuna con i propri vantaggi e limitazioni specifiche. Alcuni esempi:

- RGB: questa modalità di informazione cattura i colori e la luminosità presenti nell'immagine, ed è la migliore per la rappresentazione della texture degli oggetti in quanto consente di distinguere dettagli come rugosità e pattern, fornendo una visione ad alta risoluzione del contesto visivo circostante.
- Profondità: fornisce dati sulla distanza tra fotocamera e gli oggetti nell'ambiente, ed è particolarmente utile per misurare la distanza del gesto rispetto alla fotocamera, consentendo di comprendere meglio la posizione tridimensionale del gesto.

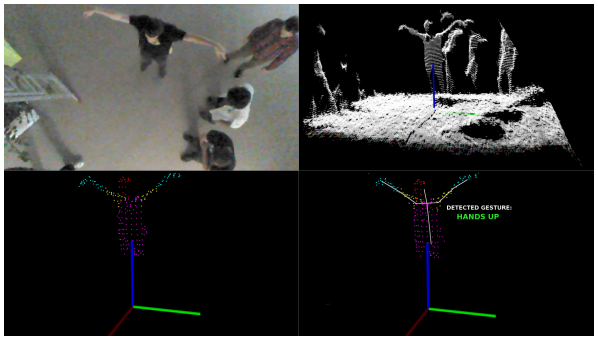



Figura 7: Dati di profondità.

[8]SHI Yuanyuan et al. "Review of dynamic gesture recognition". In: *Virtual Reality & Intelligent Hardware* 3.3 (2021), pp. 183–206.

- Flusso ottico: cattura il movimento relativo tra gli oggetti nell'immagine nel corso del tempo, ed è efficace per ottenere informazioni sul movimento del gesto nel contesto dal momento che è possibile rilevare cambiamenti nel movimento e direzione del corso del tempo.

La combinazionazione di dati di diversa natura, come quelle illustrate in precedenza, viene definita informazione multimodale e può essere utilizzata per migliorare la comprensione o l'analisi di un sistema complesso. Queste informazioni possono essere generate attraverso un procedimento definito fusione, che può essere di diversi tipi:

- Fusione a livello di dati: le informazioni vengono combinate in un'unica rappresentazione prima di essere fornite agli algoritmi di apprendimento automatiche (es. i dati RGB e di profondità possono essere combinati in una unica rappresentazione 3D prima dell'elaborazione).
- Fusione a livello di feature: le caratteristiche estratte da diverse modalità vengono combinate durante la fase di feature extraction.
- Fusione a livello di decisione: in questo caso, le decisioni ottenute da modelli addestrati su dati monomodali vengono combinate alla fine del processo di apprendimento, utilizzando metodi come la media o il massimo delle decisioni dei modelli monomodali.

## 3.3 Tecniche di riconoscimento dei gesti dinamici

Diverse tecniche sono state sviluppate per il riconoscimento dei gesti dinamici, e i ricercatori adottano approcci diversificati in base alle proprie esigenze.

Un esempio significativo è rappresentato dal lavoro di[9], il quale si avvale di coordinate 2D della mano e di specifici marker per la ricostruzione tridimensionale della stessa. Questa metodologia si basa sull'utilizzo di una telecamera per acquisire le immagini, le quali vengono elaborate nel formato HSV (Hue, Saturation, Value) per rimuovere il rumore e compensare eventuali variazioni di luce. Grazie a una tecnica di riferimento del colore, è possibile identificare le mani e i marker sulle dita, agevolando così una rapida identificazione della mano in movimento. Successivamente, le coordinate 2D vengono trasformate in coordinate 3D utilizzando una proiezione inversa e la cinematica inversa per determinare gli angoli delle articolazioni delle dita. Tale approccio fornisce una rappresentazione tridimensionale della mano e delle sue articolazioni, facilitando il riconoscimento dei gesti. Per questo scopo, viene impiegato un Hidden Markov Model (HMM), un modello estremamente potente che si basa su due livelli di processi stocastici: uno strato nascosto che rappresenta il processo dinamico che si intende riconoscere e uno strato di osservazione che rappresenta la sequenza osservata

[9]Chris Joslin et al. "Dynamic gesture recognition". In: *2005 IEEE Instrumentationand Measurement Technology Conference Proceedings*. Vol. 3. IEEE. 2005, pp. 1706–1711.

di posture della mano e degli angoli delle articolazioni. L'addestramento dell'HMM avviene mediante l'utilizzo di dati statistici variabili che rappresentano gesti specifici, seguendo un approccio simile a quello di una rete neurale. Durante la fase di riconoscimento, l'HMM valuta la probabilità di ogni gesto dinamico dato l'input di coordinate tridimensionali della mano e degli angoli delle articolazioni, identificando infine il gesto con la probabilità più alta sopra una soglia di accettazione.

Altri approcci, come quelli intrapresi da[10], sono:

- Approcci basati sul machine learning: un esempio sono gli stessi HMM visti in precedenza e i Dynamic Time Warping (DTW), un algoritmo che misura la similarità tra due serie temporali di diverse lunghezze, ed è stato originariamente utilizzato per il riconoscimento vocale, anche se spesso impiegato nel riconoscimento dei gesti dinamici (il suo uso richiede il pre-processing delle sequenze video, l'estrazione delle caratteristiche e la normalizzazione rispetto ad un modello di sequenza).

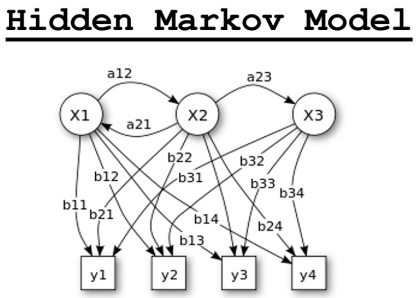



Figura 8: Struttura di un HMM.

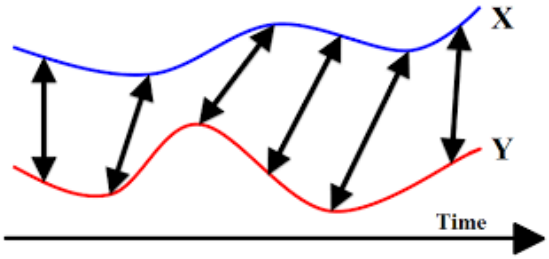



Figura 9: Modello di DTW.

[10]SHI Yuanyuan et al. "Review of dynamic gesture recognition". In: *Virtual Reality & Intelligent Hardware* 3.3 (2021), pp. 183–206.

- Approcci basati sul deep learning: viene fatto riferimento a tre tipi di reti neurali diverse:
  - Le reti a due stream si compongono di due sotto-reti distinte: una dedicata all'analisi spaziale dei gesti utilizzando immagini RGB e l'altra focalizzata sull'analisi temporale basata sul flusso ottico. Questo approccio permette di catturare sia informazioni spaziali che temporali, le quali verranno infine fuse per ottenere una rappresentazione completa e accurata dei gesti. Ad esempio, l'analisi spaziale può concentrarsi sui dettagli visivi dei movimenti della mano, mentre l'analisi temporale può considerare la sequenza di cambiamenti nel flusso di movimento nel tempo.

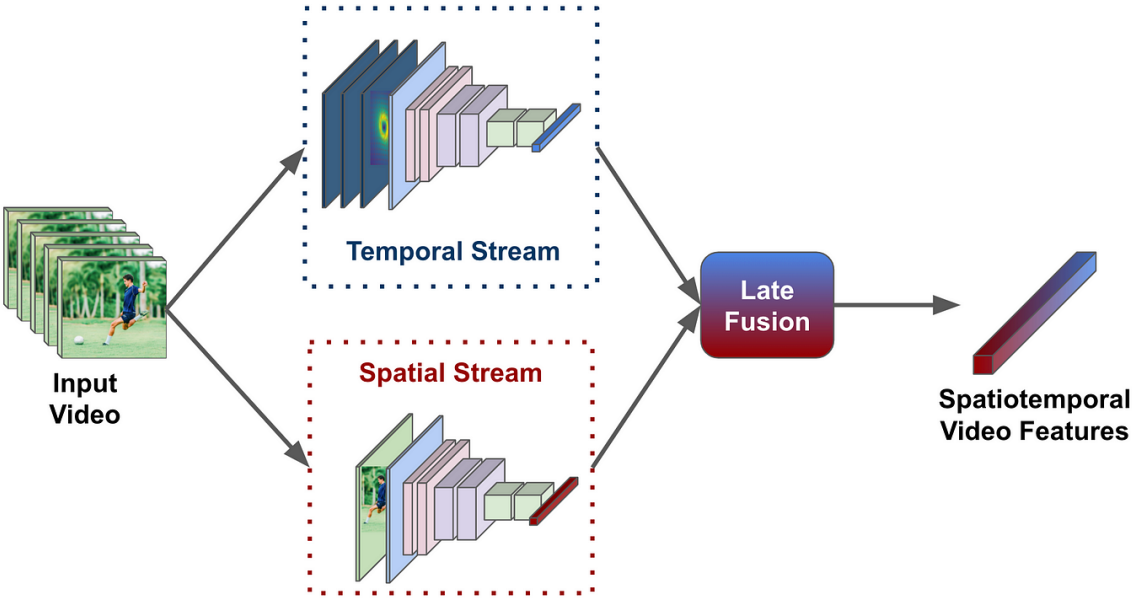



Figura 10: Rete a due stream.

– Le reti neurali convoluzionali tridimensionali (3D CNN) sono progettate specificamente per l'estrazione di informazioni spaziali e temporali da sequenze di dati tridimensionali, come i video. Queste reti sono in grado di analizzare i cambiamenti nel tempo e nello spazio all'interno di un video, consentendo una migliore comprensione dei gesti dinamici. Ad esempio, una 3D CNN potrebbe individuare pattern di movimento caratteristici che indicano un particolare gesto durante una sequenza di video.

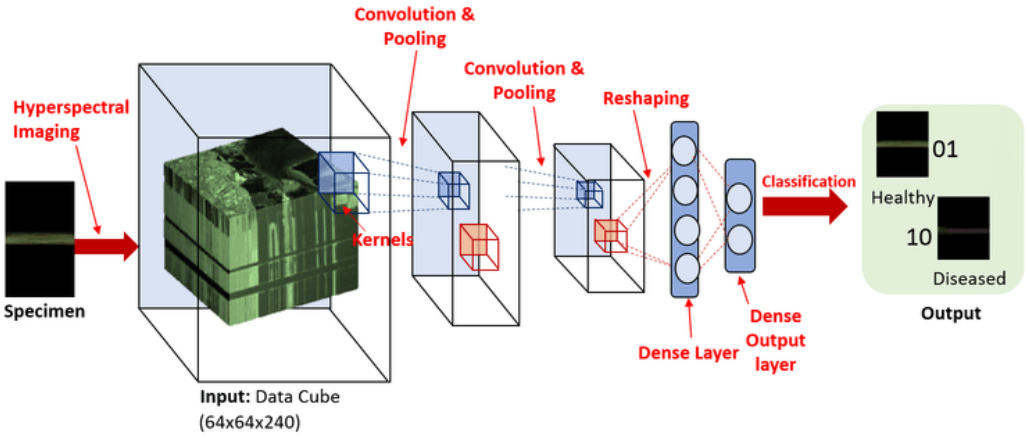



Figura 11: CNN tridimensionale.

– Le RNN, o reti neurali ricorrenti, sono un'ottima soluzione per l'elaborazione di serie temporali, e quindi particolarmente utili per il riconoscimento di gesti dinamici. Le tratteremo nel dettaglio fra qualche capitolo.

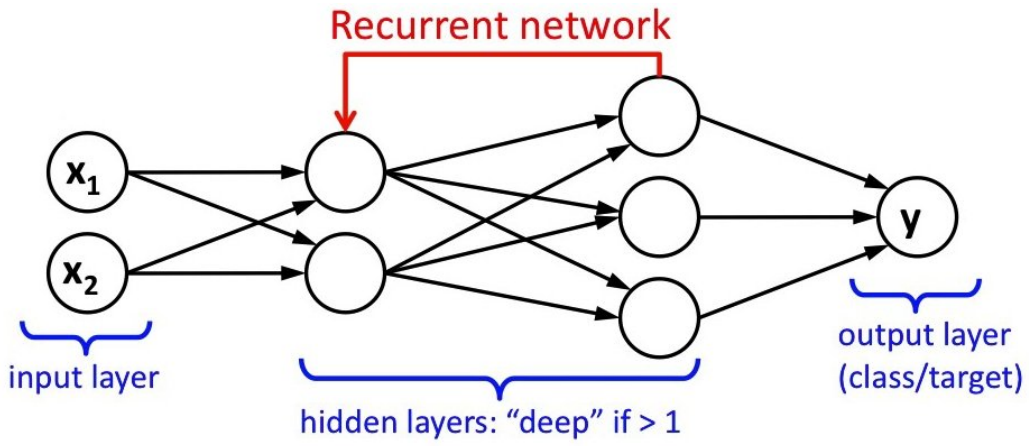



Figura 12: Struttura di una RNN.

# 4 Panoramica su dispositivi a basso costo per il riconoscimento dei gesti

## 4.1 Introduzione

Il successo del campo del riconoscimento dei gesti è strettamente correlato alla crescente disponibilità di dispositivi sempre più accessibili ed economici che integrano questa tecnologia. L'evoluzione di tali dispositivi ha reso questa tecnologia accessibile a una vasta gamma di utenti e ne ha ampliato l'impiego in molteplici settori, tra cui l'elettronica di consumo, la salute, l'intrattenimento e l'automazione domestica. Questi dispositivi utilizzano diverse tecnologie per rilevare e interpretare i gesti, tra cui telecamere 3D, sensori a infrarossi e sensori di movimento.

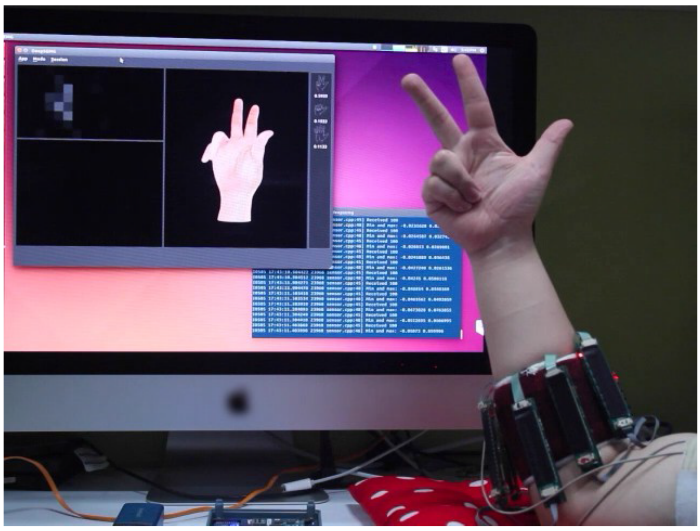


Figura 13: Generico sensore per il riconoscimento dei gesti.

## 4.2 Panoramica

Poiché nel seguente elaborato verrà utilizzato un particolare dispositivo adoperato per l'hand tracking, è bene fare una panoramica dei dispositivi presenti nel mercato a disposizione per tale obiettivo:

- Telecamere 3D: le telecamere 3D rappresentano una tecnologia avanzata che utilizza telecamere dotate di computer vision e sensori 3D per rilevare e tracciare i movimenti delle mani e del corpo in uno spazio tridimensionale. Questi dispositivi offrono una precisione e una sensibilità maggiori rispetto ai sistemi bidimensionali, consentendo un'interazione più naturale e immersiva con i dispositivi digitali. Un esempio di telecamera 3D dotata di sensori di profondità per la misura della distanza è l'Intel RealSense D435i[11]. Questa telecamera offre una precisione e una sensibilità elevate nel rilevare i movimenti e nella misurazione della distanza, consentendo un'ampia gamma di applicazioni nel campo della realtà aumentata, della robotica, dell'analisi del movimento e molto altro ancora.

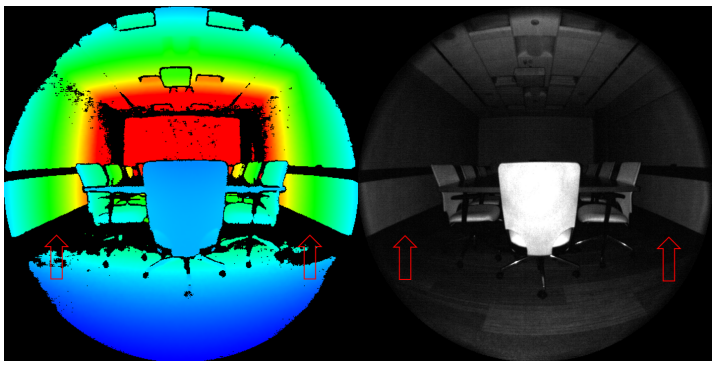


Figura 14: Vista di una telecamera 3D.

- Sensori a infrarossi: i sensori a infrarossi rappresentano una tecnologia fondamentale impiegata in diversi dispositivi per il riconoscimento dei gesti e il monitoraggio della presenza umana. Questi sensori sfruttano onde infrarosse per rilevare la presenza o i movimenti delle persone all'interno di un determinato ambiente. Tale tecnologia è particolarmente utile per dispositivi destinati all'automazione domestica e alla sicurezza. Inoltre, i sensori a infrarossi sono impiegati anche in dispositivi di realtà virtuale come il Meta Quest 2: questo visore VR[12] utilizza una combinazione di sensori, inclusi sensori a infrarossi, per tracciare i movimenti dell'utente nello spazio tridimensionale e consentire un'esperienza di gioco immersiva e interattiva. In particolare, i sensori a infrarossi contribuiscono al tracciamento

[11] *Intel RealSense Depth Camera D435i*. `https://www.intelrealsense.com/depth-camera-d435i/`. Recuperato il 4 marzo 2024. 2024.

[12] Smartworld.it. "Oculus Quest 2, la recensione: il VR per tutti (anche per chi ha un iPhone)". In: (2022). URL: `https://www.smartworld.it/recensioni/oculus-quest-2`.

preciso delle mani e dei movimenti della testa, consentendo agli utenti di interagire con l'ambiente virtuale e con gli oggetti digitali tramite gesti delle mani e del corpo.


Figura 15: Hand tracking su Meta Quest 2.

