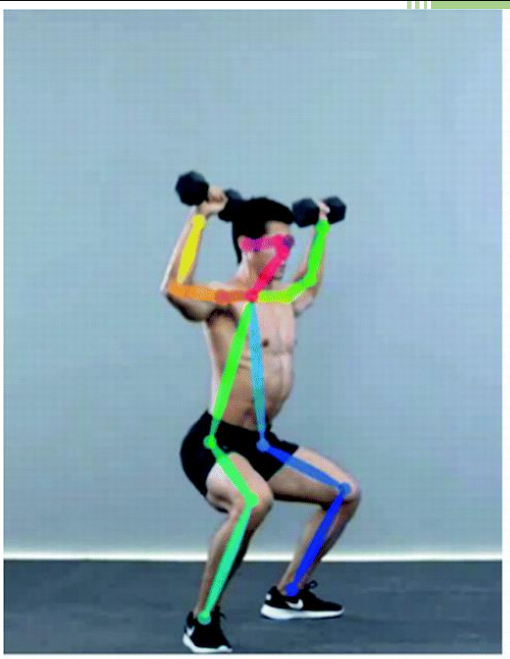
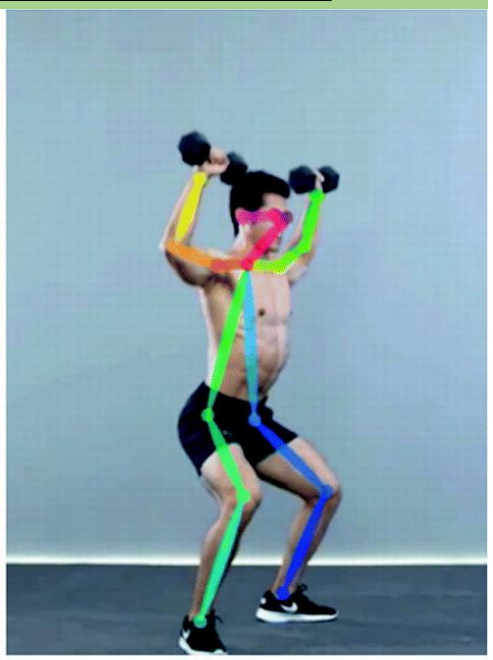
2021

TRAINER ESTIMATION





A cura di

Mangione Luigi **Mat: 699620**

Marzano Savino **Mat: 699921**

https://github.com/Savino-M/TrainerEstimation

1.0 INTRODUZIONE

Trainer estimation è un'applicazione pensata per aiutare chi svolge attività fisica, in casa o in palestra, a monitorare la propria sessione di allenamento. È infatti, utile per tenere conto delle ripetizioni svolte di un esercizio e per verificarne il corretto svolgimento, pertanto, vengono contate solamente le ripetizioni eseguite correttamente.

2.0 UTILIZZO

L'interazione avviene quasi tutta vocalmente: l'utente infatti, sceglie, nella fase iniziale, l'esercizio da eseguire. Per questa fase, l'idea iniziale era quella di consentire un input sia testuale che vocale. Ma data la bassa precisione del riconoscimento vocale, si è optato per il solo input da tastiera. In tutte le altre fasi di esecuzione, TrainerEstimation comunica con la voce: conta, ad esempio, le ripetizioni o guida l'utente nell'eseguire correttamente l'esercizio.

3.0 PROGETTAZIONE E REALIZZAZIONE

Nella fase di progettazione si sono definiti innanzitutto quali siano i casi d'uso del programma ed eventuali risorse, come un modello di esempio per ogni esercizio e il modo in cui stabilire se l'esercizio viene eseguito correttamente o meno.

Per fare ciò, si è deciso in primo luogo di utilizzare come linguaggio di sviluppo, Python. Questo, perché esistono numerose librerie, utili per le funzionalità del software. Si sono infatti utilizzate diverse librerie, ognuna con uno scopo ben preciso e dedita ad una fase specifica dell'esecuzione. Le due librerie principali utilizzate sono OpenCV e MediaPipe.

- OpenCV viene utilizzata per acquisire le immagini da webcam ed eventualmente modificarle;
- MediaPipe invece, è una libreria realizzata da Google, sviluppata in Python e utile per il rilevamento della posizione di una persona: è infatti in grado di rilevare fino a 33 punti fisici diversi.

Per riconoscere la posizione di partenza, è stato creato un classificatore apposito per ogni esercizio. Se quindi l'utente viene rilevato in posizione iniziale (inizio ripetizione), si passa alla fase successiva.

