UNIVERSITÀ DEGLI STUDI DI CATANIA

Dipartimento di Matematica e Informatica

Corso di Laurea Triennale in Informatica

Luigi Seminara

Riconoscimento di azioni su oggetti mediante Microsoft HoloLens 2

Relazione Progetto Finale

Relatore: Prof. Antonino Furnari

Correlatore: Rosario Leonardi

Anno Accademico 2020 - 2021

Abstract

L'obiettivo di questa tesi è lo sviluppo di un metodo per il riconoscimento di interazioni uomo-oggetto a partire da coordinate 3D di punti chiave di mani ottenute tramite il dispositivo indossabile Microsoft HoloLens2. Riconoscere le interazioni utilizzando dei dispositivi indossabili permette di costruire dei sistemi in grado di migliorare la sicurezza dei lavoratori in una fabbrica o di assisterli durante le loro attività. Ad esempio se un utente prende un trapano il dispositivo potrebbe visualizzare informazioni su di esso, come il livello di batteria, o una guida su come utilizzarlo. Nello specifico il metodo proposto cercherà di distinguere tre azioni: *take*, *release* e *push*. Questo verrà fatto utilizzando una rete neurale ricorrente chiamata *Long-Short Term Memory* (LSTM) che prende in input le sequenze di punti chiave delle mani.
Lo studio riportato in questo documento è stato strutturato considerando gli strumenti utilizzati, le fasi che hanno permesso di raggiungere dei risultati e l'analisi su questi ultimi. Dai risultati ottenuti è emerso che nessuno dei modelli sviluppati è in grado di risolvere il problema in maniera ottimale, ma trasformando il problema iniziale in due macro problemi si è dedotto che è possibile utilizzare il metodo proposto per distinguere i momenti in cui si eseguono azioni, dai momenti in cui non si eseguono.

Indice

Capitolo 1

Introduzione

Lo scopo di questa tesi è lo sviluppo di un metodo in grado di riconoscere interazioni tra uomo e oggetto a partire da coordinate 3D di punti chiave di mani. In particolare, il metodo proposto dovrà essere in grado di riconoscere tre tipologie di azioni: *take*, *release* e *push*. Inoltre, in assenza di azioni, esso dovrà assegnare un'etichetta di "background" chiamata *No_action*. Per ottenere le coordinate che fanno riferimento ai punti chiave delle mani, sarà utilizzato il dispositivo di realtà mista Microsoft HoloLens 2. L'idea alla base è raggruppare questi punti chiave e creare delle sequenze temporali da utilizzare come input di una LSTM. Quest'ultima, dopo aver analizzato una sequenza, dovrà restituire in output un vettore di quattro valori (ognuno associato ad un'azione) che verrà utilizzato per associare alla sequenza una specifica azione.
La Figura 1.1 mostra un diagramma che riassume il procedimento descritto sopra.

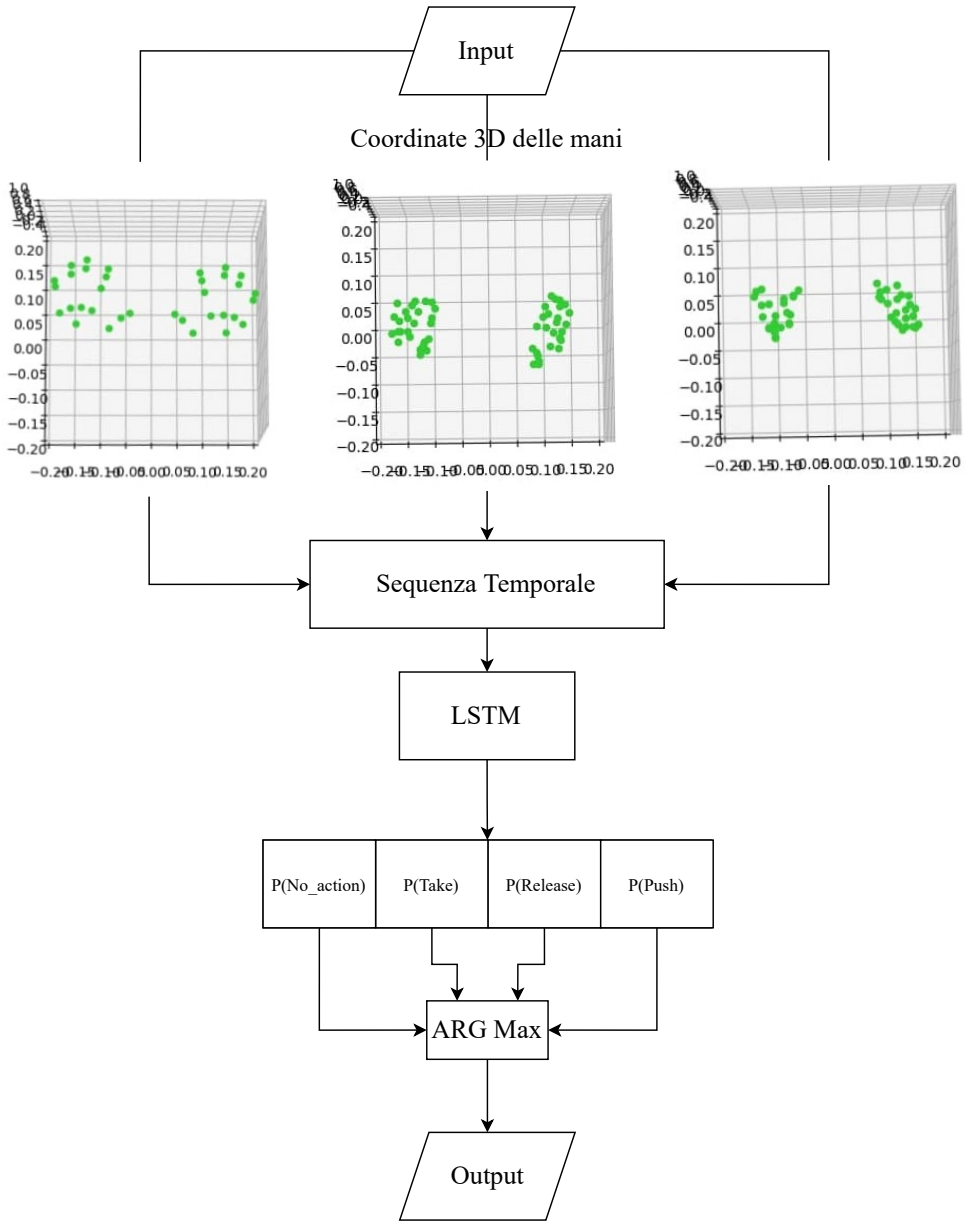

Figura 1.1: Diagramma del problema

La risoluzione di questo problema può trovare applicazione nel campo dell'interazione uomo-macchina, con particolare riferimento all'uso di dispositivi indossabili in contesti lavorativi.
Negli ultimi anni, l'impiego di tecnologie a supporto del lavoro umano è cresciuto in maniera esponenziale. Capire il tipo di azione che una persona sta svolgendo permette a un dispositivo di intelligenza artificiale, come HoloLens2, di poter dare informazioni e suggerimenti su ciò che l'utente può o

deve fare.
Ad esempio, se un utente sta montando un sistema di iniezione del carburante, HoloLens visualizzerà una guida olografica per ogni pezzo preso dall'utente, come mostrato in Figura 1.2.

Figura 1.2: Esempio di istruzioni olografiche

Lo studio riportato in questo documento è stato strutturato nel seguente modo:

- nel Capitolo 2 verranno discussi gli strumenti utilizzati e i lavori correlati a quello presentato all'interno di questo studio;
- nel Capitolo 3 si discuteranno l'acquisizione e l'etichettatura dei dati. Inoltre, verranno mostrate alcune statistiche su di essi e verrà descritta la fase di suddivisione dei dati in Training, Validation e Test set;
- nel Capitolo 4 verranno descritti gli algoritmi sviluppati e il modo in cui i dati sono stati utilizzati;
- nel Capitolo 5 verranno descritti gli esperimenti effettuati e discussi i risultati ottenuti, spiegando quali sono i problemi riscontrati.

A conclusione saranno brevemente discussi anche i possibili sviluppi futuri del presente studio.

Capitolo 2

Strumenti utilizzati

In questo capitolo verranno discussi gli strumenti utilizzati che hanno permesso di raggiungere i risultati di questa tesi. Nella sezione 2.1 è presente una breve descrizione su HoloLens 2. Nella sezione 2.2 si parlerà del linguaggio di programmazione utilizzato, ovvero Python, e delle motivazioni legate alla scelta di questo linguaggio. Nella sezione 2.3 verrà descritta un'architettura di rete neurale artificiale ricorrente, chiamata **LSTM**, acronimo di *Long-Short Term Memory*, e infine nella sezione 2.4 verranno discussi i lavori correlati a questo studio.

2.1 HoloLens 2

Microsoft HoloLens 2 è un paio di occhiali smart per realtà mista, prodotto e sviluppato da Microsoft. Esso viene definito come un computer olografico senza thering, ovvero ha una conoscenza dinamica del mondo basata su sensori, ottenuta adattando continuamente le proprie conoscenze nel corso del tempo in base all'ambiente dell'utente. In Figura 2.1 è riportata un'immagine del dispositivo.

Figura 2.1: HoloLens2

In HoloLens, gli ologrammi combinano il mondo digitale con l'ambiente fisico per apparire e sembrare parte del mondo. Anche quando gli ologrammi sono tutti intorno, è sempre possibile vedere l'ambiente circostante, spostarsi liberamente e interagire con persone e oggetti. Questa esperienza viene chiamata "realtà mista".
HoloLens verrà utilizzato per salvare ad ogni istante le coordinate che fanno riferimento a dei punti chiave delle mani, usando un'apposita applicazione sviluppata in Unity. Ad ogni mano vengono associati 26 punti nello spazio tridimensionale e, se si considerano entrambe le mani, in tutto si hanno 52 punti. La Figura 2.2 mostra i punti chiave della mano destra, gli stessi punti sono presenti nella mano sinistra.

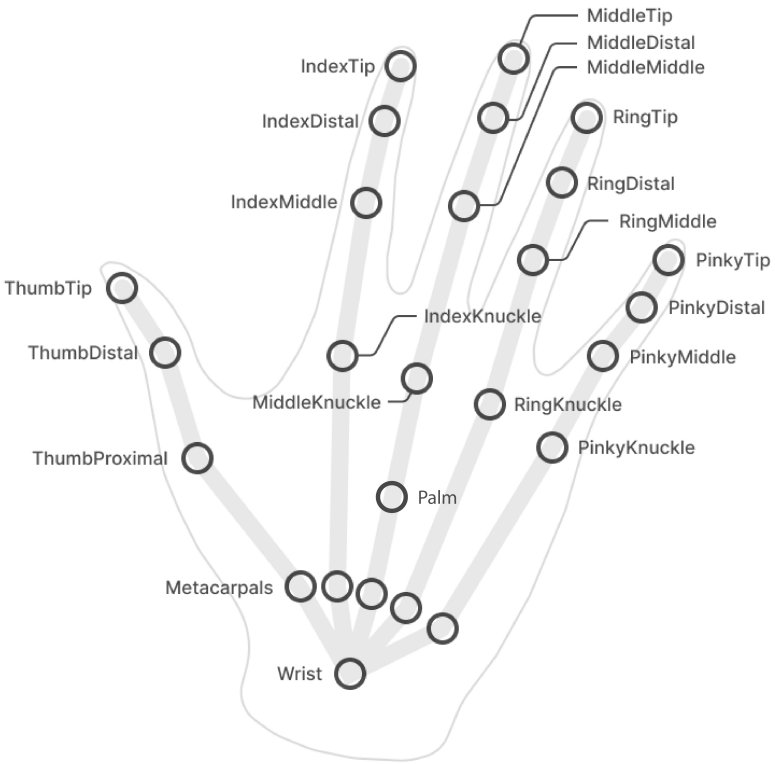

Figura 2.2: Punti chiave della mano destra

I valori delle coordinate dei punti chiave saranno utilizzati per distinguere le azioni *take*, *release* e *push*.

2.2 Python

Il linguaggio di programmazione utilizzato in questa tesi per lo sviluppo di algoritmi è Python.

Python è un linguaggio di programmazione interpretato, orientato agli oggetti e di alto livello con semantica dinamica. Progettato per essere facile da leggere e semplice da utilizzare. È open source e gratuito, anche per applicazioni commerciali.

Per questa tesi è stato scelto Python come linguaggio di programmazione

perché offre una vasta gamma di pacchetti e moduli. In particolare, semplifica la creazione di reti neurali, la rappresentazione di grafici, la creazione di processi in grado di poter comunicare tra loro, e molto altro. Esistono pacchetti in grado di risolvere la maggior parte dei problemi, difficilmente risolvibili con altri linguaggi di programmazione.

2.2.1 Librerie utilizzate

Le librerie principali sono le seguenti:

- **numpy**: è utilizzata per lavorare con gli array. Ha anche funzioni per lavorare nel dominio dell'algebra lineare, della trasformata di Fourier e delle matrici.
- **pandas**: è utilizzata per la manipolazione e l'analisi dei dati. In particolare, offre strutture dati e operazioni per la manipolazione di tabelle numeriche e serie temporali.
- **matplotlib**: fornisce un'API orientata agli oggetti per incorporare grafici in applicazioni utilizzando toolkit GUI generici come Tkinter, wxPython, Qt o GTK.
- **PyQt5**: è un toolkit GUI multipiattaforma, un set di collegamenti Python per Qt v5. Qt è un insieme di librerie C++ e strumenti di sviluppo che include astrazioni indipendenti dalla piattaforma per interfacce grafiche, networking, thread, espressioni regolari, database SQL, SVG, OpenGL, XML, servizi di localizzazione, comunicazioni a corto raggio (NFC e Bluetooth), navigazione web, animazione 3D, grafici, visualizzazione dati 3D e interfacciamento con app store.
- **PyTorch**: è una libreria di tensori ottimizzata per applicazioni di deep learning che utilizzano GPU e CPU. È mantenuta principalmente dal team di Facebook *AI Research* ed è una delle librerie di machine learning ampiamente utilizzate, altre sono TensorFlow e Keras.

2.3 LSTM - Long-Short Term Memory

Le unità o i blocchi di memoria LSTM sono un tipo di **rete neurale ricorrente**, che si differenzia dalle reti neurali classiche, chiamate **feedforward**, che trasmettono i dati dall'input all'output, mentre le reti ricorrenti possiedono un *feedback loop* in cui i dati possono essere reimmessi nell'input ad un

certo punto prima di essere inoltrati per ulteriori elaborazioni e output finale. Le reti neurali ricorrenti sono realizzate per utilizzare determinati tipi di processi di memoria artificiale, che possono aiutare i programmi ad imitare in modo più efficace il pensiero umano.

Una rete LSTM ha tre capacità:

- *Forget*: la capacità di eliminare dalla memoria informazioni che non sono più utili.
- *Save*: la capacità di salvare determinate informazioni in memoria.
- *Focus*: la capacità di focalizzarsi solo su aspetti della memoria immediatamente rilevanti.

Per realizzare tali proprietà è necessaria una **memoria a lungo termine** che mantenga informazioni sulla parte di sequenza già analizzata, e di una **memoria corrente** che mantenga informazioni di immediata rilevanza.

La LSTM verrà utilizzata per analizzare le sequenze di punti chiave delle mani e classificare l'azione in tali sequenze.

2.3.1 Struttura di una LSTM

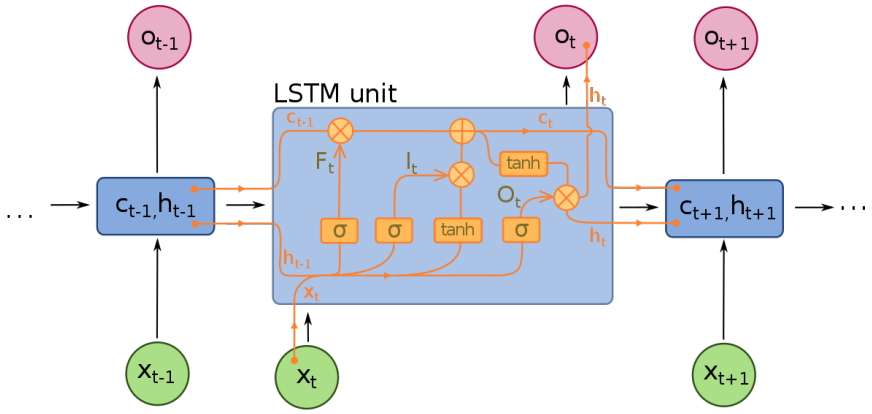

Figura 2.3: Struttura di una cella LSTM

Il modello LSTM ha due rappresentazioni latenti ad ogni istante di tempo t:

- h_t: *hidden state vector*, che indica la memoria corrente.
- c_t: *cell state*, che indica la memoria a lungo termine.

L'architettura che abilita la capacità di *forget*, *save* e *focus* è il **gate**. Un gate **g** è un vettore della stessa lunghezza dell'*hidden state vector*; ogni elemento del gate è compreso tra 0 e 1. Solitamente il *gate* è calcolato a partire da una combinazione lineare dello stato nascosto con l'input corrente. In generale, un LSTM può usare differenti tipi di gate, ognuno per uno scopo preciso. Al tempo t è possibile determinare un gate **g** come segue:

$$g = \sigma(W \cdot x_t + U \cdot h_t + b)$$

W e U sono matrici peso e b il vettore di bias.

Per determinare ad ogni istante di tempo t quali aspetti della memoria a lungo termine bisogna trattenere, si calcola il *forget gate*:

$$F_t = \sigma(W_F \cdot x_t + U_F \cdot h_{t-1} + b_F)$$

Per stabilire quali elementi del vettore h_t bisogna salvare nella memoria a lungo termine, si definisce un *input gate*:

$$I_t = \sigma(W_I \cdot x_t + U_I \cdot h_{t-1} + b_I)$$

Segue l'aggiornamento del *cell state*:

$$c_t = F_t \circ c_{t-1} + I_t \circ \tanh\left(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h\right)$$

Per aggiornare la memoria corrente, invece, bisogna:

- determinare un *output gate*:

$$O_t = \sigma(W \cdot x_t + U_O \cdot O_{t-1} + b_O)$$

- applicare l'output gate alla memoria a lungo termine usando la funzione di attivazione tanh:

$$h_t = \tanh(c_t \circ O_t)$$

L'output al tempo t è dato dalla seguente espressione:

$$o_t = \sigma(V \cdot h_t + d)$$

dove σ è una funzione di attivazione, V una matrice dei pesi e d il vettore di bias.

Di seguito vengono riassunti alcuni lavori correlati a questa tesi.

2.4 Lavori correlati

- *An Efficient PointLSTM for Point Clouds Based Gesture Recognition* [1]. Questo lavoro considera i punti chiave delle mani come dei nodi di un grafo e combina le informazioni di stato dei punti vicini in passato con le caratteristiche attuali, per aggiornare gli stati tramite un livello LSTM.

- *Dynamic Hand Gesture Recognition Using 3DCNN and LSTM with FSMContext-Aware Model* [2]. Su questo lavoro una combinazione di una rete neurale convoluzionale tridimensionale (3DCNN) seguita dal modello LSTM è stata utilizzata per estrarre feature spazio-temporali. Viene inoltre utilizzata una Finite State Machine (FSM) per limitare alcuni flussi di gesti e per limitare il numero di classi da riconoscere. Riconoscere un numero piccolo di classi tende a mostrare una precisione maggiore.

- *Skeleton Based Dynamic Hand Gesture Recognition using LSTM and CNN* [3]. In questo articolo viene proposto un approccio basato su una rete neurale convoluzionale (CNN) e su una LSTM. Questo sistema è addestrato per analizzare sequenze di dati di input 3D insieme alle informazioni di velocità e posizione delle mani apprese dal Leap Motion Controller (LMC).

Va notato che, benché questi lavori siano correlati, essi differiscono da ciò che è stato fatto all'interno di questo studio perché utilizzano anche input video nella fase di training e valutazione del modello.

Capitolo 3

Acquisizione ed etichettatura dei dati

In questo terzo capitolo si discuterà dell'acquisizione e dell'etichettatura dei dati. In particolare, nella sezione 3.1 verrà descritta la fase di acquisizione dei dati. Nella sezione 3.2.1 verrà descritta l'etichettatura manuale. Questa garantisce sicuramente precisione, tuttavia quando bisogna etichettare una grande quantità di dati, questa non è la soluzione più efficiente. Proprio per questo motivo, nella sezione 3.2.2 verrà descritto il processo di etichettatura automatica dei dati a partire da comandi vocali impartiti durante l'acquisizione. Sono stati effettuati, inoltre, due studi, uno qualitativo e uno quantitativo, che mettono a confronto i due tipi di etichettatura. Nella sezione 3.3 verrà descritto il processo di costruzione del dataset e verranno mostrate delle statiche su di esso.

3.1 Acquisizione dei dati

3.1.1 Applicazione HoloLens2

Per acquisire i dati con HoloLens2 è stata utilizzata un'applicazione Unity sviluppata dal *PhD Student* **Rosario Leonardi**. Quest'ultima permette di acquisire i seguenti dati:

1. un video RGB acquisito dal punto di vista dell'utente;
2. le coordinate dei punti chiave delle mani e associati timestamp rilevati durante l'acquisizione;
3. delle etichette temporali indicanti l'occorrenza delle azioni svolte dall'utente.