- Sensori di movimento: questi sensori, come giroscopi e accelerometri, costituiscono un componente fondamentale nei dispositivi indossabili come gli smartwatch e i braccialetti fitness. Questi dispositivi utilizzano i sensori di movimento per monitorare e rilevare i cambiamenti nella posizione, nell'orientamento e nell'accelerazione del dispositivo stesso. L'uso di tali sensori consente ai dispositivi indossabili di tracciare con precisione i movimenti delle mani e dei polsi dell'utente. Nei braccialetti fitness, ad esempio, i sensori di movimento vengono impiegati per monitorare l'attività fisica dell'utente, rilevando il numero di passi, la distanza percorsa, le calorie bruciate e altri parametri relativi all'esercizio fisico. Gli smartwatch, invece, utilizzano i sensori di movimento non solo per monitorare l'attività fisica, ma anche per fornire funzionalità aggiuntive come il tracciamento del sonno, il monitoraggio del battito cardiaco e il riconoscimento dei gesti delle mani.

In letteratura ci sono diversi articoli scientifici che focalizzano la loro attenzione su dispositivi a basso costo per il riconoscimento dei gesti della mano, per un qualunque genere di utilizzo. Un esempio è riportato in[13], che descrive un sistema di riconoscimento dei gesti basato sull'uso di telecamere 3D, come il Microsoft Kinect, insieme a webcam USB, per rilevare e tracciare i movimenti delle mani e del corpo.

Il sistema realizzato in questo articolo opera attraverso diversi moduli:

[13]Hui-Shyong Yeo, Byung-Gook Lee e Hyotaek Lim. "Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware". In: *Multimedia Tools and Applications* 74 (2015), pp. 2687–2715.

- Modulo della telecamera: questo modulo recupera i frame generati dalla webcam o dal Kinect, e li elabora attraverso varie tecniche, per essere successivamente passati al modulo di rilevamento dei gesti.
- Modulo di rilevamento: questo modulo riceve i frame elaborati dalla telecamera e estrae informazioni aggiuntive come la posizione delle mani, lo stato aperto/chiuso delle mani, il conteggio e la posizione delle dita.

### 4.2.1 Intel RealSense D435i

Le telecamere Intel RealSense rappresentano una tecnologia avanzata nel campo del rilevamento della profondità 3D, ampiamente utilizzata in settori come la realtà virtuale, la computer vision e la scansione 3D. Tra le loro funzionalità principali vi è il tracciamento facciale, che consente il riconoscimento delle espressioni facciali e la creazione di avatar animati, e il tracciamento scheletrico, che registra le posizioni tridimensionali delle articolazioni del corpo.

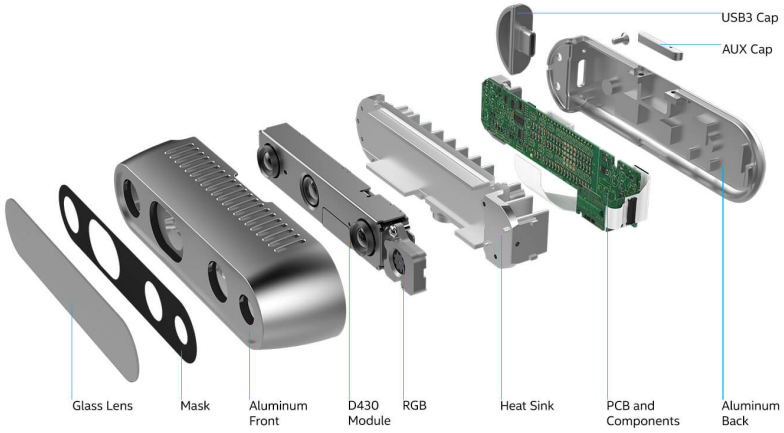



Figura 16: Intel RealSense D435i.

Il modello D435i della Intel RealSense Depth Camera rappresenta un notevole avanzamento tecnologico in questo settore. Oltre alla visione stereo per calcolare la profondità degli oggetti, questa fotocamera è dotata di un'unità di misurazione inerziale (IMU), che misura l'accelerazione e la rotazione lungo gli assi x, y e z. Questo consente una stima precisa del movimento e dell'orientamento nello spazio tridimensionale, rendendo la D435i ideale per applicazioni che richiedono il tracciamento degli oggetti in movimento o la monitoraggio della posizione della fotocamera stessa.

L'integrazione dei dati provenienti dall'IMU con quelli della visione stereo consente alla D435i di compensare i movimenti della fotocamera durante l'acquisizione delle immagini, migliorando la coerenza e l'accuratezza delle mappe di profondità generate. Questa combinazione di visione stereo e IMU rende la D435i estremamente versatile e adatta a una vasta gamma di applicazioni, come la realtà aumentata, la navigazione autonoma, la robotica e altro ancora.

### 4.2.2 Orbbec Astra

L'Orbbec Astra è un sensore di movimento compatto progettato per il rilevamento preciso dei movimenti del corpo umano e il tracciamento delle giunture in tempo reale. Simile al Microsoft Kinect, l'Astra è dotato di una fotocamera RGB, una fotocamera ad infrarossi, un proiettore IR ed un microfono integrato. Questi componenti consentono al sensore di acquisire dati di profondità e tracciare le giunture del corpo umano, permettendo interazioni naturali e intuitive con il computer e le applicazioni digitali.


Figura 17: Orbbec Astra.

Secondo quanto riportato in[14], l'Orbbec Astra ha una risoluzione di 1280x960 pixel per la fotocamera RGB, mentre quella ad infrarossi è di 640x480 pixel, entrambe con un framerate di 30 fotogrammi al secondo, garantendo una cattura fluida dei movimenti. I sensori permettono una copertura ampia e dettagliata dell'area circostante grazie ad un campo visivo di 60 gradi orizzontalmente, 49.5 gradi verticalmente e 73 gradi diagonalmente.

Una caratteristica importante dell'Orbbec Astra è la sua compatibilità con diversi sistemi operativi e framework di sviluppo. Il sensore è supportato da diversi framework come l'Astra SDK e l'OpenNI framework, consentendo agli sviluppatori di creare applicazioni personalizzate per una vasta gamma di piattaforme e dispositivi.

Grazie alle sue caratteristiche tecniche e alla sua flessibilità, l'Orbbec Astra viene utilizzato in una varietà di contesti, tra cui applicazioni di realtà virtuale, interazione uomo-macchina, monitoraggio del movimento e applicazioni di gioco. La sua capacità di tracciare con precisione i movimenti del corpo lo rende uno strumento prezioso per lo sviluppo di esperienze immersive e interattive.

[14]Alina Delia Călin Adriana Coroiu e Adriana Coroiu. "Interchangeability of kinect and orbbec sensors for gesture recognition". In: *2018 IEEE 14th international conference on intelligent computer communication and processing (ICCP)*. IEEE. 2018, pp. 309–315.

### 4.2.3 Google Soli

Il Google Project Soli[15] è un innovativo sensore basato su radar che emette segnali radio a lunghezza d'onda millimetrica. Quando questi segnali interagiscono con gli oggetti nell'ambiente, l'energia viene assorbita o dispersa, e i dati risultanti vengono rilevati dal trasduttore del sensore. Successivamente, un software appositamente progettato analizza questi dati per descrivere le dinamiche e le proprietà dei centri di dispersione.

Questo sensore è stato studiato per diverse applicazioni, tra cui l'identificazione dei materiali. Tuttavia, è importante notare che l'energia dei segnali riflessi può essere influenzata dalla posizione e dalla geometria degli oggetti. Ad esempio, oggetti con geometrie angolari o superfici piatte perpendicolari al segnale trasmesso tendono a riflettere più energia. Per mitigare questo fenomeno e garantire risultati più accurati, è stato implementato un design meccanico che riduce le variazioni nella posa finale degli oggetti.

Il sensore Google Soli è caratterizzato da una notevole versatilità e precisione nel rilevamento delle caratteristiche degli oggetti. Grazie alla sua capacità di identificare materiali e analizzare le dinamiche degli oggetti, può essere utilizzato in una vasta gamma di contesti, dalla robotica alla realtà aumentata e alla domotica. La sua dimensione compatta e la sua affidabilità lo rendono un componente ideale per sistemi interattivi e di controllo gestuale.

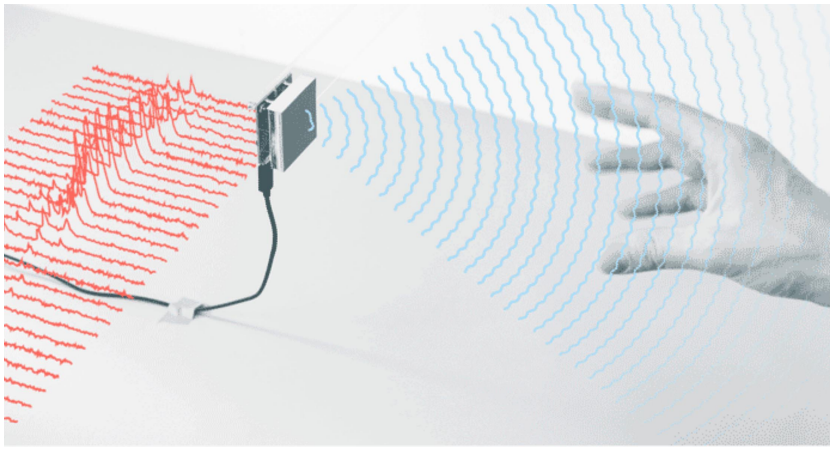


Figura 18: Google Project Soli.

[15] Zak Flintoff, Bruno Johnston e Minas Liarokapis. "Single-grasp, model-free object classification using a hyper-adaptive hand, google soli, and tactile sensors". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1943–1950.

### 4.2.4 Microsoft Kinect

Il Kinect, sviluppato da Microsoft, è un sensore di movimento inizialmente pensato per il gaming, ma presto adottato anche in settori come l'educazione, la medicina e la robotica. Dotato[16] di fotocamera RGB, proiettore IR, sensore di profondità e array di microfoni, il Kinect può rilevare e tracciare le giunture del corpo umano e i movimenti in tempo reale.

La prima versione del Kinect è in grado di tracciare 20 diverse giunture del corpo umano, con una frequenza di 30Hz sia per i dati a colori che per quelli di profondità. La telecamera ad infrarossi ha una risoluzione di 640x480 pixel, mentre quella di profondità è di 320x240 pixel, con un ampio campo visivo di 57 gradi orizzontalmente e 43 gradi verticalmente. Il software SDK si basa sulla luce strutturata dei dati di profondità ed è in grado di tracciare fino a due utenti con una gamma di profondità compresa tra 0.4 e 6 metri, utilizzando un emettitore infrarosso[17] per calcolare le distanze in base alle distorsioni dei punti infrarossi.

La seconda versione del Kinect presenta diverse migliorie, come una fotocamera a colori FULL-HD e una fotocamera di profondità 512x424 pixel, con un campo visivo più ampio (70 gradi orizzontalmente e 60 gradi verticalmente) ed un frame rate di 60Hz per la fotocamera di profondità. Le giunture tracciabili adesso sono un massimo di 25 per 6 utenti in contemporanea, con una gamma di profondità tra 0.5 e 4.5 metri, utilizzando un metodo Time-of-Flight (ToF), basato sulla misurazione della velocità della luce per calcolare la distanza.

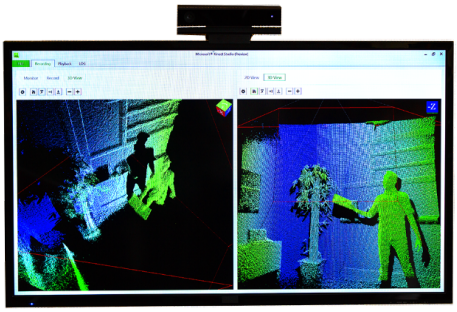


Figura 19: Microsoft Kinect.

Entrambe le versioni includono un flusso di scheletro nel loro SDK, che permette ai programmatori di tracciare le articolazioni dell'utente in tempo reale.

[16] Alina Delia Călin Adriana Coroiu e Adriana Coroiu. "Interchangeability of kinect and orbbec sensors for gesture recognition". In: *2018 IEEE 14th international conference on intelligent computer communication and processing (ICCP)*. IEEE. 2018, pp. 309–315.

[17] Tibor Guzsvinecz, Veronika Szucs e Cecilia Sik-Lanyi. "Suitability of the Kinect sensor and Leap Motion controller—A literature review". In: *Sensors* 19.5 (2019), p. 1072.

# 5 Leap Motion Controller 2

In questo capitolo analizzeremo più nel dettaglio il dispositivo Leap Motion Controller 2, che verrà utilizzato nel capitolo 7 per l'implementazione pratica di un algoritmo di gesture recognition mediante reti neurali ricorrenti.

## 5.1 Introduzione

Il Leap Motion Controller 2 è un dispositivo rivoluzionario sviluppato da Ultraleap che trasforma le mani in controller naturali per l'interazione con il mondo digitale. Questo dispositivo offre un tracciamento preciso in 3D, monitorando 27 punti distinti della mano, inclusi ossa e giunture, anche quando non completamente visibili. Ciò consente un'interazione fluida e intuitiva con computer, software VR e giochi, sostituendo i tradizionali controller.

Figura 20: Leap Motion Controller 2

Il design compatto e minimalista lo rende adatto a qualsiasi ambiente di lavoro, mentre il potente SDK (Software Development Kit) di Ultraleap offre agli sviluppatori gli strumenti necessari per creare esperienze immersive e coinvolgenti. Utilizzando il Leap Motion Controller 2, gli utenti possono immergersi in ambienti virtuali, controllare personaggi e oggetti in giochi 3D, modellare oggetti virtuali con precisione, comunicare attraverso gesti non verbali e sperimentare simulazioni coinvolgenti per la formazione e l'istruzione.

Alcune specifiche tecniche riportate sul sito ufficiale[18]:

| **Parametro** | **Valore** |
|---|---|
| Alimentazione | 5VDC tramite USB, 500 mA |
| Connessione dati | USB 3.0 tramite connettore USB Type-C |
| Gamma di tracciamento e campo visivo | Profondità tra 10 cm (4 pollici) a 110 cm (43 pollici); campo visivo di 160° x 160° |
| Frequenza dei fotogrammi | Massimo 115 fps |
| Lunghezza d'onda operativa | 850 nm |
| Costruzione | Alluminio e vetro resistente ai graffi |
| Dimensioni | 84 mm x 20 mm x 12 mm |
| Peso | 29 g |

Tabella 1: Specifiche del Leap Motion Controller 2

## 5.2 Leap Motion SDK

Il Leap Motion SDK offre agli sviluppatori un'interfaccia intuitiva per integrare il tracciamento delle mani e dei gesti del Leap Motion Controller nelle loro applicazioni. Questo kit fornisce accesso in tempo reale ai dati di tracciamento, consentendo di monitorare con precisione la posizione, l'orientamento e i movimenti delle mani e delle dita, includendo funzionalità di riconoscimento di gesti predefiniti.

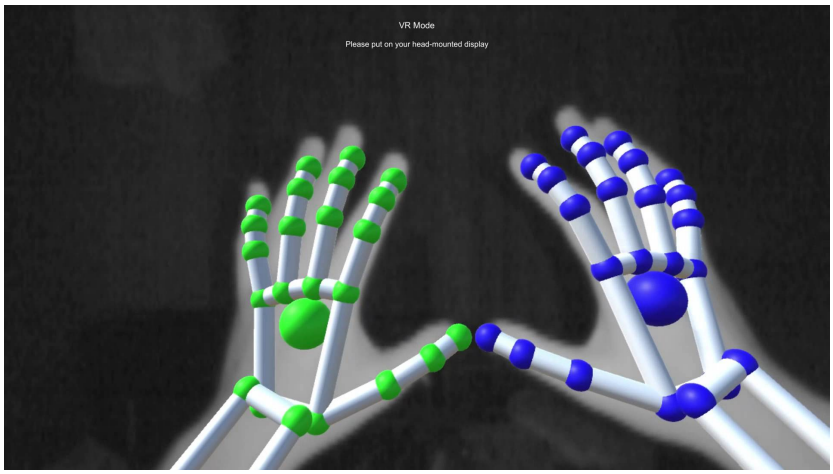



Figura 21: Interfaccia di acquisizione del Leap Motion Controller.

L'utilizzo del Leap Motion SDK consente la creazione di interazioni 3D intuitive all'interno di applicazioni, giochi e ambienti virtuali, eliminando la necessità di con-

[18]UltraLeap. *UltraLeap Official Website*. URL: `https://www.ultraleap.com/`.

troller tradizionali come mouse e tastiera, ed è disponibile per diverse piattaforme di sviluppo, garantendo una vasta compatibilità. Tra le applicazioni pratiche di questo SDK vi sono lo sviluppo di giochi coinvolgenti, la progettazione di interfacce utente innovative, la creazione di esperienze più realistiche e interattive in ambito di realtà virtuale e aumentata, lo sviluppo di strumenti di modellazione 3D più intuitivi, e la creazione di strumenti didattici interattivi.

I principali vantaggi del Leap Motion SDK includono un controllo intuitivo e naturale, una maggiore precisione e controllo nel tracciamento del movimento, la possibilità di esplorare nuove modalità di interazione e una vasta gamma di applicazioni possibili, che spaziano dai giochi all'istruzione.

## 5.3 API e developer tools

Il Leap Motion Controller 2 (LMC2) offre agli sviluppatori un'ampia gamma di strumenti e API[19] per integrare il tracciamento delle mani e dei gesti nelle proprie applicazioni in modo fluido e intuitivo. Il Leap Motion SDK offre plugin e integrazioni per piattaforme e motori di gioco popolari come Unity e Unreal Engine, semplificando notevolmente l'integrazione del tracciamento delle mani nelle applicazioni esistenti. Questi plugin forniscono un'API user-friendly e documentazione dettagliata, consentendo agli sviluppatori di iniziare rapidamente e di implementare facilmente il tracciamento delle mani nel loro progetto.

Con una documentazione completa sull'API e strumenti per lo sviluppo, come simulatori, debuggers ed emulatori, il Leap Motion SDK offre agli sviluppatori tutto il necessario per creare applicazioni innovative e coinvolgenti. La compatibilità multipiattaforma del SDK, che include Windows, macOS, Linux e Android, permette agli sviluppatori di creare esperienze che possono essere fruite su una vasta gamma di dispositivi, garantendo una maggiore accessibilità e flessibilità.