Da questo momento il programma inizia a registrare le azioni dell'utente, ricavare le coordinate dei landmarks frame per frame e creare un dataframe, che viene costantemente aggiornato. Una volta che l'utente torna in posizione di partenza (fine ripetizione), viene effettuato un confronto tra il dataframe creato in 'live' e quello preesistente, riferito ad un'esecuzione eseguita correttamente.

Il confronto viene effettuato tramite Dynamic Time Warping, una tecnica utile per confrontare sequenze temporali. Questo tipo di funzione restituisce una certa distanza, la quale, se minore della soglia impostata, reputa la ripetizione corretta e ne incrementa quindi il numero.

Il codice è stato strutturato in modo da essere il più generale possibile, è infatti molto semplice aggiungere un nuovo esercizio da supportare. Basta creare una nuova cartella contenente i file che servono all'esecuzione.

In particolare, servirà inserire:

- un classificatore in grado di rilevare la posizione di partenza dell'esercizio;
- un dataframe che rappresenterà l'esempio di esecuzione corretta dell'esercizio;
- un video di esempio (di esecuzione corretta)
- video della posizione iniziale

Aggiunti questi file alla cartella delle risorse, basterà semplicemente definire una soglia di accettazione per la valutazione dell'esecuzione.

3.1 COSTRUZIONE DEL DATAFRAME

Per far sì che l'applicazione funzioni, è necessario un dataframe. Crearlo però è complesso, ecco perché si è creato uno script "di utilità", BuildDataFrame. Questo ha lo scopo di analizzare un video e crearne un dataframe. Solitamente questo tipo di strutture dati sono in formato json, ma per rendere più intuitiva la sua visualizzazione, nel caso in cui ce ne sia bisogno, viene convertito in un file Excel.

Utilizzare questo script, quindi, ha semplificato molto il lavoro. Soprattutto nel caso in cui si volessero aggiungere dei nuovi esercizi.

3.2 BASICS

Per guidare l'utente nell'utilizzo del software, ma anche per aiutarlo nello svolgimento dell'esercizio, abbiamo creato lo script Basics. Questo ha il compito di analizzare il video della posizione iniziale, in base all'esercizio scelto dall'utente, ricavare quindi i landmarks e "disegnarli" a schermo. A questo punto, l'utente, in maniera molto intuitiva e semplice, si posiziona in maniera corretta. Questo perché, oltre alle linee guida, vengono mostrati anche i landmarks dell'utente stimati in tempo reale.

3.3 THREAD RILEVAMENTO

Come si evince dal nome, questo script è un Thread, che lavora quindi in concorrenza con Interfaccia. Mentre quest'ultimo script ha il solo scopo di mostrare le linee guida all'utente, calcolate da Basics e i landmarks rilevati dalla webcam, ThreadRilevamento svolge la componente principale dell'applicazione. Essendo sempre in esecuzione, si occupa di rilevare in ogni frame se l'utente è in posizione di partenza, tenendo ovviamente conto dell'esercizio scelto. Affinché avvenga, viene utilizzato il classificatore, che, se rileva l'utente in posizione, passa alla fase successiva. In particolare, si aspetta che l'utente "si muova" per poi tornare in posizione di partenza, ovvero quando avrà completato una ripetizione dell'esercizio. Fino a questo momento, le coordinate dei landmarks, rilevate continuamente, vengono inserite in un dataframe volatile. Una volta eseguita la ripetizione, questo dataframe viene confrontato con quello già presente, corrispondente cioè ad una corretta esecuzione dell'esercizio. Qui, viene restituito un valore, una distanza, che se minore della soglia, ritiene la ripetizione eseguita correttamente, incrementandone il numero, che viene dettato. Fatto ciò, il dataframe volatile viene svuotato. Questo procedimento avviene per ogni ripetizione eseguita, strategia utile che consente di non appesantire la dimensione del dataframe "live" e quindi non caricare ulteriormente la CPU.

3.4 DYNAMIC TIME WARPING

Abbiamo utilizzato il DTW come tecnica di confronto, perché tale algoritmo è particolarmente utile per trattare sequenze in cui singole componenti hanno caratteristiche che variano nel tempo, le sue maggiori applicazioni le ritroviamo infatti nello Speech Recognition e nella online Signature Recognition.