Per avviare una nuova ripresa è necessario pronunciare il comando vocale: "avvia", il dispositivo dopo aver recepito il comando emetterà un segnale acustico per confermare l'inizio della ripresa. A questo punto il sistema genererà i seguenti file di log:

- *handPose3D*: in cui vengono salvate le coordinate dei punti chiave delle mani;
- *action*: in cui si tiene traccia delle azioni svolte da chi sta indossando HoloLens.

Mentre il salvataggio delle coordinate dei punti chiave avviene in maniera automatica (sfruttando delle apposite API di HoloLens), il salvataggio delle azioni richiede un supporto vocale, ovvero, qualche istante prima di effettuare un'azione, l'utente dovrà pronunciare una delle seguenti parole:

- *prendi*: nel caso in cui si volesse prendere un oggetto;
- *rilascia*: nel caso in cui si volesse rilasciare un oggetto;
- *clicca*: nel caso in cui si volesse premere su un oggetto.

Nel momento in cui comparirà un log visibile all'utente, come mostrato in Figura 3.1, allora potrà effettuare l'azione.

Figura 3.1: Log visibile all'utente (in basso a sinistra)

Per concludere una ripresa bisogna pronunciare la parola *"stop"*.

3.1.2 Struttura dei log

```
01-28-22-463
01-28-23-643 RIGHT Palm_x_0,1879202_y_-0,3933441_z_0,4217097
01-28-23-643 RIGHT ThumbTip_x_0,1306434_y_-0,359126_z_0
    ,3759811
01-28-23-643 RIGHT IndexTip_x_0,1128896_y_-0,3236867_z_0
    ,3954177
01-28-23-643 RIGHT MiddleTip_x_0,1028432_y_-0,3365593_z_0
    ,4202012
01-28-23-643 RIGHT RingTip_x_0,1073983_y_-0,3661949_z_0
    ,4363913
01-28-23-643 RIGHT PinkyTip_x_0,1219814_y_-0,378962_z_0
    ,4529103
```

File 3.1: HandPose3D log

```
01-28-22-463
01-28-27-813 Prendi
01-28-33-664 Prendi
01-28-41-364 Prendi
01-28-47-764 Rilascia
01-28-54-724 Rilascia
01-28-59-174 Premi
01-29-04-397 Premi
```

File 3.2: Action log

I File 3.1 e 3.2 mostrano la struttura dei file di log generati. Nella prima riga è riportato un timestamp che coincide con l'inizio della ripresa e corrisponde all'ora locale di HoloLens2.

Il file **handPose3D** contiene le posizioni dei joints delle mani secondo il seguente formato:

- timestamp, momento in cui viene salvato il log;
- mano;
- lista dei nomi dei joint e rispettive coordinate (separate da virgola).

Questi tre elementi sono separati da uno spazio.

Il file **action** presenta il seguente formato:

- timestamp, momento in cui l'utente visualizza il log a schermo su HoloLens, come mostrato in Figura 3.1;

- nome dell'azione.

Questi due elementi sono separati da uno spazio.

3.2 Etichettatura dei video

Dopo l'acquisizione dei video, è seguita l'etichettatura delle azioni, ovvero il processo attraverso il quale vengono individuate le occorrenze di ciascuna azione nei video acquisiti. Verrano presentati due metodi di etichettatura, manuale e automatica, e uno studio che mette a confronto le due metodologie.

3.2.1 Etichettatura manuale

L'etichettatura manuale è stata eseguita utilizzando il software VGG Image Annotator (VIA) che permette l'annotazione manuale di immagini, audio e video. Esso funziona in un browser web e non richiede alcuna installazione o configurazione. L'utilizzo approfondito di questo tool è presentato nell'appendice A.1.

Per etichettare i video utilizzando VIA, come inizio dell'etichetta è stato selezionato il frame nel momento in cui si vede (vedono) apparire la mano (le mani), come mostrato nelle Figure 3.2, 3.3, mentre come fine dell'etichetta è stato selezionato il momento in cui la mano (le mani) sta (stanno) per uscire dai margini del video, come mostrato nelle Figure 3.4, 3.5.

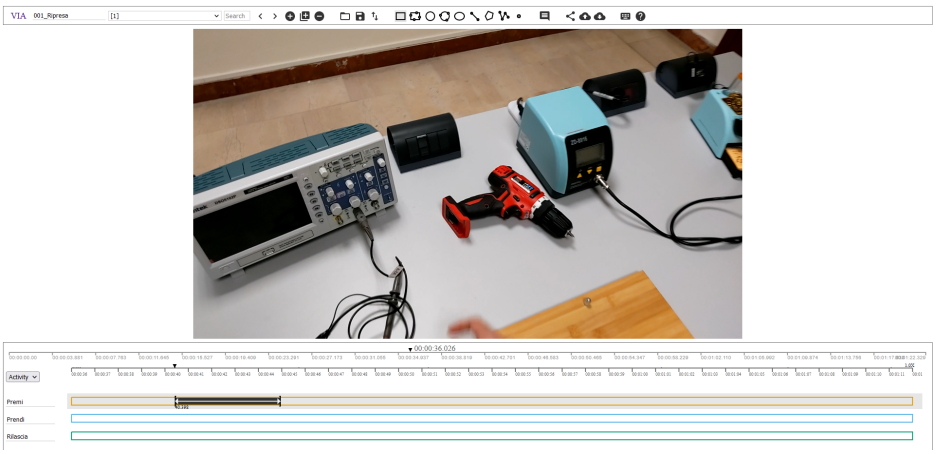

Figura 3.2: Esempio dell'inizio di un'etichetta con una mano

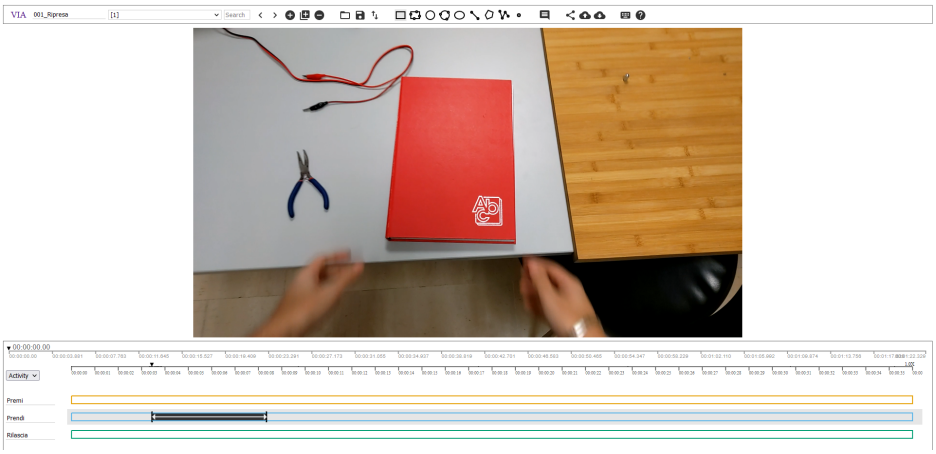

Figura 3.3: Esempio dell'inizio di un'etichetta con due mani

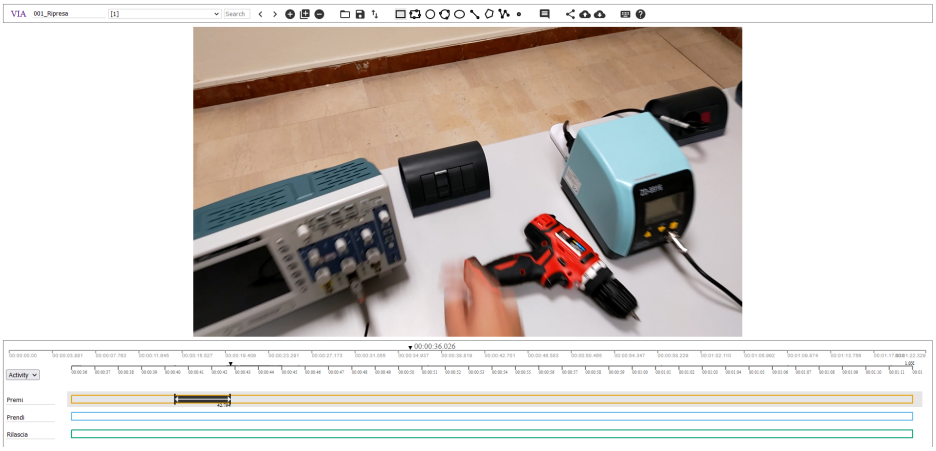

Figura 3.4: Esempio della fine di un'etichetta con una mano

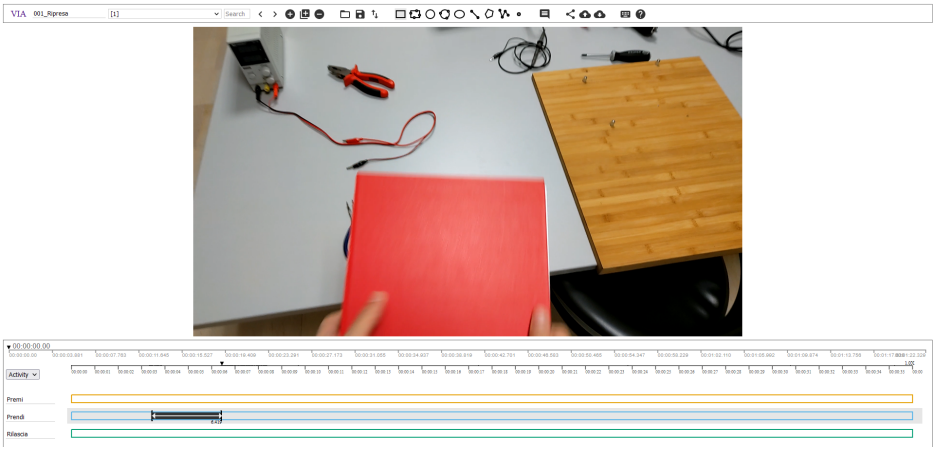

Figura 3.5: Esempio della fine di un'etichetta con due mani

3.2.2 Etichettatura automatica

Effettuare l'etichettatura manuale non è sempre una buona soluzione, ad esempio, per raggiungere l'obiettivo di questo studio, è necessario dover etichettare una grande quantità di video (che è dispendioso in termini di tempo e costi) ed è proprio in questi casi che bisogna trovare delle soluzioni alternative, come l'etichettatura automatica.

Per effettuare l'etichettatura automatica sono stati utilizzati i file *action*, descritti nella sezione 3.1.2, attraverso i quali è possibile determinare i frame da etichettare utilizzando i log dei comandi vocali. Attorno ad ogni frame estratto viene generato un segmento temporale di 6 secondi. Se x è il tempo di inizio del frame estratto dal file action, allora l'inizio e la fine dell'etichetta sono definite nel seguente modo:

$$\text{inizio} = x - 1500$$

$$\text{fine} = x + 4500$$

La Figura 3.6 mostra una timeline in cui si vede la posizione dell'annotazione iniziale e il segmento generato.

Figura 3.6: Timeline

3.2.3 Confronto tra etichettatura manuale e automatica

Aver etichettato in maniera corretta il video, significa avere dei buoni risultati nel momento in cui bisognerà identificare il tipo di azione: *push*, *release*, o *take*.
In questo paragrafo, verrano eseguiti due tipi di studi sull'etichettatura, uno **qualitativo** e uno **quantitativo**. L'obiettivo è capire se l'etichettatura automatica non differisce molto da quella manuale, ovvero sono simili.

L'etichettatura manuale garantisce precisione perché è effettuata da annotatori umani, quindi il confronto tra manuale e automatica serve per capire se anche l'automatica, con la quale si risparmia tempo, permette di ottenere una buona etichettatura dei video.

3.2.3.1 Studio qualitativo

Per effettuare lo studio qualitativo, sono state considerate le etichette automatiche e manuali di un video e riportate su VIA come mostrato in Figura 3.7.

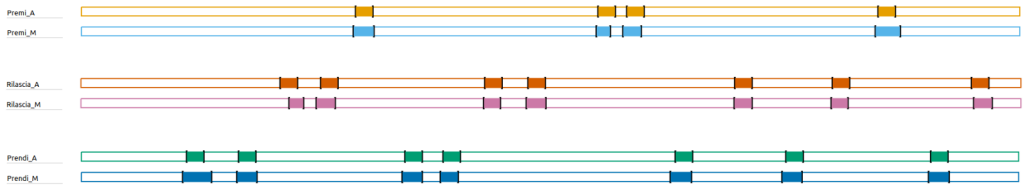

Figura 3.7: Azioni a confronto

Le etichette che terminano per `_A` sono state generate in maniera *automatica*, mentre le etichette che terminano con `_M` sono quelle *manuali*.
Dall'immagine sopra, possiamo osservare che alcune etichette automatiche sono simili a quelle manuali, ma nessuna (o quasi) coincide perfettamente. Di seguito verranno riportati dei dettagli delle etichette di alcune azioni.

Take

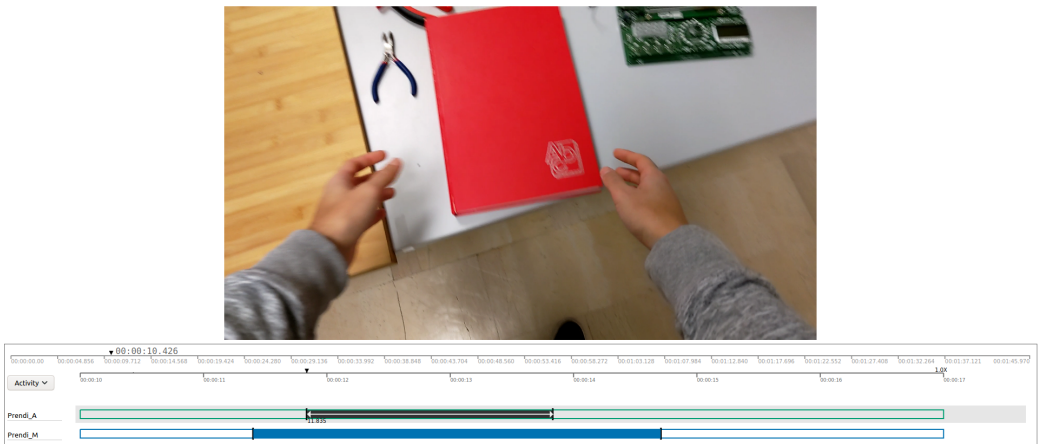

Figura 3.8: Inizio automatico di un'azione di Take

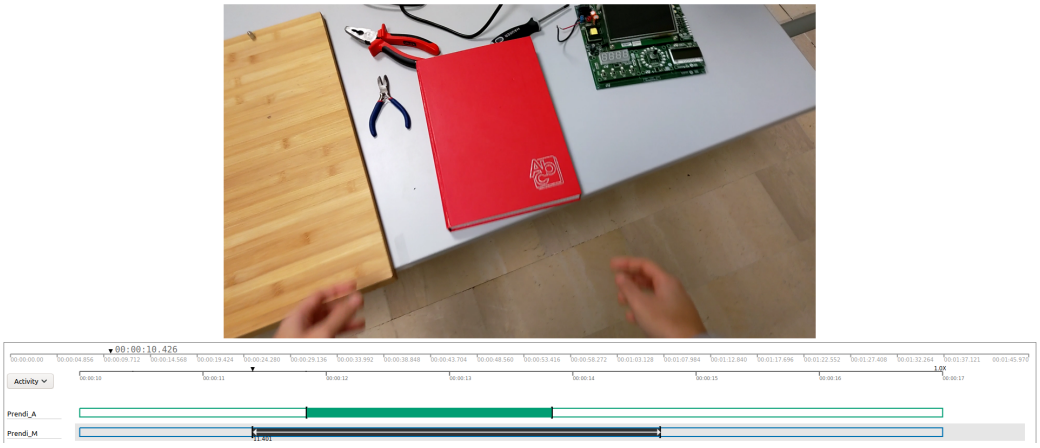

Figura 3.9: Inizio manuale di un'azione di Take

Nelle Figure 3.8, 3.9, i tempi iniziali delle due etichette non coincidono, ma nonostante ciò i frame sono molto simili tra di loro e si riesce a riconoscere in entrambi un'azione di *take*.

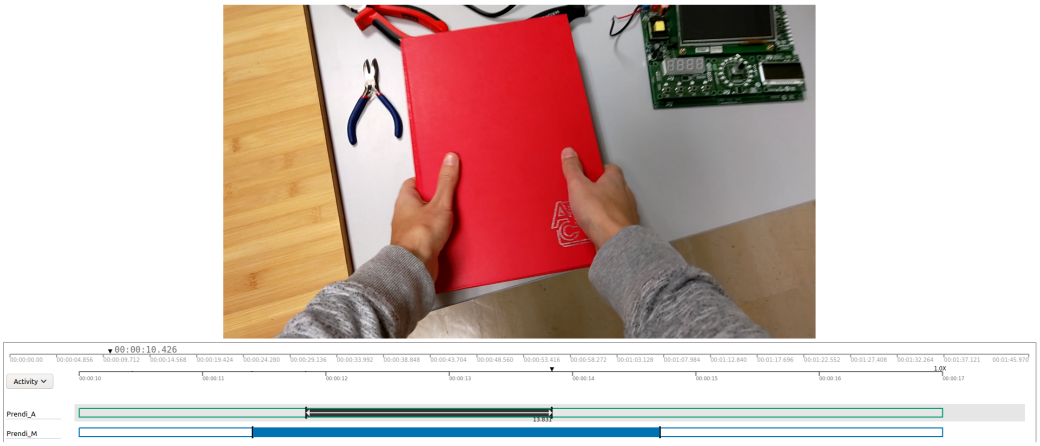

Figura 3.10: Fine automatica di un'azione di Take

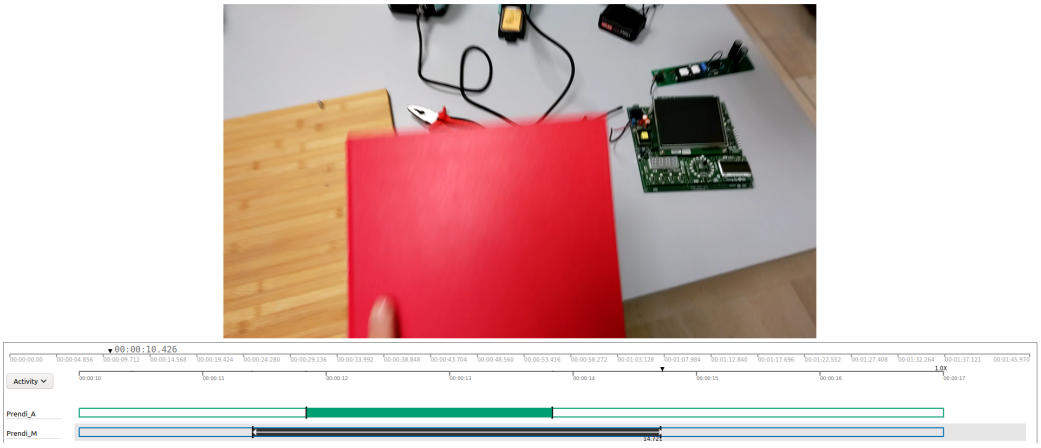

Figura 3.11: Fine manuale di un'azione di Take

Nelle Figure 3.10, 3.11, i tempi finali delle due etichette non coincidono, ma anche in questo caso nei frame è possibile riconoscere un'azione di *take*.