È presente inoltre LeapC, un'API nativa che offre agli sviluppatori un controllo diretto sul tracciamento delle mani e dei gesti, consentendo loro di creare applicazioni personalizzate e di massimizzare le prestazioni del Leap Motion Controller. LeapC è particolarmente utile per gli sviluppatori che necessitano di un controllo dettagliato e di basso livello sulle funzionalità del dispositivo, e offre una gamma completa di funzioni per l'interazione con il tracciamento delle mani, tra cui il controllo dei puntatori, la gestione degli eventi e la calibrazione del dispositivo.

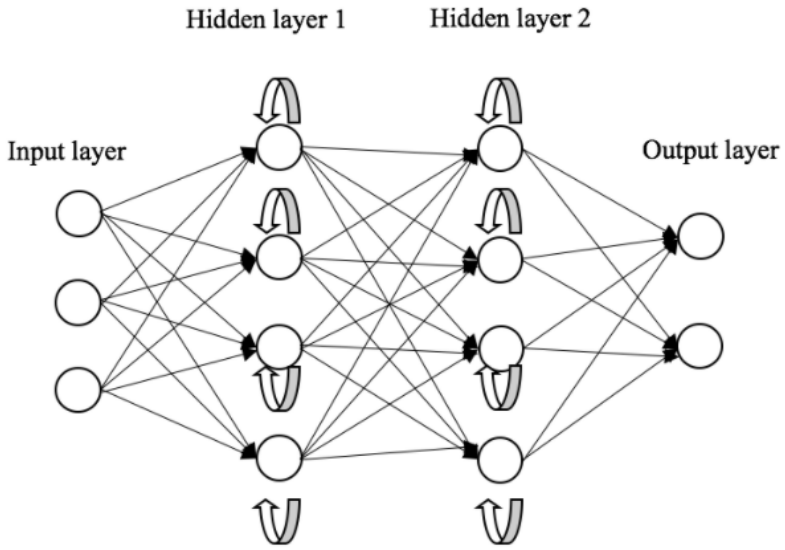
[19] UltraLeap. *Leap Concepts - Tracking API - UltraLeap Documentation.* [Online; accessed 26-April-2024]. 2024. URL: `https://docs.ultraleap.com/api-reference/tracking-api/leapc-guide/leap-concepts.html`.

# 6 Reti neurali ricorrenti

Una descrizione teorica più dettagliata delle RNN permetterà di gettare le basi per la comprensione della loro implementazione nel riconoscimento dei gesti dinamici presentati nel capitolo successivo.

## 6.1 Che cosa sono?

Le RNN[20] sono un particolare tipo di reti neurali adatte all'elaborazione di sequenze, conservando uno stato interno che contiene informazioni sugli elementi precedentemente osservati. Questo ciclo interno consente alla RNN di mantenere informazioni rilevanti nel tempo, consentendo la trasformazione di sequenze temporali complesse in un output significativo.




General Form of RNNs

Un aspetto cruciale è la struttura dell'input accettato dalla rete neurale, rappresentato come (batchsize, timesteps, features). Il batchsize indica il numero di sequenze presenti nel dataset, mentre timesteps rappresenta la lunghezza di ciascuna sequenza e features il numero di caratteristiche presenti in ciascun punto dati, ovvero una singola istanza all'interno della sequenza temporale.

[20]Francois Chollet. *Deep Learning with Python*. Manning Publications Co., 2018.

## 6.2 Neuroni ricorrenti

Le RNN utilizzano tre tipi principali di neuroni: SimpleRNN, GRU ed LSTM. Il SimpleRNN rappresenta la forma più basilare di neurone ricorrente. Tuttavia, presenta un difetto significativo noto come "scomparsa del gradiente", che si verifica quando i segnali più vecchi scompaiono gradualmente durante l'elaborazione, rendendo la rete difficile da addestrare. Per risolvere questo problema, sono stati sviluppati i neuroni GRU (Gated Recurrent Unit) ed LSTM (Long Short-Term Memory)[21]. Questi neuroni sono progettati per memorizzare informazioni a lungo termine e mitigare la scomparsa del gradiente, consentendo alle reti neurali di mantenere informazioni rilevanti su sequenze lunghe nel tempo.

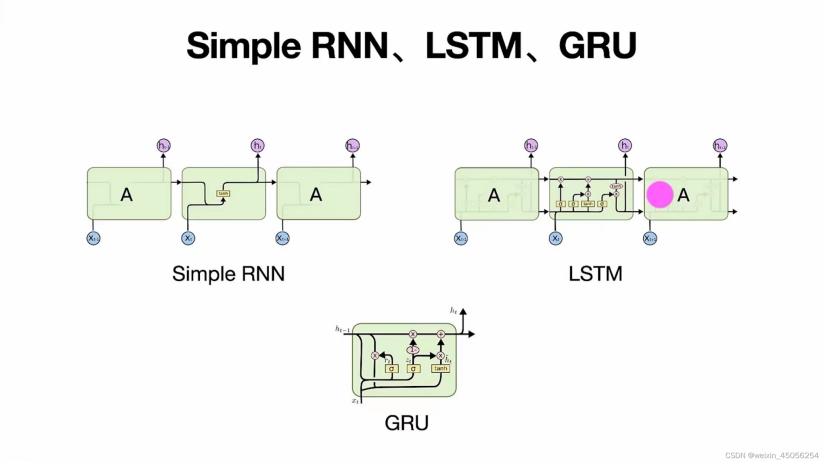



Figura 22: Tipi di neuroni per una RNN.

## 6.3 Dropout per la prevenzione dell'overfitting

L'overfitting è un fenomeno in cui il modello di rete neurale si adatta eccessivamente ai dati di addestramento, perdendo la capacità di generalizzare correttamente su nuovi dati.

Una delle tecniche più comuni per contrastare l'overfitting è il dropout, che consiste nell'azzerare in modo casuale alcuni neuroni durante l'addestramento, in modo da "rompere" le correlazioni spurie presenti nei dati di addestramento. Tuttavia, applicare il dropout prima di un layer ricorrente può risultare problematico. Questo perché il dropout, se applicato in modo casuale ad ogni iterazione temporale, può interferire con la capacità del layer ricorrente di imparare e memorizzare sequenze temporali. In altre parole, una maschera di dropout casuale potrebbe sovvertire il segnale temporale di errore, danneggiando il processo di apprendimento. Per ovviare a questo problema,

[21] Alessandro Bellini e Andrea Guidi. *Python e Machine Learning*. McGraw-Hill Education, 2022.

è possibile applicare una maschera di dropout costante rispetto al tempo, nota come maschera di dropout ricorrente, alle attivazioni interne del layer ricorrente. Questo significa che la stessa maschera di dropout viene applicata a tutte le iterazioni temporali, consentendo al layer ricorrente di apprendere e memorizzare correttamente le sequenze temporali senza interferenze.

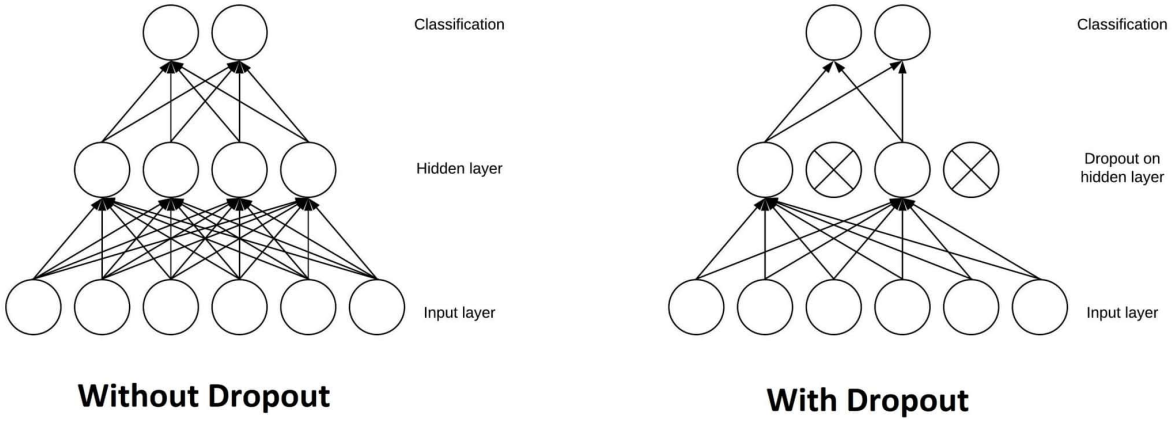



Figura 23: Rete con e senza dropout.

## 6.4 Reti bidirezionali

Una RNN bidirezionale è una variante comune delle reti neurali ricorrenti che, in certi compiti, può offrire prestazioni superiori rispetto a una RNN standard. È particolarmente diffusa nel campo del Natural Language Processing (NLP). Le RNN sono notoriamente sensibili all'ordine e alla sequenzialità dei dati: elaborano i timestep delle sequenze di input in base al loro ordine temporale e il rimescolamento o l'inversione di questi timestep può alterare radicalmente le rappresentazioni che la rete estrae dalla sequenza. Questa caratteristica le rende particolarmente adatte a problemi in cui l'ordine dei dati è rilevante.

La RNN bidirezionale sfrutta appunto questa sensibilità all'ordine tipica delle RNN. Essa consiste nell'utilizzare due RNN standard, ognuna delle quali elabora la sequenza di input in una direzione specifica: una procede in avanti, seguendo l'ordine cronologico della sequenza, mentre l'altra procede all'indietro, seguendo l'ordine inverso della sequenza. Le rappresentazioni generate da entrambe le RNN vengono poi combinate o concatenate, al fine di ottenere una rappresentazione complessiva della sequenza che incorpora informazioni sia dall'inizio alla fine che dalla fine all'inizio.

Grazie a questa doppia elaborazione della sequenza in direzioni opposte, la RNN bidirezionale è in grado di catturare pattern e relazioni che potrebbero non emergere in una RNN monodirezionale. Questo la rende estremamente efficace in problemi in cui sia importante considerare il contesto sia precedente che successivo, come nel caso dell'analisi del linguaggio naturale.

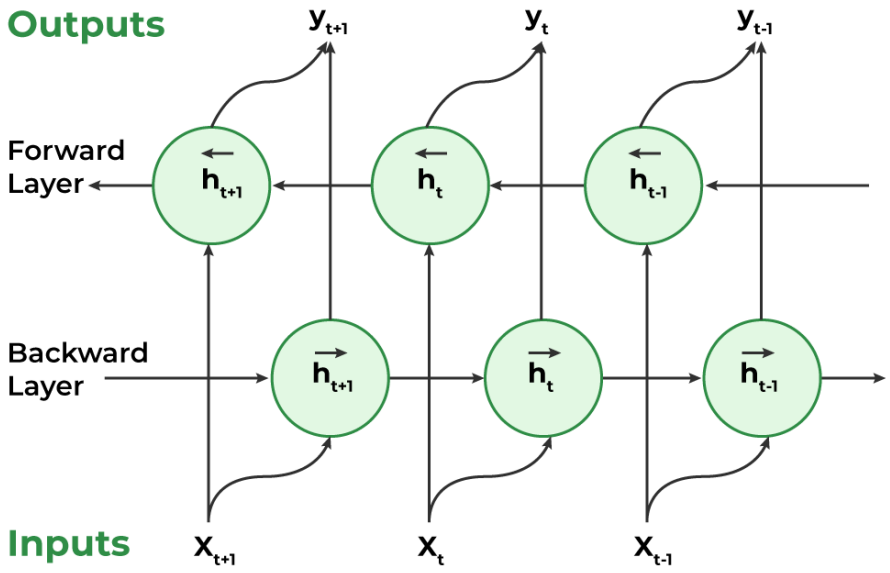



Figura 24: Struttura di una Bidirectional RNN.

# 7 Implementazione di una RNN per il riconoscimento di gesti dinamici

## 7.1 Introduzione

Considerando le discussioni finora affrontate, ci proponiamo di esplorare l'implementazione di un algoritmo di deep learning, in particolare una rete neurale ricorrente, per il riconoscimento di quattro gesti dinamici distinti:

- Finger wagging: un movimento laterale della mano, includendo il gesto tradizionale del "no" con il dito.
- Waving: un gesto di saluto caratterizzato da un movimento ondulatorio della mano.
- Air quotes: il gesto di "virgolette" che coinvolge l'indice e il medio.
- Zoom: un movimento di allontanamento o avvicinamento delle dita, comunemente associato all'azione di ingrandire o rimpicciolire.

Figura 25: Gesti acquisiti.

Per questo progetto, optiamo per la creazione di un dataset personalizzato, acquisito tramite il Leap Motion Controller 2, in collaborazione con il collega Gabriele Trovato. Questo dataset contiene sequenze temporali di gesti dinamici, catturando variazioni nel movimento delle mani durante l'esecuzione di ciascun gesto.

L'obiettivo principale è quello di addestrare una rete neurale ricorrente (RNN) per riconoscere e classificare in tempo reale i gesti dinamici eseguiti dagli utenti. Questo approccio ci consentirà di sviluppare un sistema interattivo e intuitivo che potrebbe essere utilizzato in una varietà di contesti, come interfacce utente gestuali per dispositivi mobili, controlli gestuali per applicazioni di realtà aumentata o addirittura nell'ambito dell'assistenza agli utenti con disabilità motorie.

## 7.2 Raccolta dei dati

Il primo passo cruciale per implementare con successo un algoritmo di intelligenza artificiale, come una rete neurale, è il suo addestramento, che richiede una considerevole quantità di dati. In collaborazione con il mio collega Gabriele Trovato, abbiamo raccolto dati da un campione di 100 individui, chiedendo loro di eseguire quattro gesti distinti,

mentre il LMC2, grazie ad un software sviluppato ad hoc, registra le coordinate del gesto dinamico, come meglio spiegato in seguito. L'età dei partecipanti varia dai 19 ai 60 anni; inoltre, poichè ogni soggetto esegue i quattro gesti, il dataset finale è composto da 400 campioni o sample.

Questi dati sono stati salvati in file .csv e rappresentano sequenze temporali di movimenti della mano. Ogni sequenza contiene valori di posizione, velocità e orientamento del palmo della mano, inclusi dati sulle quattro ossa per ciascun dito: metacarpo, distale, mediale e prossimale. Le informazioni includono anche il tipo di mano (destra o sinistra) e un identificativo univoco per ogni mano. Questo si traduce in un totale di 244 caratteristiche per ciascun dataframe, con ogni sequenza di dati di lunghezza variabile.

| **Feature** | **Descrizione** |
|---|---|
| hand_id | Identificatore unico della mano |
| hand_type | Tipo di mano (destra o sinistra) |
| palm_position | Posizione del palmo nello spazio 3D (x, y, z) |
| palm_velocity | Velocità del palmo nello spazio 3D (x, y, z) |
| palm_normal | Vettore normale al palmo (x, y, z) |
| palm_width | Larghezza del palmo |
| palm_direction | Direzione del palmo nello spazio 3D (x, y, z) |
| palm_orientation | Orientamento del palmo (quaternione: x, y, z, w) |
| *finger*_id | Identificatore unico per ciascun dito |
| *finger_bone*_joint_positions | Posizioni delle articolazioni per ogni dito (x, y, z) |
| *finger_bone*_joint_rotations | Rotazioni delle articolazioni per ogni dito (quaternione: x, y, z, w) |
| *finger_bone*_width | Larghezza di ciascun segmento del dito |

Tabella 2: Principali features per il tracciamento delle mani

Di seguito si riporta l'elenco completo di tutte le 244 features contenute in ciascun dataframe:

- hand_id
- hand_type
- palm_position_x
- palm_position_y
- palm_position_z
- palm_velocity_x
- palm_velocity_y
- palm_velocity_z

- palm_normal_x
- palm_normal_y
- palm_normal_z
- palm_width
- palm_direction_x
- palm_direction_y
- palm_direction_z
- palm_orientation_x
- palm_orientation_y
- palm_orientation_z
- palm_orientation_w
- thumb_id
- thumb_metacarpal_prev_joint_x
- thumb_metacarpal_prev_joint_y
- thumb_metacarpal_prev_joint_z
- thumb_metacarpal_next_joint_x
- thumb_metacarpal_next_joint_y
- thumb_metacarpal_next_joint_z
- thumb_metacarpal_width
- thumb_metacarpal_rotation_x
- thumb_metacarpal_rotation_y
- thumb_metacarpal_rotation_z
- thumb_metacarpal_rotation_w
- thumb_proximal_prev_joint_x
- thumb_proximal_prev_joint_y
- thumb_proximal_prev_joint_z
- thumb_proximal_next_joint_x
- thumb_proximal_next_joint_y
- thumb_proximal_next_joint_z
- thumb_proximal_width

- thumb_proximal_rotation_x
- thumb_proximal_rotation_y
- thumb_proximal_rotation_z
- thumb_proximal_rotation_w
- thumb_middle_prev_joint_x
- thumb_middle_prev_joint_y
- thumb_middle_prev_joint_z
- thumb_middle_next_joint_x
- thumb_middle_next_joint_y
- thumb_middle_next_joint_z
- thumb_middle_width
- thumb_middle_rotation_x
- thumb_middle_rotation_y
- thumb_middle_rotation_z
- thumb_middle_rotation_w
- thumb_distal_prev_joint_x
- thumb_distal_prev_joint_y
- thumb_distal_prev_joint_z
- thumb_distal_next_joint_x
- thumb_distal_next_joint_y
- thumb_distal_next_joint_z
- thumb_distal_width
- thumb_distal_rotation_x
- thumb_distal_rotation_y
- thumb_distal_rotation_z
- thumb_distal_rotation_w
- index_id
- index_metacarpal_prev_joint_x
- index_metacarpal_prev_joint_y
- index_metacarpal_prev_joint_z

- index_metacarpal_next_joint_x
- index_metacarpal_next_joint_y
- index_metacarpal_next_joint_z
- index_metacarpal_width
- index_metacarpal_rotation_x
- index_metacarpal_rotation_y
- index_metacarpal_rotation_z
- index_metacarpal_rotation_w
- index_proximal_prev_joint_x
- index_proximal_prev_joint_y
- index_proximal_prev_joint_z
- index_proximal_next_joint_x
- index_proximal_next_joint_y
- index_proximal_next_joint_z
- index_proximal_width
- index_proximal_rotation_x
- index_proximal_rotation_y
- index_proximal_rotation_z
- index_proximal_rotation_w
- index_middle_prev_joint_x
- index_middle_prev_joint_y
- index_middle_prev_joint_z
- index_middle_next_joint_x
- index_middle_next_joint_y
- index_middle_next_joint_z
- index_middle_width
- index_middle_rotation_x
- index_middle_rotation_y
- index_middle_rotation_z
- index_middle_rotation_w