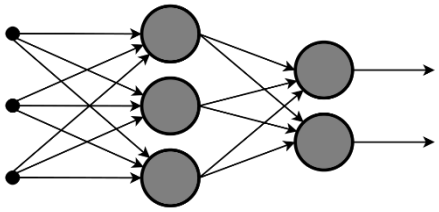
Il DTW viene utilizzato nella fase di confronto dei dataframe e quindi nella valutazione dell'esecuzione. Un dataframe non è altro che una tabella dati eterogenea bidimensionale, a dimensione variabile, con gli assi etichettati. Nel nostro specifico caso, una riga rappresenta un frame, mentre le colonne rappresentano le coordinate dei landmarks in quel frame. Per creare un dataframe e gestirlo, ci viene in aiuto la libreria Pandas. In particolare, calcoliamo per ogni landmark un valore di distanza, ovvero, la differenza tra le coordinate che quel landmark assume nell'esercizio corretto, e le coordinate assunte dall'utente. Viene poi calcolata la media tra queste distanze, in modo da avere una misura generale di distanza rispetto all'esecuzione corretta dell'esercizio. Se questa media è minore della soglia prestabilita, si valuta correttamente l'esercizio, e si incrementano il numero di ripetizioni eseguite.

Un esempio di implementazione del DTW è la seguente:

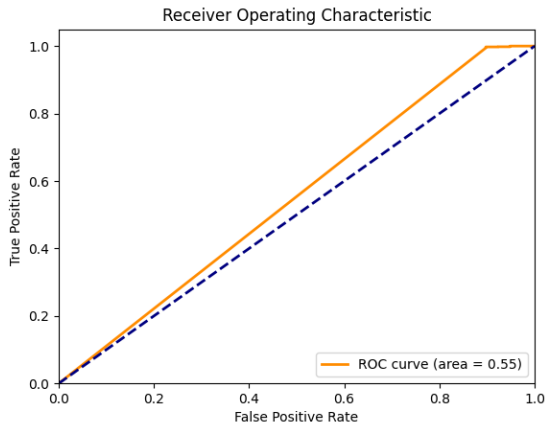
```
int DTWDistance(s: array [1..n], t: array [1..m]) {
    DTW := array [0..n, 0..m]

    for i := 0 to n
        for j := 0 to m
            DTW[i, j] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := 1 to m
            cost := d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],    // insertion
                                        DTW[i  , j-1],    // deletion
                                        DTW[i-1, j-1])    // match

    return DTW[n, m]
}
```

Abbiamo utilizzato una libreria esterna, Fastdtw: dati in input due strutture dati, siano esse monodimensionali o bidimensionali, restituisce la distanza fra esse.

4.0 CLASSIFICATORE



Per quanto riguarda il rilevamento della posizione iniziale e quindi l'inizio e la fine di una ripetizione, è stato deciso di utilizzare un classificatore basato su Rete Neurale Convoluzionale. In particolare, abbiamo costruito un modello sequenziale costituito da tre livelli di neuroni. L'addestramento è stato eseguito con un dataset popolato manualmente, sia con foto prodotte autonomamente, sia con immagini recuperate dal web. Non essendo un dataset estremamente ricco, non abbiamo riscontrato un alto livello di accuratezza nel rilevamento (55%), ma, nonostante ciò, il sistema risulta usufruibile senza troppi problemi.



5.0 CONCLUSIONI

L'applicazione si dimostra funzionare egregiamente su pc, unica piattaforma per ora supportata dall'app, anche se, come verificato, la maggior parte degli utenti avrebbe preferito una sua implementazione su smartphone, per ovvie ragioni di portabilità. È stato inoltre effettuato un test per valutarne le prestazioni in termini di fps, e con una media di 18 frame al secondo, l'app risulta utilizzabile e non particolarmente fastidiosa.

Da sottolineare inoltre, che non avendo a disposizione una webcam collegata direttamente al pc, sono state utilizzate delle webcam esterne, connesse tramite Wi-Fi, che forse hanno leggermente appesantito il carico su CPU, già abbastanza grande. Inoltre, la libreria utilizzata, MediaPipe, lavora su CPU. E quindi rende molto pesante il carico sulla componente hardware. Senza contare il fatto che il confronto dell'esecuzione, utilizzando il DTW, richiede molte risorse computazionali. Riassumendo, tutto il programma pesa sulla CPU, ma questo non compromette l'usabilità e la fluidità. I problemi di questo genere sorgono nel momento in cui sulla macchina si manda in esecuzione un altro programma. Ad esempio, se si utilizza MS Team, con condivisione dello schermo attivata, l'applicazione va in crash.