UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Analisi del movimento umano mediante tecniche di machine learning per il monitoraggio di esercizi fisici"

Relatore: Prof.ssa Gloria Beraldo

Laureando: Giacomo D'Andria

ANNO ACCADEMICO 2022–2023

Data di laurea 29 Settembre 2023

Abstract

La diminuzione dell'esercizio fisico nella popolazione mondiale rappresenta una grande sfida per la società contemporanea. Per permettere una diminuzione delle patologie legate all'inattività motoria e all'avanzamento d'età, mantenendo una buona salute fisica, mentale e sociale, l'OMS promuove dunque il cosiddetto *active ageing*, che mira a migliorare la qualità di vita degli anziani e della popolazione in generale, ponendo alla propria base la promozione di una regolare attività fisica.

Questa tesi dunque ha lo scopo di progettare e implementare un sistema basato su algoritmi di Machine Learning (ML) per l'analisi automatizzata delle coordinate dei giunti principali del corpo umano al fine di monitorare l'esecuzione degli esercizi fisici svolti dagli utenti e permettere il conteggio delle ripetizioni effettuate per diverse classi di allenamento.

Utilizzando una *webcam* o un dispositivo di registrazione esterno (anche di media capacità computazionale), il sistema sviluppato processa le immagini acquisite in *real-time*, estraendo le coordinate dei giunti d'interesse per classificare il tipo di movimento, e allo stesso tempo contare le ripetizioni effettuate.

La metodologia utilizzata si è rivelata un efficace punto di partenza per l'ottenimento dei risultati preposti, riuscendo spesso a conteggiare correttamente il numero di ripetizioni svolte dagli utenti, ma mostrando allo stesso tempo delle debolezze dovute all'utilizzo di *dataset* di dimensioni limitate che, nel caso di angolazioni di registrazione insolite, portano talvolta a una bassa confidenza delle previsioni effettuate.

Indice

1 Introduzione

In un'era spesso dominata da comportamenti sedentari e prolungati periodi di immobilità, la tecnologia ha avuto profonde implicazioni sull'inattività di una percentuale sempre maggiore della popolazione mondiale.

L'OMS riporta che il 28% degli adulti nel mondo non esegue una sufficiente quantità di attività fisica per mantenere una buona salute. Questo problema si estende anche agli adolescenti, tra i quali è stato stimato che almeno l'80% non svolge sufficiente attività motoria.

L'obiettivo di questa tesi è dunque quello di proporre una modalità di esercizio fisico che possa essere eseguita senza il supporto fisico di un istruttore, fornendo feedback agli utenti tramite un sistema digitale basato su tecniche di Machine Learning che permette il conteggio delle ripetizioni eseguite per diverse tipologie di esercizi fisici.

Naturalmente, tramite tale sistema di feedback è inoltre possibile motivare gli utenti tramite messaggi personalizzati che possono essere utilizzati come mezzo per migliorarne lo stato d'animo, oppure, semplicemente, per ricompensarli in seguito alla corretta esecuzione di particolari movimenti.

Per raggiungere tale obiettivo sono state utilizzate delle tecniche di Machine Learning che coinvolgono il riconoscimento delle pose tramite l'analisi dei giunti principali che caratterizzano il corpo umano.

1.1 L'importanza del riconoscimento delle pose

Il riconoscimento delle pose umane tramite l'Intelligenza Artificiale e il Machine Learning rende necessario lo studio della configurazione spaziale di oggetti ed entità presenti all'interno di immagini e video, ricercandone le caratteristiche più rilevanti e significative nel tentativo di riconoscere in esse delle informazioni che permettano di trarre conclusioni sulla natura stessa del problema—ovvero, in altre parole, sulla sua risoluzione.

Tramite tali tecniche si possono eseguire studi sulle azioni umane con il

relativo riconoscimento di movimenti e gesti tramite algoritmi di Computer Vision (CV), che cercano di decrittare e comprendere le intricate relazioni che coesistono tra i diversi giunti del corpo umano durante il movimento [1]. I dati ottenuti tramite questo tipo di analisi trascendono la mera difficoltà tecnica impiegata per ottenerli ed estendono la propria utilità a una grande moltitudine di situazioni reali nelle quali diventa di fondamentale importanza la corretta comprensione e interpretazione della disposizione dei giunti umani, ad esempio, nei seguenti ambiti:

- **Robotica e sanità**: nella robotica viene fatto ampio ricorso al riconoscimento delle pose: i robot che sono in grado di comprendere e interpretare le posture umane possono infatti interagire con gli esseri umani in maniera più naturale, efficiente ed efficace. Si tratta di una sfida di cruciale importanza specialmente in applicazioni come l'assistenza sanitaria, dove i robot assistono la cura dei pazienti aiutandoli e supportandoli come mostrato in Figura 1, ma anche nel caso di ambienti di produzione collaborativa nei quali i robot si trovano a lavorare al fianco di operatori umani—dunque devono essere in grado di interagire correttamente con questi ultimi, nonché reagire di conseguenza alle loro azioni [2].

Figura 1: Esempio di robot dedicato all'assistenza sanitaria

- **Biomeccanica**: nel campo della biomeccanica, il riconoscimento delle posture è di inestimabile valore per diagnosticare disturbi muscoloscheletrici, monitorare il progresso della riabilitazione e progettare piani di trattamento personalizzati, affiancando così l'opinione esperta dei medici alle capacità computazionali dei calcolatori e degli algoritmi per migliorare le diagnosi proposte [3].

- **Sorveglianza**: anche nell'ambito della sorveglianza l'analisi e il relativo riconoscimento delle pose gioca un ruolo centrale, permettendo di potenziare le capacità dei moderni dispositivi per la sicurezza raffinando così la possibilità di individuare e seguire attività sospette in soggetti umani, identificandone le posture o i movimenti del corpo che caratterizzano delle azioni specifiche. Queste tecnologie possono essere utilizzate come fondamentale strumento per garantire la sicurezza pubblica e proteggere le infrastrutture critiche [4]. In Figura 2 viene mostrato un esempio di riconoscimento delle pose umane in un luogo pubblico.

Figura 2: Riconoscimento delle pose umane in un luogo pubblico

- **Guida autonoma di veicoli**: il riconoscimento delle pose è inoltre di fondamentale importanza per i sistemi utilizzati durante la guida dai

veicoli autonomi. Una corretta identificazione delle posture permette infatti una maggior comprensione della situazione nella quale si trovano i veicoli sulla strada e al di fuori di essa, permettendo di interagire in maniera sicura, ad esempio, con pedoni e ciclisti, prendendo decisioni pressoché istantanee e di alta qualità per garantire la sicurezza dei passeggeri e di chi circonda il veicolo come mostrato in Figura 3 [5].

Figura 3: Riconoscimento delle pose umane in relazione alla guida autonoma di veicoli

- **Realtà aumentata e virtuale (AR e VR)**: specie in tempi recenti, il riconoscimento delle pose contribuisce in maniera significativa all'esperienza degli utenti all'interno di realtà aumentata e realtà virtuale. Tramite la comprensione dei movimenti umani viene infatti reso possibile l'adattamento in tempo reale dell'ambiente virtuale nel quale l'utente si trova, garantendo un'esperienza fluida e interattiva che si adatta alle situazioni specifiche nelle quali l'utente si trova [6].

- **Analisi sportiva**: il riconoscimento delle pose riveste inoltre un ruolo cruciale nello studio dei movimenti degli atleti, di cui viene mostrato un

esempio in Figura 4, fornendo una grande quantità di dati fondamentali per la valutazione delle performance, la prevenzione degli infortuni e la pianificazione strategica dei programmi di allenamento, massimizzando le probabilità di successo e minimizzando al tempo stesso i rischi d'infortunio [7].

Figura 4: Il riconoscimento delle pose nell'analisi sportiva

1.2 Tipologie di riconoscimento delle pose

Il riconoscimento delle pose umane si pone l'obiettivo di estrarre informazioni riguardanti l'orientamento e le posizioni dei giunti principali del corpo umano. In questa tesi considereremo 33 giunti totali (numerati da 0 a 32), riportati in Figura 5, a cui corrisponde la rispettiva *label* riportata a piè di pagina[1].

[1] **0**: Nose, **1**: Left Eye (Inner), **2**: Left Eye, **3**: Left Eye (Outer), **4**: Right Eye (Inner), **5**: Right Eye, **6**: Right Eye (Outer), **7**: Left Ear, **8**: Right Ear, **9**: Mouth (Left), **10**: Mouth (Right), **11**: Left Shoulder, **12**: Right Shoulder, **13**: Left Elbow, **14**: Right Elbow, **15**: Left Wrist, **16**: Right Wrist, **17**: Left Pinky, **18**: Right Pinky, **19**: Left Index, **20**: Right Index, **21**: Left Thumb, **22**: Right Thumb, **23**: Left Hip, **24**: Right Hip, **25**: Left Knee, **26**: Right Knee, **27**: Left Ankle, **28**: Right Ankle, **29**: Left Heel, **30**: Right Heel, **31**: Left Foot Index, **32**: Right Foot Index

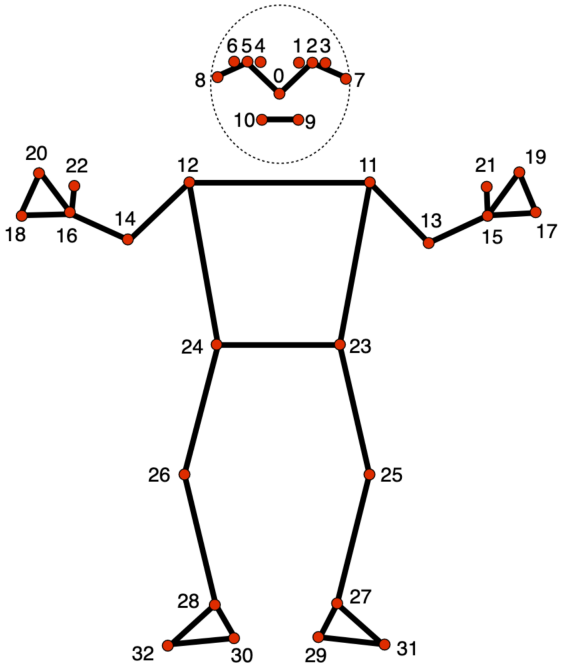

Figura 5: Numerazione dei giunti del corpo umano seguendo la convenzione utilizzata da MediaPipe

Tale divisione dei giunti deriva da quella utilizzata dalla libreria MediaPipe, che adopereremo nello sviluppo dei programmi del presente elaborato e avremo modo di analizzare nel Capitolo 2.