- index_distal_prev_joint_x
- index_distal_prev_joint_y
- index_distal_prev_joint_z
- index_distal_next_joint_x
- index_distal_next_joint_y
- index_distal_next_joint_z
- index_distal_width
- index_distal_rotation_x
- index_distal_rotation_y
- index_distal_rotation_z
- index_distal_rotation_w
- middle_id
- middle_metacarpal_prev_joint_x
- middle_metacarpal_prev_joint_y
- middle_metacarpal_prev_joint_z
- middle_metacarpal_next_joint_x
- middle_metacarpal_next_joint_y
- middle_metacarpal_next_joint_z
- middle_metacarpal_width
- middle_metacarpal_rotation_x
- middle_metacarpal_rotation_y
- middle_metacarpal_rotation_z
- middle_metacarpal_rotation_w
- middle_proximal_prev_joint_x
- middle_proximal_prev_joint_y
- middle_proximal_prev_joint_z
- middle_proximal_next_joint_x
- middle_proximal_next_joint_y
- middle_proximal_next_joint_z
- middle_proximal_width

- middle_proximal_rotation_x
- middle_proximal_rotation_y
- middle_proximal_rotation_z
- middle_proximal_rotation_w
- middle_middle_prev_joint_x
- middle_middle_prev_joint_y
- middle_middle_prev_joint_z
- middle_middle_next_joint_x
- middle_middle_next_joint_y
- middle_middle_next_joint_z
- middle_middle_width
- middle_middle_rotation_x
- middle_middle_rotation_y
- middle_middle_rotation_z
- middle_middle_rotation_w
- middle_distal_prev_joint_x
- middle_distal_prev_joint_y
- middle_distal_prev_joint_z
- middle_distal_next_joint_x
- middle_distal_next_joint_y
- middle_distal_next_joint_z
- middle_distal_width
- middle_distal_rotation_x
- middle_distal_rotation_y
- middle_distal_rotation_z
- middle_distal_rotation_w
- ring_id
- ring_metacarpal_prev_joint_x
- ring_metacarpal_prev_joint_y
- ring_metacarpal_prev_joint_z

- ring_metacarpal_next_joint_x
- ring_metacarpal_next_joint_y
- ring_metacarpal_next_joint_z
- ring_metacarpal_width
- ring_metacarpal_rotation_x
- ring_metacarpal_rotation_y
- ring_metacarpal_rotation_z
- ring_metacarpal_rotation_w
- ring_proximal_prev_joint_x
- ring_proximal_prev_joint_y
- ring_proximal_prev_joint_z
- ring_proximal_next_joint_x
- ring_proximal_next_joint_y
- ring_proximal_next_joint_z
- ring_proximal_width
- ring_proximal_rotation_x
- ring_proximal_rotation_y
- ring_proximal_rotation_z
- ring_proximal_rotation_w
- ring_middle_prev_joint_x
- ring_middle_prev_joint_y
- ring_middle_prev_joint_z
- ring_middle_next_joint_x
- ring_middle_next_joint_y
- ring_middle_next_joint_z
- ring_middle_width
- ring_middle_rotation_x
- ring_middle_rotation_y
- ring_middle_rotation_z
- ring_middle_rotation_w

- ring_distal_prev_joint_x
- ring_distal_prev_joint_y
- ring_distal_prev_joint_z
- ring_distal_next_joint_x
- ring_distal_next_joint_y
- ring_distal_next_joint_z
- ring_distal_width
- ring_distal_rotation_x
- ring_distal_rotation_y
- ring_distal_rotation_z
- ring_distal_rotation_w
- pinky_id
- pinky_metacarpal_prev_joint_x
- pinky_metacarpal_prev_joint_y
- pinky_metacarpal_prev_joint_z
- pinky_metacarpal_next_joint_x
- pinky_metacarpal_next_joint_y
- pinky_metacarpal_next_joint_z
- pinky_metacarpal_width
- pinky_metacarpal_rotation_x
- pinky_metacarpal_rotation_y
- pinky_metacarpal_rotation_z
- pinky_metacarpal_rotation_w
- pinky_proximal_prev_joint_x
- pinky_proximal_prev_joint_y
- pinky_proximal_prev_joint_z
- pinky_proximal_next_joint_x
- pinky_proximal_next_joint_y
- pinky_proximal_next_joint_z
- pinky_proximal_width

- pinky_proximal_rotation_x
- pinky_proximal_rotation_y
- pinky_proximal_rotation_z
- pinky_proximal_rotation_w
- pinky_middle_prev_joint_x
- pinky_middle_prev_joint_y
- pinky_middle_prev_joint_z
- pinky_middle_next_joint_x
- pinky_middle_next_joint_y
- pinky_middle_next_joint_z
- pinky_middle_width
- pinky_middle_rotation_x
- pinky_middle_rotation_y
- pinky_middle_rotation_z
- pinky_middle_rotation_w
- pinky_distal_prev_joint_x
- pinky_distal_prev_joint_y
- pinky_distal_prev_joint_z
- pinky_distal_next_joint_x
- pinky_distal_next_joint_y
- pinky_distal_next_joint_z
- pinky_distal_width
- pinky_distal_rotation_x
- pinky_distal_rotation_y
- pinky_distal_rotation_z
- pinky_distal_rotation_w

Il sistema di coordinate utilizzato[22] è destrorso e segue un sistema cartesiano, con l'origine posizionata al centro della telecamera del Leap Motion. Gli assi x e z giacciono nel piano dei sensori della telecamera, con l'asse x che corre lungo la linea di base della telecamera e l'asse y che è verticale, con valori positivi che aumentano verso l'alto. L'asse z ha valori positivi che aumentano verso l'utente. Ogni frame rappresenta

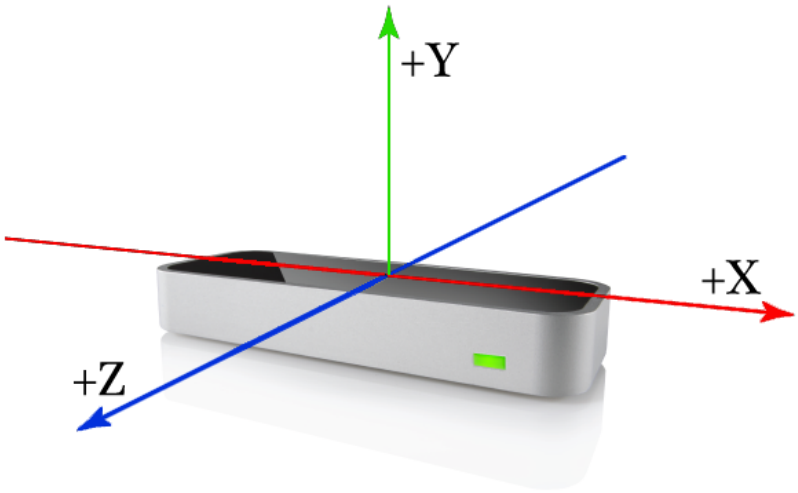



Figura 26: Sistema di assi di coordinate del Leap Motion Controller 2.

la collezione di dati di tracciamento derivati da una singola coppia stereo di immagini. Il software di tracciamento analizza ogni coppia di immagini stereo dai sensori del dispositivo per rilevare la presenza, la postura e il movimento delle mani. Il software di tracciamento delle mani si basa su modelli interni della mano umana per adattarsi alla scena osservata. Se parti di una mano sono occluse o non visibili, il software stima i dati in base a ciò che è visibile, alle osservazioni passate e ai vincoli del modello.

[22]UltraLeap. *Leap Concepts - Tracking API - UltraLeap Documentation.* [Online; accessed 26-April-2024]. 2024. URL: `https://docs.ultraleap.com/api-reference/tracking-api/leapc-guide/leap-concepts.html`.

Il quaternione è una struttura utilizzata per rappresentare rotazioni tridimensionali. Inoltre, il sistema di tracciamento delle mani Ultraleap utilizza il quaternione per rappresentare l'orientamento dei vari elementi, come le dita e l'avambraccio.

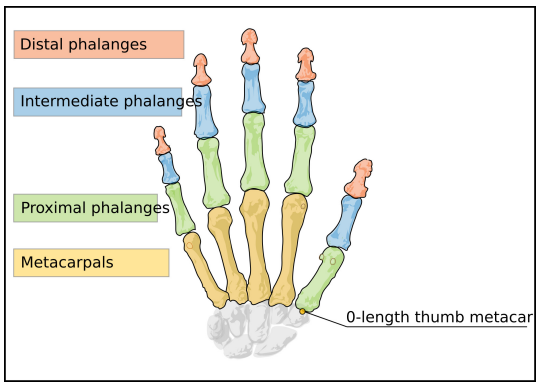



Figura 27: Struttura scheletrica della mano.

Per l'acquisizione e il salvataggio dei dati, è stata utilizzata repository creata da Harry Mills e resa disponibile gratuitamente su GitHub[23]. Questa repository fornisce dei bindings Python open-source per l'API Gemini LeapC, consentendo agli sviluppatori di integrare la tecnologia di tracciamento delle mani di Ultraleap nei propri progetti Python. Nel nostro caso, è stato modificato lo script tracking_event_example.py presente nella repository, in modo che, durante il processo di acquisizione dei dati, questi venissero immagazzinati in un dizionario; da quest'ultimo viene generato il dataframe pandas e salvato in formato csv nella cartella inerente a seconda del tipo di gesto catturato.

[23]Ultraleap. *Gemini LeapC Python Bindings*. `https://github.com/ultraleap/leapc-python-bindings`. 2024.

Al fine di rendere più user friendly il processo di acquisizione del programma, è stata implementata un'interfaccia grafica per la selezione veloce del gesto dinamico da catturare grazie al modulo Python Flet: l'utente è in grado così di selezionare in anticipo il tipo di gesto che verrà catturato successivamente, far partire l'acquisizione dei dati attraverso il tasto record, e fermare il recording mediante l'opportuno tasto di stop. Il gesto selezionato in questo caso verrà utilizzato dal programma per smistare il dataframe acquisito nella cartella corretta per il salvataggio.

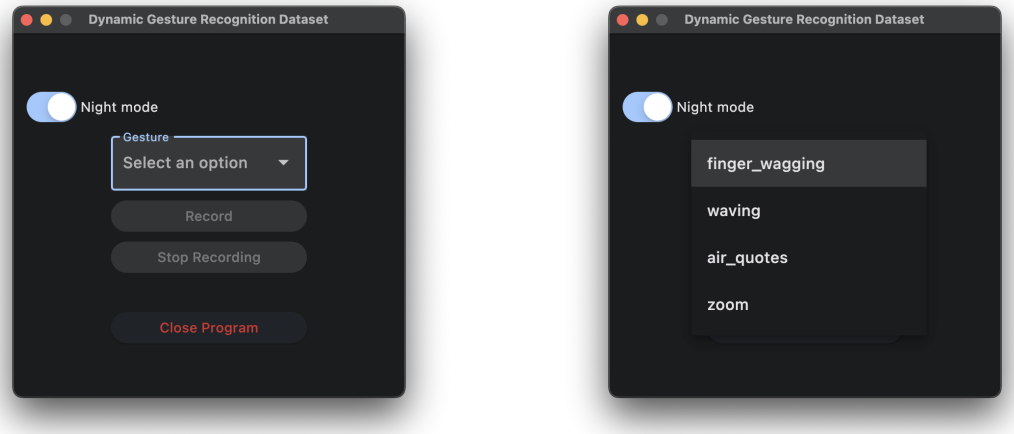



Figura 28: Interfaccia utente per l'acquisizione dei dati implementata con il modulo Flet.

A seguire lo script utilizzato per la cattura e l'immagazzinamento dei dati, presente nella sua versione non modificata su GitHub[24]:

```
"""
Prints the palm position of each hand, every frame. When
    a device is connected we set the tracking mode to
   desktop and then generate logs for every tracking
   frame received. The events of creating a connection
   to the server and a device being plugged in also
   generate logs.
"""

import leap
import os
import sys
import time
```

[24]Ultraleap. *Gemini LeapC Python Bindings*. `https://github.com/ultraleap/leapc-python-bindings`. 2024.

```
import flet as ft
import pandas as pd
from feature_structures import FeatureStruct
from flet import ElevatedButton, Row, Column, Dropdown
from flet_core.control_event import ControlEvent

class MyListener(leap.Listener):
    def __init__(self, gesture):
        self._gesture = gesture
        path = os.path.join(os.getcwd(), f'
           dynamic_gesture_recognition_dataset/{gesture}
           ')
        self._file_name = f'{len(os.listdir(path))}.csv'

    def on_connection_event(self, event):
        print("Connected")

    def on_device_event(self, event):
        try:
            with event.device.open():
                info = event.device.get_info()
        except leap.LeapCannotOpenDeviceError:
            info = event.device.get_info()

        print(f"Found device {info.serial}")

    def on_tracking_event(self, event):
        tmp = FeatureStruct()
        print(f"Frame {event.tracking_frame_id} with {
           len(event.hands)} hands.")
        if len(event.hands) != 0:
            for hand in event.hands:
                tmp.add_feature(hand)
            dct = tmp.return_dict()
            df = pd.DataFrame(dct)
            subfolder = '
               dynamic_gesture_recognition_dataset'
            save_csv_in_subfolder(df, subfolder, self.
               _gesture, self._file_name)

def save_csv_in_subfolder(df, subfolder, gesture,
   filename):
    path = os.path.join(os.getcwd(), f'{subfolder}/{
       gesture}')
    if filename not in os.listdir(path):
        df.to_csv(f'{path}/{filename}', index=False,
```

```
            mode='w')
    else:
        df.to_csv(f'{path}/{filename}', index=False,
            mode='a', header=False)

def main(page: ft.Page):
    page.title = "Dynamic Gesture Recognition Dataset"
    page.vertical_alignment = ft.MainAxisAlignment.
        CENTER
    page.theme_mode = ft.ThemeMode.DARK
    page.window_width = 400
    page.window_height = 400
    page.window_resizable = False

    gesture_text = ''
    flag = shutdown = stop = False

    def record(e: ControlEvent) -> None:
        nonlocal gesture_text, flag, stop
        gesture_text = dropdown_gesture.value
        flag, stop = True, False
        button_stop.disabled, button_record.disabled =
            False, True
        page.update()

    def stop_record(e: ControlEvent) -> None:
        nonlocal stop
        button_record.disabled, button_stop.disabled =
            False, True
        stop = True
        page.update()

    def change_mode(e: ControlEvent) -> None:
        if night_mode_switch.value:
            page.theme_mode = ft.ThemeMode.DARK
        else:
            page.theme_mode = ft.ThemeMode.LIGHT
        page.update()

    def close(e: ControlEvent) -> None:
        nonlocal shutdown, stop
        shutdown = stop = True
        page.window_close()

    night_mode_switch = ft.CupertinoSwitch(label="Night
        mode",
```

```
                                        value=True,
                                        on_change=
                                           change_mode
                                           )

    dropdown_gesture = ft.Dropdown(width=200,
                                   label='Gesture',
                                   hint_text='Select an
                                      option',
                                   options=[
                                       ft.dropdown.
                                          Option('
                                          finger_wagging
                                          '),
                                       ft.dropdown.
                                          Option('waving
                                          '),
                                       ft.dropdown.
                                          Option('
                                          air_quotes'),
                                       ft.dropdown.
                                          Option('zoom')
                                   ],
                                   autofocus=True,
                                   )
    button_record = ElevatedButton(text='Record',
                                   width=200,
                                   disabled=True,
                                   on_click=record)
    button_stop = ElevatedButton(text='Stop Recording',
                                 width=200,
                                 disabled=True,
                                 on_click=stop_record,
                                 color=ft.colors.RED_700
                                    )
    button_close = ElevatedButton(text='Close Program',
                                  width=200,
                                  on_click=close,
                                  color=ft.colors.
                                     RED_700)

    def validate_record(e: ControlEvent) -> None:
        if dropdown_gesture.value:
            button_record.disabled = False
        page.update()
```

```
dropdown_gesture.on_change = validate_record

page.add(
    Row(
        controls=[night_mode_switch],
        alignment=ft.MainAxisAlignment.START
    ),
    Row(
        controls=[
            Column(
                [dropdown_gesture,
                 button_record,
                 button_stop],
            )
        ],
        alignment=ft.MainAxisAlignment.CENTER
    ),
    Row(
        controls=[],
        alignment=ft.MainAxisAlignment.CENTER
    ),
    Row(
        controls=[],
        alignment=ft.MainAxisAlignment.CENTER
    ),
    Row(
        controls=[],
        alignment=ft.MainAxisAlignment.CENTER
    ),
    Row(
        controls=[button_close],
        alignment=ft.MainAxisAlignment.CENTER
    )
)

while True:
    if flag:
        my_listener = MyListener(gesture_text)

        connection = leap.Connection()
        connection.add_listener(my_listener)

        running = True

        with connection.open():
            connection.set_tracking_mode(leap.
```

```
                    TrackingMode.Desktop)
                while running:
                    time.sleep(1)
                    if stop:
                        break
            if shutdown:
                sys.exit()
            flag = False

if __name__ == "__main__":
    ft.app(main)
```

## 7.3 Pulizia e normalizzazione dei dati

Una volta ottenuto il dataset, il passo successivo consiste nella scrittura dello script Python per la nostra RNN, utilizzando Google Colab come ambiente di sviluppo. Il primo compito da affrontare è recuperare i dataframe dai file CSV e caricarli nel nostro programma. È stata creata una funzione apposita per questa operazione:

```
def _data_reader():
    """
    Read data from CSV files and split into train and
       test sets.

    Returns:
    - tmp1: List of training dataframes
    - tmp2: List of testing dataframes
    - target1: List of training labels
    - target2: List of testing labels
    """
    path = '/drive/MyDrive/
       dynamic_gesture_recognition_dataset'
    progress_bar = tqdm(total=400, desc='Loading data',
       leave=True)
    tmp1, tmp2, target1, target2 = [], [], [], []
    path = os.getcwd() + '/' + path
    for type_ in os.listdir(path):
        type_path = os.path.join(path, type_)
        for gesture in os.listdir(type_path):
            gesture_path = os.path.join(type_path,
               gesture)
            for csv in os.listdir(gesture_path):
                if csv.endswith('.csv'):
                    if type_ == 'train':
                        tmp1.append(pd.read_csv(os.path.
                           join(gesture_path, csv)))
```

```
                        target1.append(gesture)
                    elif type_ == 'test':
                        tmp2.append(pd.read_csv(os.path.
                            join(gesture_path, csv)))
                        target2.append(gesture)
                    progress_bar.update(1)

    return tmp1, tmp2, target1, target2
```

Grazie a questa funzione, otteniamo direttamente i dati di train e test, così come i target di training e test, già divisi e pronti ad essere puliti e preprocessati.