Release

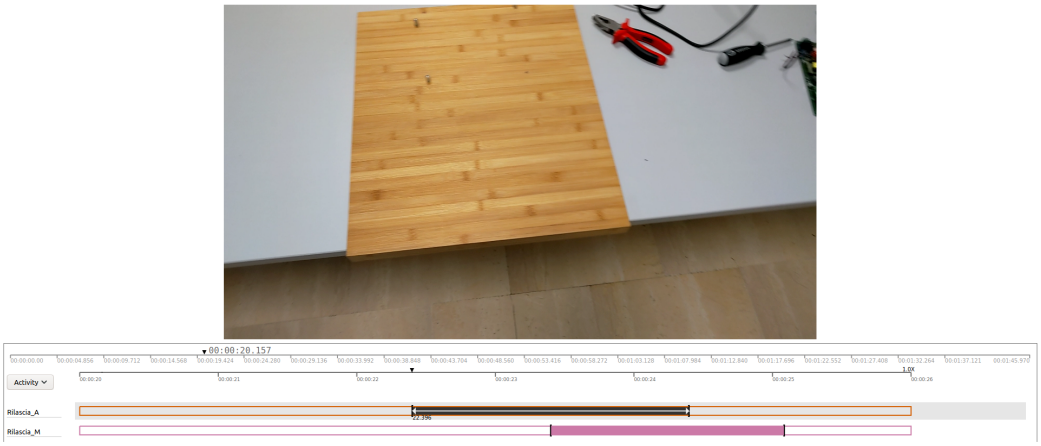

Figura 3.12: Inizio automatico di un'azione di Release

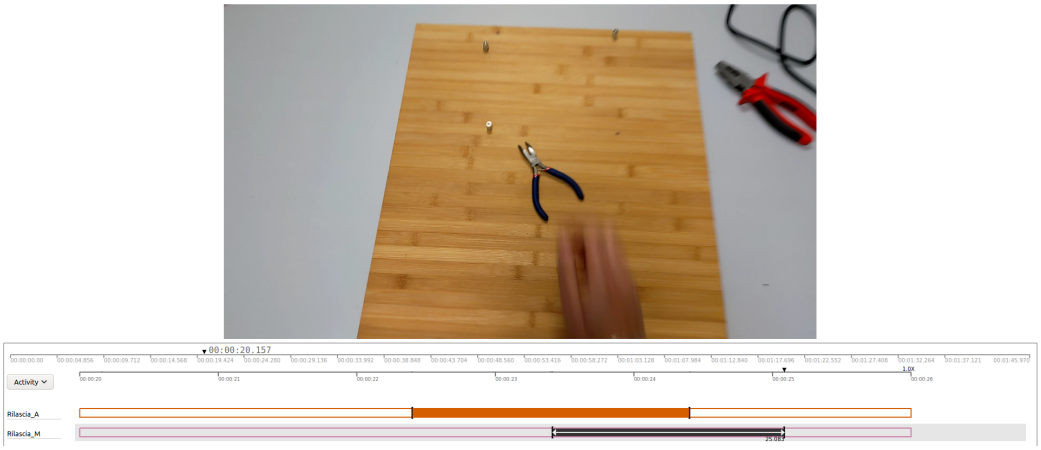

Figura 3.13: Inizio manuale di un'azione di Release

Nelle Figure 3.12, 3.13, possiamo osservare che tra l'inizio dell'etichettatura automatica e quella dell'etichettatura manuale c'è una differenza di circa 1 secondo. Inoltre, nel caso dell'etichettatura automatica, la scena risulta essere vuota.

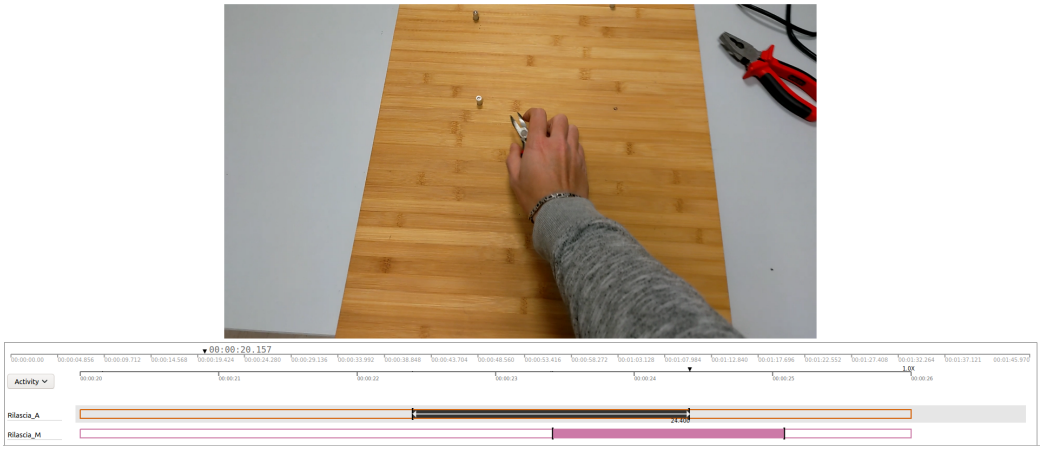

Figura 3.14: Fine automatica di un'azione di Release

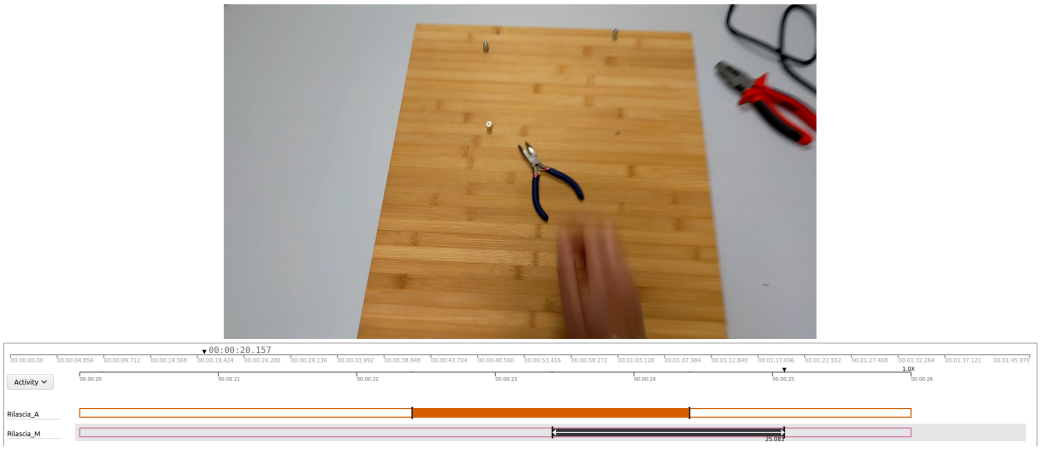

Figura 3.15: Fine manuale di un'azione di Release

Anche tra le due parti finali è presente una differenza di 1 secondo, ma quello che possiamo osservare è che, se si guardasse l'intera azione di entrambe le etichette, si potrebbe riconoscere un'azione di *release*.

Push

Figura 3.16: Inizio automatico di un'azione di Push

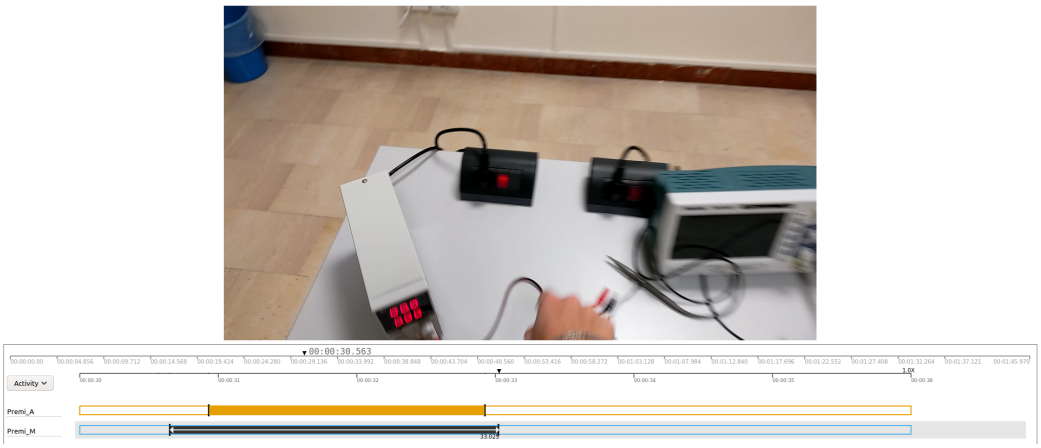

Figura 3.17: Inizio manuale di un'azione di Push

Figura 3.18: Fine automatica di un'azione di Push

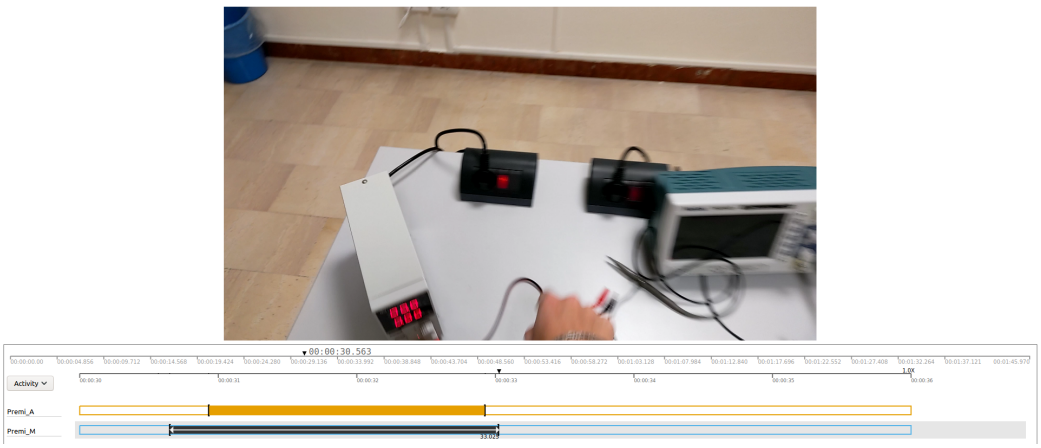

Figura 3.19: Fine manuale di un'azione di Push

Sia nelle parti iniziali, sia nelle parti finali, dei due tipi di etichettatura, si è in grado di riconoscere che si tratta di un'azione di *push*.

Conclusioni sullo studio qualitativo.

Quello che è stato riportato è lo studio di un solo video, ma lo stesso procedimento è stato applicato per altri video, qui non riportati per brevità. Osservando le etichette manuali e automatiche, possiamo concludere che esiste una certa differenza tra queste, ma nella maggior parte dei casi è minima. Quindi, è possibile concludere, in maniera qualitativa, che l'etichetta automatica può essere utilizzata.

3.2.3.2 Studio quantitativo

Per effettuare lo studio quantitativo, invece, verrà utilizzato l'**indice di Jaccard**, il quale misura la similarità tra insiemi campionari, ed è definito come la dimensione dell'intersezione divisa per la dimensione dell'unione degli insiemi campionari.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Supponiamo di avere le seguenti etichette per un'azione di *push*:

Figura 3.20: Etichette di un'azione di Push

dove:

- x_{11} rappresenta l'inizio dell'etichetta automatica;
- x_{12} rappresenta la fine dell'etichetta automatica;
- x_{21} rappresenta l'inizio dell'etichetta manuale;
- x_{22} rappresenta la fine dell'etichetta manuale.

L'intersezione (I) e l'unione (U) possono essere definite come segue:

$$I = \min(x_{12}, x_{22}) - \max(x_{11}, x_{21})$$

$$U = \max(x_{12}, x_{22}) - \min(x_{11}, x_{21})$$

Allora:

$$J(Premi_A, Premi_M) = \frac{|Premi_A \cap Premi_M|}{|Premi_A \cup Premi_M|} = \frac{I}{U} =$$
$$= \frac{\min(x_{12}, x_{22}) - \max(x_{11}, x_{21})}{\max(x_{12}, x_{22}) - \min(x_{11}, x_{21})}$$

Sfruttando questa formula è possibile calcolare la similarità tra le etichette automatiche e quelle manuali.

La Figura 3.21 mostra un Box Plot, che riporta la distribuzione delle similarità senza distinguere le azioni. La Figura 3.22 mostra un Box Plot, che riporta la distribuzione delle similarità distinguendo le azioni.

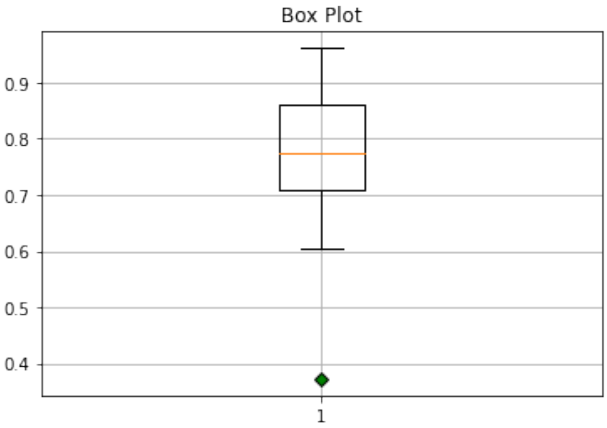

Figura 3.21: Distribuzione della similarità senza distinzione delle azioni

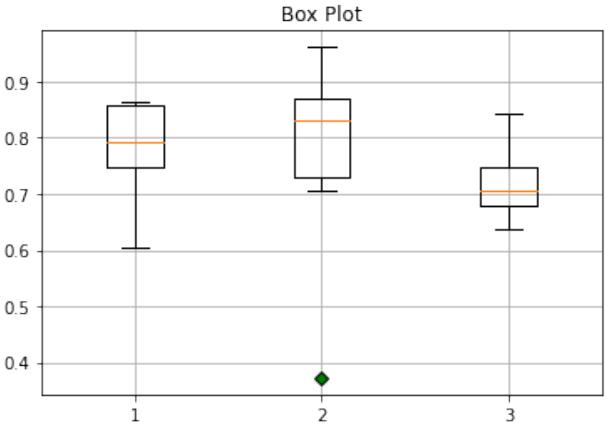

Figura 3.22: Distribuzione della similarità con distinzione delle azioni

Conclusioni sullo studio quantitativo.

Quello che è stato riportato nei Box Plot è lo studio fatto su un solo video, ma risultati simili sono stati ottenuti per altri video, qui non riportati per brevità.
Osservando i risultati, ricaviamo che ci sono pochi outliers, ovvero poche etichette automatiche che non sono simili a etichette manuali, ed, inoltre, i valori di similarità, in media, sono sempre superiori a 0.5.

3.2.3.3 Conclusioni sul confronto

Visti i due studi, qualitativo e quantitativo, si decide di utilizzare l'etichettatura automatica, che richiede meno tempo della manuale. Quest'ultima richiederebbe 5 minuti per etichettare una singola azione. Considerati i 56 video contenuti nel dataset e supponendo che ci siano 15 azioni per video, in totale sarebbero necessarie 70 ore per l'etichettatura manuale.

3.3 Dataset

Nella sezione 3.3.1 verranno presentate le statistiche sui video e sui soggetti. Nella sezione 3.3.2 verranno spiegati i passaggi che hanno portato alla costruzione del dataset e del modo in cui questo è stato suddiviso in *training set*, *test set* e *validation set*. Infine, nella sezione 3.3.3 verranno presentate le caratteristiche dei tre set.

3.3.1 Statistiche

La Figura 3.23 mostra la distribuzione delle azioni presenti su tutti i video.

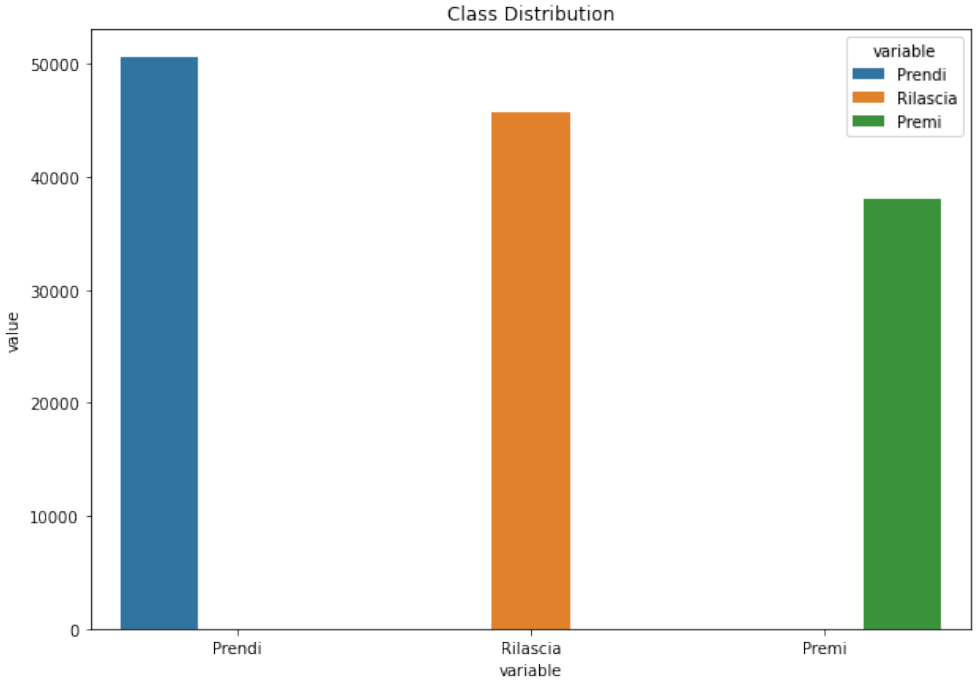

Figura 3.23: Distribuzione delle azioni su tutti i video

La Tabella 3.1 mostra le statistiche sui video.

Statistiche sui video				
Video totali	**Durata totale (min)**	**Durata media (min)**	**Durata minima (min)**	**Durata massima (min)**
56	72.95	1.30	0.45	3.09

Tabella 3.1: Tabella contenente le statistiche sui video

I soggetti in totale sono 9. Ognuno di essi ha effettuato delle riprese nei seguenti modi:

1. interagendo con oggetti muovendosi all'interno di una stanza (modalità 1);
2. interagendo con oggetti rimanendo seduti (modalità 2);
3. interagendo con oggetti presenti all'interno di uno scaffale (modalità 3).

La Figura 3.24 mostra il numero di riprese effettuate da ogni soggetto:

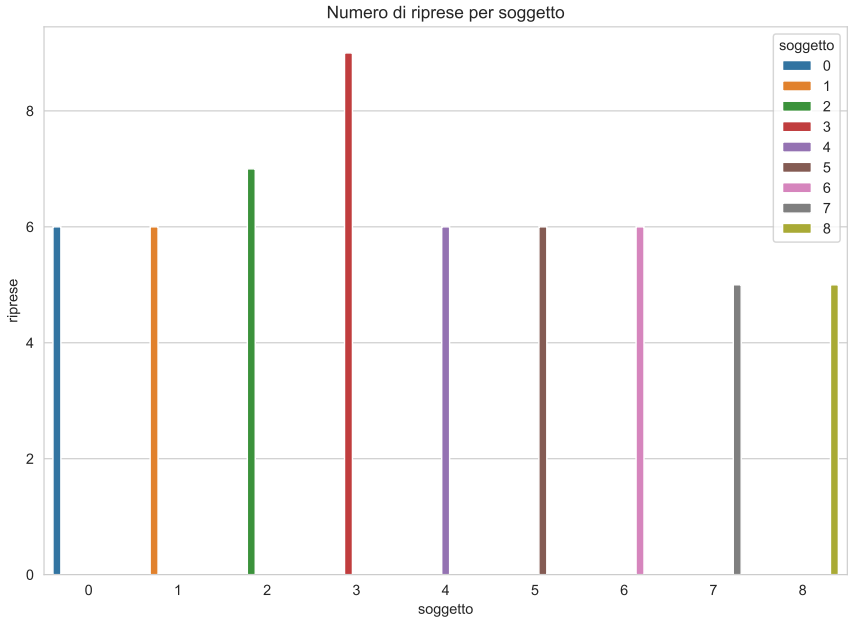

Figura 3.24: Numero di riprese per soggetto

La Tabella 3.2 mostra le statistiche sui soggetti.

Statistiche sui soggetti				
Soggetti totali	**Riprese totali**	**Riprese medie**	**Riprese minime**	**Riprese massime**
9	56	6.22	5	9

Tabella 3.2: Tabella contenente le statistiche sui soggetti

La Tabella 3.3 mostra il numero di riprese effettuate da ogni soggetto nelle 3 diverse modalità.

Soggetto	**Modalità 1**	**Modalità 2**	**Modalità 3**	**Totali**
0	2	2	2	6
1	2	2	2	6
2	3	2	2	7
3	3	3	3	9
4	2	2	2	6
5	2	2	2	6
6	2	2	2	6
7	2	2	1	5
8	2	2	1	5

Tabella 3.3: Tabella contenente il numero di riprese dei soggetti

3.3.2 Suddivisione in frame dei video

La suddivisione in frame rappresenta un procedimento importante, in quanto permette di suddividere i video in istanti discreti (frame) a ciascuno dei quali viene associata l'azione eseguita dall'utente o una etichetta di "background" che indica l'assenza di azione.
Utilizzando i file di log generati nella fase d'acquisizione precedente, sono stati creati dei file csv (uno per singolo video), così composti:

- 156 colonne che si riferiscono alle coordinate dei punti chiave della mano;
- 1 colonna che contiene le etichette delle azioni associate alle coordinate;
- ogni riga rappresenta un frame.

Per creare i file csv, ogni video è stato suddiviso in frame considerando finestre di 33 millisecondi. All'interno di ognuno di questi frame, ricadono un certo numero di coordinate dei punti chiave delle mani, di quest'ultime per ogni finestra verranno considerate solo quelle più recenti, ovvero quelle con il timestamp più grande. Ad esempio, se in un finestra di tempo ricadono $[\bar{a}, \bar{b}, \bar{c}]$, che sono dei vettori contenenti 156 elementi, tra questi si considera solo quello che nel file *handPose3D* presenta un timestamp maggiore. Per associare l'etichetta dell'azione alle coordinate delle mani, viene invece sfruttato il file contenente le etichette. In mancanza di un'azione viene attribuita

di default l'etichetta di background **No_action**. Quest'ultima viene attribuita anche a quei frame su cui non ricadono punti chiave, in questo caso si considerano tutte le coordinate pari a 0 (zero).