Distinguiamo nel frattempo due diverse tipologie di riconoscimento delle pose:

1. Riconoscimento delle pose *bidimensionale*
2. Riconoscimento delle pose *tridimensionale*

Che analizzaremo separatamente nelle sezioni seguenti.

1.2.1 Riconoscimento delle pose 2D

Nel riconoscimento bidimensionale delle pose lo spazio considerato coincide con $\mathbb{R}^2$, dunque ogni giunto chiave preso in considerazione sarà formato da

tuple del tipo (x_i, y_i), dove x_i, $y_i \in \mathbb{R} \, \forall \, i$ [8].

Per effettuare questo tipo di riconoscimento vengono comunemente utilizzate le reti neurali convoluzionali (ovvero CNN) tramite cui si ottengono degli ottimi risultati sia nel riconoscimento del movimento di persone singole, come, nondimeno, nel caso in cui sia presente un numero maggiore di persone [9].

Un importante modello utilizzato per tale scopo è OpenPose[2] [10]: il primo sistema di riconoscimento delle pose multi-persona in tempo reale che riesce a rilevare contemporaneamente i giunti chiave di busto, mani, viso e piedi umani (per un totale di 135 punti chiave) su singole immagini. Tramite tale modello si è dunque in grado di ricostruire la struttura del corpo utilizzando in input immagini, video, feed webcam o telecamere ad alte performance per applicazioni di computer vision, restituendo come output la visualizzazione dei *key-points* sovrapposti alle immagini originali, permettendo inoltre il salvataggio delle coordinate degli stessi per consentire ulteriori elaborazioni.

Un'altra tecnica utilizzata per il riconoscimento delle pose bidimensionali è quella che utilizza i Pose ResNet [11].

I ResNet, acronimo di *Residual Networks* (Reti Residuali), emersero come una rivoluzionaria innovazione nel campo del deep learning, principalmente progettata per affrontare il problema della scomparsa del gradiente [3] nei neural network molto profondi. Le reti neurali tradizionali esperivano infatti un

[2] https://github.com/CMU-Perceptual-Computing-Lab/openpose

[3] Il problema della scomparsa del gradiente si riferisce alla tendenza del gradiente a diventare sempre più piccolo (eventualmente infinitesimo) durante la fase di back-propagation. A causa di ciò, i pesi dell'intero network sono modificati in maniera molto lenta, dunque ci vorrà più tempo per arrivare alla convergenza dei medesimi.

La back-propagation riguarda invece la tecnica utilizzata nell'allenamento di modelli di Machine Learning partendo dal nodo di output della rete e calcolando "all'indietro" tutti gli errori che vengono commessi rispetto all'output desiderato utilizzando la funzione di loss. Per i diversi nodi viene successivamente calcolato il suo contributo all'errore totale commesso in output, in maniera tale da aggiornare i pesi sulla base dell'errore ottenuto cercando di minimizzare la funzione di loss. Nel caso in cui in gradiente svanisca, tale errore non può essere calcolato correttamente, dunque rende la tecnica meno efficace.

indebolimento dei valori di gradiente con l'aumentare della propria profondità, portando a difficoltà di addestramento e limitati miglioramenti delle prestazioni. La soluzione introdotta tramite i ResNet fu quella di introdurre i cosiddetti blocchi residui (*residual blocks*).

Tali blocchi residui contengono delle "connessioni salto" (*skip connections*), le quali consentono al gradiente di fluire attraverso la rete senza incontrare degli strati di attivazione che possano produrre la scomparsa dello stesso. Queste connessioni consentono alla rete di far fluire alcune delle informazioni senza bisogno di astrazioni, mantenendo dunque delle informazioni dettagliate su alcuni aspetti del problema che non si vogliono astrarre tramite delle rappresentazioni ad alto livello. Abilitando tale flusso diretto è stato reso possibile l'addestramento di reti estremamente profonde in maniera piuttosto efficace [12].

Nel contesto del riconoscimento delle pose, l'utilizzo di reti di questo tipo permette ai modelli di conservare contemporaneamente informazioni locali e globali sui giunti, portando così a una maggior precisione e accuratezza. Un ulteriore vantaggio dell'utilizzo di architetture ResNet è la loro capacità di supportare il transfer learning [4], permettendone così un loro utilizzo ancora più efficiente e dinamico.

1.2.2 Riconoscimento delle pose 3D

Il riconoscimento tridimensionale delle pose tenta di individuare correttamente anche la profondità dei vari giunti rispetto agli altri. Si tratta dunque di individuare le corrette tuple (x_i, y_i, z_i), $x_i, y_i, z_i \in \mathbb{R} \, \forall \, i$ che rappresentano le tre coordinate spaziali tridimensionali [13]. Tale caratteristica risulta essere di fondamentale importanza specialmente nello sviluppo di applicazioni di robotica o realtà aumentata, nelle quali è necessaria anche la terza dimen-

[4] Il transfer learning è una tecnica utilizzata per raffinare l'allenamento di modelli già di per sé allenati. Viene utilizzato nella risoluzione di problemi che hanno molte caratteristiche in comune e hanno soltanto bisogno di imparare gli esempi specifici che sono di rilevanza per il problema.

sione per comprendere il mondo e muoversi al suo interno.

Distinguiamo due diverse tipologie di stime delle pose tridimensionali:

- **Monoculare**: questo tipo di stima della posizione coinvolge la presenza di un solo "occhio", tramite il quale si cerca di calcolare la posizione tridimensionale dei soggetti partendo da un'immagine bidimensionale o da un singolo frame video [14]. Questa operazione costituisce una sfida di non poca difficoltà, specialmente a causa dell'intrinseca ambiguità legata alla trasformazione delle caratteristiche bidimensionali nello spazio tridimensionale.

  Una tecnica utilizzata per risolvere tale problematica è la cosiddetta MonoDepth [15], la quale coniuga la previsione della profondità con la stima della posizione bidimensionale per calcolare una possibile coordinata rappresentante la terza dimensione dei giunti umani.

- **Multi-vista**: nel caso in cui siano disponibili più di una singola prospettiva dalla quale visualizzare un oggetto o una scena, è possibile sfruttare il confronto incrociato tra i diversi dati per ottenere una stima, più o meno accurata, della terza dimensione [16].

  Una tecnica utilizzata a tale scopo è la cosiddetta OpenMVG [17], che tenta di estrarre le caratteristiche chiave dalle diverse immagini, calcolando le corrispondenze tra esse nelle diverse prospettive e, tramite triangolazione, calcolare la geometria tridimensionale dell'intero sistema.

Naturalmente, alcuni problemi di stima della posizione traggono vantaggio da approcci ibridi che combinano le capacità sia delle tecniche 2D che di quelle 3D. Ad esempio, un modello di stima della posizione 2D può essere utilizzato per valutare le coordinate dei punti chiave in ciascun frame di un video, e successivamente un modello separato di stima tridimensionale può essere impiegato per ricostruire la posizione tridimensionale a partire dai punti chiave 2D nel corso del tempo.

1.3 Scopo della tesi

Il principale obiettivo della presente tesi è quello di valutare le performance di diversi algoritmi di Machine Learning per il riconoscimento di alcuni esercizi fisici, al fine di contare le ripetizioni dei medesimi eseguite dagli utenti. In particolare, ci proponiamo di addestrare algoritmi in grado di riconoscere tre diversi esercizi eseguiti di fronte a una webcam o a un dispositivo di acquisizione video (anche di media capacità computazionale), al fine di consentirne il monitoraggio automatizzato e l'eventuale feedback tecnico o motivazionale durante l'esecuzione degli esercizi stessi.

I dati utilizzati per l'addestramento dei modelli sono costituiti da sequenze preregistrate di video dai quali sono state estratte le coordinate dei giunti d'interesse dei soggetti umani contenuti in essi. Per ognuno degli esercizi sono state estratte le coordinate delle posizioni terminali in molteplici situazioni, etichettando ognuna di esse con il nome identificativo dell'esercizio che viene eseguito e la corrispettiva denominazione terminale. Verranno analizzati tre diversi esercizi: pushup, situp e squat, per i quali verranno distinte le classi 'up' e 'down', in riferimento alle relative posizioni dei giunti.

Per create questo dataset[5] sono stati utilizzati circa 500 video prevenienti dal Countix Dataset[6], un'importante risorsa utilizzata nella ricerca scientifica nel campo dell'analisi delle attività umane e del riconoscimento delle pose fisiche.

Il lavoro svolto nella presente tesi riguarda dunque il processo di sviluppo relativo al riconoscimento delle posizioni terminali dei predetti esercizi, da cui può essere successivamente estratto il numero di ripetizioni eseguite nello svolgimento degli stessi.

1.4 Struttura della tesi

La presente tesi è strutturata nei seguenti capitoli:

[5] https://github.com/giacomodandria/tesi-triennale/tree/main/training_data
[6] https://www.deepmind.com/open-source/kinetics

- **Capitolo 1**: in questo capitolo viene introdotta la tematica del riconoscimento delle pose umane e la sua significatività nel mondo contemporaneo, esplorando le diverse tipologie (bidimensionali e tridimensionali) di riconoscimento delle pose.

- **Capitolo 2**: in questo capitolo vengono brevemente descritti gli algoritmi utilizzati (regressione logistica, gradient descent stocastico, random forest, naive Bayes multinomiale, naive Bayes gaussiano, k-nearest neighbors, support vector machines, multi-layer perceptron), nonché le relative librerie utilizzate per l'implementazione in Python (sklearn, pandas, numpy, openCV, pickle, matplotlib, time, csv, os), assieme a una digressione più estesa riguardante la libreria MediaPipe, della quale vengono discusse diverse funzionalità principali.

- **Capitolo 3**: in questo capitolo vengono implementati i diversi algoritmi (regressione logistica, gradient descent stocastico, random forest, naive Bayes multinomiale, naive Bayes gaussiano, k-nearest neighbors, support vector machines, multi-layer perceptron), dei quali vengono testate le performance e la miglior configurazione degli iperparametri.

- **Capitolo 4**: in questo capitolo viene eseguito il test del miglior modello in *real-time*, utilizzando la funzionalità del conteggio delle ripetizioni.