A questo punto, i dati risultano ancora grezzi ed è necessario pulirli per rimuovere le feature non rilevanti o non utili alla nostra rete neurale. A tale scopo, è stata definita la seguente funzione:

```
def _leap_data_cleaner(df_list):
    """
    Clean the Leap Motion dataframes.

    Args:
    - df_list (list): List of dataframes to clean.

    Returns:
    - Cleaned list of dataframes.
    """
    progress_bar = tqdm(total=len(df_list), desc='
        Cleaning data', leave=True)
    for k, df_tmp in enumerate(df_list):
        # Remove duplicate columns
        df_tmp = df_tmp.T.drop_duplicates().T
        # Remove columns with the same value for each
            row
        same_value_cols = [col for col in df_tmp.columns
                           if all(df_tmp[col] == df_tmp[
                               col][0])]
        df_tmp = df_tmp.drop(columns=same_value_cols)
        # Remove 'hand_id' and 'hand_type' columns
        if 'hand_id' in df_tmp.columns:
            df_tmp = df_tmp.drop(columns=['hand_id'])
        if 'hand_type' in df_tmp.columns:
            df_tmp = df_tmp.drop(columns=['hand_type'])
        # Fill NaN values with backward fill
        nan_values = df_tmp.isna().sum().sum()
        if nan_values > 0:
            df_tmp = df_tmp.bfill(axis=0)
        df_list[k] = df_tmp
```

```
            progress_bar.update(1)

        return df_list
```

Questa funzione accetta una lista di dataframe e, per ciascuno di essi, esegue una pulizia in tempo reale delle colonne. Durante questa pulizia, vengono rimosse le colonne duplicate o le colonne i cui valori sono identici su tutte le righe. Questo processo di verifica e pulizia viene effettuato dinamicamente ad ogni iterazione, poiché le colonne da rimuovere non sono conosciute in anticipo e vengono identificate solo al momento dell'analisi dei valori presenti nel dataframe. Inoltre, vengono eliminate le colonne 'hand_id' e 'hand_type', che non sono rilevanti per l'addestramento della rete. Infine, viene controllata la presenza di eventuali valori NaN (Not a Number) e, se presenti, vengono sostituiti con il valore immediatamente precedente. È importante notare che questo è solo un precauzione, poiché il dataset è stato già verificato alla fine della fase di raccolta dati e non presenta valori NaN.

Prima di dare in pasto i dati alla nostra rete neurale, dobbiamo prepararli adeguatamente. Questo processo, chiamato preprocessing, è fondamentale per assicurarci che la nostra rete possa trarre il massimo vantaggio dai dati disponibili. A seguire lo script:

```
def _data_preprocessing(df_train_list, df_test_list,
    y_train, y_test):
    """
    Preprocess the data and labels.

    Args:
    - df_train_list (list): List of training dataframes.
    - df_test_list (list): List of testing dataframes.
    - y_train (list): List of training labels.
    - y_test (list): List of testing labels.

    Returns:
    - Preprocessed training and testing data and labels.
    """
    # Map labels to numeric values
    cl = {'air_quotes':0, 'finger_wagging':1, 'waving'
        :2, 'zoom':3}
    for i in range(len(y_train)):
        y_train[i] = cl[y_train[i]]
    for i in range(len(y_test)):
        y_test[i] = cl[y_test[i]]

    # Pad sequences to have equal length
    max_len_train = max([len(df) for df in df_train_list
        ])
```

```
df_train_list = pad_sequences(df_train_list, maxlen=
    max_len_train,
                                padding='post',
                                   truncating='post')
df_test_list = pad_sequences(df_test_list, maxlen=
    max_len_train,
                               padding='post',
                                  truncating='post')

# Scale data using MinMaxScaler
scale_ = MinMaxScaler(feature_range=(-1, 1))
X_train_list, X_test_list = [], []
scale_.fit(df_train_list[0])
for df, label in zip(df_train_list, y_train):
    X_train = scale_.transform(df)
    X_train = X_train.reshape(-1, X_train.shape[0],
       X_train.shape[1])
    X_train_list.append(X_train)
for df, label in zip(df_test_list, y_test):
    X_test = scale_.transform(df)
    X_test = X_test.reshape(-1, X_test.shape[0],
       X_test.shape[1])
    X_test_list.append(X_test)

# Shuffle data and labels
random.seed(1)
random.shuffle(X_train_list)
X_train = np.concatenate(X_train_list)
random.seed(1)
random.shuffle(y_train)
y_train = np.array(y_train)
random.seed(1)
random.shuffle(X_test_list)
X_test = np.concatenate(X_test_list)
random.seed(1)
random.shuffle(y_test)
y_test = np.array(y_test)

return X_train, X_test, y_train, y_test
```

Per prima cosa, convertiamo le etichette delle nostre categorie in valori numerici. Questo passaggio semplifica il lavoro della rete, consentendole di comprendere meglio le relazioni tra le diverse classi. Successivamente, ci assicuriamo che tutte le sequenze di dati abbiano la stessa lunghezza. Questo è importante perché molte reti neurali, inclusa la nostra, richiedono input di lunghezza uniforme. Quindi, determiniamo la lunghezza

massima tra tutte le sequenze di addestramento e la utilizziamo per "padding" (riempire) o "troncamento" delle altre sequenze. Questo garantisce che tutte abbiano la stessa dimensione. Dopo aver uniformato le lunghezze, normalizziamo i dati utilizzando la tecnica MinMaxScale. Questo passaggio è utile per assicurarci che le varie caratteristiche dei nostri dati abbiano valori comparabili all'interno di un range comune, compreso tra -1 e 1. Infine, per evitare che la rete impari l'ordine dei dati durante l'addestramento, mescoliamo casualmente sia i dati che le relative etichette. Questo aiuta a rendere l'addestramento più robusto e a prevenire eventuali bias dovuti all'ordine dei dati. Una volta completato il preprocessing, i dati sono pronti per essere utilizzati per addestrare la nostra rete neurale.

## 7.4 Creazione del modello di rete

Per affrontare la natura sequenziale dei nostri dati, abbiamo adottato un modello di rete neurale ricorrente (RNN) bidirezionale con layer GRU di tipo Many-to-One. Questo tipo di architettura consente alla rete di analizzare le sequenze temporali sia in avanti che all'indietro, permettendo una comprensione più profonda dei dati e migliorando la capacità di riconoscimento dei gesti. Per la costruzione della nostra rete, abbiamo utilizzato i moduli Tensorflow e Keras, che forniscono un'implementazione intuitiva e potente delle reti neurali.

```
def _build_model():
    """
    Build and compile the bidirectional GRU model.

    Returns:
    - model (tensorflow.keras.Model): Compiled
       bidirectional GRU model.
    """
    model = Sequential()
    model.add(layers.Bidirectional(layers.GRU(32,
                                              input_shape
                                                 =(
                                                 X_train
                                                 .shape
                                                 [1],
                                                 X_train
                                                 .shape
                                                 [2]),
                                              return_sequences
                                                 =False,
                                              dropout
                                                 =0.2,
                                              recurrent_dropout
                                                 =0.2)))
```

```
    model.add(layers.Dense(32,
                           activation='relu'))
    model.add(layers.Dense(4,
                           activation='softmax'))

    model.compile(optimizer='nadam',
                  loss='sparse_categorical_crossentropy'
                      ,
                  metrics=['accuracy'])

    return model
```

Il cuore del modello è costituito da un layer di 32 neuroni GRU bidirezionali. Successivamente, abbiamo aggiunto un layer di 32 neuroni densi, che servono a introdurre non linearità nel sistema. Questo layer è completamente connesso al precedente, garantendo che ogni neurone riceva input da tutti i neuroni del layer GRU. Infine, il layer di output è composto da 4 neuroni densi, uno per ciascuna classe da predire. Si utilizza la funzione di attivazione softmax per ottenere una distribuzione di probabilità sulle diverse classi, consentendo alla rete di effettuare predizioni.

| **Layer (type)** | **Output Shape** |
|---|---|
| Bidirectional | (None, 32) |
| Dense | (None, 32) |
| Dense | (None, 4) |

Tabella 3: Sommario dell'architettura del modello.

Per quanto riguarda l'ottimizzazione della funzione di costo, la scelta è ricaduta sull'algoritmo Nadam. Nadam combina i vantaggi di Adam, un metodo di ottimizzazione che adatta automaticamente il tasso di apprendimento durante l'addestramento, con il Nesterov Momentum, che accelera la convergenza del modello. La funzione di costo utilizzata è la sparse categorical crossentropy, che è una scelta comune quando si lavora con problemi di classificazione. Questa funzione di loss calcola l'entropia tra le classi, permettendo al modello di apprendere le relazioni tra di esse durante l'addestramento. Infine, valutiamo le prestazioni del modello utilizzando l'accuratezza come metrica di valutazione, che ci fornisce una misura della percentuale di predizioni corrette rispetto al totale.

Questa configurazione è stata scelta dopo numerose prove di ottimizzazione degli iperparametri, durante le quali sono state variate il numero di layer, il numero di neuroni per layer, le funzioni di attivazione e gli ottimizzatori, in quanto si è dimostrata quella con i risultati migliori.

Qui è stata definita la funzione per l'addestramento del modello. Abbiamo incluso una tecnica chiamata "early stopping", che interrompe l'addestramento se non si

verifica alcun miglioramento nelle prestazioni per un certo numero di epoche consecutive e ritorna i pesi dei neuroni che hanno ottenuto le performance migliori in termini di validation accuracy. Questo ci aiuta a evitare il sovraddestramento e a risparmiare tempo. Il modello viene addestrato per un massimo di 1000 epoche, con i dati di addestramento divisi in batch da 64 dataframe ciascuno. Utilizziamo anche i dati di validazione per monitorare le prestazioni del modello durante l'addestramento e per applicare l'early stopping.

```
def _training(X_train, y_train):
    """
    Train the model.

    Args:
    - X_train (numpy.ndarray): Training data.
    - y_train (numpy.ndarray): Training labels.

    Returns:
    - Training history.
    """
    # Train the model
    early_stopping_callback = EarlyStopping(
        monitor='val_accuracy',
        patience=100,
        restore_best_weights=True
    )

    history = model.fit(X_train, y_train,
                        epochs=1000,
                        batch_size=64,
                        validation_data=(X_test, y_test)
                            ,
                        callbacks=[
                           early_stopping_callback])

    return (history.history['loss'], history.history['
       val_loss'],
            history.history['accuracy'], history.history
               ['val_accuracy'])
```

In particolare, la misura delle performance del modello è stata effettuata con un 80% di dati utilizzati per il training (320 dataframe) ed il restante 20% per effettuarne il test (80 dataframe).

Non è stato utilizzato un validation split perché il numero di campioni raccolti era limitato. In una situazione con pochi dati disponibili, suddividere ulteriormente il dataset per la validazione avrebbe ridotto ulteriormente la quantità di dati utilizzabili

per l'addestramento del modello, compromettendone l'efficacia e la capacità di generalizzare. Perciò, si è deciso di utilizzare l'intera parte di training per l'addestramento, al fine di massimizzare l'uso delle informazioni disponibili.

Con questa funzione, il modello viene addestrato e e viene restituita la storia dell'addestramento, che include le perdite e le metriche di accuratezza sia per i dati di addestramento che per quelli di validazione.

## 7.5 Analisi delle performance

I risultati dell'addestramento si sono rivelati molto promettenti, soprattutto grazie all'ottimo utilizzo dell'ottimizzatore Nadam per un totale di circa 450 epoche. Questa scelta ha dimostrato di essere azzeccata e ha portato a risultati soddisfacenti.

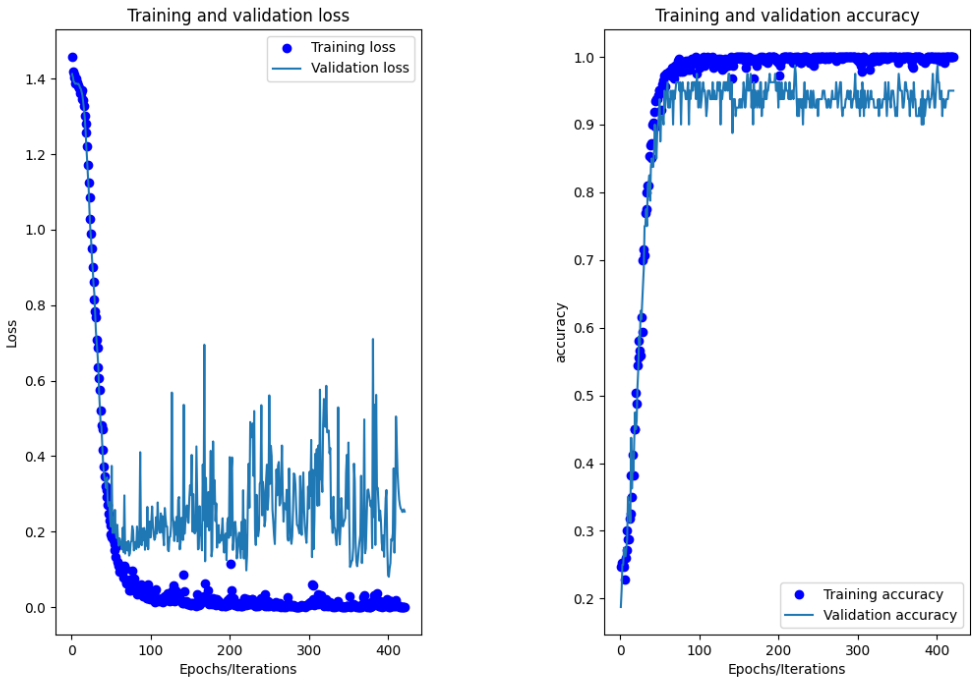



Figura 29: Andamento delle performance durante il training.

Dopo aver completato l'addestramento, è stata condotta un'analisi dettagliata delle performance sul test set. La rete è stata in grado di classificare correttamente ben 79 su 80 nuovi gesti presentati, che è un buon risultato per una implementazione sperimentale di questo tipo.

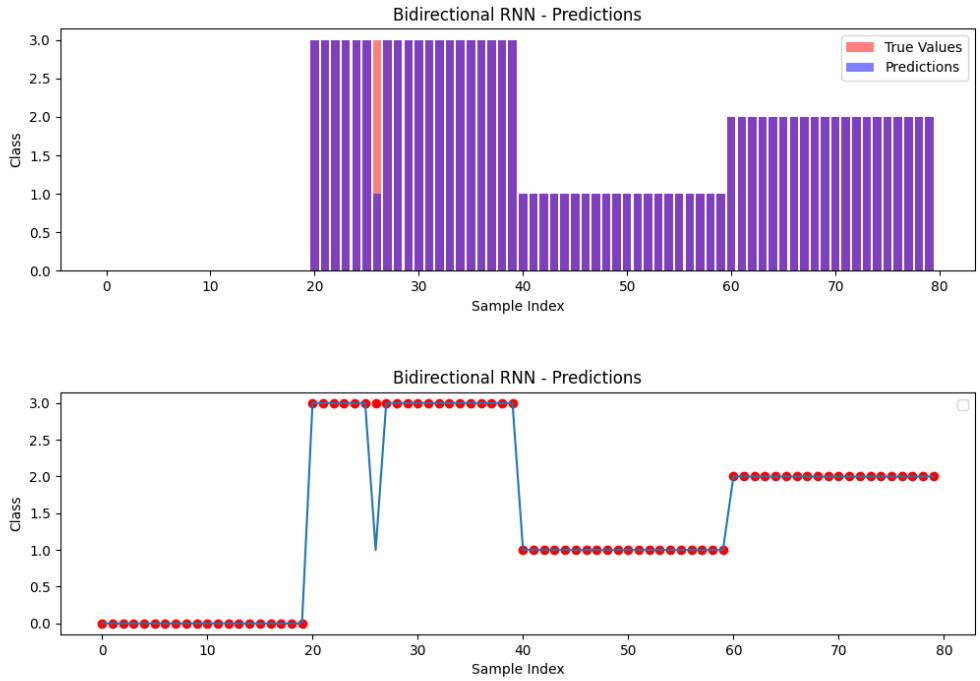



Figura 30: Valori reali e predizioni.

Come si può vedere nella Figura 30, abbiamo confrontato le predizioni del modello con i valori reali. Questo confronto chiarisce visivamente l'efficacia del modello nell'interpretare correttamente i dati di input.

La matrice di confusione è uno strumento fondamentale per valutare le prestazioni dei nostri modelli di classificazione. In pratica, assegna le classi reali degli input alle righe e le classi predette dai modelli alle colonne. Ogni cella della matrice registra il numero di campioni che appartengono contemporaneamente alla classe reale (riga) e alla classe predetta (colonna).

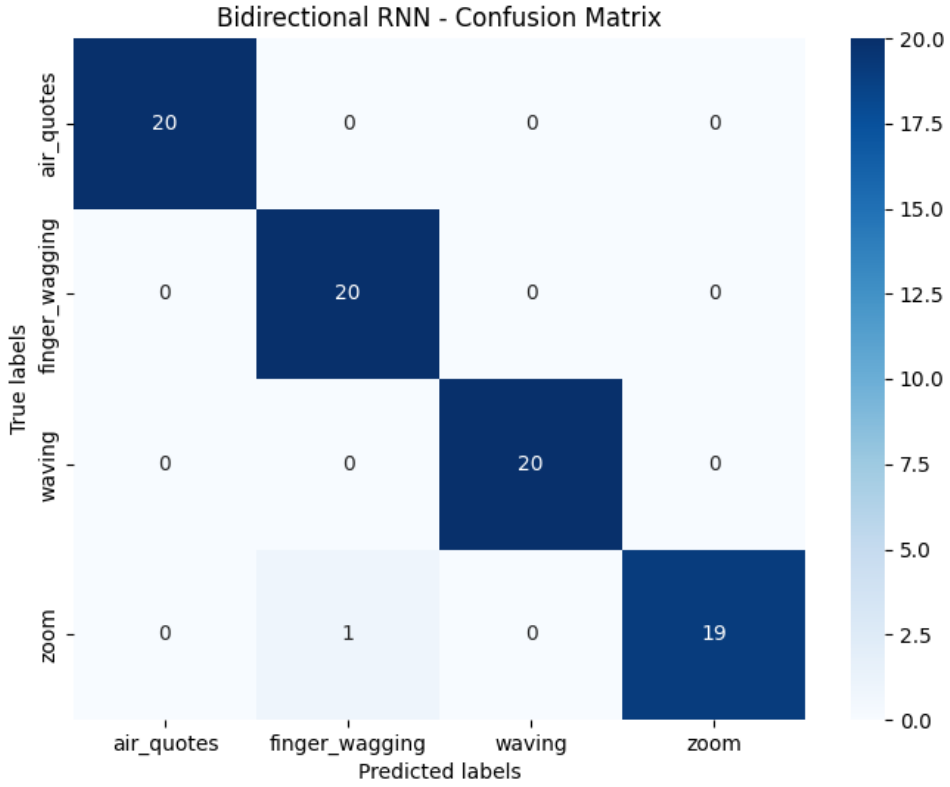



Figura 31: Matrice di confusione della rete.