Basandosi sul procedimento descritto sopra, è stato realizzato uno script Python, in grado di automatizzare tutti i passaggi e di creare un file csv per ogni video. Un estratto del codice è riportato nel blocco 3.3.

```
def get_dataframe(self) -> None:
        for videos in self.__video_in_folder:
            for video in videos:
                records = copy.deepcopy(self.__hand_joints)
                video_duration:int = ExtractVideoDuration.
    get_duration(video)
                print(video[:-10] + "_handPose3D.txt")
                print(video[:-10] + "_action.json")
                hand_pose:HandPose = HandPose(video[:-10] + "
    _handPose3D.txt")
                self.__index_data = 0
                self.__create_records(
                    video_duration,
                    hand_pose.get_start_time(),
                    hand_pose.get_hand_pose_dict(),
                    records,
                    Target.get(video[:-10] + "_action.json",
    hand_pose.get_start_time())
                )
                pd.DataFrame(records, columns=self.
    __hand_joint_names).to_csv(video[:-3] + "csv", index=False
    )
```

Codice 3.3: Codice per effettuare il framing dei video

3.3.3 Training set, Test set e Validation set

I file csv ottenuti sono stati suddivisi in tre cartelle, al fine di ottenere un *training set*, un *test set* e un *validation set*.

Il training set è utilizzato per addestrare il modello alla classificazione delle azioni. Mentre il validation set e il training set sono utilizzati per valutare il modello.

La suddivisione dei file è stata effettuata in modo che nel *training set* sia

presente una certa varietà nello svolgimento delle azioni, i tre set finali sono così composti:

- 38 file csv per il training set;
- 10 file csv per il test set;
- 8 file csv per il validation set.

Le distribuzioni delle azioni nei tre set, alla fine del processo di suddivisione in frame, sono riportate nelle Figure 3.25, 3.26, 3.27

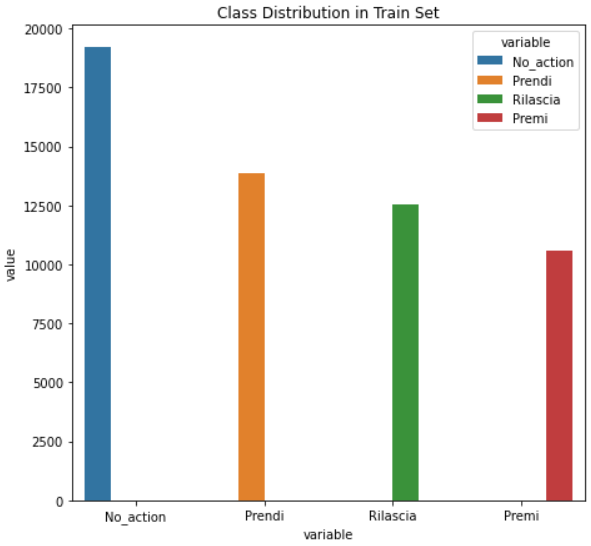

Figura 3.25: Training Set

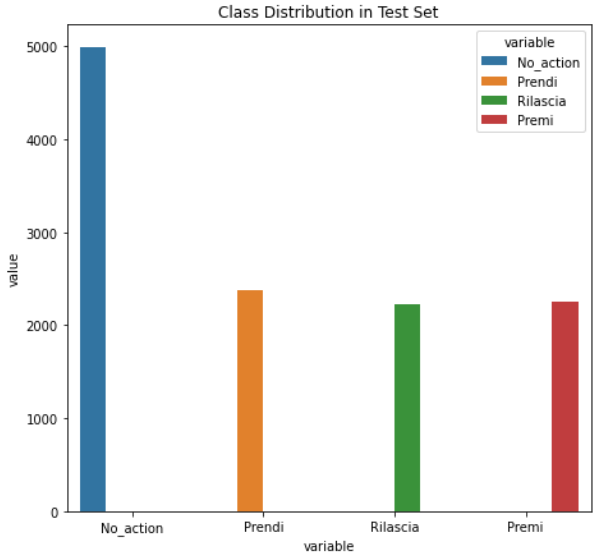

Figura 3.26: Test Set

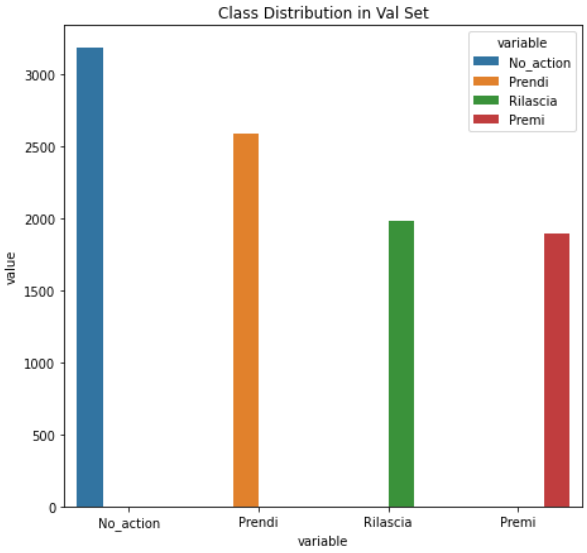

Figura 3.27: Validation Set

Capitolo 4

Algoritmi sviluppati

In questo capitolo verranno discussi gli algoritmi sviluppati. In particolare, nella sezione 4.1 verrà presentata l'implementazione dell'algoritmo proposto basato su **LSTM**, discussa nel capitolo 2, mentre nella sezione 4.2 verrà descritto come i tre set di dati, mostrati nella sezione 3.3.3, sono stati utilizzati.

4.1 Implementazione dell'algoritmo proposto

In Figura 4.1 è riportato lo schema di funzionamento dell'algoritmo proposto.

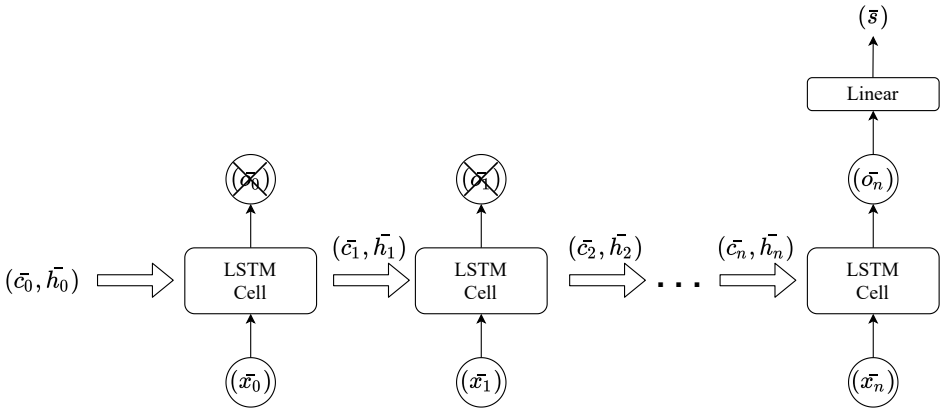

Figura 4.1: Schema dell'algoritmo proposto

La prima cella LSTM prende in input 3 vettori:

- *cell state*, indicato con $\bar{c_0}$, i cui valori sono nulli ($\bar{c_0} = \bar{0}$);
- *hidden state*, indicato con $\bar{h_0}$, i cui valori sono nulli ($\bar{h_0} = \bar{0}$);

- *input*, indicato con $\bar{x_0}$, che contiene 156 valori, che rappresentano le coordinate dei punti chiave delle mani.

Gli output sono:

- *cell state*, indicato con $\bar{c_1}$, che viene trasferito alla cella LSTM successiva;
- *hidden state*, indicato con $\bar{h_1}$, che viene trasferito alla cella LSTM successiva ed è anche l'output del livello, indicato con $\bar{o_1}$.

Nelle celle successive alla prima, in ingresso si avranno sempre 3 vettori:

- *cell state*, indicato con $\bar{c_t}$, i cui valori provengono dall'output della cella LSTM precedente;
- *hidden state*, indicato con $\bar{h_t}$, i cui valori provengono dall'output della cella LSTM precedente;
- *input*, indicato con $\bar{x_t}$, che contiene 156 valori, che rappresentano le coordinate dei punti chiave delle mani al tempo t.

Dallo schema del sistema è possibile notare come l'output di ogni cella, indicato con $\bar{o_t}$, venga totalmente scartato con l'unica eccezione dell'ultima cella, quest'ultimo è utilizzato come input per un layer lineare. L'output di questo layer, indicato con $\bar{s}$, è un vettore contenente 4 valori, che fanno riferimento alle quattro possibili classi: *No_action*, *Prendi*, *Rilascia* e *Premi*. Per implementare il sistema è stata utilizzata la libreria **PyTorch**, il codice dell'implementazione è mostrato nel blocco 4.1.

```
1  class Model(nn.Module):
2      def __init__(self, input_size, output_size,
       hidden_layer_size, num_layers):
3          super(Model, self).__init__()
4          self.num_layers = num_layers
5          self.hidden_size = hidden_layer_size
6          self.output_size = output_size
7          self.lstm = nn.LSTM(
8              input_size,
9              hidden_layer_size,
10             num_layers
11         )
12         self.linear = nn.Linear(
13             hidden_layer_size,
14             output_size
15         )
```

```
16
17     def forward(self, x, hidden=None):
18         if hidden is not None:
19             h0 = hidden[0]
20             c0 = hidden [1]
21         else:
22             h0 = torch.zeros(self.num_layers, x.size()[0],
       self.hidden_size).to(device)
23             c0 = torch.zeros(self.num_layers, x.size()[0],
       self.hidden_size).to(device)
24         e = x.view(x.size(1), x.size(0), x.size(2))
25         o, hn = self.lstm(e, (h0, c0))
26         o = o.view(o.size(1), o.size(0), o.size(2))
27         o = o[:,-1,:]
28         s = self.linear(o)
29         return s, hn
```

Codice 4.1: Codice che implementa il modello proposto

I parametri utilizzati per l'inizializzazione della LSTM sono:

- *input_size*: il numero di elementi d'input, che corrispondono agli elementi del vettore $\bar{x}$;
- *output_size*: il numero di elementi di output: $\bar{s}$ (corrisponde al numero di classi);
- *hidden_layer_size*: il numero di features nello stato nascosto $\bar{h}$;
- *num_layers*: il numero di strati ricorrenti, ovvero il numero di celle LSTM presenti nel sistema.

In particolare la modifica degli ultimi due parametri, che verrà descritta nel capitolo 5, è stata fondamentale per ottimizzare le performance del modello.

4.2 Sequenze temporali

Il modello definito permette di classificare delle sequenze temporali. In particolare si è scelto di lavorare con finestre di lunghezza fissa, per cui è necessario suddividere le lunghe sequenze del dataset in sequenze più brevi mediante una finestra mobile.
I tre set che sono stati ricavati dalla fase di suddivisione in frame, descritta nella sezione 3.3.2, sono costituiti da un insieme di file csv. Per utilizzare questi dati, è stato definito un parametro denominato **window_size**, che rappresenta il numero di righe da selezionare per costruire una sequenza temporale.

Ad ogni sequenza costruita verrà associata la classe più frequente all'interno della finestra, ad esempio si supponga che il parametro **window_size** sia pari a 4 e che una sequenza $\bar{x}$ sia costituita da $[\bar{a}, \bar{b}, \bar{c}, \bar{d}]$. Le etichette di questi quattro vettori siano rispettivamente uguali a $[0, 1, 1, 2]$, allora l'etichetta della sequenza temporale è 1. Le sequenze generate attraverso il parametro window_size costituiscono l'input del sistema.
Questa è una fase di pre-processing necessaria per ottenere dati di training e test. La Figura 4.2 mostra il funzionamento di un sistema allenato con i dati ottenuti considerando window_size pari a 4. In Figura 4.2 si può osservare una time-line che indica un video e come istante per istante si campiona una diversa sequenza temporale che viene passata al modello ottenendo la predizione dell'azione.

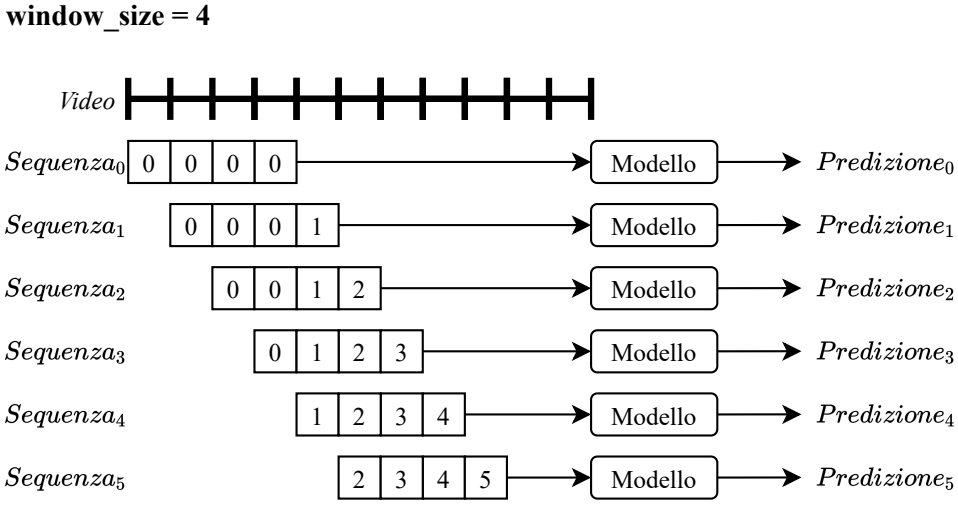

Figura 4.2: Funzionamento a test-time con window_size pari a 4

4.2.1 Pre-processing del training set

Il training set è stato pre-processato in due modi differenti:

1. le sequenze temporali sono state campionate mediante la finestra mobile in modo da non essere sovrapposte, come mostrato in Figura 4.3, tranne in un caso specifico che verrà mostrato in seguito;

2. le sequenze temporali sono state sovrapposte, come mostrato in Figura 4.4.

Le time-line nelle Figure 4.3 e 4.4 indicano i frame di un video.

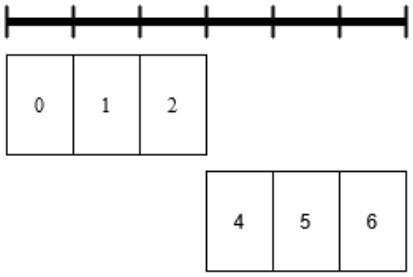

Figura 4.3: Sequenze temporali senza sovrapposizione

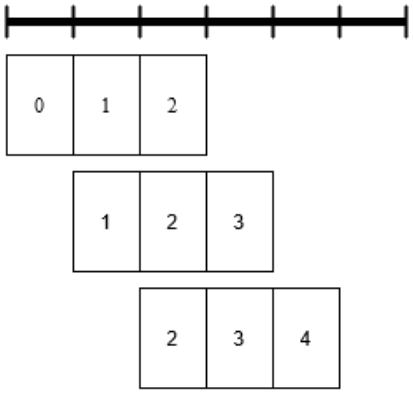

Figura 4.4: Sequenze temporali con sovrapposizione

Metodo in assenza di sovrapposizione

Ogni riga di un file csv rappresenta un intervallo temporale di 33 millisecondi. Impostando, ad esempio, il parametro window_size a 5 verranno generate sequenze di cinque elementi che coprono un intervallo temporale pari a: $5 \cdot 33$ ms $= 165$ ms. Inoltre, ogni sequenza generata sarà composta da

frame consecutivi campionati da un singolo video. In Figura 4.5 è mostrato un esempio di campionamento di sequenze da vari video (file csv) con window_size pari a 2.

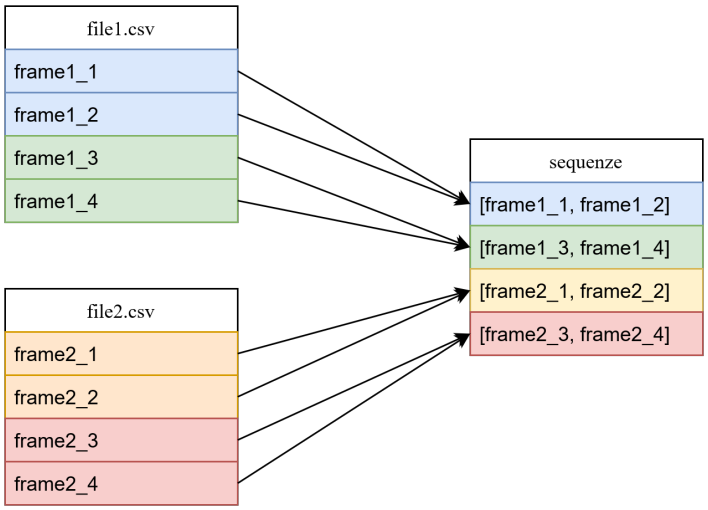

Figura 4.5: Esempio con window_size pari a 2

Considerando l'esempio precedente e supponendo che il **file1** contenga una quinta riga, adottando questa procedura quest'ultima verrebbe scartata. Una delle strategie per evitare tale perdita è utilizzare le righe precedenti fino al riempimento della sequenza, come mostrato in Figura 4.6. Questo è l'unico caso di sovrapposizione.

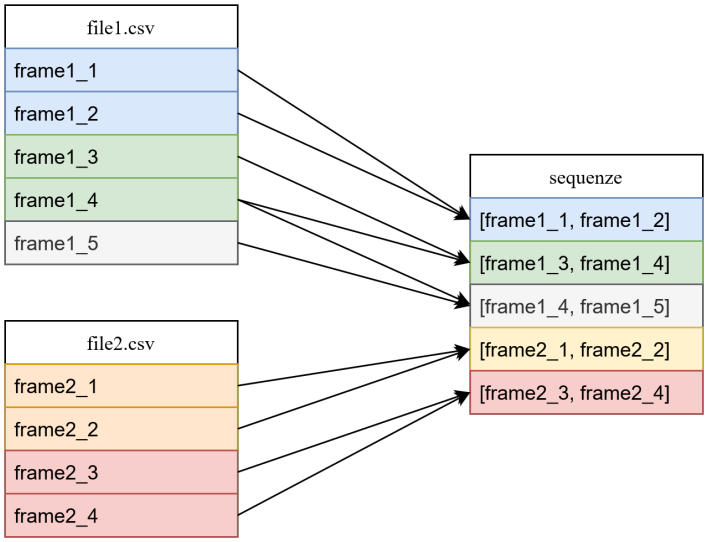

Figura 4.6: Esempio con window_size pari a 2 di un caso particolare

```
def create_sequence_train(array_data, label_data, window):
    X = []
    files_num = len(array_data)
    for index, array, label in zip(range(files_num),
    array_data, label_data):
        l = array.shape[0]
        i = 0
        while l - i >= window:
            j = i + window
            index_window = np.array([index, i, j]).astype(int)
            X.append(index_window)
            i += window
        if i < l:
            w = l - i
            i = i - w - window
            j = i + window
            index_window = np.array([index, i, j]).astype(int)
        X.append(index_window)
    X_data = torch.from_numpy(np.array(X).astype(int))
    return ClassifierDataset(array_data, label_data, X_data,
    window)
```

Codice 4.2: Codice per dividere il training set in sequenze senza sovrapposizione

Il Codice 4.2 riporta la funzione che permette di suddividere il dataset utilizzando il metodo senza sovrapposizioni.

Metodo con sovrapposizione

Questo metodo permette di avere un maggior numero di dati durante la fase di training, come mostrato in Figura 4.7, che riporta l'esempio visto in precedenza, ma con sovrapposizione.

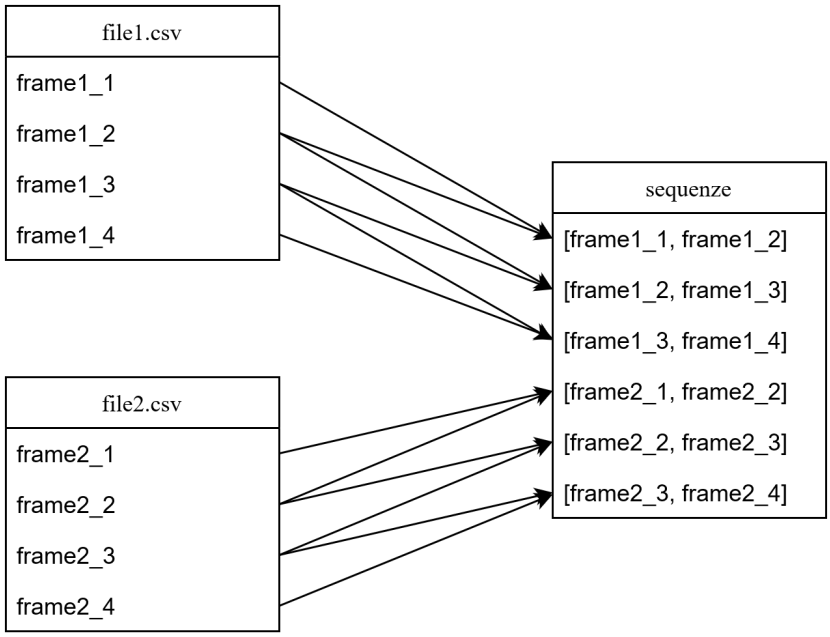

Figura 4.7: Esempio con window_size pari a 2

Il blocco di Codice 4.3 riporta l'implementazione di questo secondo metodo.

```
def create_sequence_train_overlap(array_data, label_data, window):
    X = []
    files_num = len(array_data)
    for index, array, label in zip(range(files_num), array_data, label_data):
        l = array.shape[0]
        i = 0
        while l - i >= window:
            j = i + window
            index_window = np.array([index, i, j]).astype(int)
            X.append(index_window)
            i += 1
    X_data = torch.from_numpy(np.array(X).astype(int))
    return ClassifierDataset(array_data, label_data, X_data, window)
```

Codice 4.3: Codice per dividere il training set in sequenze con sovrapposizione

Il numero di sequenze temporali che si possono ricavare con il primo metodo da un singolo file, può essere calcolato attraverso la seguente formula:

$$\#sequenze\_temporali = \left\lfloor \frac{n}{k} \right\rfloor + \left[\!\left[\frac{n}{k} - \left\lfloor \frac{n}{k} \right\rfloor > 0 \right]\!\right]$$

Dove n è il numero di righe del file, k è il valore di **window_size** e le doppie parentesi quadre indicano le *Iverson Brackets*.