- **Capitolo 5**: in questo capitolo vengono riassunti i risultati ottenuti nel corso della tesi, proponendo direzioni future per ampliare il lavoro svolto nel campo del riconoscimento delle pose.

2 Metodologia

2.1 Librerie e moduli Python utilizzati

Analizziamo di seguito le diverse librerie che sono state utilizzate nello sviluppo dei programmi di questa tesi, ponendo particolare attenzione agli aspetti più strettamente rilevanti ai programmi sviluppati.

2.1.1 `scikit-learn` (sklearn)

Scikit-learn[7] fornisce una vasta gamma di algoritmi per la classificazione, la regressione, il clustering, la riduzione della dimensionalità e molte altre utilità di grande importanza nello sviluppo di applicazioni per la risoluzione di una vasta gamma di problemi nell'ambito del machine learning. Scikit-learn è progettata per essere allo stesso tempo facile da usare e potente, fornendo una vasta gamma di algoritmi di apprendimento supervisionato e non.

Oltre agli algoritmi in sé, Scikit-learn offre anche diversi strumenti per la preparazione dei dati. Inoltre, la libreria offre diverse funzionalità di valutazione dei modelli, associando all'allenamento degli algoritmi report e metriche per valutarne le performance.

2.1.2 `Pandas`

Pandas[8] è una delle librerie più utilizzate per la manipolazione e l'analisi dei dati. Fornisce strutture dati che permettono di organizzare i dati di un problema in maniera tale da renderli accessibili direttamente all'interno del programma, filtrando, aggregando e organizzando i dati senza doverli importare da fonti esterne ogni qualvolta siano necessari.

I DataFrame (`df`) di pandas sono strutture dati bidimensionali simili a delle tabelle, i quali consentono di organizzare e manipolare dati in maniera flessibile ed efficiente.

[7] https://scikit-learn.org/stable/
[8] https://pandas.pydata.org/

Tramite Pandas è inoltre possibile calcolare importanti statistiche sui dati disponibili per cercare di comprenderne meglio la struttura o le caratteristiche specifiche, permettendo anche di gestire eventuali dati mancanti stimandone il valore, o eventualmente eliminandoli.

2.1.3 NumPy

NumPy[9] fornisce molti degli strumenti utilizzati dalle diverse librerie di machine learning. L'efficienza e la versatilità delle strutture dati offerte da NumPy per eseguire i calcoli e le elaborazioni di dati sono infatti molto utilizzate nei diversi algoritmi.

Un importante tipo di dati introdotto da NumPy è il cosiddetto `ndarray`, che è una struttura dati multidimensionale simile a un array in algebra lineare nello spazio $\mathbb{R}^n$. Su tali `ndarray` possono essere eseguite diverse operazione matematiche e statistiche, permettendone l'analisi in maniera piuttosto celere e dinamica.

Le operazioni tra ndarray eseguite tramite NumPy sono rese molto efficienti grazie alla funzionalità di broadcasting, che consiste nella capacità di allineare automaticamente le dimensioni di array diversi e replicare i dati, quando necessario, per effettuare operazioni tra array incompatibili senza la necessità di espliciti cicli o copie di dati aggiuntive.

2.1.4 OpenCV (cv2)

OpenCV[10] (*Open-source Computer Vision library*) è una libreria ampiamente utilizzata nell'elaborazione delle immagini e video in computer vision.

L'elaborazione di immagini tramite OpenCV avviene utilizzando filtri e trasformazioni geometriche, rilevando i bordi e le caratteristiche principali di ciò che si sta analizzando per evidenziarne le *features* più rilevanti. Tramite OpenCV è inoltre possibile effettuare il riconoscimento automatico di oggetti,

[9] https://numpy.org/
[10] https://opencv.org/

volti e diverse altre funzionalità. Nell'ambito specifico della presente tesi OpenCV verrà utilizzata primariamente per interagire con video preregistrati o *feed* live ottenuti tramite webcam.

2.1.5 `Pickle`

Il modulo pickle[11] fa parte della libreria standard di Python e fornisce strumenti per la serializzazione e la de-serializzazione di oggetti.

La serializzazione è il processo di conversione di un oggetto in una sequenza di byte, mentre la de-serializzazione è il processo inverso, ovvero la conversione di una sequenza di byte in un oggetto Python.

Tramite tale tecnica viene reso possibile il salvataggio di strutture dati come i dizionari o le liste, nonché classi personalizzate o modelli già allenati di algoritmi che possono essere direttamente caricati in memoria quando necessario.

2.1.6 `Matplotlib`

Matplotlib[12] è una libreria che fornisce una vasta gamma di strumenti per la creazione di grafici e visualizzazioni di dati, tra cui istogrammi, grafici a barre, grafici a torta e molte altre tipologie utili alla visualizzazione di dati.

È possibile personalizzare pressoché ogni aspetto dei grafici creati con Matplotlib, compresi colori, stili, nomi degli assi, legende e titoli. Questo consente di adattare i grafici alle specifiche esigenze di visualizzazione, nonché la possibilità di esportare i grafici in vari formati (PNG, JPEG, PDF, SVG, etc.).

Matplotlib si integra facilmente con NumPy e Pandas, consentendo di visualizzare facilmente dati da array o DataFrame.

[11] https://docs.python.org/3/library/pickle.html
[12] https://matplotlib.org/

2.1.7 Time

Il modulo time[13] fa parte della libreria standard di Python e fornisce funzioni per misurare il tempo e creare ritardi nei programmi. La funzione `time.time()` restituisce il tempo corrente in secondi dal 1 gennaio 1970 (noto come l'*epoch* time). La funzione `time.sleep()` permette invece di mettere in pausa l'esecuzione del programma per un certo numero di secondi, utile per gestire i ritardi o le pause tra le operazioni.

È inoltre possibile utilizzare la libreria time per misurare quanto tempo impiega una parte specifica del codice durante l'esecuzione.

2.1.8 CSV

Il modulo csv[14] fa parte della libreria standard di Python e offre strumenti per la lettura e la scrittura di file in *Comma-Separated Values* (CSV), un formato di file comunemente utilizzato per memorizzare dati tabulari. Se il file CSV utilizza un separatore diverso dalla virgola è possibile specificarlo tramite un apposito parametro.

2.1.9 OS

Il modulo os[15] fa parte della libreria standard di Python e fornisce una serie di funzioni per interagire con il sistema operativo. È possibile utilizzare le funzioni di os per creare, spostare, rinominare, eliminare file e cartelle, oltre a ottenere informazioni sulle proprietà dei file. È inoltre possibile accedere alle variabili di ambiente del sistema e modificarle; avviare nuovi processi, ottenere informazioni sui processi in esecuzione e comunicare con essi.

[13] https://docs.python.org/3/library/time.html
[14] https://docs.python.org/3/library/csv.html
[15] https://docs.python.org/3/library/os.html

2.1.10 Mediapipe

MediaPipe[16] è un framework open source, cross-platform sviluppato da Google Research che consente a sviluppatori e ricercatori di creare applicazioni in diversi ambiti dell'intelligenza artificiale tramite dei modelli di machine learning preallenati [18].

MediaPipe è stato ufficialmente introdotto da Google Research nel giugno 2019 con la principale motivazione di fornire un framework unificato, flessibile ed efficiente per la creazione di programmi e applicazioni che potessero sfruttare la potenza e versatilità dei modelli preallenati anche tramite l'utilizzo di dispositivi di media capacità computazionale. Inoltre, tramite MediaPipe è diventato spesso possibile semplificare il processo di sviluppo di nuovi programmi, eliminando l'utilizzo diretto di molte delle librerie utilizzate in precedenza come ad esempio TensorFlow o PyTorch, sostituendole con le soluzioni ML formite da MediaPipe.

L'architettura di MediaPipe è caratterizzata da due diverse soluzioni: (1) MediaPipe Studio e (2) MediaPipe Low-Code API. Tramite (1) si utilizzano dei blocchi di costruzione modulari e personalizzabili noti come "grafi MediaPipe". Un grafo MediaPipe è un grafo aciclico diretto (DAG) che rappresenta il flusso di dati e le diverse fasi di elaborazione. Ogni nodo del grafo rappresenta un'unità di elaborazione, la quale può, ad esempio, includere attività di pre-elaborazione di immagini, estrazione di caratteristiche, inferenza di reti neurali e post-elaborazione. MediaPipe Studio è inoltre *web-based*, dunque permette di eseguire dei test direttamente all'interno del browser con i modelli preallenati e/o personalizzabili.

La versione Low-Code API di MediaPipe è stata aggiunta da poco al framework per semplificare lo sviluppo di applicazioni tramite l'utilizzo di codice. Con tale API molte fasi della programmazione vengono astratte proprio dall'utilizzo della libreria, permettendo così agli sviluppatori di focalizzare la propria attenzione sulle componenti più importanti delle proprie

[16] https://developers.google.com/mediapipe

applicazioni.

Funzionalità di MediaPipe

Le funzionalità di MediaPipe si estendono su un ampio spettro di impieghi. Alcune delle sue funzioni più notevoli sono le seguenti:

- **Rilevamento e identificazione di oggetti in immagini e video**: il rilevamento degli oggetti è un compito fondamentale in computer vision, e MediaPipe fornisce modelli efficienti per questo scopo. Questi modelli sono ampiamente utilizzati nella navigazione autonoma, sorveglianza e robotica. In Figura 6 viene riportato un esempio di rilevamento di oggetti e classificazione d'immagini.

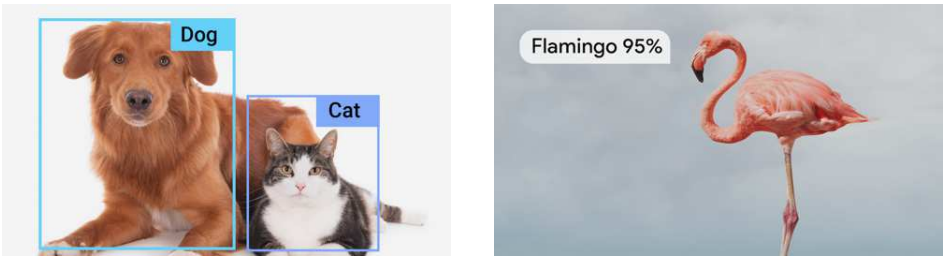

Figura 6: Sulla sinistra un esempio di rilevamento di oggetti, sulla destra un esempio di classificazione d'immagini

- **Riconoscimento dei *landmark* e dei gesti delle mani**: il framework include modelli per il tracciamento delle mani in tempo reale ad alta precisione, spesso utilizzati in applicazioni come il riconoscimento dei gesti e la manipolazione delle mani virtuali, di cui vengono riportati degli esempi in Figura 7.