In quest matrice di confusione, possiamo notare solo un errore di predizione da parte del modello: la rete ha scambiato uno "zoom" per un "finger wagging". Ciononostante, su un totale di 80 campioni, 79 sono stati classificati correttamente. Sono stati effettuati diversi run dell'algoritmo ma non si è riusciti comunque a replicare le performance di questo tentativo in particolare.

In termini numerici, il modello ha raggiunto un'accuratezza di quasi il 99% sul test set, con una loss di soli 0.104. In figura 32 è rappresentato l'istogramma dell'accuratezza, che evidenzia ulteriormente la coerenza e l'affidabilità delle previsioni del modello.

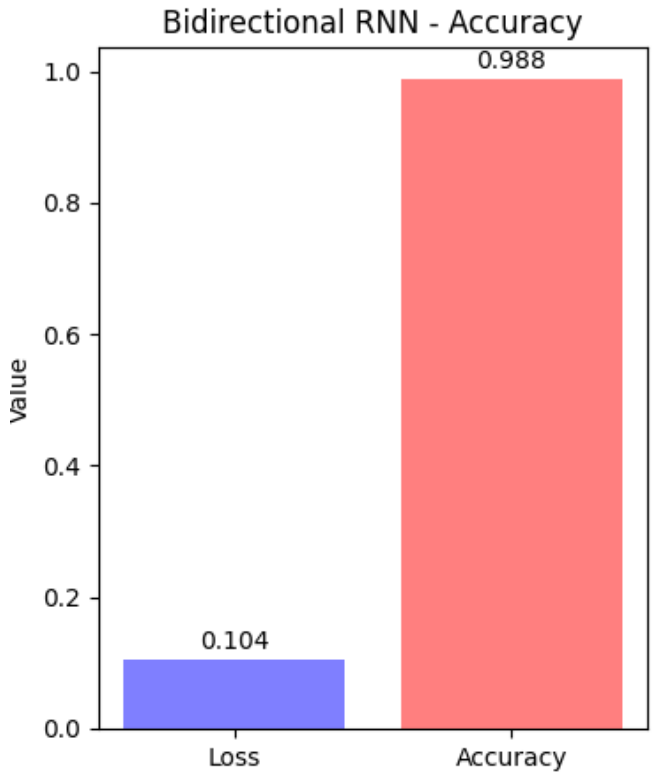



Figura 32: Loss e accuracy sul test set.

È stata inclusa una heatmap dei pesi nel primo layer GRU della nostra RNN. Questa visualizzazione a colori mette in risalto le regioni dell'input che hanno maggior peso nel processo decisionale del modello. I colori più chiari indicano valori più alti dei pesi, mentre i colori più scuri indicano valori più bassi.

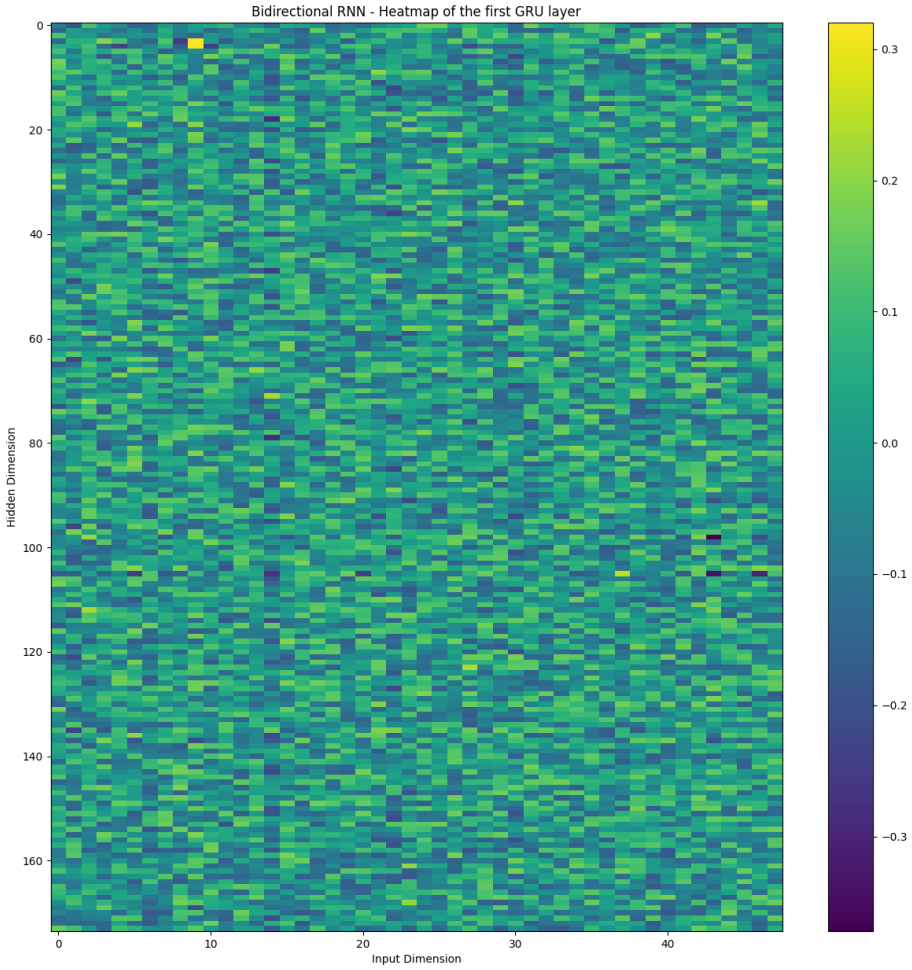



Figura 33: Heatmap dei pesi del layer GRU.

Questa analisi è fondamentale per migliorare le prestazioni del nostro modello. Se notiamo che parti dell'input che dovrebbero essere rilevanti non ricevono un peso significativo, potremmo dover rivedere l'architettura del modello o il processo di addestramento.

Infine, abbiamo a disposizione la ROC curve del nostro modello, che consente di valutarne le prestazioni analizzando i tassi di True Positive Rate (TPR) e False Positive Rate (FPR). Questi due tassi sono cruciali per comprendere quanto bene il nostro modello distingue tra le classi positive e negative.

Grazie alle buone prestazioni del modello, le curve di classe sono praticamente indistinguibili ad occhio. Questo perché tendono ad avvicinarsi il più possibile al valore 1, che rappresenta l'ideale per un classificatore perfetto. In altre parole, il modello ha dimostrato una notevole capacità di discriminazione tra le classi, con un TPR elevato e un FPR basso. Inoltre, è degno di nota che le AUC (Area Under Curve) di tutte le quattro classi siano pari ad 1, il che indica una perfetta capacità di separazione delle classi. La ROC curve rappresentata in figura 34 offre una visualizzazione chiara di queste prestazioni, mostrandoci quanto vicino il nostro modello sia al punto ideale in alto a sinistra del grafico.

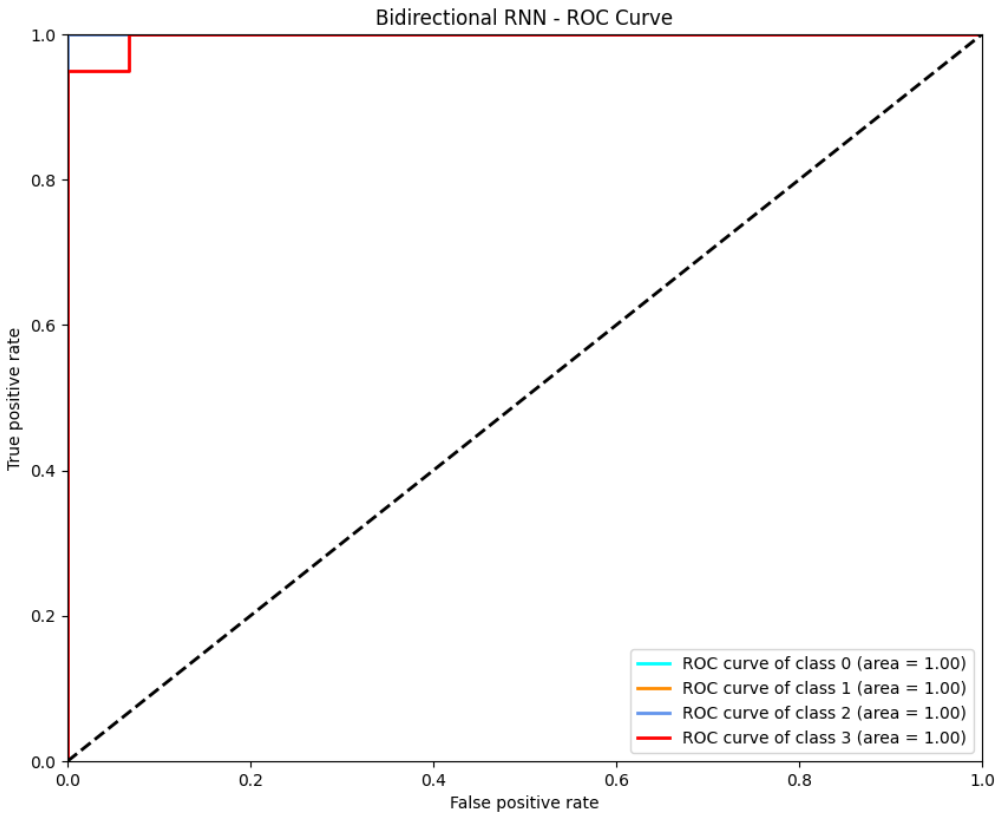



Figura 34: ROC curve del modello

## 7.6 Ingegnerizzazione delle feature

È stato sperimentato un nuovo approccio modificando la struttura della rete neurale ricorrente e applicando un'operazione aggiuntiva al dataset. Invece di addestrare il modello con sequenze temporali, si è la RNN utilizzando singoli campioni.

Per prima cosa, è stato aggiunto un nuovo metodo allo script per l'ingegnerizzazione delle features. Questo metodo, chiamato *_feature_engineering*(), calcola quattro valori per ogni singola feature: media, deviazione standard, massimo e minimo. Questi valori aggiuntivi ci permettono di rappresentare ogni campione in modo più dettagliato e sotto forma di una singola riga, non di una sequenza temporale di campioni. Il problema di questo approccio è il costo computazionale: il calcolo di 4 valori statistici per 174 feature dà luogo a 696 feature totali, ed il tempo impiegato per l'ingegnerizzazione di queste non è trascurabile, rendendo difatti la classificazione real time abbastanza più lenta rispetto all'analisi di sequenze temporali.

```
def _feature_engineering(df_list):
    progress_bar = tqdm(total=len(df_list), desc='
       Engineering features', leave=True)
    new_list = []
    for df in df_list:
        tmp1 = pd.DataFrame()
        for col in df.columns:
            tmp2 = {
              f'{col}_mean': df[col].mean(),
              f'{col}_std': df[col].std(),
              f'{col}_max': df[col].max(),
              f'{col}_min': df[col].min()
            }
            tmp2 = pd.DataFrame(tmp2, index=[0])
            tmp1 = pd.concat([tmp1, tmp2], axis=1)
        new_list.append(tmp1)
        progress_bar.update(1)

    return new_list
```

Dovendo trattare i dati in maniera diversa, è necessario modificare anche il metodo *_data_preprocessing*(). In particolare, dopo aver ingegnerizzato ogni singolo dataframe in possesso e aver ottenuto una lista di singoli campioni, questi devono essere concatenati in un unico dataframe, e bisogna mappare nuovamente le label in valori interi.

```
def _data_preprocessing(X_train, X_test, y_train, y_test
   ):
    cl = {'air_quotes':0, 'finger_wagging':1, 'waving'
       :2, 'zoom':3}
    for i in range(len(y_train)):
```

```
        y_train[i] = cl[y_train[i]]
    for i in range(len(y_test)):
        y_test[i] = cl[y_test[i]]
    y_train = np.array(y_train)
    y_test = np.array(y_test)

    tmp = []
    for df in X_train:
        tmp.append(df.values)
    X_train = np.concatenate(tmp, axis=0)
    tmp = []
    for df in X_test:
        tmp.append(df.values)
    X_test = np.concatenate(tmp, axis=0)

    return X_train, X_test, y_train, y_test
```

Una volta ottenuti i dati e le label sia di test che di train, si effettua una normalizzazione sempre di tipo MinMaxScale, per facilitare l'apprendimento del modello, e si pratica un nuovo reshape dei dati, questa volta impostando il timestep (lunghezza della sequenza di input) a 1 in quanto adesso abbiamo singoli campioni.

```
scaler = MinMaxScaler(feature_range=(-1, 1))
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = X_train.reshape(X_train.shape[0], 1, X_train.
   shape[1])
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape
   [1])
```

Per questo nuovo tentativo, è stata modificata l'architettura della RNN. Il modello è ora una rete neurale profonda (DNN), poiché presenta due hidden layer, per un totale di quattro layer complessivi.

```
model = Sequential(
    [
        GRU(32,
            input_shape=(None, X_train.shape[2]),
            dropout=0.1,
            recurrent_dropout=0.1,
            return_sequences=True),
        GRU(64,
            dropout=0.1,
            recurrent_dropout=0.1),
        Dense(64,
              activation='relu'),
```

```
            Dense(4,
                  activation='softmax')
        ]
    )

    model.compile(optimizer='adam', loss='
       sparse_categorical_crossentropy', metrics=['accuracy'
       ])
```

Il primo layer è il layer di input, composto da 32 neuroni GRU (Gated Recurrent Unit). Il secondo layer è sempre un layer ricorrente, con 64 neuroni GRU. Il terzo layer è un layer densamente connesso, con 64 neuroni e una funzione di attivazione di tipo ReLU. Infine, l'ultimo layer è il layer di output, che possiede 4 neuroni con una funzione di attivazione softmax, adatto per problemi di classificazione multiclasse.

È degno di nota che questa volta non sono stati implementati layer bidirezionali: questo perchè, dal momento che adesso abbiamo sequenze composte da un unico valore rappresentativo con valori statistici, l'analisi temporale in avanti e indietro non avrebbe alcun vantaggio.

Inoltre, abbiamo cambiato l'ottimizzatore della funzione di costo e provato ad utilizzare Adam, un algoritmo di ottimizzazione ampiamente utilizzato per addestrare reti neurali.

| **Layer (type)** | **Output Shape** |
|---|---|
| GRU | (None, None, 32) |
| GRU | (None, 64) |
| Dense | (None, 64) |
| Dense | (None, 4) |

Tabella 4: Sommario dell'architettura del modello ingegnerizzato.

Dopo una serie di prove sull'architettura del modello, provando un diverso numero di layer e di neuroni per layer, si è ottenuta una corretta classificazione di tutti i campioni di test.

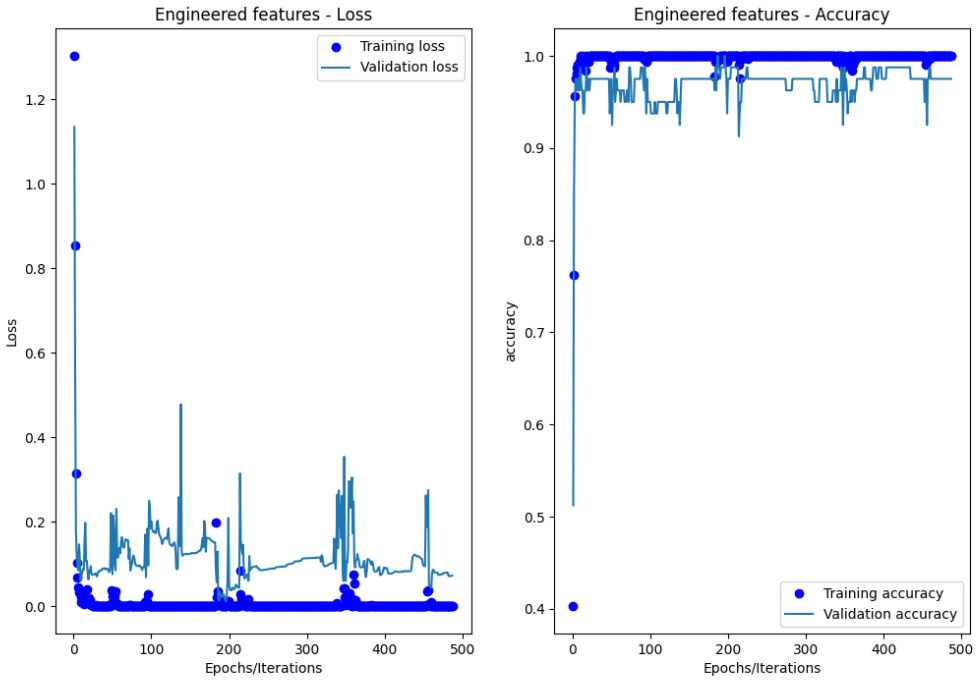



Figura 35: Andamento delle performance dopo il training del modello ingegnerizzato.

Nonostante i valori di validation loss e validation accuracy oscillino, il modello si è praticamente assestato approssimativamente al 98.88% di accuracy.

Grazie alla early_stopping_callback, sono stati ritornati i pesi del modello che ha ottenuto le migliori prestazioni: il miglior risultato ottenuto è una rete con una classificazione perfetta su tutti i campioni di test, quindi un risultato di 80 gesti correttamente classificati su 80.