Il numero di sequenze temporali che si possono ricavare con il secondo metodo da un singolo file, può essere calcolato attraverso la seguente formula:

$$\#sequenze\_temporali = \frac{1}{2}\binom{n}{k} = \frac{n!}{2k!(n-k)!}$$

Dove n è il numero di righe del file e k è il valore di **window_size**.

4.2.2 Pre-processing del test set e del validation set

Come fatto per il training set, anche per il test set e per il validation set bisogna costruire delle sequenze temporali. Per questi due set non vengono utilizzate le tecniche viste nella sezione precedente, poiché nella valutazione dei vari modelli allenati è utile testare quest'ultimi utilizzando lo stesso test e validation set (a parità di window_size), indipendentemente dal modo in cui è stato suddiviso il training set. Basti osservare gli esempi precedenti per capire che, a parità di finestra, utilizzando le strategie precedenti si avrebbero dimensioni differenti.

Per ovviare a questo problema è stata proposta la seguente strategia di generazione delle sequenze:

1. si consideri un file csv e una lista di lunghezza pari a **window_size**;
2. tutti i valori della lista sono inizializzati con il valore presente nella prima riga del file, come mostrato in Figura 4.8;
3. ad ogni passo successivo, il primo elemento della lista viene eliminato, mentre in coda alla lista viene aggiunta la riga successiva del file, come mostrato in Figura 4.9.

Le liste di elementi costruite ad ogni istante di tempo t, costituiscono le sequenze temporali utilizzate per valutare il modello, come mostrato in Figura 4.10.

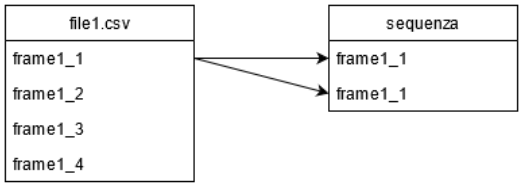

Figura 4.8: Esempio di inizializzazione con window_size pari a 2

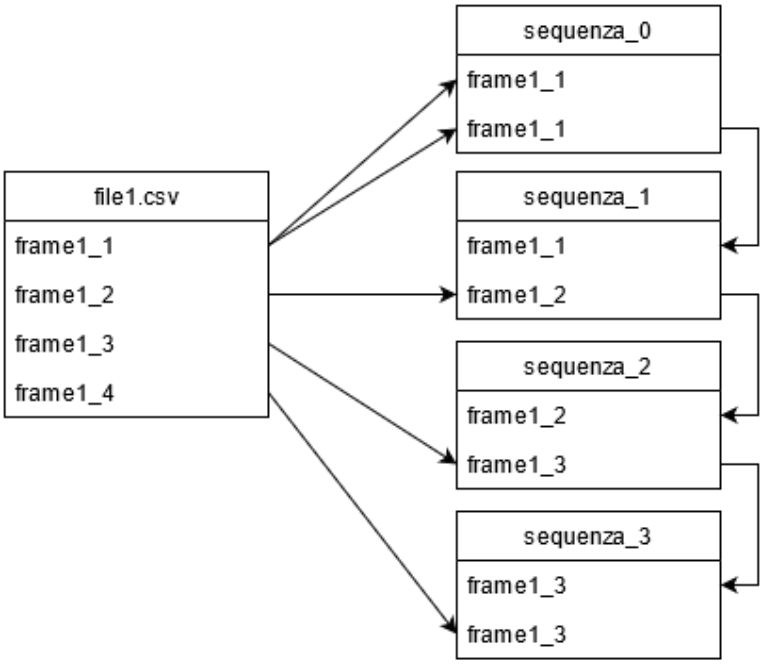

Figura 4.9: Esempio con window_size pari a 2

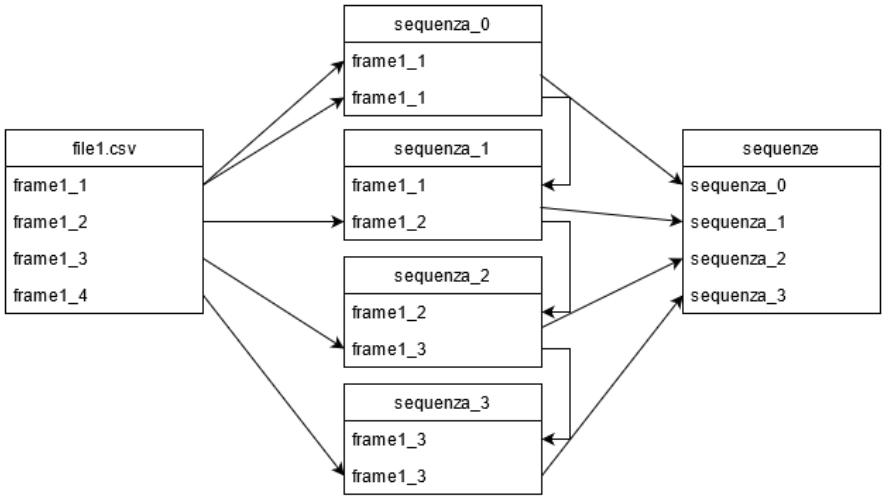

Figura 4.10: Esempio con window_size pari a 2

Il blocco di Codice 4.4 riporta l'implementazione della suddivisione del test set o del validation set.

```
def create_sequence_test_val(array_data, label_data, window):
    X = []
    files_num = len(array_data)
    for index, array, label in zip(range(files_num),
    array_data, label_data):
        l = array.shape[0]
        i = -1
        w = [0] * window
        while i < l:
            i += 1
            w.pop(0)
            w.append(i)
            index_window = np.array([index, w[0], i]).astype(
    int)
            X.append(index_window)
    X_data = torch.from_numpy(np.array(X).astype(int))
    return ClassifierDataset(array_data, label_data, X_data,
    window)
```

Codice 4.4: Codice per dividere il test set o il validation set in sequenze

4.3 Parametri da ottimizzare

La Tabella 4.1 mostra i parametri da ottimizzare.

Parametri da ottimizzare				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**

Tabella 4.1: Tabella contenente i parametri da ottimizzare

I parametri vengono descritti di seguito:

- **window_size**: rappresenta il numero di elementi contenuti all'interno di una sequenza temporale;
- **batch size**: si riferisce al numero di sequenze temporali utilizzate in fase di addestramento in una singola iterazione;
- **learning rate**: determina la rapidità con cui si aggiornano i parametri;
- **numero di layer**: il numero di strati ricorrenti, ovvero il numero di celle LSTM presenti nel sistema;
- **hidden layer size**: rappresenta il numero di features nello stato nascosto $\bar{h}$.

Capitolo 5

Risultati

In questo capitolo verranno descritti gli esperimenti effettuati e discussi i risultati ottenuti. Inizialmente nella sezione 5.1 verranno descritte le metriche utilizzate nel processo di valutazione delle performance dei modelli. Nella sezione 5.2 saranno invece mostrati gli esperimenti effettuati utilizzando il metodo proposto al variare dei parametri descritti nella sezione 4.3. Infine, nella sezione 5.3 verranno riportati degli esperimenti supplementari su delle semplificazioni del task iniziale.

5.1 Misure di valutazione

Nel processo di valutazione delle performance di un modello, la scelta delle misure di valutazione da utilizzare ricopre un ruolo fondamentale.

Le misure di valutazione utilizzate sono le seguenti:

- *Precision*: definita come il numero di **true positive** diviso il numero di **true positive** più il numero di **false positive**.

$$\text{Precision} = \frac{tp}{tp + fp}$$

  Considerata un'azione, la precision misura la percentuale delle sequenze temporali classificate con l'azione considerata che sono effettivamente appartenenti a quella classe.

- *Recall*: definita come il numero di **true positive** diviso il numero di **true positive** più il numero di **false negative**.

$$\text{Recall} = \frac{tp}{tp + fn}$$

La recall misura la frazione delle sequenze temporali appartenenti ad una data classe che sono state classificate correttamente.

- *F1-Score*: è una misura che combina la precision e la recall del modello, ed è definita come la media armonica di precision e recall.

$$\text{F1-Score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Avere valori alti di F1-Score per una determinata azione significa che i **false positive** e i **false negative** dell'azione considerata hanno valori bassi.

- *Matrice di confusione*: per avere una visione più completa delle performance dei modelli utilizziamo la matrice di confusione (**confusion matrix**). Nella matrice di confusione le righe indicano le etichette vere, mentre le colonne indicano le etichette previste. Se si considerano ad esempio M classi, la matrice di confusione ha dimensione $M \times M$ e il valore C_{ij} indica il numero o la percentuale di elementi appartenenti alla classe i, che sono stati classificati come appartenenti alla classe j. Quindi nella matrice di confusione ci si aspetta di avere valori grandi sulla diagonale e valori piccoli in tutte le altre celle. La Figura 5.1 mostra un esempio di matrice di confusione con tre classi.

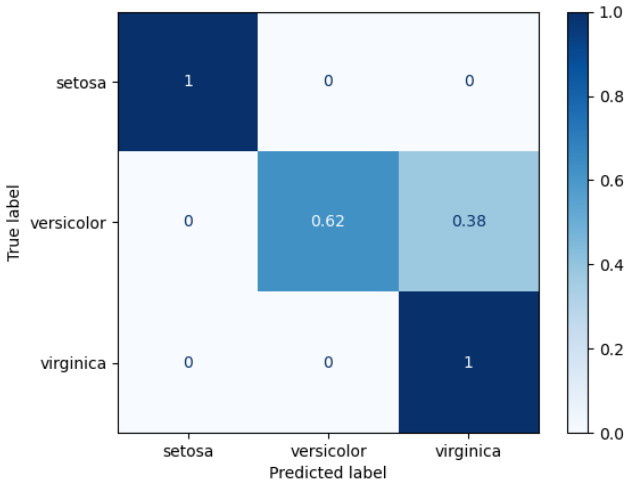

Figura 5.1: Matrice di confusione con tre classi

5.2 Esperimenti

Gli esperimenti sono stati effettuati utilizzando le due modalità di pre-processamento dei dati del training set discusse nel capitolo precedente. Nella sezione 5.2.1 sono discussi 5 esperimenti con la modalità di pre-processamento del training set in sequenze senza sovrapposizione. Nella sezione 5.2.2 sono discussi 5 esperimenti con la modalità di pre-processamento del training set in sequenze sovrapposte. In entrambe le sezioni, l'ultimo esperimento è quello con i risultati migliori. Nella sezione 5.2.3 verranno discussi i risultati ottenuti e descritti i problemi riscontrati del metodo proposto.

5.2.1 Sequenze non sovrapposte

Esperimento n. 1

In questo esperimento è stato impostato il valore del parametro *window_size* pari a 182 frame per studiare il comportamento del modello con una finestra temporale pari a 6 secondi. I valori dei parametri *batch size*, *learning rate*, *numero di layer* e *hidden layer size* sono stati impostati in maniera arbitraria. La Tabella 5.1 mostra i parametri utilizzati nell'esperimento 1.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
182	64	0.004	1	32

Tabella 5.1: Tabella contenente i parametri dell'esperimento 1

La Figura 5.2 mostra la matrice di confusione dell'esperimento 1.

Figura 5.2: Matrice di confusione dell'esperimento 1

La Tabella 5.2 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.72	0.14	0.27	0.29	0.36
Precision	0.75	0.36	0.36	0.20	0.42
Recall	0.70	0.09	0.21	0.48	0.37

Tabella 5.2: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che:

- diverse azioni di classe *take* sono confuse con azioni di classe *push* o con *No_action*;
- diverse azioni di classe *release* sono confuse con azioni di classe *push* o con *No_action*;
- diverse azioni di *push* sono confuse con *No_action*.

Inoltre, osservando la prima colonna della matrice di confusione è possibile dedurre che il modello non riesce a distinguere le sequenze temporali in cui si sta effettuando un'azione dalle sequenze temporali in cui non si effettuano azioni. Queste limitazioni del modello sono dovute al fatto che il valore del parametro *window_size* è grande.
Dalla Tabella 5.2 si osserva che i valori di F1 Score delle azioni sono bassi, questo significa che i **false positive** e i **false negative** delle azioni sono molto numerosi.

Esperimento n. 2

In questo esperimento è stato impostato il valore del parametro *window_size* pari a 33 frame per studiare il comportamento del modello con una finestra temporale pari a 1 secondo, più piccola dell'esperimento precedente. I valori dei parametri *batch size*, *learning rate*, *numero di layer* e *hidden layer size* sono gli stessi dell'esperimento 1. La Tabella 5.3 mostra i parametri utilizzati nell'esperimento 2.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
33	64	0.004	1	32

Tabella 5.3: Tabella contenente i parametri dell'esperimento 2

La Figura 5.3 mostra la matrice di confusione dell'esperimento 2.

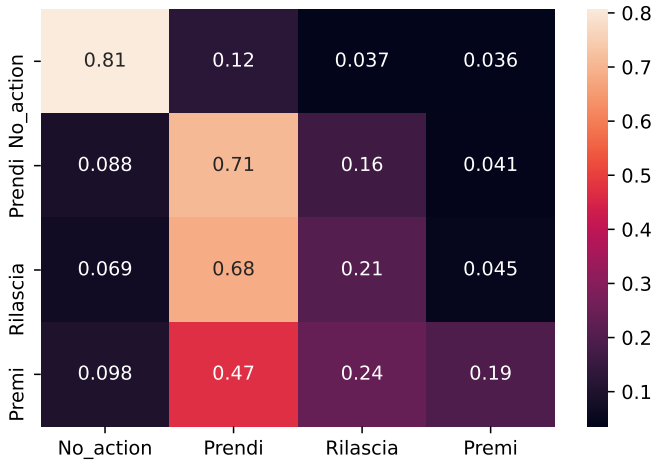

Figura 5.3: Matrice di confusione dell'esperimento 2

La Tabella 5.4 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.84	0.46	0.24	0.28	0.46
Precision	0.87	0.35	0.29	0.54	0.51
Recall	0.81	0.71	0.21	0.19	0.48

Tabella 5.4: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che:

- diverse azioni di classe *push* sono confuse con azioni di classe *take*;
- diverse azioni di classe *release* sono confuse con azioni di classe *take*.

Inoltre, osservando la prima colonna della matrice di confusione è possibile dedurre che il modello riesce a distinguere le sequenze temporali in cui si sta effettuando un'azione dalle sequenze temporali in cui non si effettuano azioni.
Dalla Tabella 5.4 si osserva che i valori di F1 Score sono migliorati per tutte le classi. Questo significa che utilizzare un valore del parametro *window_size* più piccolo permette di ottenere un modello con migliori performance.

Esperimento n. 3

In questo esperimento è stato impostato il valore del parametro *window_size* pari a 6 frame per studiare il comportamento del modello con una finestra temporale pari a 198 millisecondi, più piccola degli esperimenti precedenti. Il valore del parametro *batch size* è impostato a 8 per capire se il modello migliora quando i pesi della rete neurale in fase di addestramento si aggiornano utilizzando un numero più piccolo di sequenze temporali. Per provare ad aumentare ulteriormente le performance del modello il valore del parametro *numero di layer* è impostato a 3, ovvero si hanno 3 celle LSTM, e il valore del parametro *hidden layer size* è impostato a 184. Il valore del parametro *learning rate* è impostato in maniera da adattarsi ai cambiamenti degli altri parametri. La Tabella 5.5 mostra i parametri utilizzati nell'esperimento 3.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
6	8	0.007	3	184

Tabella 5.5: Tabella contenente i parametri dell'esperimento 3

La Figura 5.4 mostra la matrice di confusione dell'esperimento 3.

Figura 5.4: Matrice di confusione dell'esperimento 3

La Tabella 5.6 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.88	0.43	0.52	0.48	0.58
Precision	0.91	0.46	0.41	0.68	0.62
Recall	0.86	0.41	0.69	0.38	0.59

Tabella 5.6: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che:

- diverse azioni di classe *take* sono confuse con azioni di classe *release*;
- diverse azioni di classe *push* sono confuse con azioni di classe *release*.

Inoltre, osservando la prima colonna della matrice di confusione è possibile dedurre che il modello riesce a distinguere le sequenze temporali in cui si sta effettuando un'azione dalle sequenze temporali in cui non si effettuano azioni.
Dalla Tabella 5.6 si osserva che i valori di F1 Score sono migliorati per tutte le classi, questo significa che diminuire i valori dei parametri *window_size* e *batch size*, aumentare i valori dei parametri *numero di layer* e *hidden layer size* permette di ottenere un modello con migliori performance.

Esperimento n. 4

Per migliorare la capacità del modello nel distinguere le azioni è stato ulteriormente diminuito il valore del parametro *window_size*, ma aumentato il valore del parametro *batch size* per capire se il modello migliora quando i pesi della rete neurale in fase di addestramento si aggiornano utilizzando un numero più grande di sequenze temporali. Per provare ad aumentare ulteriormente le performance del modello il valore del parametro *numero di layer* è impostato a 2, ovvero si hanno 2 celle LSTM, e il valore del parametro *hidden layer size* è impostato a 192. Il valore del parametro *learning rate* è impostato in maniera da adattarsi ai cambiamenti degli altri parametri. La Tabella 5.7 mostra i parametri utilizzati nell'esperimento 4.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
5	16	0.0099	2	192

Tabella 5.7: Tabella contenente i parametri dell'esperimento 4

La Figura 5.5 mostra la matrice di confusione dell'esperimento 4.

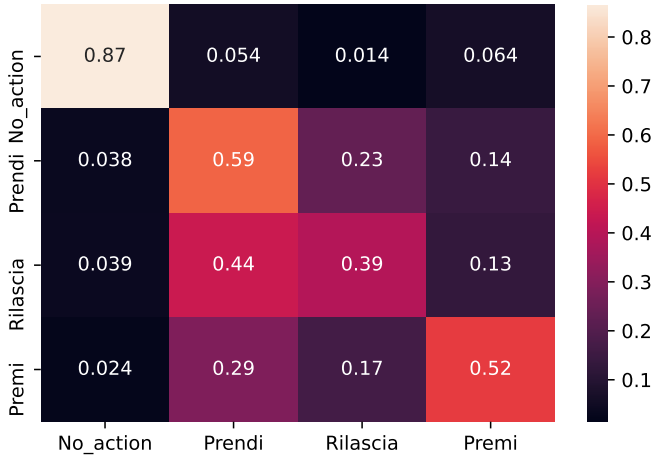

Figura 5.5: Matrice di confusione dell'esperimento 4

La Tabella 5.8 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.91	0.49	0.43	0.54	0.59
Precision	0.95	0.42	0.47	0.55	0.60
Recall	0.87	0.59	0.39	0.52	0.59

Tabella 5.8: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che:

- diverse azioni di classe *take* sono confuse con azioni di classe *release*;
- diverse azioni di classe *release* sono confuse con azioni di classe *take*;
- diverse azioni di classe *push* sono confuse con la classe *take* oppure con la classe *release*.

Inoltre, osservando la prima colonna della matrice di confusione è possibile dedurre che il modello riesce a distinguere le sequenze temporali in cui si sta effettuando un'azione dalle sequenze temporali in cui non si effettuano azioni.
Dalla Tabella 5.8 si osserva che i valori di F1 Score sono migliorati per tutte le classi a meno dell'azione di *release* in cui il valore è diminuito rispetto all'esperimento precedente, perché il modello tende a confondere diverse azioni di classe *release* con la classe *take*.

Esperimento n. 5

Questo esperimento ha gli stessi parametri di quello precedente a meno dei seguenti parametri: *window_size*, che è stato impostato a 3 e *learning rate*, che è stato impostato a 0.05. La Tabella 5.9 mostra i parametri utilizzati nell'esperimento 5.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
3	16	0.05	2	192

Tabella 5.9: Tabella contenente i parametri dell'esperimento 5

La Figura 5.6 mostra la matrice di confusione dell'esperimento 5.

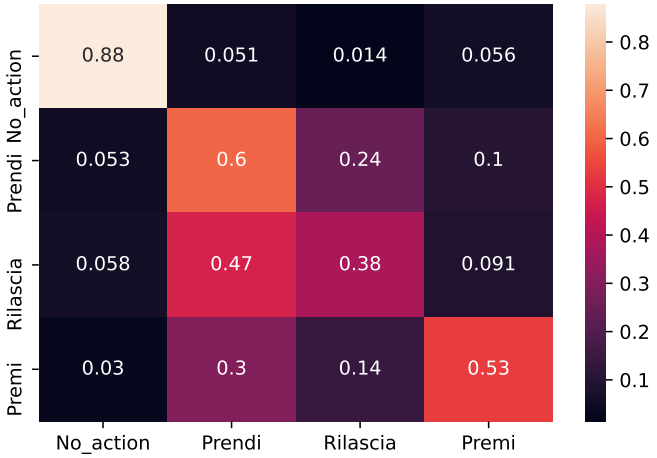

Figura 5.6: Matrice di confusione dell'esperimento 5

La Tabella 5.10 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.90	0.49	0.42	0.57	0.60
Precision	0.93	0.42	0.47	0.62	0.61
Recall	0.88	0.60	0.38	0.53	0.60

Tabella 5.10: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

I risultati ottenuti sono simili a quelli dell'esperimento 4. Non ci sono stati miglioramenti nelle performance del modello diminuendo ulteriormente il parametro *window_size*.