Figura 7: Sulla sinistra un esempio di riconoscimento dei gesti delle mani, sulla destra un esempio di estrazione dei *landmark* delle mani

- **Rilevamento del volto**: MediaPipe offre modelli altamente efficienti per il rilevamento e il tracciamento del viso, di cui viene riportato un esempio in Figura 8, consentendo la programmazione di applicazioni che utilizzano effetti di realtà aumentata, riconoscimento facciale e analisi delle emozioni.

Figura 8: Sulla sinistra un esempio di rilevamento dei volti, sulla destra un esempio di estrazione dei *landmark* facciali

- **Rilevamento di landmark di posa**: MediaPipe supporta la stima della posa sia nel caso di persone singole, come riportato in Figura 9, come anche in presenza di situazioni multi-persona, rendendolo uno strumento essenziale per l'analisi sportiva e le applicazioni di fitness.

Figura 9: Esempio di estrazione dei *landmark* di posa nel caso di singola persona

MediaPipe rappresenta un punto di riferimento importante nell'evoluzione dei framework di computer vision. La sua architettura modulare, le prestazioni efficienti, la compatibilità multi piattaforma e l'ampia gamma di capacità lo rendono infatti uno strumento indispensabile per sviluppatori e ricercatori.

2.2 Algoritmi utilizzati

In questa sezione verranno analizzati i diversi algoritmi utilizzati in seguito per allenare i classificatori. Verranno discusse le seguenti tecniche:

- Regressione logistica
- Stochastic Gradient Descent (SGD)
- Random Forest
- Naive Bayes Multinomiale
- Naive Bayes Gaussiano
- K-Nearest Neighbors (KNN)
- Support Vector Machines (SVM)
- Multi-Layer Perceptron (MLP)

2.2.1 Regressione logistica

La regressione logistica è un algoritmo di classificazione che sfrutta la funzione logistica definita come

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

Tramite tale funzione si è in grado di ottenere un output i cui valori sono contenuti nel range [0, 1]. Si può dunque interpretare il risultato come la probabilità che un dato esempio appartenga a una classe specifica in base al problema in questione.

Inoltre, tramite l'utilizzo di una funzione derivabile infinite volte (D^{∞}) si potrà ottenere la derivata n-esima di tale funzione senza generare punti di non derivabilità. Durante il processo di allenamento tale caratteristica può risultare molto utile, permettendo al modello di imparare i pesi ottimali che massimizzano la verosimiglianza dei dati di addestramento rispetto ai parametri del modello.

Adattato da scikit-learn.org:

> Il modulo `sklearn.linear_model.LogisticRegression` implementa il classificatore di regressione logistica, noto anche come `logit` o `MaxEnt`. È utilizzato per problemi di classificazione binaria o multiclasse. Il classificatore utilizza l'implementazione di regressione logistica regolarizzata con diverse opzioni di penalità (`L1`, `L2`, `elasticnet` o nessuna penalità). È in grado di gestire dati sia densi che sparsi ed è compatibile con varie opzioni di solver, tra cui `liblinear`, `newton-cg`, `sag`, `saga` e `lbfgs`.

2.2.2 Classificatore *Gradient Descent* stocastico

L'algoritmo di *Gradient Descent* (GD) di tipo stocastico (SGD) consiste nel cercare di minimizzare la funzione di loss finché non si arriva alla convergenza dei pesi della rete a valori stazionari. La versione stocastica del GD è da preferire nel caso in cui l'efficienza sia importante, infatti tramite questo

algoritmo si esplora lo spazio "a tentativi", cercando di avvicinarsi il più possibile al valore del minimo cercato—senza perdere tempo nella ricerca dell'intero spazio in ogni suo dettaglio.

Adattato da scikit-learn.org:

> `sklearn.linear_model.SGDClassifier` è un modello di classificazione lineare implementato utilizzando SGD. L'addestramento avviene campione per campione e il modello viene aggiornato con un tasso di apprendimento decrescente. Il metodo `partial_fit` consente l'addestramento in mini-batch, utile per dataset di grandi dimensioni.

2.2.3 Random Forest

L'algoritmo di *Random Forest* è composto da un insieme di alberi decisionali che vengono addestrati su diversi sottoinsiemi dei dati di allenamento, combinando le loro previsioni con l'obiettivo di ottenere una maggior robustezza della previsione finale, dunque una maggior attendibilità della stessa.

Ogni albero all'interno della "foresta" è un albero decisionale autonomo. Questo processo di suddivisione si basa su criteri come l'entropia, tramite cui si è in grado di misurare la quantità e qualità delle informazioni che ognuno degli alberi contribuisce alla decisione finale.

Adattato da scikit-learn.org:

> Il `RandomForestClassifier` è un meta-estimatore che addestra un insieme di alberi decisionali su vari sotto campioni del dataset e utilizza la media delle previsioni per migliorarne l'accuratezza predittiva e controllare l'overfitting. È possibile controllare la dimensione dei sotto campioni con il parametro `max_samples`, altrimenti viene utilizzato l'intero dataset per costruire ogni albero. Alcuni dei parametri principali includono il numero di alberi, la funzione per misurare la qualità di una divisione, la massima profondità dell'albero, il numero minimo di campioni per dividere un nodo interno e il numero minimo di campioni per una foglia.

2.2.4 Naive Bayes Multinomiale

L'algoritmo Naive Bayes Multinomiale è una variante dell'algoritmo di classificazione Naive Bayes, basato sul teorema di Bayes. Esso fa uso di assunzioni "Naïve" (ingenue) sull'indipendenza condizionale tra le diverse caratteristiche del problema, il che significa che considera come condizionalmente indipendenti anche delle caratteristiche che potrebbero in realtà non esserlo.

Adattato da scikit-learn.org:

> Il classificatore Naive Bayes multinomiale è un modello adatto per la classificazione con caratteristiche discrete. Questo modello utilizza la distribuzione multinomiale, che è un modello statistico utilizzato per descrivere la probabilità di osservare un insieme di eventi discreti, ognuno con una propria probabilità di successo, in un dato numero di prove indipendenti.

2.2.5 Naive Bayes Gaussiano

In maniera simile all'algoritmo multinomiale appena descritto, anche la versione gaussiana dell'algoritmo di classificazione "Naïve" Bayes può assumere l'indipendenza delle diverse variabili del problema anche quando essa non è strettamente verificata. Il suo funzionamento migliore avviene dunque nel caso in cui i dati di allenamento siano normalmente distribuiti rispetto ai valori di media (μ) e varianza (σ).

Adattato da scikit-learn.org:

> La classe `sklearn.naive_bayes.GaussianNB` è un implementazione del classificatore Naive Bayes per dati con distribuzione Gaussiana. Questo classificatore può essere utilizzato per eseguire l'allenamento incrementale dei parametri del modello tramite il metodo `partial_fit`.

2.2.6 K-Nearest Neighbors (KNN)

L'algoritmo KNN può essere utilizzato per la classificazione studiando i K valori più prossimi rispetto a quello corrente. Assumendo dunque la similarità

tra il nuovo esempio osservato e quelli presenti tra i dati di allenamento, lo confronta rispetto ai K valori più prossimi disponibili, inserendo la nuova osservazione nella categoria più simile rispetto ai dati su cui il modello è stato allenato. Un valore troppo piccolo di K può rendere il modello eccessivamente sensibile al rumore, mentre un valore troppo grande può portare a una perdita di dettaglio nelle previsioni.

Adattato da scikit-learn.org:

> Il `KNeighborsClassifier` è un classificatore che implementa l'algoritmo dei k-nearst neighbors. Questo modello di machine learning è utilizzato per la classificazione di dati in base ai vicini più prossimi in uno spazio multidimensionale.

2.2.7 Support Vector Classifier (SVC)

L'algoritmo SVC rappresenta una versione del più generale algoritmo di *Support Vector Machines* (SVM). L'obiettivo principale dell'SVM è quello di trovare un iperpiano (per problemi di classificazione binaria) o una superficie (per problemi di classificazione multi-classe) che separi in maniera ottimale le diverse classi nell'insieme di dati considerato, facendo eventualmente uso del cosiddetto *kernel-trick*[17].

L'addestramento di un SVC comporta la risoluzione di un problema di ottimizzazione convesso, in cui l'obiettivo è massimizzare la larghezza del margine tra i punti di supporto (support vectors) e l'iperpiano di separazione. Questo iperpiano viene scelto in modo da massimizzare il margine di classificazione e minimizzarne l'errore.

Adattato da scikit-learn.org:

> Il `sklearn.svm.SVC` è utilizzato per la classificazione mediante Support Vector Machine (SVM) con supporto per la classificazione mul-

[17] Il *kernel-trick* consiste nel rappresentare i dati del problema in questione all'interno di uno spazio vettoriale di dimensione maggiore rispetto a quello originale, consentendo così la separazione lineare dei dati tramite l'utilizzo di iperpiani che, nello spazio originale, avrebbero necessitato l'utilizzo di superfici non-lineari.

ticlasse.

I principali parametri includono il parametro di regolarizzazione `C`, il tipo di kernel, il grado del kernel, il coefficiente gamma, il peso delle classi (`class_weight`) e diversi altri parametri che possono essere utilizzati per migliorare le performance del modello.

2.2.8 Multi-Layer Perceptron (MLP) Classifier

Il Multi-Layer Perceptron è un neural network *feedforward* composto da uno o più *hidden layer*. Ogni neurone dei diversi strati è connesso a tutti i neuroni del livello precedente e di quello successivo. Gli MLP sono in grado di modellare relazioni piuttosto complesse nei dati, specialmente tramite le trasformazioni non lineari che utilizzano, le quali consentono di catturare informazioni più astratte e complesse nei dati rispetto a quelle lineari.

Adattato da scikit-learn.org:

> Il `MLPClassifier` è un modello di neural network in grado di ottimizzare la funzione di perdita logaritmica utilizzando l'ottimizzatore LBFGS o l'algoritmo precedentemente analizzato di SGD.

3 Codice sviluppato

Lo sviluppo dei programmi utilizzati nella presente tesi è partito dalla scrittura di un programma per ottenere le coordinate dei giunti tramite MediaPipe, salvandole successivamente all'interno di un file `csv` per poterle, eventualmente, utilizzare successivamente per altri scopi—ad esempio per ottenere le coordinate dei giunti durante l'esecuzione di esercizi specifici sui quali si vuole allenare uno degli algoritmi analizzati nelle precedenti pagine.