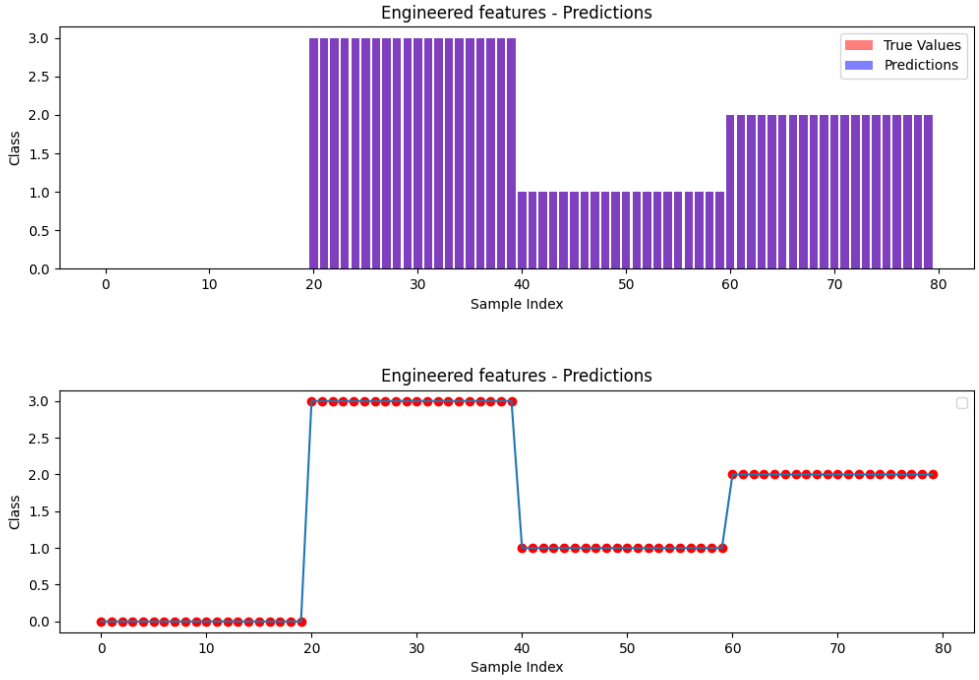



Figura 36: Predizioni del modello ingegnerizzato.

Come c'è da aspettarsi i valori finali di accuracy e loss sul test set sono perfetti: l'accuratezza del modello è del 100%, mentre il valore di loss è appena 0.013, quasi del tutto trascurabile.

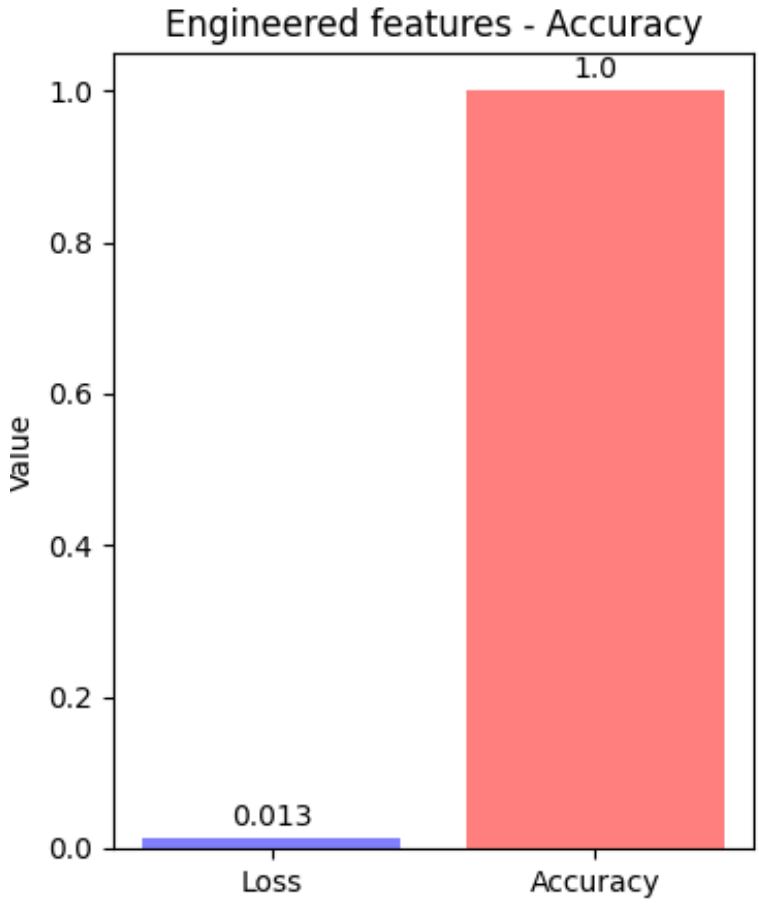



Figura 37: Loss e accuracy sul test set del modello ingegnerizzato.

Mostriamo anche qui la confusion matrix del modello, che dimostra nuovamente di aver classificato correttamente tutti gli 80 campioni presentati, rispettivamente 20 campioni per gesto.

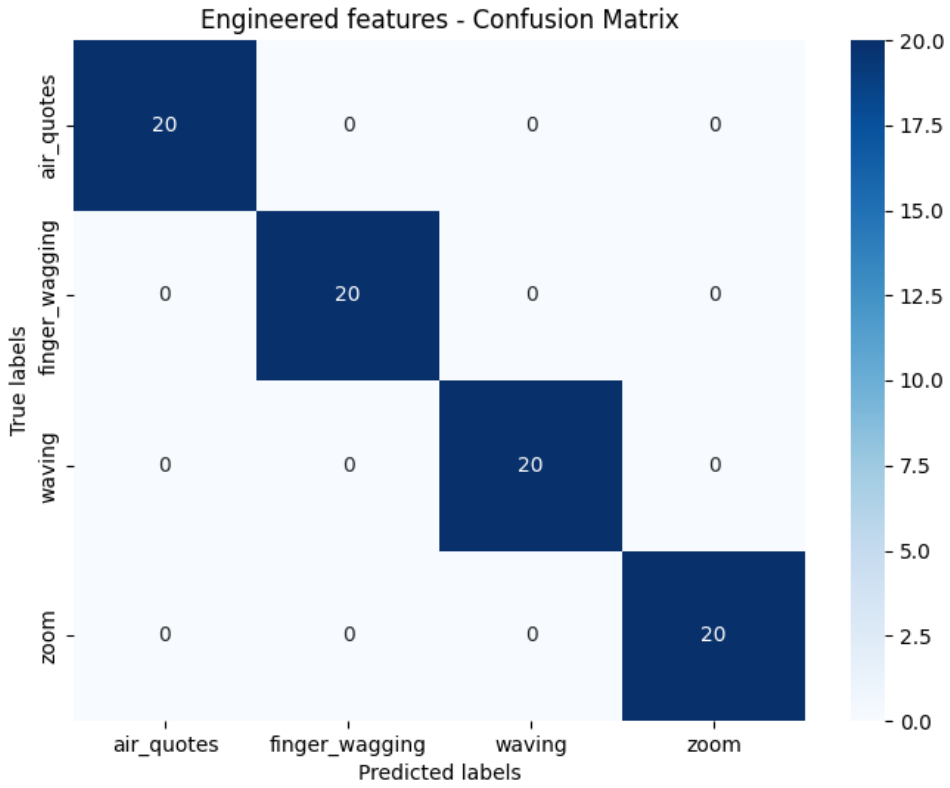



Figura 38: Confusion Matrix del modello ingegnerizzato.

Infine mostriamo la ROC Curve del modello ingegnerizzato, che ovviamente presenta valori perfetti: tutte le curve di classificazione intersecano il valore 1, ovvero quello del classificatore perfetto, mentre tutte le AUC presentano anch'esse un valore di 1.

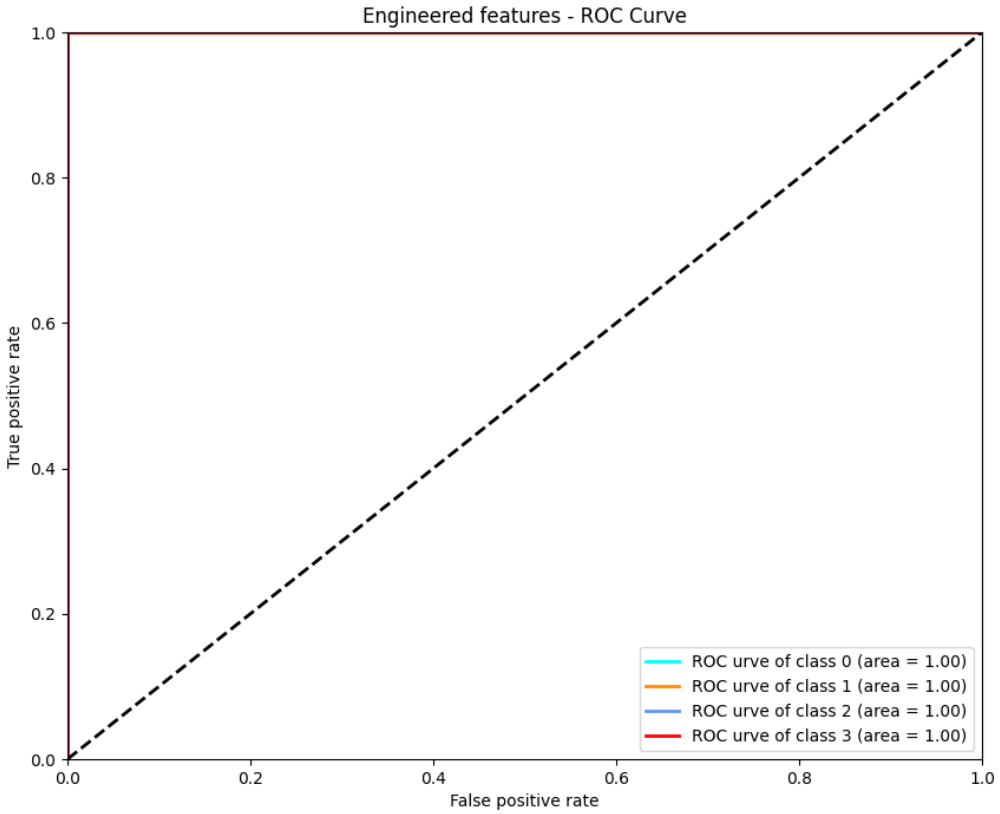



Figura 39: ROC Curve del modello ingegnerizzato.

## 7.7 Confronto con un algoritmo non-neural

Per completezza, si è scelto di confrontare due approcci di apprendimento differenti: la nostra rete neurale ricorrente e un algoritmo di apprendimento non-neural, il Random Forest.

Il Random Forest è un tipo di algoritmo di machine learning basato su un insieme di alberi decisionali, noto per la sua robustezza e versatilità. A differenza delle reti neurali, che sono modelli complessi con molti parametri, il Random Forest è relativamente semplice da comprendere e interpretare. Funziona combinando i risultati di diversi alberi decisionali indipendenti, ciascuno costruito su un sottoinsieme casuale dei dati di addestramento e delle caratteristiche, e prendendo una media o una maggioranza delle loro predizioni.

Una delle principali ragioni per cui il Random Forest gode di una grande explainability è la sua natura basata su alberi decisionali. Ogni albero nel foresta può essere visualizzato e interpretato individualmente, consentendo agli analisti di comprendere come vengono fatte le decisioni in base alle caratteristiche dei dati di input. Inoltre, il Random Forest fornisce misure di importanza delle caratteristiche, che indicano quanto ciascuna variabile influenzi le predizioni del modello. Questo è un contrasto significativo rispetto alle reti neurali, specialmente quelle ricorrenti, che spesso non forniscono una chiara spiegazione del processo decisionale interno. Le reti neurali, essendo modelli più complessi e non lineari, possono essere difficili da interpretare e spiegare. Le loro predizioni sono il risultato di un intricato processo di calcolo distribuito su molti strati e nodi, il che rende difficile tracciare il ragionamento esatto che porta a una particolare previsione. Questo ci aiuterà a valutare non solo quale modello sia migliore in termini di accuratezza, ma anche quale sia più adatto a contesti in cui è fondamentale comprendere il ragionamento dietro le previsioni.

Ovviamente, un algoritmo non-neural non è in grado di analizzare sequenze temporali, per cui sono stati riutilizzati gli stessi dati ingegnerizzati utilizzati in precedenza.

Per ottimizzare le prestazioni del classificatore, è stata eseguita una ricerca degli iperparametri migliori utilizzando la grid search. Questo permette di esplorare una griglia di combinazioni di iperparametri e trovare quelle che massimizzano l'accuratezza sul test set. La grid search funziona nel seguente modo: definiamo una griglia di valori per gli iperparametri che vogliamo ottimizzare, come ad esempio il numero di alberi nella foresta (n_estimators), la profondità massima di ogni albero (max_depth), e così via. Successivamente, la grid search esegue l'addestramento del classificatore su tutte le possibili combinazioni di iperparametri specificate nella griglia, utilizzando una tecnica di cross-validation per valutare le prestazioni di ogni combinazione. A seguire l'implementazione:

```
param_grid = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
```

```
    'min_samples_leaf': [1, 2, 4]
}

clf = RandomForestClassifier()
grid_search = GridSearchCV(estimator=clf, param_grid=
    param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best parameters found:", grid_search.best_params_
    )
best_scores = grid_search.best_score_
print(f'Accuracy: {round(best_scores, 3)}%')
```

Una volta completata la ricerca degli iperparametri, è stato individuato il classificatore ottimale, che ha raggiunto un'accuratezza di circa il 97.2%. Tuttavia, questo risultato non è paragonabile alla prestazione della RNN, che ha raggiunto un'accuracy del 100%.

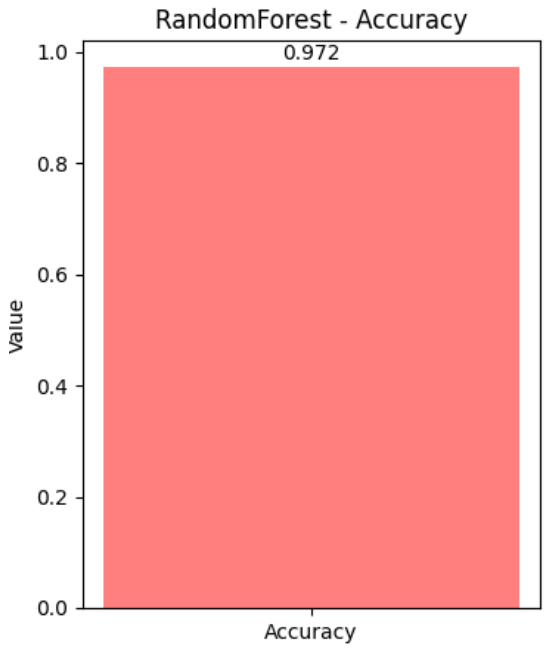



Figura 40: RF accuracy su test set.

Come accennato in precedenza, una delle grandi caratteristiche del Random Forest è la sua straordinaria capacità di spiegare le decisioni. È possibile esplorare la struttura di un albero decisionale interno al modello per comprendere i criteri utilizzati nelle predizioni. Di seguito è riportata la struttura del primo albero del nostro Random Forest.

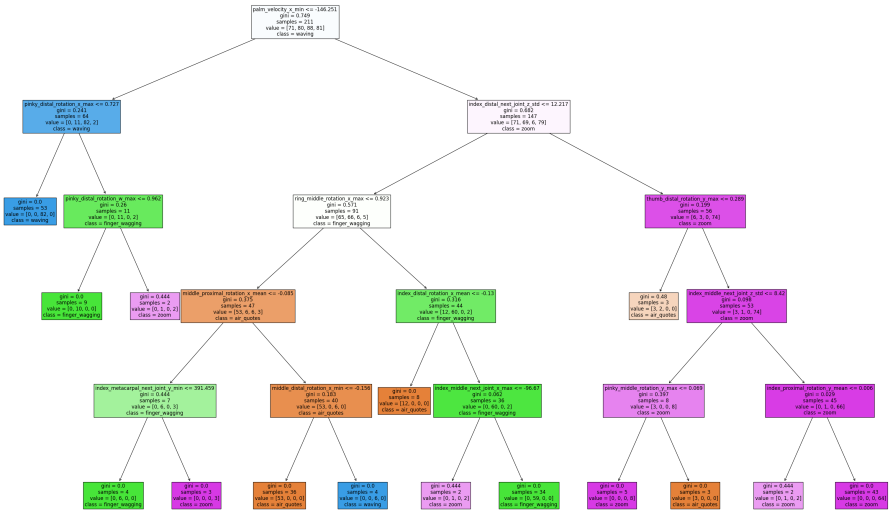


Figura 41: Struttura di un albero decisionale del miglior RF.

Notiamo come l'indice decisionale utilizzato in questo Random Forest è l'indice di Gini. L'indice di Gini è una misura di impurità utilizzata nell'albero decisionale per valutare quanto un determinato nodo sia eterogeneo in termini di classi target. Più basso è il valore di Gini, più "puro" è il nodo, cioè contiene principalmente esempi di una sola classe. L'indice di Gini è calcolato sull'insieme dei dati nel nodo e tiene conto della frequenza delle diverse classi. Durante la costruzione dell'albero decisionale, l'algoritmo cercherà di suddividere i nodi in modo tale da ridurre l'impurità (ovvero, ridurre l'indice di Gini) il più possibile in ciascuno dei nodi figli.

Questo particolare albero ha effettuato le sue scelte basandosi su feature specifiche, come ad esempio palm_velocity_x_min e altre caratteristiche rilevanti.

Grazie all'utilizzo del modulo Lime di Python, siamo stati in grado di analizzare quali feature il Random Forest utilizza maggiormente per effettuare le predizioni. Questo fornisce una preziosa comprensione dei processi decisionali del modello.