Analisi dei risultati

Analizzando gli esperimenti riportati con la modalità di pre-processamento dei dati del training set in sequenze senza sovrapposizione è possibile osservare che i parametri che hanno influenzato maggiormente gli esperimenti sono:

- *window_size*;
- *batch size*;
- *hidden layer size*.

Dagli esperimenti 1 e 2 si evince che riducendo il valore del parametro *window_size*, il modello proposto inizia a distinguere le **No_action** da tutte le altre azioni. Dagli esperimenti 3, 4 e 5 è possibile osservare che diminuendo ulteriormente i parametri *window_size* e *batch_size*, aumentando il valore dell'*hidden layer size* e scegliendo un opportuno *numero di layer* si ottengono dei modelli con i migliori risultati di F1-Score. Per quanto riguarda il *numero di layer*, i risultati migliori sono stati ottenuti impostando tale parametro con un valore pari a 2. La Figura 5.7 mostra la F1 media al variare del parametro *window_size* e mostra una curva per ogni valore del parametro *numero di layer* utilizzato. Il grafico mostra anche i valori di F1 media di esperimenti non discussi nelle sezioni precedenti.

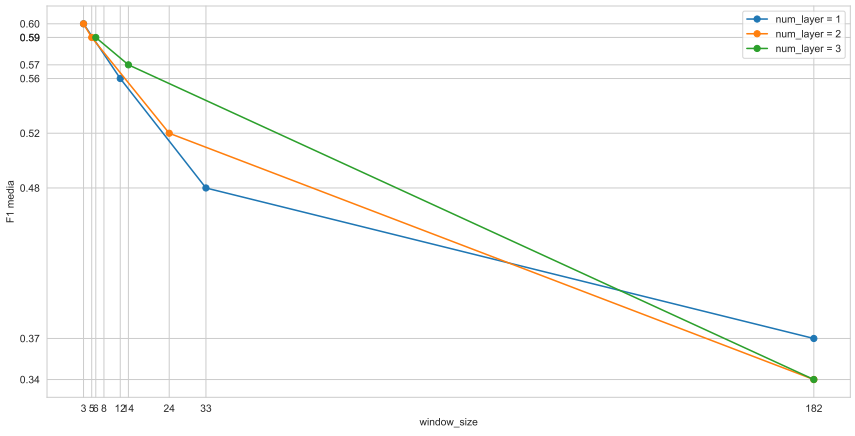

Figura 5.7: F1 media al variare del parametro window_size e del numero di layer

5.2.2 Sequenze sovrapposte

Esperimento n. 6

In questo esperimento è stato impostato il valore del parametro *window_size* pari a 60 frame per studiare il comportamento del modello con una finestra temporale pari a 2 secondi. I valori dei parametri *batch size*, *learning rate*, *numero di layer* e *hidden layer size* sono stati scelti in maniera arbitraria. La Tabella 5.11 mostra i parametri utilizzati nell'esperimento 6.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
60	512	0.04	1	32

Tabella 5.11: Tabella contenente i parametri dell'esperimento 6

La Figura 5.8 mostra la matrice di confusione dell'esperimento 6.

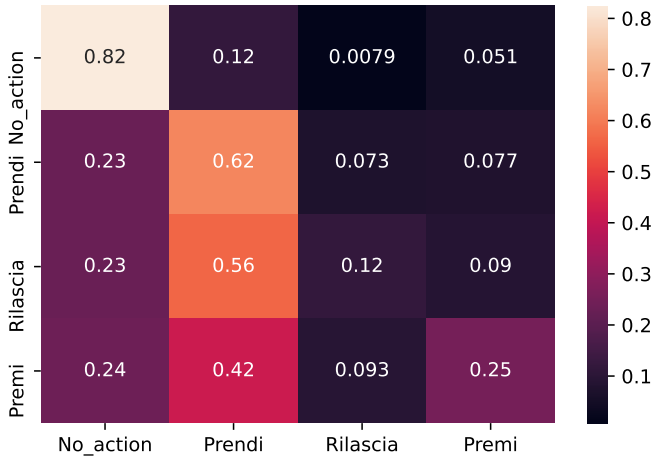

Figura 5.8: Matrice di confusione dell'esperimento 6

La Tabella 5.12 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.75	0.45	0.18	0.34	0.43
Precision	0.69	0.36	0.37	0.49	0.48
Recall	0.83	0.52	0.12	0.26	0.43

Tabella 5.12: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che:

- diverse azioni di classe *release* sono confuse con azioni di classe *take* o con *No_action*;
- diverse azioni di classe *push* sono confuse con azioni di classe *take* o con *No_action*;
- diverse azioni di classe *take* sono confuse con *No_action*.

Inoltre, osservando la prima colonna della matrice di confusione è possibile dedurre che il modello non riesce a distinguere le sequenze temporali in cui si sta effettuando un'azione dalle sequenze temporali in cui non si effettuano azioni.
Dalla Tabella 5.12 si osserva che i valori di F1 Score delle azioni sono inferiori a 0.5, ma in particolare quello dell'azione *release* risulta essere il più basso, questo vuol dire che il modello non riesce a distinguere questo tipo di azione dalle altre.

Esperimento n. 7

In questo esperimento è stato impostato il valore del parametro *window_size* pari a 33 frame per studiare il comportamento del modello con una finestra temporale pari a 1 secondo, più piccola dell'esperimento precedente. Il valore del parametro *batch size* è stato impostato a 128 per capire se il modello migliora quando i pesi della rete neurale in fase di addestramento si aggiornano utilizzando un numero più piccolo di sequenze temporali. I valori dei parametri *learning rate*, *numero di layer* e *hidden layer size* sono gli stessi dell'esperimento 6. La Tabella 5.13 mostra i parametri utilizzati nell'esperimento 7.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
33	128	0.04	1	32

Tabella 5.13: Tabella contenente i parametri dell'esperimento 7

La Figura 5.9 mostra la matrice di confusione dell'esperimento 7.

Figura 5.9: Matrice di confusione dell'esperimento 7

La Tabella 5.14 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.83	0.47	0.35	0.43	0.52
Precision	0.83	0.42	0.36	0.52	0.53
Recall	0.84	0.54	0.34	0.36	0.53

Tabella 5.14: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che:

- diverse azioni di classe *take* sono confuse con azioni di classe *release*;
- diverse azioni di classe *release* sono confuse con azioni di classe *take*;
- diverse azioni di classe *push* sono confuse con la classe *take* oppure con la classe *release*.

Inoltre, osservando la prima colonna della matrice di confusione è possibile dedurre che il modello riesce a distinguere le sequenze temporali in cui si sta effettuando un'azione dalle sequenze temporali in cui non si effettuano azioni.
Dalla Tabella 5.14 si osserva che i valori di F1 Score sono migliorati per tutte le classi, questo significa che utilizzare un valore dei parametri *window_size* e *batch size* più piccolo permette di ottenere un modello con migliori performance.

Esperimento n. 8

In questo esperimento sono stati mantenuti i parametri dell'esperimento precedente a meno del parametro *batch size* che è stato impostato a 64 per capire se il modello migliora quando i pesi della rete neurale in fase di addestramento si aggiornano utilizzando un numero più piccolo di sequenze temporali. La Tabella 5.15 mostra i parametri utilizzati nell'esperimento 8.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
33	64	0.04	1	32

Tabella 5.15: Tabella contenente i parametri dell'esperimento 8

La Figura 5.10 mostra la matrice di confusione dell'esperimento 8.

Figura 5.10: Matrice di confusione dell'esperimento 8

La Tabella 5.16 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.83	0.50	0.45	0.47	0.56
Precision	0.83	0.45	0.43	0.60	0.58
Recall	0.84	0.56	0.46	0.38	0.56

Tabella 5.16: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che:

- diverse azioni di classe *take* sono confuse con azioni di classe *release*;
- diverse azioni di classe *release* sono confuse con azioni di classe *take*;
- diverse azioni di classe *push* sono confuse con la classe *take* oppure con la classe *release*.

Inoltre, osservando la prima colonna della matrice di confusione è possibile dedurre che il modello riesce a distinguere le sequenze temporali in cui si sta effettuando un'azione dalle sequenze temporali in cui non si effettuano azioni.
Dalla Tabella 5.16 si osserva che i valori di F1 Score sono aumentati per tutte le classi, questo significa che diminuendo il valore del parametro *batch size* si ottiene un modello con migliori performance.

Esperimento n. 9

In questo esperimento è stato impostato il valore del parametro *window_size* pari a 8 frame per studiare il comportamento del modello con una finestra temporale pari a 264 millisecondi, più piccola degli esperimenti precedenti. Il valore del parametro *batch size* è stato impostato a 512 per capire se il modello migliora quando i pesi della rete neurale in fase di addestramento si aggiornano utilizzando un numero più grande di sequenze temporali. Per provare ad aumentare le performance del modello, il valore del parametro *numero di layer* è impostato a 2, ovvero si hanno 2 celle LSTM, e il valore del parametro *hidden layer size* è impostato a 78. Il valore del parametro *learning rate* è lo stesso dell'esperimento 8. La Tabella 5.17 mostra i parametri utilizzati nell'esperimento 9.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
8	512	0.04	2	78

Tabella 5.17: Tabella contenente i parametri dell'esperimento 9

La Figura 5.11 mostra la matrice di confusione dell'esperimento 9.

Figura 5.11: Matrice di confusione dell'esperimento 9

La Tabella 5.18 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.92	0.54	0.43	0.58	0.62
Precision	0.94	0.45	0.48	0.67	0.64
Recall	0.89	0.67	0.40	0.51	0.62

Tabella 5.18: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che:

- diverse azioni di classe *take* sono confuse con azioni di classe *release*;
- diverse azioni di classe *release* sono confuse con azioni di classe *take*;
- diverse azioni di classe *push* sono confuse con la classe *take* oppure con la classe *release*.

Inoltre, osservando la prima colonna della matrice di confusione è possibile dedurre che il modello riesce a distinguere le sequenze temporali in cui si sta effettuando un'azione dalle sequenze temporali in cui non si effettuano azioni.
Dalla Tabella 5.18 si osserva che i valori di F1 Score sono migliorati per tutte le classi, questo significa che diminuendo il valore del parametro *window_size* e aumentando i valori dei parametri *batch size* e *hidden layer size* si ottiene un modello con migliori performance.

Esperimento n. 10

In questo esperimento sono stati utilizzati gli stessi parametri dell'esperimento 9 a meno del parametro *numero di layer* che è stato impostato a 1, per provare ad aumentare le performance del modello utilizzando una sola cella LSTM. La Tabella 5.19 mostra i parametri utilizzati nell'esperimento 5.

Parametri				
window_size	**Batch Size**	**Learning Rate**	**Numero di Layer**	**Hidden Layer Size**
8	512	0.04	1	78

Tabella 5.19: Tabella contenente i parametri dell'esperimento 5

La Figura 5.12 mostra la matrice di confusione dell'esperimento 5.

Figura 5.12: Matrice di confusione dell'esperimento 5

La Tabella 5.20 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Prendi**	**Rilascia**	**Premi**	**Media**
F1-Score	0.93	0.56	0.54	0.53	0.64
Precision	0.97	0.50	0.48	0.71	0.67
Recall	0.89	0.64	0.61	0.43	0.64

Tabella 5.20: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che:

- diverse azioni di classe *take* sono confuse con azioni di classe *release*;
- diverse azioni di classe *release* sono confuse con azioni di classe *take*;
- diverse azioni di classe *push* sono confuse con la classe *take* oppure con la classe *release*.

Inoltre, osservando la prima colonna della matrice di confusione è possibile dedurre che il modello riesce a distinguere le sequenze temporali in cui si sta effettuando un'azione dalle sequenze temporali in cui non si effettuano azioni.
Dalla Tabella 5.20 si osserva che i valori di F1 Score sono migliorati per tutte le classi, questo significa che diminuendo il valore del parametro *numero di layer* si ottiene un modello con migliori performance.

Analisi dei risultati

Analizzando gli esperimenti riportati con la modalità di pre-processamento dei dati del training set in sequenze sovrapposte è possibile osservare che i parametri che hanno influenzato maggiormente gli esperimenti sono:

- *window_size*;
- *batch size*;
- *hidden layer size*.

Dagli esperimenti 6, 7 e 8 si evince che diminuendo i parametri *window_size* e *batch size* il modello inizia a distinguere le **No_action** dal resto delle azioni e in particolare nell'esperimento 8 si hanno dei valori di F1-Score comparabili con quelli ottenuti dai migliori esperimenti nella sezione precedente. I parametri degli esperimenti 9 e 10 sono stati definiti a partire da quelli dell'esperimento 8. In questi ultimi è stato diminuito ulteriormente il valore del parametro *window_size*, mentre i valori dei parametri *batch size* e *hidden layer size* sono stati aumentati. Nell'esperimento 9, il *numero di layer* è stato impostato a 2, mentre nell'esperimento 10 è stato impostato a 1. In entrambi gli esperimenti, sono stati ottenuti i migliori risultati di F1-Score. In particolare, l'esperimento 10 è l'unico ad avere tutti i valori di F1-Score maggiori di 0.5. La Figura 5.13 mostra la F1 media al variare del parametro *window_size* e mostra una linea per ogni valore del parametro *numero di layer* utilizzato. Il grafico mostra anche i valori di F1 media di esperimenti non discussi nelle sezioni precedenti.

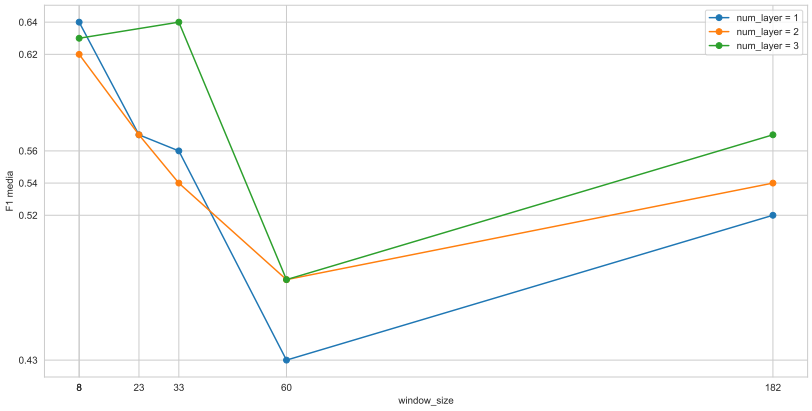

Figura 5.13: F1 media al variare del parametro window_size e del parametro numero di layer

5.2.3 Discussione comparativa dei risultati

Dai risultati ottenuti si evince che il metodo di pre-processamento dei dati del training set più efficace è quello con sequenze sovrapposte. Ciò è dovuto probabilmente al fatto che sequenze sovrapposte forniscono esempi di training più vari riducendo l'overfitting.
Inoltre, dai risultati ottenuti si evince come nessun modello sviluppato è in grado di risolvere in maniera ottimale il problema che era stato posto all'inizio di questo studio. Infatti, dalle matrici di confusione degli esperimenti migliori (esperimenti 4, 5, 9 e 10) è possibile osservare che:

- diverse azioni di classe *take* sono confuse con azioni di classe *release*;
- diverse azioni di classe *release* sono confuse con azioni di classe *take*;
- diverse azioni di classe *push* sono confuse con la classe *take* oppure con la classe *release*.

Le Figure 5.14 e 5.15 mostrano qualitativamente due esempi di azioni di tipo *push*. Quando l'azione è eseguita tenendo le dita delle mani chiuse a meno dell'indice, il modello riesce a capire che si tratta di un'azione di classe *push*, come mostrato in Figura 5.14. Quando invece si esegue l'azione con la mano aperta, il modello tende a confondersi con un'altra azione, come mostrato in Figura 5.15.

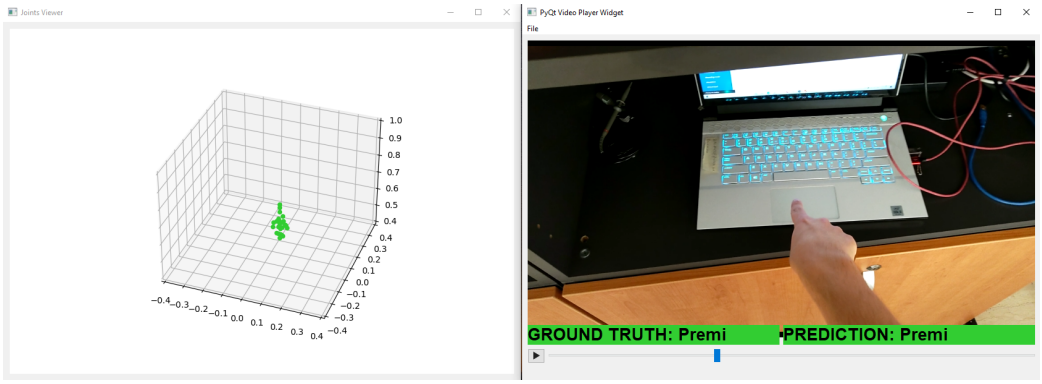

Figura 5.14: Azione premi classificata correttamente

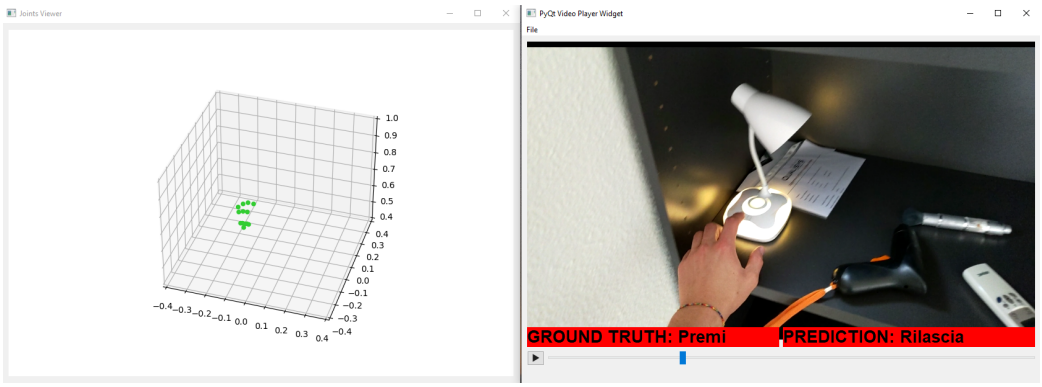

Figura 5.15: Azione premi non classificata correttamente

Dai risultati ottenuti però si evince come il metodo proposto riesce a distinguere le **No_action** dal resto delle azioni. Partendo da questa supposizione, sono stati effettuati ulteriori esperimenti descritti nella sezione successiva.

5.3 Esperimenti Supplementari

Gli esperimenti supplementari sono stati effettuati considerando due macro-classi:

- **action** composta dagli esempi delle classi *take*, *release* e *push*. L'uso di questa macro-classe permette di valutare la capacità del modello nel distinguere i casi di **No_action** dalle altre azioni.
- **P/R** composta dagli esempi delle classi *take* e *release*. L'uso di questa macro-classe permette di valutare il comportamento del modello quando *take* e *release* sono considerate come un'unica azione.

La Figura 5.16 mostra la relazione tra classi e macro-classi.

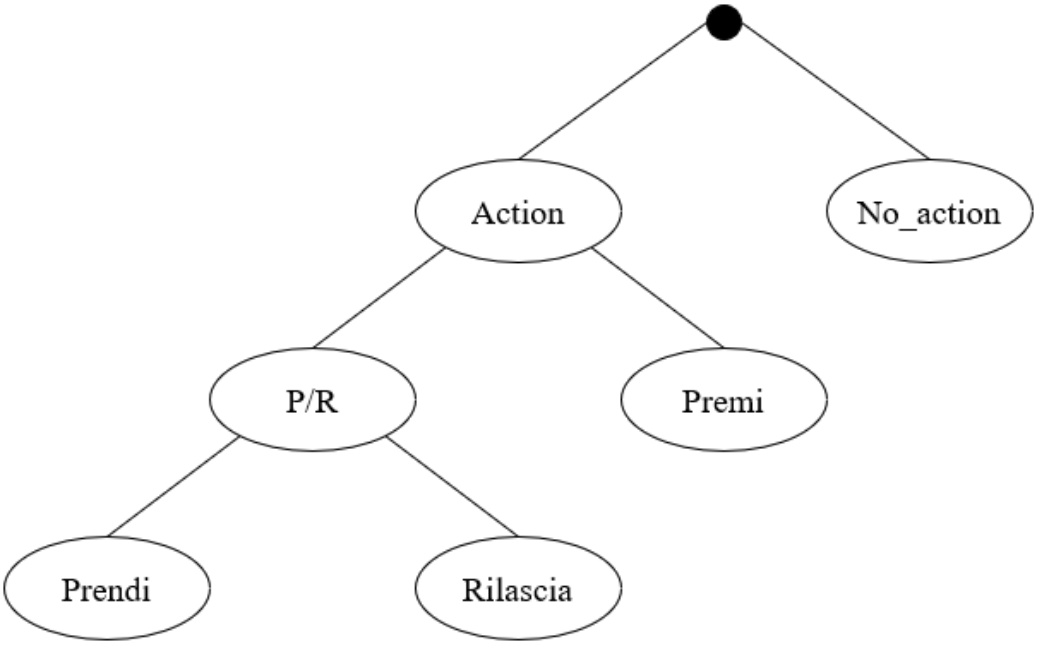

Figura 5.16: Relazione tra classi e macro-classi

5.3.1 Action vs No action

Si consideri l'esperimento 10, che utilizza il pre-processamento dei dati del training set con sequenze sovrapposte, utilizzando la macro-classe **action**. La Figura 5.17 mostra la relativa matrice di confusione.