Inoltre, una versione simile dello stesso programma verrà utilizzata in seguito per ottenere le coordinate in tempo reale a partire dal *feed* della webcam ed effettuare il confronto delle medesime tramite il modello di machine learning allenato, contando così il numero di ripetizioni svolte dall'utente.

Analizziamo dapprima il codice all'interno del notebook

`get_joint_positions.ipynb`[18]

a titolo esemplificativo dell'utilizzo della libreria MediaPipe.

All'interno dello stesso notebook verranno utilizzate anche le librerie `cv2` (OpenCV), `mediapipe` (`mp`), `csv` e `time`.

Per rendere più immediato l'utilizzo delle predette funzionalità è stata definita la seguente funzione

```
1 def save_joint_positions_to_csv(csv_filename: str, duration:
  ↪  int = 10) -> None:
```

Alla quale bisogna passare la stringa contenente il nome del file su cui si vogliono salvare le coordinate dei giunti e la durata del video che si vuole registrare. Nel caso in cui non venga passato il parametro `duration` viene usato il valore di default di 10 secondi.

All'interno di tale funzione vengono assegnati i nomi `mp_pose`, `pose` e `mp_drawing` rispettivamente alla creazione di un alias per la creazione di modelli

[18] https://github.com/giacomodandria/tesi-triennale/blob/main/get_joint_positions.ipynb

sulla stima di posizione di MediaPipe (servirà più tardi come parametro della funzione `draw_landmarks` utilizzata per tracciare le connessioni tra i giunti del corpo come mostrato in Figura 10), al modello vero e proprio di stima della posizione di MediaPipe e, infine, un alias al modulo utilizzato per disegnare i giunti sopra il video o l'immagine che si vuole analizzare.

Figura 10: Esempio di tracciamento delle connessioni tra i giunti del corpo sovrapposte a un'immagine tramite la funzione `draw_landmarks`

Di seguito sono riportate le relative righe di codice

```
mp_pose = mp.solutions.pose
pose = mp_pose.Pose()
mp_drawing = mp.solutions.drawing_utils
```

Utilizzando `OpenCV` (`cv2`) si può ora aprire il *feed* della webcam tramite il comando `cv2.VideoCapture(0)`, dove `0` indica il valore di default che indicizza la webcam. Naturalmente, se quest'ultima non è accessibile il programma termina. Inoltre, tramite `cap.set(3, 640)` e `cap.set(4, 480)` impostiamo e larghezza e altezza (i campi 3 e 4 della funzione `set`) del video come 640×480 px.

Le righe di codice corrispondenti sono riportate di seguito

```
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print('Impossibile aprire webcam.')
    return

cap.set(3, 640)
cap.set(4, 480)
```

Si può ora passate all'ottenimento delle coordinate dal video e al relativo salvataggio in formato `csv`. Innanzitutto, scegliamo di salvare le coordinate x, y, z per ognuno dei giunti individuati da MediaPipe, nonché il frame corrispondente nel quale sono stati riconosciuti.

```
with open(csv_filename, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Timestamp', 'Joint', 'X', 'Y', 'Z'])
```

Poiché avremo bisogno di analizzare, uno a uno, tutti i frame degli s secondi durante i quali si vuole registrare il video, utilizziamo un ciclo `while`.

```
start_time = time.time()
while (time.time() - start_time) < duration:
```

All'interno del quale leggiamo il primo *frame* dal *feed* della webcam utilizzando la funzione `read()` di OpenCV. Tale funzione, oltre al frame corrente, restituisce anche il valore `ret` (return), che indica se l'azione è avvenuta con successo o meno, permettendoci di gestire di conseguenza il fallimento della lettura del frame.

```
ret, frame = cap.read()
if not ret:
    print('Impossibile registrare video.')
```

```
    break
```

Poiché OpenCV utilizza lo spazio dei colori BGR e MediaPipe quello RGB serve convertire il *frame* appena ottenuto da uno spazio di colori all'altro.

```
image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

Soltanto a questo punto si possono estrarre, tramite MediaPipe, i nomi e le coordinate dei giunti

```
results = pose.process(image_rgb)
```

Salvandole poi tramite la funzione `writer.writerow()`.

```
if results.pose_landmarks:
    for joint_id, landmark in enumerate(results.pose_landmarks.landmark):
        writer.writerow([time.time(), f'Joint {joint_id}', landmark.x, landmark.y, landmark.z])
```

E mostrare i giunti e le relative connessioni direttamente nel *live-feed* della webcam

```
mp_drawing.draw_landmarks(frame, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)
```

Un esempio di file generato tramite il programma appena dettagliato viene riportato sotto

```
Timestamp,Joint,X,Y,Z
1692631522.58,Joint 0,0.9084,0.6928,-1.2596
1692631522.58,Joint 1,0.9288,0.6209,-1.1696
1692631522.58,Joint 2,0.9475,0.6190,-1.1697
```

```
1692631522.58,Joint 3,0.9636,0.6175,-1.1700
1692631522.58,Joint 4,0.8642,0.6299,-1.1878
1692631522.58,Joint 5,0.8398,0.6345,-1.1869
...
```

Nota: i valori decimali sono stati troncati per una maggior facilità di lettura.

Nel caso in cui non si desideri leggere il *feed* della webcam ma un video preregistrato si può utilizzare la stessa funzione `VideoCapture()` di OpenCV, passando però in argomento il *path* del video che si vuole analizzare.

```
video_path = 'example/path/to/video.mp4'
cap = cv2.VideoCapture(video_path)
```

Tale funzionalità sarà utilizzata durante la fase di test dei modelli allenati per provare a conteggiare il numero di ripetizioni eseguite in video preregistrati scaricati dal web.

3.1 I dati utilizzati per l'allenamento

Prima di utilizzare il dataset per allenare gli algoritmi descritti nel Capitolo 2 è necessario modificarne marginalmente la struttura. I dati sono infatti organizzati in due diversi file: `landmarks.csv` e `labels.csv`, che contengono, rispettivamente, le coordinate x, y, z dei vari giunti, associate ognuna di esse a un numero intero nella colonna titolata `pose_id`, la quale rappresenta l'intero corrispondente all'interno del file `labels.csv` dove sono invece contenuti i corrispettivi nomi degli esercizi associati alle coordinate.

Per aumentare la robustezza dei modelli, invece di mantenere nello stesso file le coordinate per tutti gli esercizi, sono stati creati degli altri file con struttura

`landmarks_exercise.csv` `labels_exercise.csv`

dove `exercise` rappresenta l'istanza specifica dell'esercizio considerato sul quale si sta allenando il modello.

Utilizzando `pandas` (`pd`) possiamo associare in un unico DataFrame le coordinate con il corrispettivo nome dell'esercizio utilizzando la funzione `merge()`.