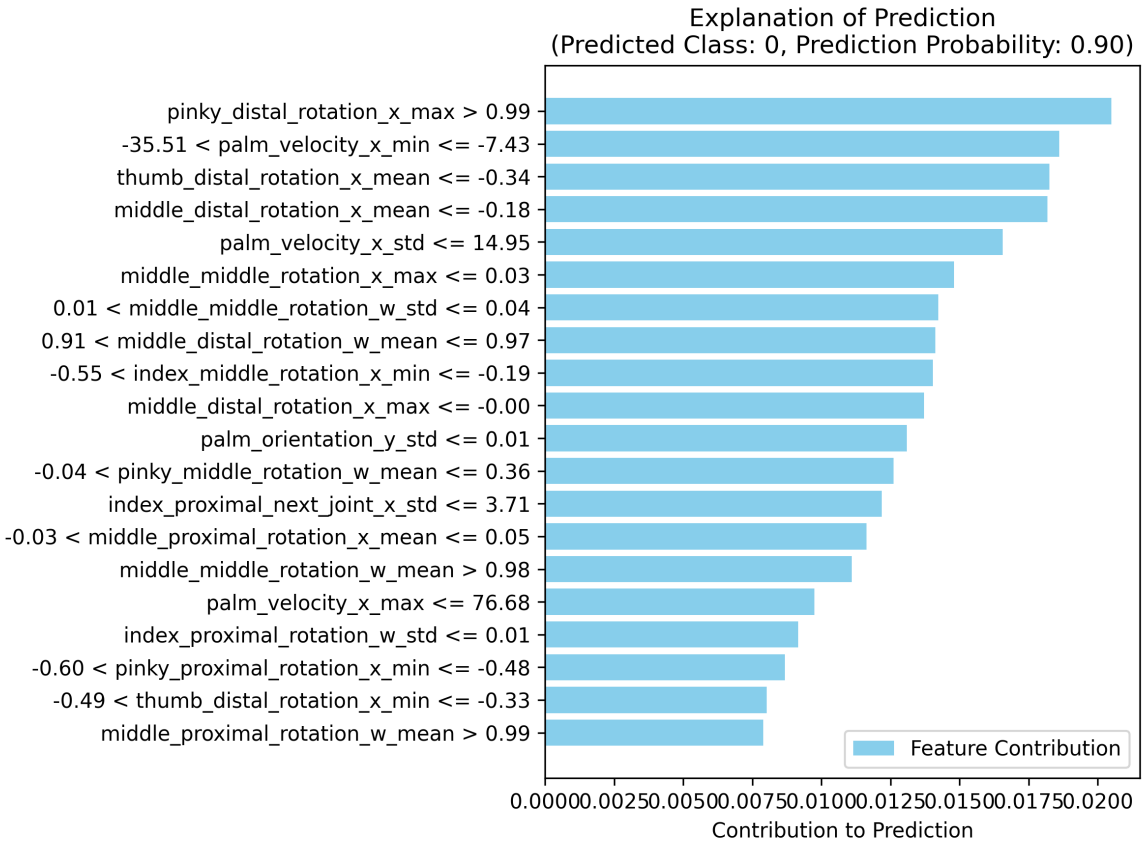



Figura 42: Contributo delle feature alla predizione.

Ad esempio, durante la predizione su un determinato campione, è stato osservato che il modello ha attribuito una probabilità del 90% alla sua predizione. È interessante notare che questa decisione è stata basata principalmente su alcune feature specifiche, suggerendo che tali caratteristiche svolgono un ruolo significativo nel processo di classificazione del modello. Questo tipo di analisi aiuta a comprendere meglio come il modello prende le sue decisioni e quali aspetti dei dati sono considerati più importanti nella fase di predizione. Ci fornisce anche un'opportunità per valutare la coerenza delle predizioni del modello e individuare eventuali pattern o tendenze nei dati che potrebbero influenzare le sue decisioni.

Considerando le prestazioni predittive, il Random Forest ha classificato correttamente 76 gesti su 80, il che è un risultato degno di nota data la complessità del dataset. Tuttavia, è importante notare che la RNN si è dimostrata ancora più performante in termini di accuratezza predittiva, con una corretta classificazione di 80 gesti su 80.

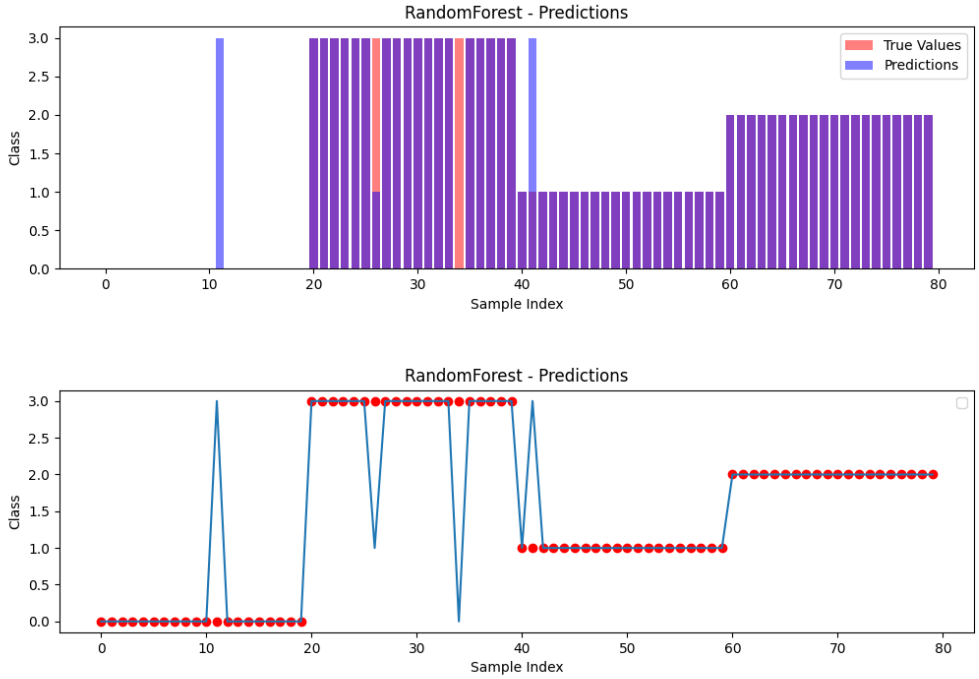



Figura 43: Predizioni del RF.

Stampiamo anche la matrice di confusione per comprendere meglio dove il modello non-neural commette più errori.

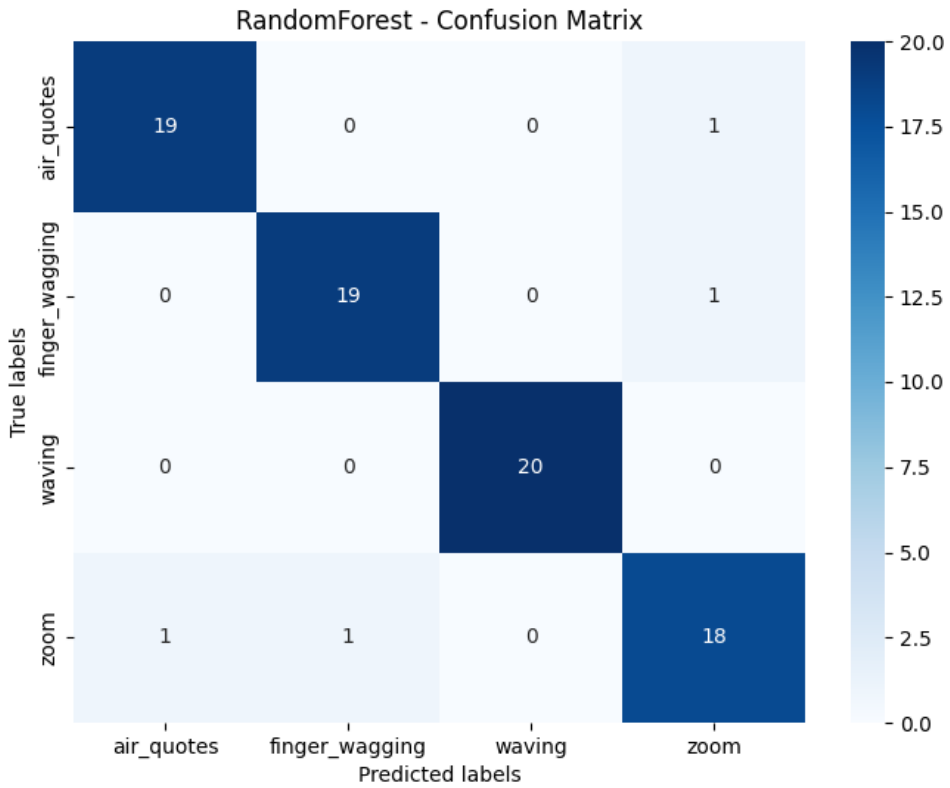



Figura 44: Confusion matrix del RF

È evidente che il modello ha più difficoltà nel riconoscere il gesto dello zoom e spesso lo confonde con gesti come le air quotes o il finger wagging. Questo potrebbe essere dovuto al fatto che tutti e tre i gesti coinvolgono principalmente il movimento delle dita, ma non necessariamente del palmo, rendendoli più simili tra loro e quindi più difficili da distinguere per il modello. Questa osservazione ci fornisce un'importante intuizione sul tipo di errori che il modello commette e suggerisce possibili miglioramenti nel processo di classificazione, ad esempio considerando altre caratteristiche o utilizzando tecniche di fusione dei dati per migliorare la discriminazione tra gesti simili.

Anche la curva ROC del nostro modello non-neural ha mostrato risultati eccellenti, con valori molto vicini a quelli ottenuti dalla RNN. In particolare, tre delle quattro classi hanno ottenuto un'area sotto la curva (AUC) approssimativamente pari a 1, il che indica un'eccellente capacità discriminante del modello. Solo per la quarta classe abbiamo ottenuto un punteggio leggermente inferiore, ma comunque molto alto, pari a 0.99.

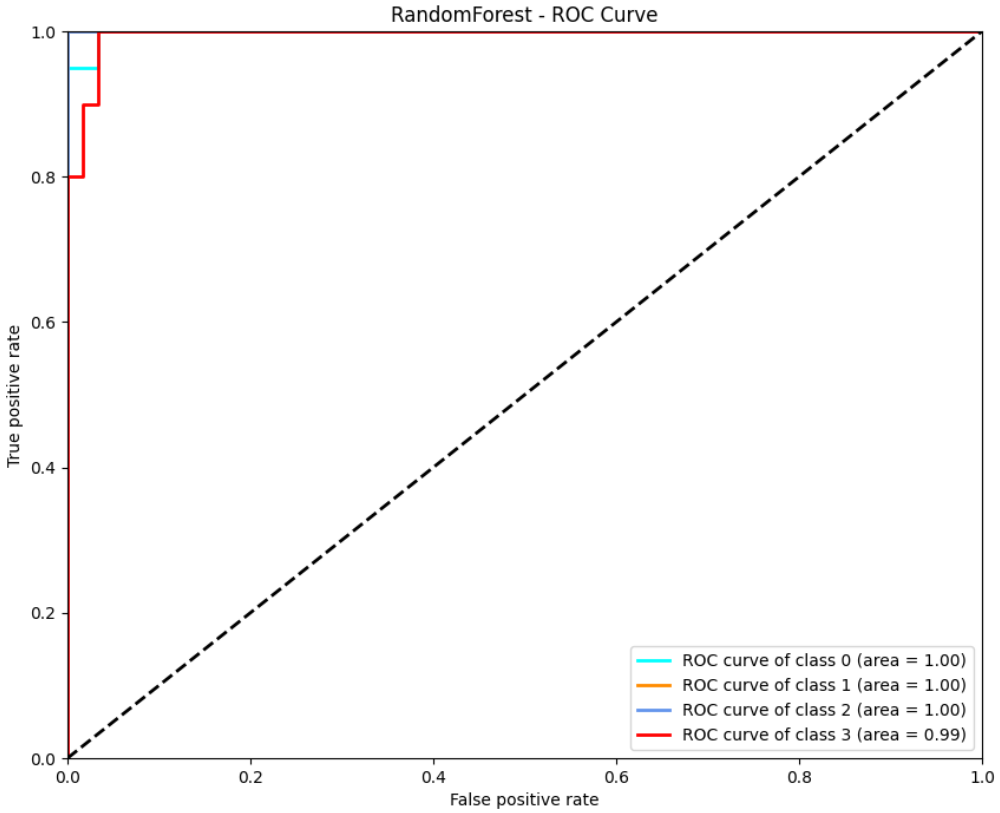



Figura 45: ROC Curve del RF

# 8 Conclusione

In conclusione, è stata offerta una panoramica generale sull'intelligenza artificiale e sugli algoritmi adottati, con un focus sulla vasta gamma di gesti, sia statici che dinamici, e sulle tecniche di preprocessing, riconoscimento e classificazione utilizzate per affrontare questa sfida.

Particolare attenzione è stata dedicata all'esplorazione dei dispositivi a basso costo disponibili per il riconoscimento dei gesti, concentrandosi soprattutto sul Leap Motion Controller 2, che ha rappresentato il fulcro dell'implementazione pratica.

Successivamente, è stato approfondito il funzionamento delle reti neurali ricorrenti (RNN), mettendo in luce le logiche e le scelte di progettazione necessarie per creare un algoritmo efficace per il riconoscimento dei gesti dinamici.

È stato sviluppato e implementato un algoritmo completo per il riconoscimento dei gesti dinamici, partendo dalla raccolta dei dati fino alla creazione e all'addestramento di una RNN per la classificazione. I risultati ottenuti sono stati notevoli, soprattutto considerando la complessità del dataset sperimentale utilizzato. Durante il lavoro, sono stati esplorati due approcci diversi: uno basato sull'analisi delle sequenze temporali e l'altro sull'addestramento della RNN su un dataset ingegnerizzato. Quest'ultimo, sebbene più oneroso a livello computazionale e meno adatto alle previsioni in tempo reale, ha mostrato un'accuratezza ancora maggiore rispetto al primo approccio.

Inoltre, sono state confrontate le prestazioni della RNN con un approccio basato su Random Forest (RF). Mentre la RNN ha ottenuto risultati eccellenti, il RF si è dimostrato leggermente meno performante, soprattutto considerando l'ingegnerizzazione delle feature richiesta. Una differenza significativa è stata la trasparenza del modello: la RNN non è interpretabile allo stesso modo del RF, che gode di una maggiore explainability, facilitando la comprensione delle decisioni del modello.

È importante notare che, grazie alla sua efficienza e velocità nell'elaborazione delle previsioni, dovuta alla mancanza di necessità di ingegnerizzare le features e al conseguente ridotto costo computazionale, l'intero sistema basato su RNN bidirezionale è già stato implementato con successo in un ambiente di realtà virtuale. In questo contesto, i gesti eseguiti in ambiente virtuale vengono classificati in tempo reale, permettendo di associare azioni specifiche a ciascun tipo di gesto effettuato. Questo approccio apre la strada a molteplici applicazioni con scopi diversi, offrendo ampi margini di manovra per l'innovazione e l'ottimizzazione in vari settori.

Guardando al futuro, si auspica che questo lavoro possa ispirare lo sviluppo di tecnologie innovative in vari settori, dalla riabilitazione sanitaria al divertimento ludico. Il riconoscimento dei gesti ha il potenziale per rivoluzionare molti ambiti, rappresentando un processo di cambiamento e innovazione entusiasmante.

# Riferimenti bibliografici


[1] Monika Bansal, Munish Kumar e Manish Kumar. "2D object recognition: a comparative analysis of SIFT, SURF and ORB feature descriptors". In: *Multimedia Tools and Applications* 80.12 (2021), pp. 18839–18857.

[2] Alessandro Bellini e Andrea Guidi. *Python e Machine Learning*. McGraw-Hill Education, 2022.

[3] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., 2018.

[4] Alina Delia Călin Adriana Coroiu e Adriana Coroiu. "Interchangeability of kinect and orbbec sensors for gesture recognition". In: *2018 IEEE 14th international conference on intelligent computer communication and processing (ICCP)*. IEEE. 2018, pp. 309–315.

[5] Zak Flintoff, Bruno Johnston e Minas Liarokapis. "Single-grasp, model-free object classification using a hyper-adaptive hand, google soli, and tactile sensors". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1943–1950.

[6] Vito Gentile et al. "Gesture recognition using low-cost devices: Techniques, applications, perspectives". In: *Mondo Digitale* 15.63 (2016), pp. 161–169.

[7] Tibor Guzsvinecz, Veronika Szucs e Cecilia Sik-Lanyi. "Suitability of the Kinect sensor and Leap Motion controller—A literature review". In: *Sensors* 19.5 (2019), p. 1072.

[8] *Intel RealSense Depth Camera D435i*. `https://www.intelrealsense.com/depth-camera-d435i/`. Recuperato il 4 marzo 2024. 2024.

[9] Chris Joslin et al. "Dynamic gesture recognition". In: *2005 IEEE Instrumentationand Measurement Technology Conference Proceedings*. Vol. 3. IEEE. 2005, pp. 1706–1711.

[10] Fabio Nelli. "OpenCV & Python – La binarizzazione di Otsu". In: *Meccanismo Complesso* (2018). URL: `https://www.meccanismocomplesso.org/opencv-python-otsu-binarization-thresholding/`.

[11] Duong Hai Nguyen et al. "Hand segmentation and fingertip tracking from depth camera images using deep convolutional neural network and multi-task segnet". In: *arXiv preprint arXiv:1901.03465* (2019).

[12] Stuart J. Russell, Peter Norvig e Francesco Amigoni. *Intelligenza artificiale. Un approccio moderno. Vol. 1*. Pearson, 2021.

[13] Ashish Sharma et al. "Hand gesture recognition using image processing and feature extraction techniques". In: *Procedia Computer Science* 173 (2020), pp. 181–190.

[14] Smartworld.it. "Oculus Quest 2, la recensione: il VR per tutti (anche per chi ha un iPhone)". In: (2022). URL: `https://www.smartworld.it/recensioni/oculus-quest-2`.

[15] Renjie Song, Ziqi Zhang e Haiyang Liu. "Edge connection based Canny edge detection algorithm". In: *Pattern Recognition and Image Analysis* 27 (2017), pp. 740–747.

[16] UltraLeap. *Leap Concepts - Tracking API - UltraLeap Documentation*. [Online; accessed 26-April-2024]. 2024. URL: https://docs.ultraleap.com/api-reference/tracking-api/leapc-guide/leap-concepts.html.

[17] UltraLeap. *UltraLeap Official Website*. URL: https://www.ultraleap.com/.

[18] Ultraleap. *Gemini LeapC Python Bindings*. https://github.com/ultraleap/leapc-python-bindings. 2024.

[19] Hui-Shyong Yeo, Byung-Gook Lee e Hyotaek Lim. "Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware". In: *Multimedia Tools and Applications* 74 (2015), pp. 2687–2715.

[20] SHI Yuanyuan et al. "Review of dynamic gesture recognition". In: *Virtual Reality & Intelligent Hardware* 3.3 (2021), pp. 183–206.