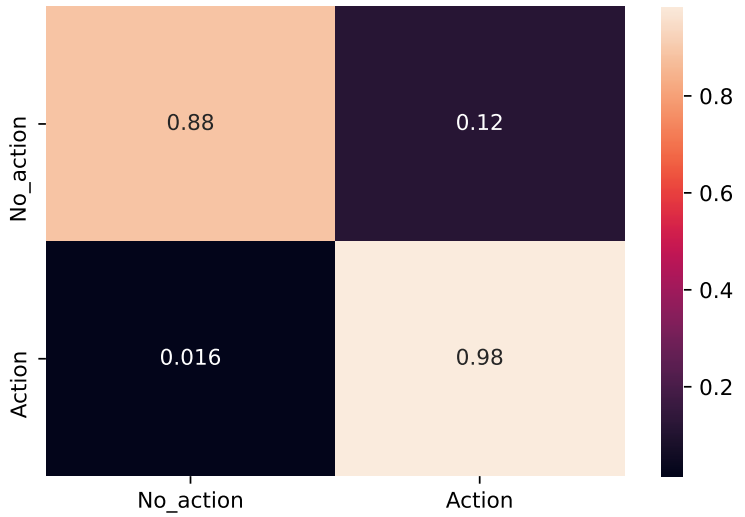

Figura 5.17: Matrice di confusione con macro-classe action

La Tabella 5.21 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**Action**	**Media**
F1-Score	0.93	0.95	0.94
Precision	0.97	0.93	0.95
Recall	0.88	0.98	0.93

Tabella 5.21: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che lungo la diagonale si hanno dei valori molto vicini a 1. Questo significa che utilizzare la macro-classe **action** con il modello dell'esperimento 10 è possibile distinguere le **No_action** dal resto delle azioni.
Dalla Tabella 5.21 si osserva che i valori di F1 Score delle due classi sono maggiori di 0.9, quindi i **false positive** e i **false negative** delle azioni considerate hanno valori bassi.

5.3.2 P/R, Push e No action

Si consideri l'esperimento 10, che utilizza il pre-processamento dei dati del training set con sequenze sovrapposte, utilizzando la macro-classe **P/R**. La Figura 5.18 mostra la relativa matrice di confusione.

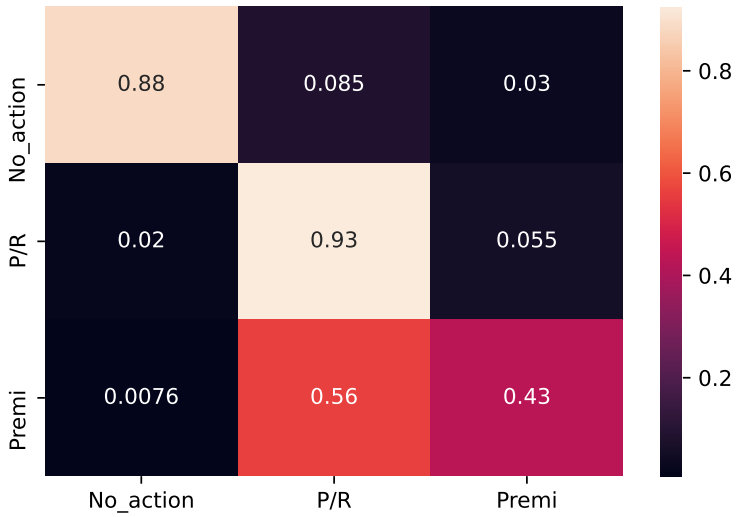

Figura 5.18: Matrice di confusione con macro-classe P/R

La Tabella 5.22 mostra i valori ottenuti dalle misure: F1-Score, Precision e Recall.

	No_action	**P/R**	**Premi**	**Media**
F1-Score	0.93	0.81	0.53	0.77
Precision	0.97	0.72	0.71	0.8
Recall	0.88	0.93	0.43	0.77

Tabella 5.22: Tabella contenente i valori delle misure: F1-Score, Precision e Recall

Dalla matrice di confusione è possibile osservare che diverse azioni di *push* sono classificate con la macro-classe **P/R**. La stessa osservazione può essere fatta considerando i valori di F1 Score dalla Tabella 5.22, dove il valore di F1 Score dell'azione *push* è molto più piccolo dei valori delle altre due classi, questo significa che i **false positive** e i **false negative** hanno valori alti per l'azione *push*. Questi risultati mostrano quantitativamente ciò che è mostrato qualitativamente dalle Figure 5.14 e 5.15.

5.3.3 Analisi dei risultati

Dai risultati degli esperimenti si evince che:

- semplificando il problema utilizzando la macro-classe **action** si ottengono delle buone performance con il metodo proposto. Le Figure 5.19 e 5.20 mostrano qualitativamente esempi di classificazione che considerano la macro-classe **action**.
- semplificando il problema utilizzando la macro-classe **P/R** si osserva che le azioni di *push* sono spesso confuse con *take* e *release*, come mostrato nelle Figure 5.14 e 5.15. Le Figure 5.21 e 5.22 mostrano qualitativamente esempi di classificazione che considerano la macro-classe **P/R**.

Le analisi effettuate riguardano solo l'esperimento 5 che utilizza il pre-processamento dei dati del training set con sequenze sovrapposte, ma risultati simili sono stati ottenuti analizzando altri esperimenti, qui non riportati per brevità.

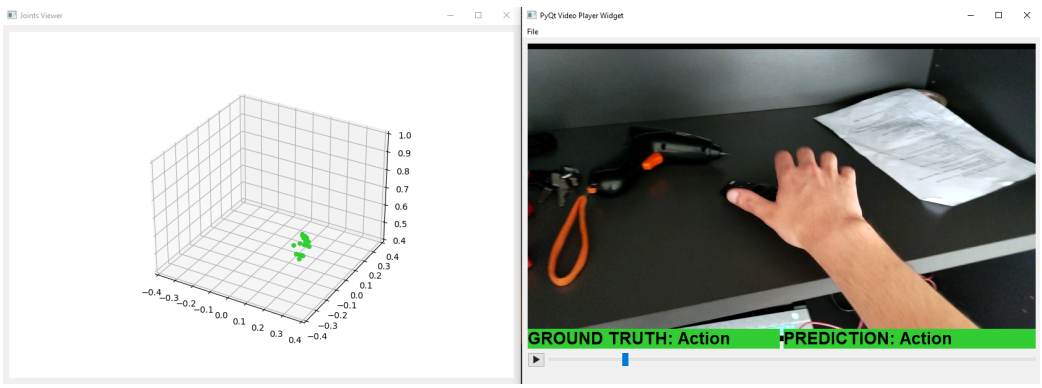

Figura 5.19: Esempio qualitativo con macro-classe action

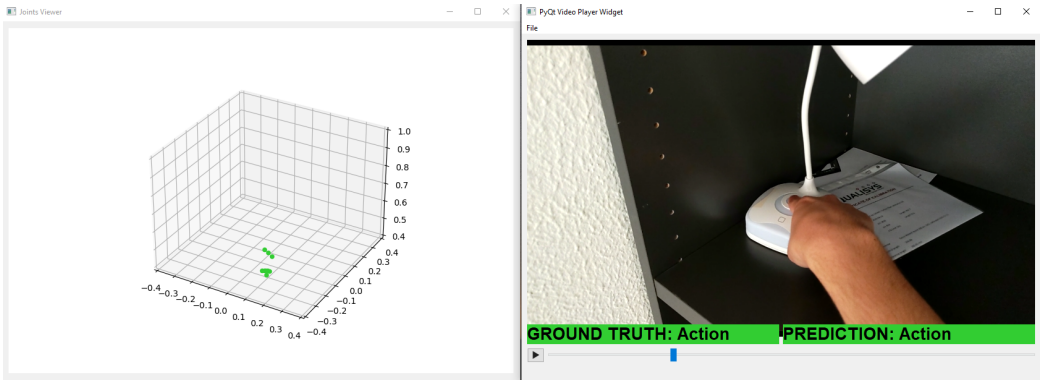

Figura 5.20: Esempio qualitativo con macro-classe action

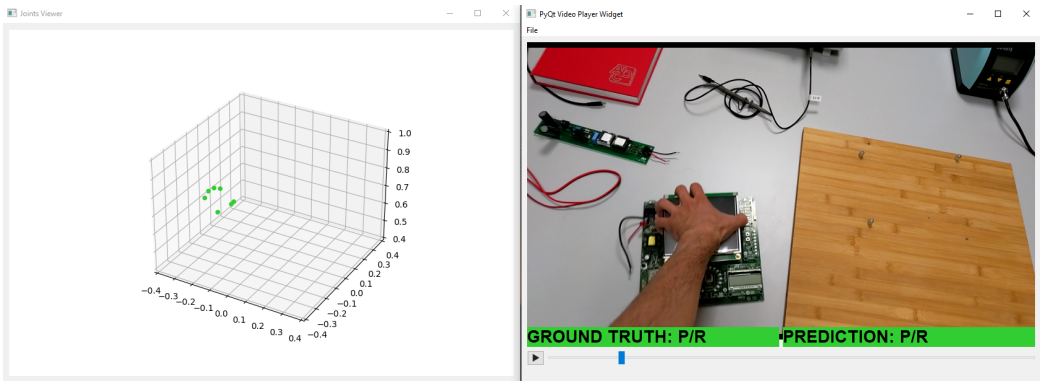

Figura 5.21: Esempio qualitativo con macro-classe P/R

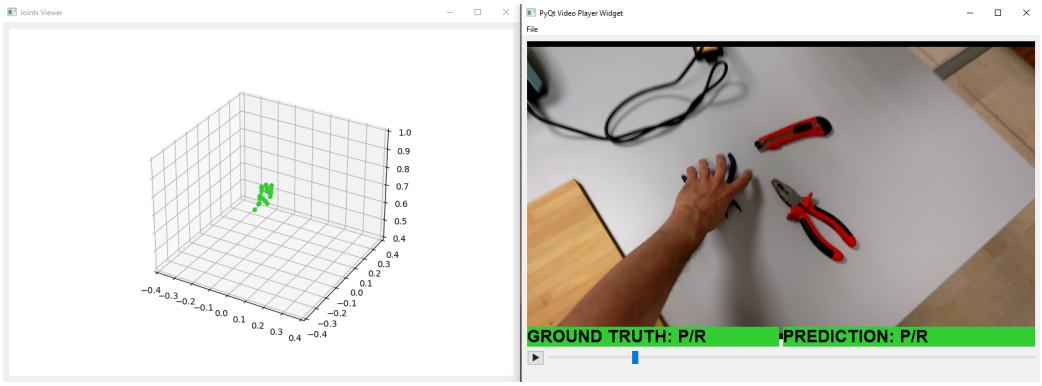

Figura 5.22: Esempio qualitativo con macro-classe P/R

Conclusione

Questo studio si è concentrato sul problema di analizzare i punti chiave delle mani acquisiti da Microsoft HoloLens2 per determinare il tipo di azione che si sta svolgendo tra *take*, *release* e *push*, distinguendo anche i casi in cui non si esegue nessuna azione.
Per raggiungere l'obiettivo di questa tesi, le coordinate 3D dei punti chiave delle mani sono state ottenute attraverso il dispositivo indossabile HoloLens2. I dati raccolti sono stati etichettati e successivamente suddivisi in Training, Validation e Test set. Attraverso degli appositi algoritmi, dai tre set sono state ottenute delle sequenze temporali, utilizzate come input per l'algoritmo proposto, che si basa su LSTM. Quest'ultima restituisce in output un vettore di quattro valori che permette di associare alle sequenze in input una specifica azione. Dai risultati ottenuti dagli esperimenti si è dedotto che nessuno dei modelli sviluppati è stato in grado di risolvere in maniera ottimale il problema proposto. Partendo da delle osservazioni sui risultati, il problema iniziale è stato trasformato in due macro problemi, utilizzando le macro-classi: **P/R** (Prendi/Rilascia) e **action**. Dai risultati degli esperimenti supplementari, in cui sono state utilizzate le macro-classi, è emerso che è possibile utilizzare il metodo proposto per distinguere i momenti in cui si eseguono azioni, dai momenti in cui non si eseguono.
Le difficoltà principali sono state riscontrate nella raccolta dei dati, dove spesso HoloLens non è stato in grado di acquisire tutti i punti chiave delle mani e questo ha creato confusione nell'interpretazione delle azioni, che non ha permesso di raggiungere l'obiettivo inizialmente definito con risultati ottimali. Per migliorare la capacità nella distinzione delle azioni si potrebbero utilizzare delle tecniche che sfruttano anche i video per cercare di ricostruire i punti chiave delle mani mancanti partendo da quelli che sono stati acquisiti correttamente e utilizzare l'ambiente circostante a supporto della classificazione dell'azione.
Questa tesi si è focalizzata sul riconoscimento delle azioni. In futuro, questo studio può essere ampliato cercando di dare dei suggerimenti all'utente in base al tipo di azione eseguita.

Appendice A

Approfondimenti

A.1 Come si utilizza VIA?

In questa sezione verrà spiegato come effettuare l'etichettatura di un video con VIA.

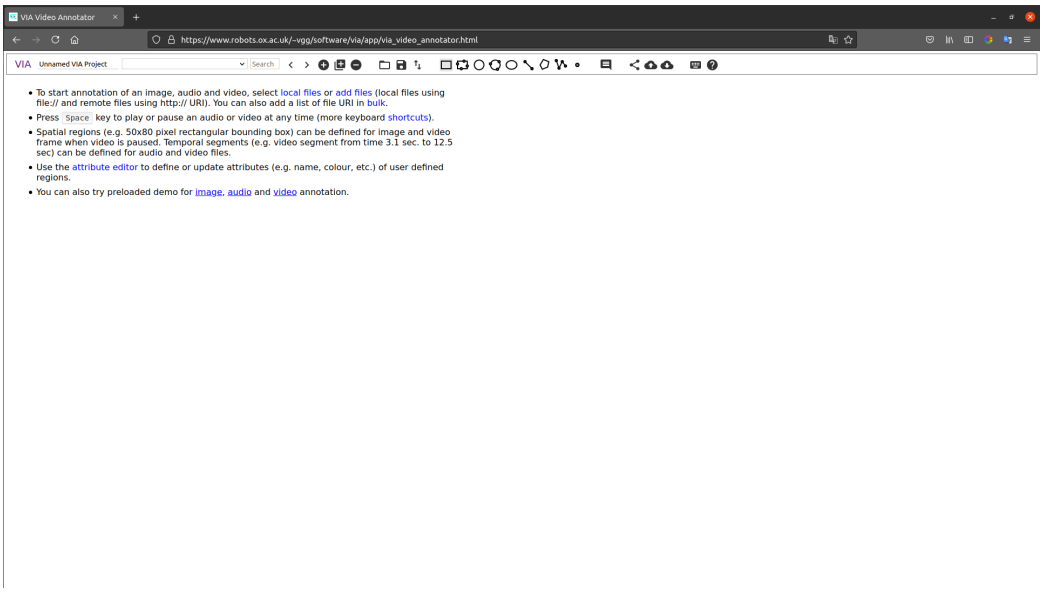

Figura A.1: VIA - Interfaccia iniziale.

L'interfaccia iniziale, rappresentata in Figura A.1, presenta un menù in alto e cliccando sul simbolo ⊕ è possibile aggiungere un file audio o video locale. Dopo aver aggiunto il file, quello che si ottiene è qualcosa di simile alla Figura A.2.

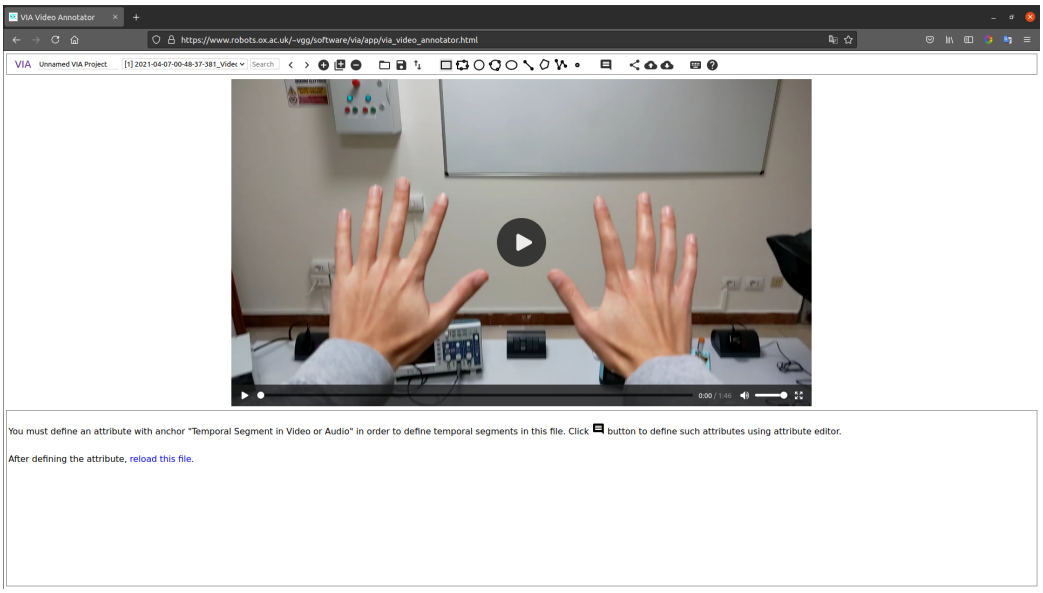

Figura A.2: VIA - Interfaccia con un file video.

Per poter procedere con l'etichettatura del video, è necessario definire un attributo di tipo "Segmento temporale in video o audio" e per farlo bisogna cliccare su e definire l'attributo utilizzando l'editor degli attributi. Le Figure A.3, A.4, A.5, A.6 riassumono la creazione di tale attributo:

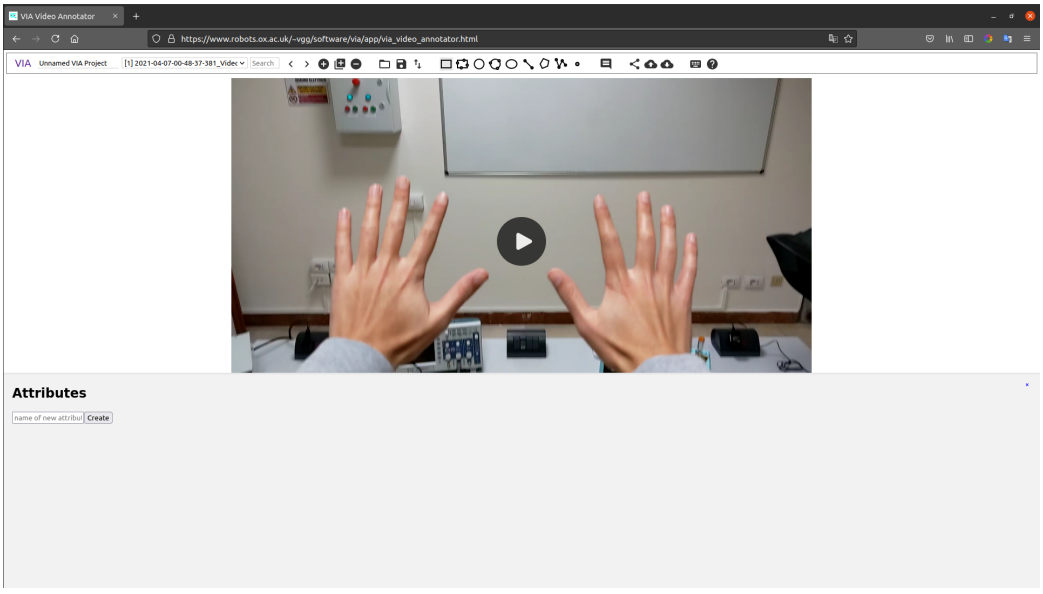

Figura A.3: VIA - Aggiungere un nome e cliccare "create".