```
coordinates_df = pd.read_csv('path/to/landmarks.csv')
labels_df = pd.read_csv('path/to/labels.csv')
data_df = pd.merge(coordinates_df, labels_df, on='pose_id')
```

3.2 Allenamento dei modelli

L'allenamento dei classificatori avviene in maniera piuttosto standard, dividendo i dati in un *train-test-split* del 20%, utilizzando gli iperparametri trovati tramite le tecniche analizzate nel Capitolo 3.3. Per ognuno degli algoritmi utilizzati si tiene traccia della quantità di tempo necessaria per l'allenamento e la ricerca degli iperparametri, salvando inoltre un *classification report* che riassuma le performance ottenute, nonché le relative *confusion-matrix*. Per qualche altro breve commento relativo all'allenamento dei modelli si faccia riferimento al seguente notebook:

exercise_recognition.ipynb[19]

3.3 La scelta degli iperparametri

L'ottimizzazione degli iperparametri è una fase cruciale nell'addestramento dei modelli di machine learning in quanto può significativamente influenzare le loro performance. Tale fase cerca dunque di tenere in considerazione diversi obiettivi come la massimizzazione dell'accuratezza, la minimizzazione dell'errore o la massimizzazione della stabilità del modello.

[19] https://github.com/giacomodandria/tesi-triennale/blob/main/exercise_recognition.ipynb

Scikit-Learn fornisce diverse tecniche per eseguire l'ottimizzazione degli iperparametri, che suddivideremo in tre principali categorie:

- **Ricerca esaustiva (*Exhaustive Search*)**: Questa tecnica prevede di definire un insieme di valori candidati per ciascun iperparametro e quindi esplorare tutte le possibili combinazioni di questi valori. Le due implementazioni più comuni in Scikit-Learn sono Grid Search e Random Search.
- **Ottimizzazione bayesiana**: Questa tecnica utilizza modelli probabilistici per predire quali combinazioni di iperparametri sono più promettenti da esplorare.
- **Algoritmi evolutivi e genetici**: Questi algoritmi prendono ispirazione dalla teoria dell'evoluzione biologica e creano una popolazione di configurazioni di iperparametri che "evolvono" nel tempo per cercare le combinazioni ottimali.

Nel contesto specifico della presente tesi verrà utilizzata la ricerca esaustiva per trovare la miglior combinazione dei valori degli iperparametri.

3.4 Iperparametri utilizzati negli algoritmi

Di seguito vengono riportate le combinazioni di iperparametri che sono state utilizzate per l'allenamento dei modelli utilizzati nella tesi, con una breve descrizione per ciascuno di essi. Vengono dapprima riportati i migliori iperparametri nel caso dell'allenamento dei modelli di riconoscimento multi-esercizio, e successivamente quelli dei modelli di riconoscimento degli esercizi singoli.

3.4.1 Regressione logistica (`LogisticRegression()`)

- `penalty`: Specifica la norma da applicare per la regolarizzazione (L_1 o L_2).

- `C`: Il parametro di regolarizzazione inverso, che controlla la "forza" della regolarizzazione.
- `solver`: Il solutore utilizzato per ottimizzare i pesi (ad esempio, 'liblinear' e 'saga').

Iperparametri modello multi-esercizio:

- `'C': 10, 'penalty': 'l1', 'solver': 'saga'`

Gli iperparametri dei modelli separati sono riportati in Tabella 1:

Esercizio	`C`	`penalty`	`solver`
Pushup	`10`	`l1`	`saga`
Squat	`10`	`l1`	`liblinear`
Situp	`10`	`l2`	`liblinear`

Tabella 1: Iperparametri del modello regressione logistica per gli esercizi di pushup, squat e situp

3.4.2 *Stochastic Gradient Descent* (`SGDClassifier()`)

- `loss`: La funzione di loss da minimizzare.
- `alpha`: Il termine di regolarizzazione.
- `learning_rate`: Il tasso di apprendimento, che controlla la dimensione dei passi durante la fase di SGD.

Iperparametri modello multi-esercizio:

- `'alpha': 0.001, 'loss': 'hinge', 'max_iter': 1000, 'penalty': 'l1'`

Gli iperparametri dei modelli separati sono riportati in Tabella 2:

Esercizio	alpha	loss	max_iter	penalty
Pushup	0.01	hinge	1000	l1
Squat	0.001	log	1000	l1
Situp	0.01	hinge	1000	l1

Tabella 2: Iperparametri del modello *Stochastic Gradient Descent* per gli esercizi di pushup, squat e situp

3.4.3 Random forest (RandomForestClassifier())

- `n_estimators`: Il numero di alberi decisionali nell'*ensemble*.
- `max_depth`: La massima profondità di ciascun albero.
- `min_samples_split`: Il numero minimo di campioni richiesti per suddividere un nodo interno.
- `max_features`: Il numero massimo di caratteristiche da considerare in ogni split.

Iperparametri modello multi-esercizio:

- `'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300`

Gli iperparametri dei modelli separati sono riportati in Tabella 3:

Esercizio	max_depth	min_samples_leaf	min_samples_split	n_estimators
Pushup	None	1	2	200
Squat	None	1	2	100
Situp	None	1	2	100

Tabella 3: Iperparametri del modello Random Forest per gli esercizi di pushup, squat e situp

3.4.4 Naive Bayes multinomiale (`MultinomialNB()`)

Nessun iperparametro significativo da regolare.

3.4.5 Naive Bayes gaussiano (`GaussianNB()`)

Nessun iperparametro significativo da regolare.

3.4.6 k-nearest neighbour (`KNeighborsClassifier()`)

- `n_neighbors`: Il numero di vicini da considerare durante la classificazione.
- `weights`: La funzione di peso utilizzata per attribuire importanza ai vicini (ad esempio, 'uniform' o 'distance').
- `p`: 1 per utilizzare la distanza Manhattan, 2 per quella euclidea

Iperparametri modello multi-esercizio:

- `'n_neighbors': 3, 'p': 2, 'weights': 'distance'`

Gli iperparametri dei modelli separati sono riportati in Tabella 4:

Esercizio	`n_neighbors`	`p`	`weights`
Pushup	3	2	`uniform`
Squat	5	2	`distance`
Situp	3	1	`uniform`

Tabella 4: Iperparametri del modello k-nearest neighbour per gli esercizi di pushup, squat e situp

3.4.7 *Support Vector Classifier* (`SVC`)

- `C`: Il parametro di regolarizzazione inverso.

- `kernel`: Il kernel utilizzato per mappare i dati in uno spazio di dimensioni superiori (ad esempio, 'linear', 'rbf', 'poly').
- `gamma`: Il coefficiente del kernel per kernel 'rbf' e 'poly'.

Iperparametri modello multi-esercizio:

- `'C': 0.1, 'gamma': 1, 'kernel': 'poly'`

Gli iperparametri dei modelli separati sono riportati in Tabella 5:

Esercizio	`C`	`gamma`	`kernel`
Pushup	`0.1`	`scale`	`poly`
Squat	`10`	`0.1`	`rbf`
Situp	`1`	`scale`	`rbf`

Tabella 5: Iperparametri del modello *Support Vector Classifier* per gli esercizi di pushup, squat e situp

3.4.8 Multi-Layer Perceptron (`MLPClassifier()`)

Sono stati scelti i seguenti valori tabulati

- `max_iter=1000, random_state=42`

3.5 Setup e specifiche dell'allenamento

Per la scrittura dei programmi analizzati nella presente tesi è stato utilizzato Jupiter[20] su Windows 11 su un PC HP EliteBook con le seguenti specifiche:

```
Processor Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
```

I tempi di ricerca della miglior combinazione di iperparametri e quelli utilizzati per l'allenamento nel caso del modello multi-esercizio sono riportati in Tabella 6.

[20] https://jupyter.org/

Algoritmo	Tempo ricerca parametri (secondi)	Tempo allenamento (secondi)
Support Vector Classifier	4.88	0.03
Logistic Regression	3.93	0.70
Stochastic Gradient Descent	3.07	0.12
Random Forest	95.67	2.51
Multinomial Naive Bayes	N/A	0.01
Gaussian Naive Bayes	N/A	0.01
K-Nearest neighboor	0.48	0.00
Multi-layer Perceptron	N/A	10.53

Tabella 6: Tempi di ricerca degli iperparametri e tempi di allenamento nel caso del modello multi-esercizio. Le celle N/A (*Not-Applicable*) corrispondono agli algoritmi nei quali non è stata necessaria la ricerca degli iperparametri tramite Grid Search

In Tabella 7 vengono invece riportati i tempi di ricerca della miglior combinazione di iperparametri e quelli utilizzati per l'allenamento nel caso dei modelli relativi ai singoli esercizi di pushup, squat e situp.

La valutazione delle prestazioni dei modelli di machine learning è fondamentale per comprendere quanto bene un modello è in grado di generalizzare su dati non visti. Le metriche di valutazione misurano dunque diversi aspetti delle prestazioni di un modello, a seconda del tipo di problema che si sta affrontando. Tra le principali metriche utilizzate da scikit-learn sono presenti precision, recall e f1-score, analizzati brevemente di seguito:

- **Precision**: La frazione di previsioni positive correttamente identificate rispetto a tutte le previsioni positive.
- **Recall**: La frazione di esempi positivi correttamente identificati rispetto a tutti gli esempi positivi reali.
- **F1-Score**: La media armonica di precision e recall.

In Tabella 8 vengono riportati i valori relativi a precision, recall e f1-score

Algoritmo	Tempo ricerca parametri (secondi)	Tempo allenamento (secondi)
Support Vector Classifier		
Pushup	4.29	0.01
Squat	5.04	0.02
Situp	4.10	0.01
Logistic Regression		
Pushup	0.42	0.07
Squat	0.46	0.02
Situp	0.24	0.01
Stochastic Gradient Descent		
Pushup	0.64	0.01
Squat	0.44	0.01
Situp	0.36	0.01
Random Forest		
Pushup	77.41	1.09
Squat	55.52	0.20
Situp	48.05	0.23
Multinomial Naive Bayes		
Pushup	N/A	0.01
Squat	N/A	0.00
Situp	N/A	0.01
Gaussian Naive Bayes		
Pushup	N/A	0.01
Squat	N/A	0.00
Situp	N/A	0.00
K-Nearest neighboor		
Pushup	0.37	0.00
Squat	0.19	0.00
Situp	0.20	0.00
Multi-layer Perceptron		
Pushup	N/A	2.47
Squat	N/A	1.14
Situp	N/A	1.29

Tabella 7: Tempi di ricerca degli iperparametri e tempi di allenamento del caso dei modelli separati per pushup, squat e situp. Le celle N/A (*Not-Applicable*) corrispondono agli algoritmi nei quali non è stata necessaria la ricerca degli iperparametri tramite Grid Search

Algoritmo	**Precision**	**Recall**	**F1-score**
Support Vector Classifier	0.93	0.93	0.93
Logistic Regression	0.91	0.91	0.91
Stochastic Gradient Descent	0.89	0.89	0.88
Random Forest	0.92	0.92	0.92
Multinomial Naive Bayes	0.79	0.76	0.75
Gaussian Naive Bayes	0.91	0.90	0.90
K-Nearest neighboor	0.95	0.95	0.95
Multi-layer Perceptron	0.93	0.93	0.93
avg $\pm$ st.dev	0.904 $\pm$ 0.046	0.899 $\pm$ 0.056	0.896 $\pm$ 0.059

Tabella 8: Metriche relative a precision, recall e f1-score per il modello multi-esercizio

del modello multi-esercizio, mentre in Tabella 9 vengono riportati quelli divisi per i diversi esercizi di pushup, squat e situp.

In Figura 11 e Figura 12 vengono riportate le *confusion-matrix* relative al modello multi-esercizio, in Figura 13 quelle relative al solo esercizio pushup, in Figura 14 quelle dell'esercizio squat, e in Figura 15 quelle dell'esercizio situp.

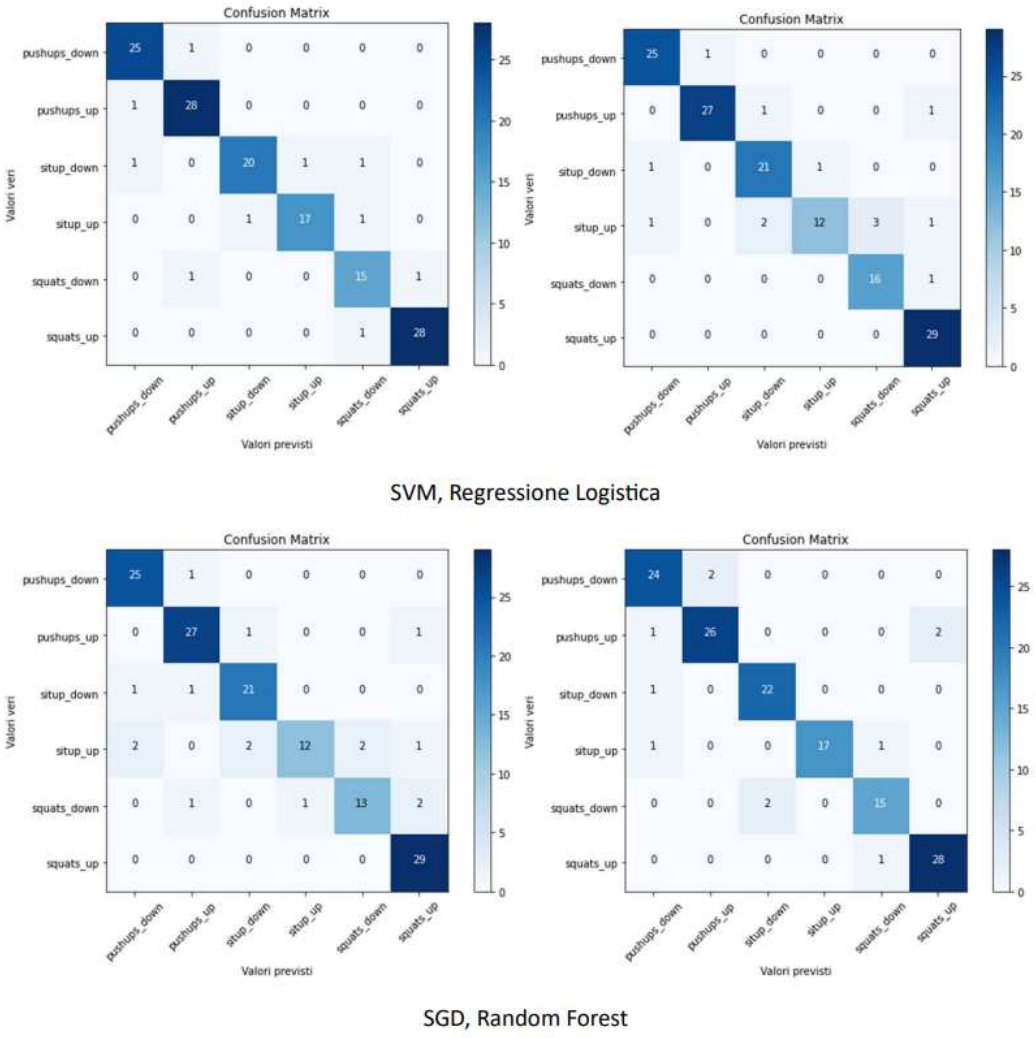

Figura 11: *Confusion-matrix* del modello multi-esercizio SVM, Regressione logistica, SGD e Random Forest

Algoritmo	Precision	Recall	F1-score
Support Vector Classifier			
Pushup	0.94	0.94	0.94
Squat	0.98	0.98	0.98
Situp	0.95	0.95	0.95
Logistic Regression			
Pushup	0.95	0.94	0.94
Squat	0.97	0.96	0.96
Situp	0.98	0.98	0.98
Stochastic Gradient Descent			
Pushup	0.98	0.98	0.98
Squat	0.96	0.96	0.96
Situp	0.89	0.88	0.88
Random Forest			
Pushup	0.96	0.96	0.96
Squat	0.96	0.96	0.96
Situp	0.98	0.98	0.98
Multinomial Naive Bayes			
Pushup	0.88	0.88	0.88
Squat	0.87	0.81	0.81
Situp	0.91	0.90	0.90
Gaussian Naive Bayes			
Pushup	0.92	0.92	0.92
Squat	0.91	0.91	0.91
Situp	0.98	0.98	0.98
K-Nearest neighboor			
Pushup	0.94	0.94	0.94
Squat	0.94	0.93	0.93
Situp	0.95	0.95	0.95
Multi-layer Perceptron			
Pushup	0.94	0.94	0.94
Squat	0.96	0.96	0.96
Situp	0.95	0.95	0.95
Pushup avg ± st.dev	0.939 ± 0.028	0.938 ± 0.027	0.938 ± 0.027
Squat avg ± st.dev	0.944 ± 0.034	0.934 ± 0.051	0.934 ± 0.051
Situp avg ± st.dev	0.949 ± 0.031	0.946 ± 0.035	0.946 ± 0.035

Tabella 9: Metriche relative a precision, recall e f1-score per i modelli separati di pushup, squat e situp

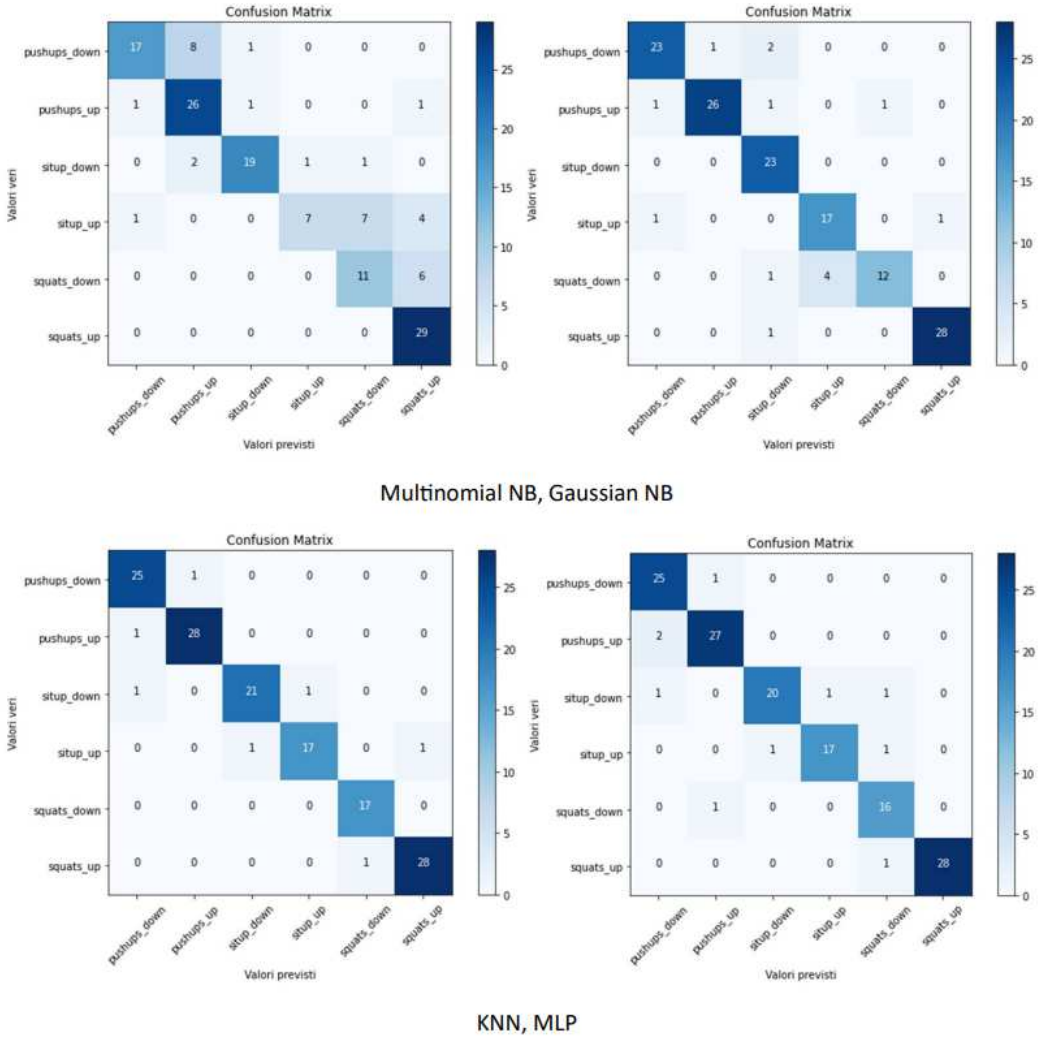

Figura 12: *Confusion-matrix* del modello multi-esercizio Naive Bayes Multinomiale, Gaussiano, KNN e MLP

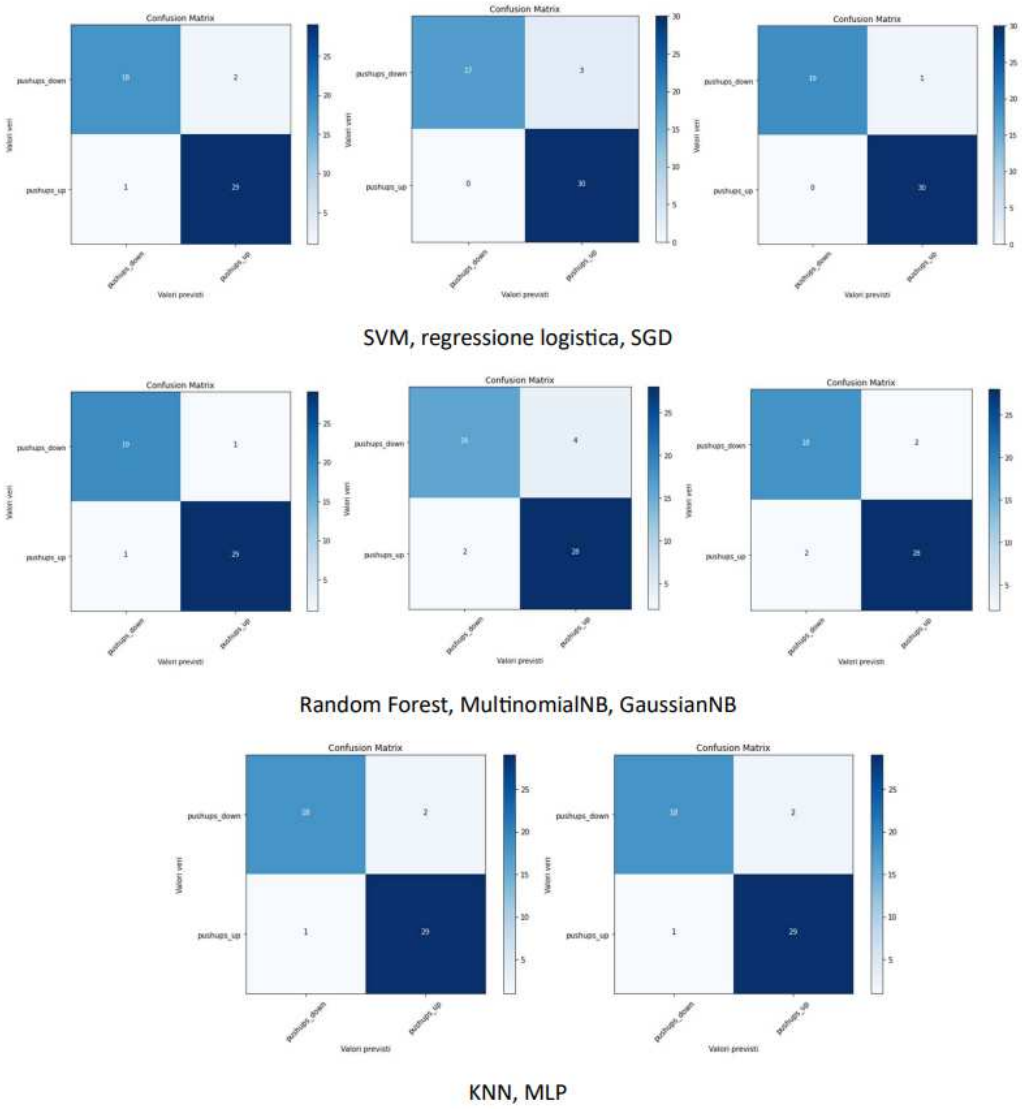

Figura 13: *Confusion-matrix* dei modelli SVM, Regressione logistica, SGD, Random Forest, Naive Bayes Multinomiale, Gaussiano, KNN e MLP relative all'esercizio pushup

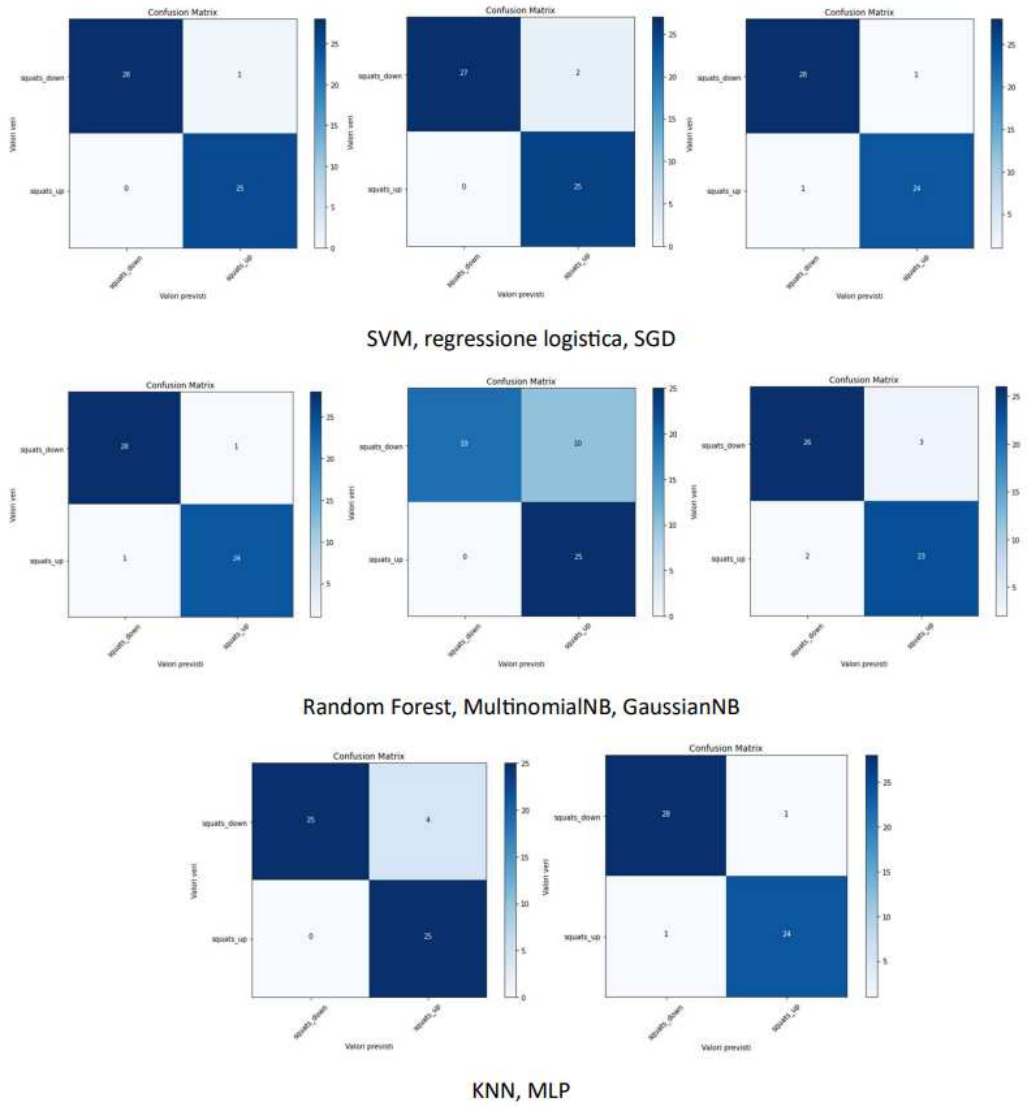

Figura 14: *Confusion-matrix* dei modelli SVM, Regressione logistica, SGD, Random Forest, Naive Bayes Multinomiale, Gaussiano, KNN e MLP relative all'esercizio squat

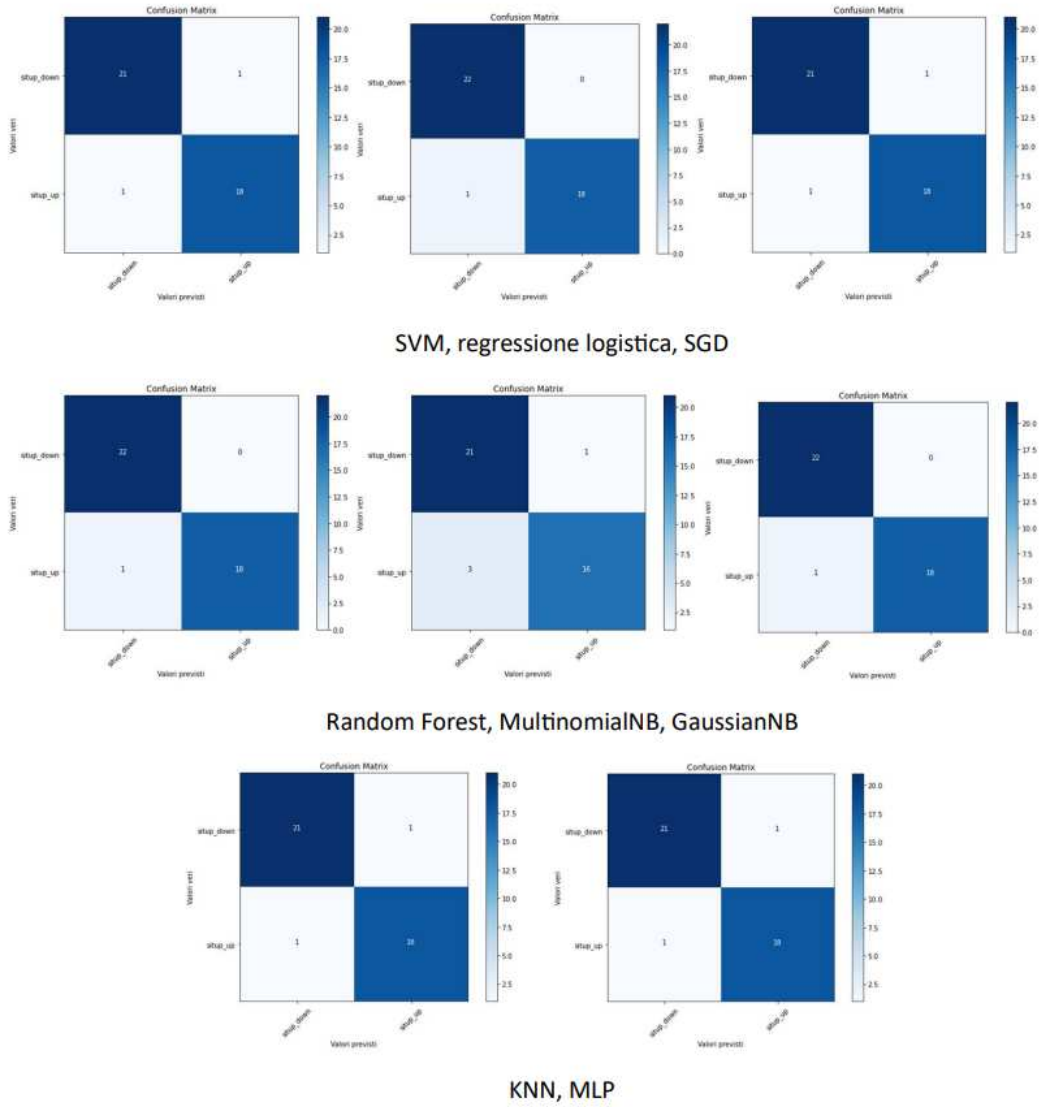

Figura 15: *Confusion-matrix* dei modelli SVM, Regressione logistica, SGD, Random Forest, Naive Bayes Multinomiale, Gaussiano, KNN e MLP relative all'esercizio situp

4 Test dei modelli in *real-time*

La fase finale della presente tesi consiste nella valutazione di uno dei modelli allenati nel tentativo di porlo all'interno di uno scenario reale nel quale possa cercare di riconoscere sia tramite webcam che tramite video preregistrati le posizioni terminali degli esercizi già discussi in precedenza, utilizzando proprio il riconoscimento di tali pose per effettuare un conteggio delle ripetizioni eseguite dall'utente durante lo svolgimento degli esercizi.

Per tale scopo è stato prodotto il seguente notebook:

exercise_recognition_test.ipynb[21]

Un esempio di output video può essere visualizzato nella cartella github[22] del progetto.

La scrittura del codice relativo a quest'ultima fase si è inizialmente focalizzata sull'estrazione e formattazione delle coordinate ottenute tramite MediaPipe per renderle confrontabili con quelle utilizzate durante l'allenamento dei modelli. In particolare, facendo riferimento al seguente codice