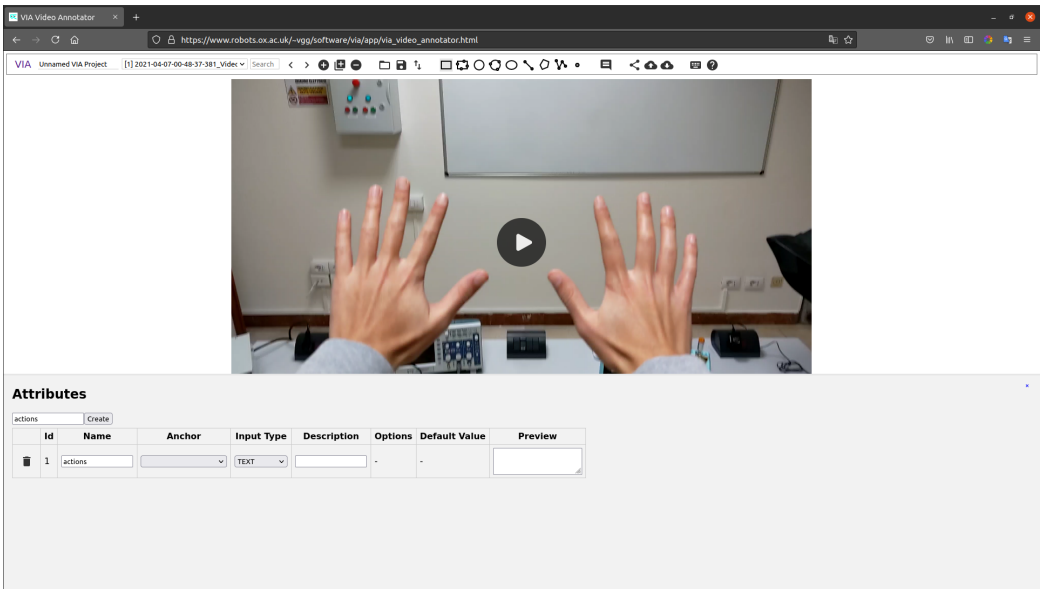

Figura A.4: VIA - Cliccare sul menù a tendina sotto "Anchor".

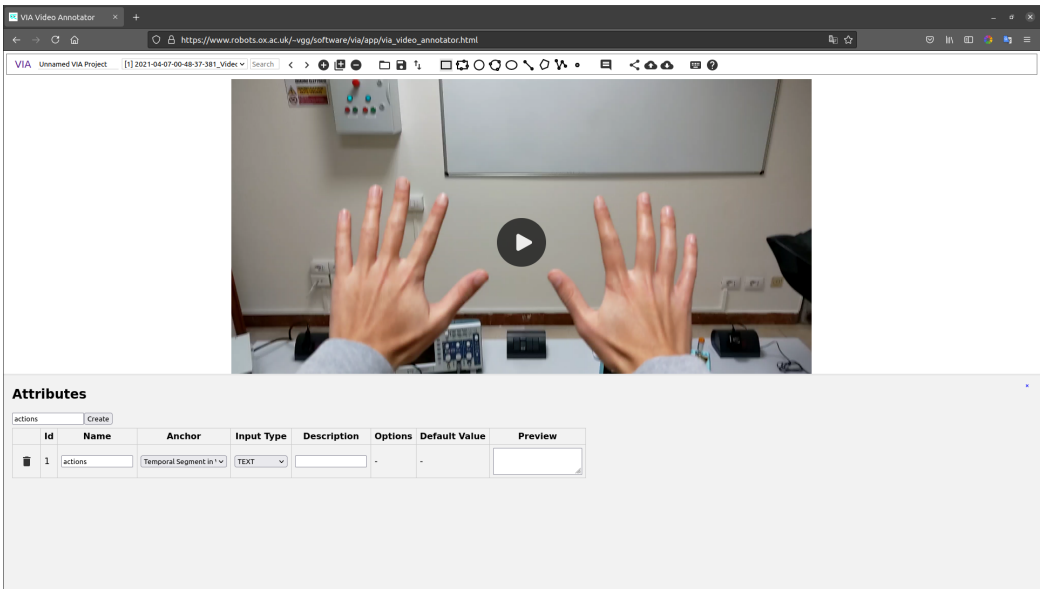

Figura A.5: VIA - Selezionare "Temporal Segment in Video or Audio".

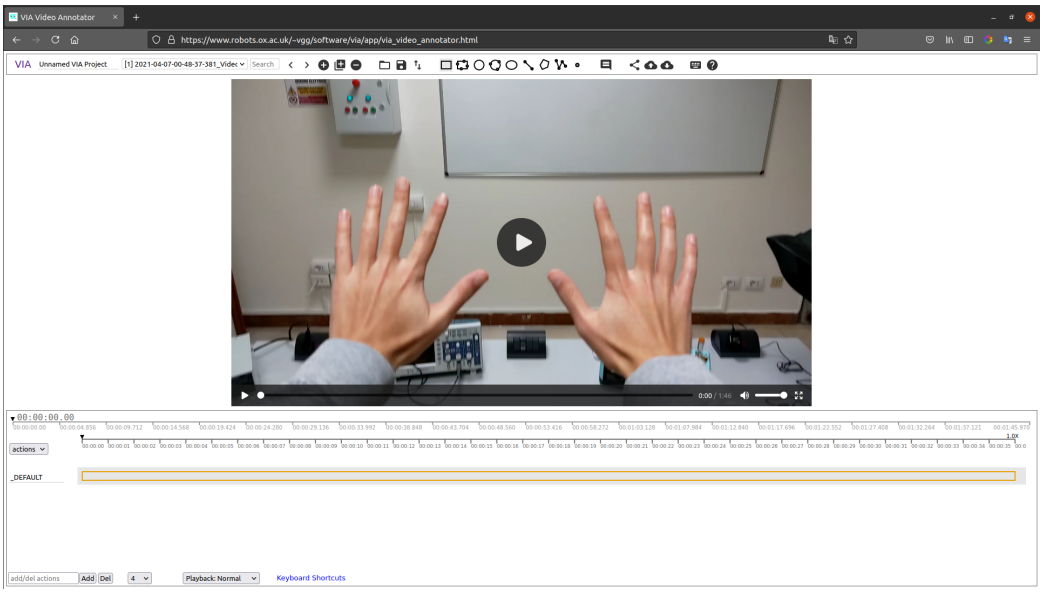

Figura A.6: VIA - Ricaricare la pagina.

Per aggiungere una nuova etichetta, bisogna scrivere il nome nella casella di testo, presente in basso a sinistra, e cliccare su "Add". Se, invece, si vuole eliminare un'etichetta bisogna scrivere il nome dell'etichetta nella stessa casella di testo e cliccare su "Del".
A questo punto è possibile selezionare gli intervalli di tempo che si vogliono considerare per una data etichetta e ottenere qualcosa di simile alla Figura A.7.

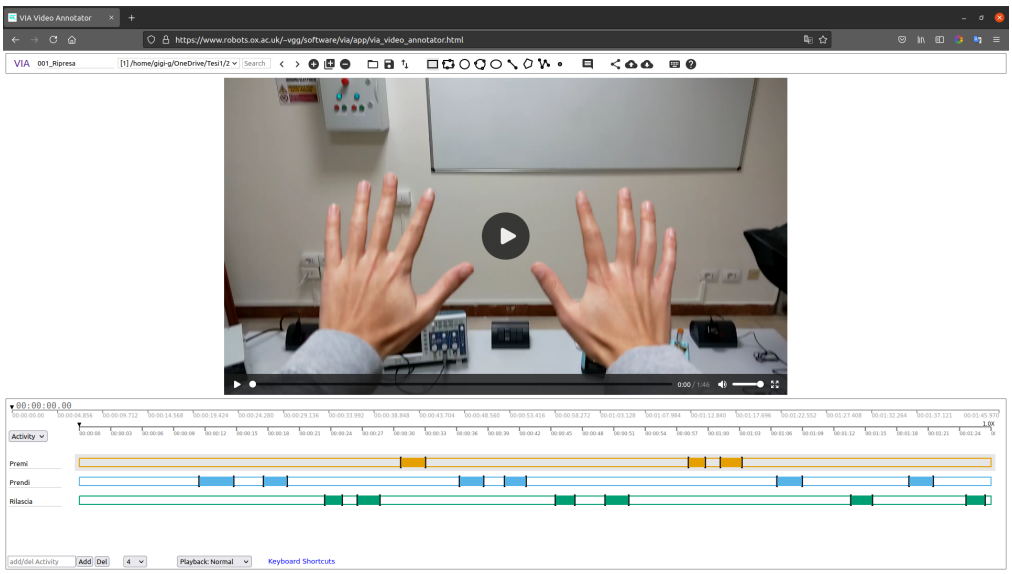

Figura A.7: VIA - Esempio di etichettatura.

Per salvare le annotazioni in formato JSON, bisogna cliccare sul pulsante . Se si volesse caricare un file contenente delle annotazioni, bisogna, prima, selezionare il video, usando e, successivamente, cliccare su per caricare il file JSON.
Sotto è riportato un esempio di annotazione contenuto nel file scaricato.

```
"1_10000015": {
  "vid": 1,
  "flg": 0,
  "z": [
    89.567,
    92.447
  ],
  "xy": [

  ],
  "av": {
    "1": "Premi"
  }
}
```

A.2 Codice utilizzato per l'etichettatura automatica

```
import json
import glob
from modules.extract_json import ExtractJSON
from modules.extract_action import ExtractAction
from modules.create_metadata import CreateMetadata

def create_filename(path:str) -> str:
    return path[:-3] + "json"

def save_json_file(path:str, js:dict) -> None:
    with open(create_filename(path), "w") as f:
        f.write(json.dumps(js))
        f.close()
    print("JSON file has just been saved!")

def main() -> None:
    via_json:str = "./via.json"
    js = json.loads(
        ExtractJSON.get_json(
            via_json
        )
    )
    folders:list = glob.glob("../data/VIDEO/*")
    for folder in folders:
        action_files:list = glob.glob(folder + "/*_action.txt"
    )
        print( "START: " + folder)
        for file in action_files:
            ex_action:ExtractAction = ExtractAction(
                file
            )
            time:int = ex_action.get_time()
            action:list = ex_action.get_action_vector()
            js["metadata"] = CreateMetadata.create(action,
    time, left_size = 1500, right_size = 4500)
            save_json_file(file, js)
        print("END\n")
```

Codice A.1: Codice per etichettare in automatico i video

Il Codice A.1 genera dei JSON che possono essere usati anche su VIA e per farlo viene utilizzato un modello, al quale verranno aggiunte le etichette automatiche.
Le operazioni eseguite da questo script sono:

1. legge il contenuto di una cartella, dove sono presenti i file delle azioni da etichettare per ogni video -generati da Hololens 2-, e restituisce una lista con i nomi dei file;

2. per ogni file viene costruito un JSON, dove ogni azione viene etichettata all'interno di una finestra temporale di 6 secondi.

I moduli **json** e **glob** sono già preinstallati con Python.
Il primo permette di serializzare dei dati in formato JSON, oppure, deserializzare quest'ultimi in un dizionario Python. Il secondo permette di leggere il contenuto di una cartella.

Il Codice A.2 riporta l'implementazione della classe **ExtractJSON** che contiene il metodo statico `get_json`, il quale prende il path di un file JSON e ne restituisce il contenuto, in caso di errore ritorna **None**.

```
class ExtractJSON:
    @staticmethod
    def get_json(path:str) -> str:
        try:
            return open(path, "r").readlines()[0]
        except ValueError:
            print("ERROR: file not found.")
            exit(-1)
        return None
```

Codice A.2: Codice per estrarre il contenuto di un file JSON

Il Codice A.3 riporta l'implementazione della classe **ExtractAction**. Quest'ultima per poter essere inizializzata necessita del path del file contenente le azioni.
Dalla lettura del file è possibile ricavare il tempo di inizio del video su HoloLens e un vettore contenente le azioni.

```
from io import FileIO

class ExtractAction:

    _action_file:FileIO

    def __init__(self, path:str) -> None:
        try:
            self._action_file = open(path, "r")
        except ValueError:
            print("ERROR: file not found.")
```

```
12              exit(-1)
13          return None
14
15      def get_time(self) -> int:
16          hours, minutes, seconds, millis =
17          [(int(elem)) for elem in self._action_file.readline().
    split("-")]
18          return (hours * 3600 + minutes * 60 + seconds)*1000 +
    millis
19
20      def get_action_vector(self) -> list:
21          return [(act) for act in self._action_file]
```

Codice A.3: Codice per estrarre le azioni dai file action di HoloLens2

Il Codice A.4 riporta l'implementazione dela classe **CreateMetadata**, la quale contiene un metodo statico, `create`, con i seguenti parametri:

- **action_vector:list**: lista contenente le azioni che devono essere etichettate.
- **time:int**: tempo di Hololens 2 che indica l'inizio del video.
- **left_size:float**: limite inferiore della finestra di tempo.
- **right_size:float**: limite superiore della finestra di tempo.

Questo metodo permette di generare un dizionario Python contenente le etichette che saranno aggiunte al modello del file VIA.

```
1  class CreateMetadata:
2      @staticmethod
3      def create(action_vector:list, time:int, left_size:int,
    right_size:int) -> dict:
4          met:dict = {}
5          start_index:int = 10000000
6          for elem in action_vector:
7              key:str = "1_" + str(start_index)
8              t, a = elem.split()
9              hours, minutes, seconds, millis = [(int(el)) for
    el in t.split("-")]
10             tim = ((hours * 60 + minutes * 60 + seconds)*1000
    + millis) - time - left_size
11             tim_f = tim + right_size
12             met[key] = {
13                 "vid": 1,
14                 "flg": 0,
15                 "z": [tim/1000, tim_f/1000],
```

```
                "xy": [],
                "av": {
                    "1": a
                }
            }
            start_index += 1
        return met
```

Codice A.4: Codice per generare i meta dati del file JSON di VIA

A.3 Codice utilizzato per lo studio quantitativo

In questo approfondimento sono riportati gli estratti di codice Python per lo studio quantitativo sul confronto tra l'etichettatura manuale e quella automatica.
Utilizzando il Codice A.5 è stato estratto il JSON di un file VIA, contenente sia etichette automatiche che manuali, ottenendo un dizionario Python.

```python
import json
from modules.extract_json import ExtractJSON

js:dict = json.loads(
    ExtractJSON.get_json(
        "via.json"
    )
)
```

Codice A.5: Codice per estrarre un JSON da un file VIA

```python
metadata:dict = js["metadata"]
```

Codice A.6: Codice per estrarre i metadata

Dal dizionario viene estratto il contenuto della chiave denominata `metadata`, che è un altro dizionario, Codice A.6. Gli elementi che lo costituiscono, a coppie, rappresentano la stessa azione, dove la prima è un'etichetta manuale, mentre la seconda è un'etichetta automatica. Quelli che si riferiscono alla stessa azione possiedono la stessa **chiave** a meno del terzo carattere, il quale è pari a **1**, se l'etichetta è automatica; **2**, se l'etichetta è manuale.

```
{
    '1_10000000': {'vid': 1,
    'flg': 0,
    'z': [11.831, 13.831],
    'xy': [],
    'av': {'1': 'Prendi_A'}},
    '1_20000000': {'vid': 1,
    'flg': 0,
    'z': [11.398, 14.71],
    'xy': [],
    'av': {'1': 'Prendi_M'}}
}
```

La funzione `jaccard_similarity`, riportata nel Codice A.7, invece, permette di calcolare la similarità di Jaccard.

```
def jaccard_similarity(x11:float, x12:float, x21:float, x22:
    float) -> float:
    I:float = min(x12, x22) - max(x11, x21)
    U:float = max(x12, x22) - min(x11, x21)
    return I/U
```

Codice A.7: jaccard_similarity

La funzione `similarity_dict`, riportata nel Codice A.8, calcola la similarità di Jaccard tra tutte le etichette automatiche e manuali, restituendo un dizionario con le seguenti chiavi:

- **key**: un numero progressivo che inizia da 10000000 a cui viene aggiunto il seguente prefisso "1_" (Es. 1_10000000);
- **action**: contiene il tipo di azione;
- **similarity**: contiene la similarità di Jaccard.

```
def similarity_dict(metadata:dict) -> dict:
    action:int = 0
    result:dict = {}
    for key1, value in metadata.items():
        if action >= len(metadata) / 2: break
        key2 = list(key1)
        key2[2] = '2'
        key2 = ''.join(key2)
        result[key1] = {
            "action": value["av"]["1"][:-2],
            "similarity": jaccard_similarity(
                value["z"][0], value["z"][1],
                metadata[key2]["z"][0],
                metadata[key2]["z"][1]
            )
        }
        action += 1
    return result
```

Codice A.8: Codice per calcolare la similarità tra tutte le etichette

Un estratto del dizionario, restituito dalla funzione, è riportato di seguito:

```
{'1_10000000': {'action': 'Prendi', 'similarity': 0
    .6038647342995167},,
 '1_10000001': {'action': 'Rilascia', 'similarity':
    0.7070993914807296},
 '1_10000002': {'action': 'Premi', 'similarity': 0.
    842105263157895}}
```

È stata, inoltre, definita una funzione chiamata `mean`, riportata nel Codice A.9, che permette di ottenere due risultati, in base ad un parametro action:

- se action è *False*, ritorna la media tra tutte le similarità;
- se action è *True*, ritorna la media tra tutte le similarità, per singola azione.

```
def mean(similarity:dict, action:boolean = False) -> list:

    if not action:
        return [sum(value["similarity"] for value in
    similarity.values()) / len(similarity)]

    s_take:float = 0
    c_take:int = 0
    s_release:float = 0
    c_release:int = 0
    s_push:float = 0
    c_push:int = 0
    for _, value in similarity.items():
        if value["action"] == "Prendi":
            s_take += value["similarity"]
            c_take += 1
        elif value["action"] == "Rilascia":
            s_release += value["similarity"]
            c_release += 1
        else:
            s_push += value["similarity"]
            c_push += 1
    return [s_take/c_take, s_release/c_release, s_push/c_push]
```

Codice A.9: Codice per il calcolo della media tra le similarità

I valori di similarità, presenti nel dizionario ottenuto dalla funzione *similarity_dict*, possono essere, inoltre, rappresentati in un Box Plot, usando il Codice A.10.

```python
import matplotlib.pyplot as plt

def box_plot(data:list) -> None:
    green_diamond = dict(markerfacecolor='g', marker='D')
    fig1, ax1 = plt.subplots()
    ax1.set_title('Box Plot')
    ax1.boxplot(data, flierprops=green_diamond)
    if len(data) == 3:
        plt.xticks([1, 2, 3], ['Take', 'Release', 'Push'])
    plt.plot()
```

Codice A.10: Codice per visualizzare dei Box Plot

È possibile ottenere un Box Plot senza distinguere le azioni come in Figura A.8 o con distinzione delle azioni come in Figura A.9.

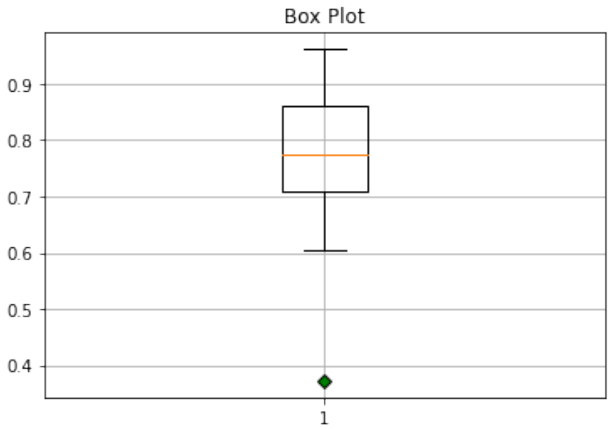

Figura A.8: Box Plot senza distinzione delle azioni

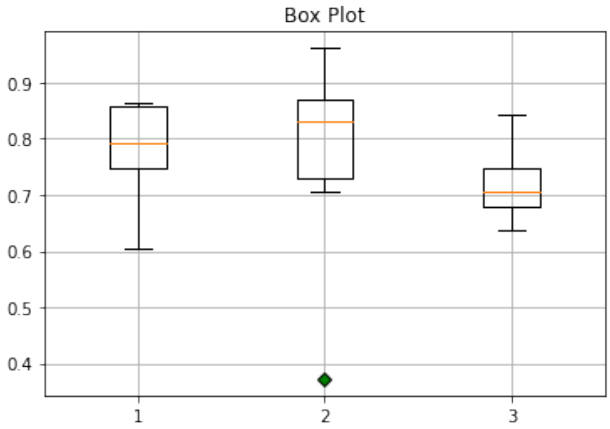

Figura A.9: Box Plot con distinzione delle azioni

Bibliografia

[1] Yuecong Min, Yanxiao Zhang, Xiujuan Chai e Xilin Chen. *An Efficient PointLSTM for Point Clouds Based Gesture Recognition*. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 5761-5770.

[2] Noorkholis Luthfil Hakim, Timothy K. Shih, Sandeli Priyanwada Kasthuri Arachchi, Wisnu Aditya, Yi-Cheng Chen and Chih-Yang Lin. *Dynamic Hand Gesture Recognition Using 3DCNN and LSTM with FSM Context-Aware Model*. Sensors, 2019 - mdpi.com

[3] Aaahm Ikram e Yue Liu. *Skeleton Based Dynamic Hand Gesture Recognition using LSTM and CNN*. 2020 2nd International Conference on Image Processing and Machine Vision, 2020 - dl.acm.org