```
landmarks_list = []
for landmark in landmarks:
    lines = str(landmark).split('\n')
    for line in lines:
        parts = line.split(':')
        if len(parts) == 2:
            key, value = parts
            key = key.strip()
            value = value.strip()
            if key in ['x', 'y', 'z']:
                try:
```

[21] https://github.com/giacomodandria/tesi-triennale/blob/main/exercise_recognition_test.ipynb
[22] https://github.com/giacomodandria/tesi-triennale/tree/main/output

```
                    number = float(value)
                    landmarks_list.append(number)
                except ValueError:
                    pass
```

vengono estratte le diverse coordiante dei giunti dall'oggetto `landmarks`, il quale contiene delle coppie del tipo nome-valore per le coordinate x, y e z di ogni giunto, nonché un valore nel range [0, 1] che rappresenta la sua visibilità.

Poiché sono d'interesse soltanto le tre coordinate spaziali, vengono estratti solo i loro valori, trasformandoli in *float* rimuovendo il campo del "nome" da ognuna delle coppie, eliminando anche eventuali caratteri di spazio. Al termine del ciclo, la lista `landmarks_list` conterrà tutte le coordinate x, y e z di ognuno dei giunti.

A questo punto, per rendere tali valori confrontabili con quelli utilizzati per l'allenamento, è necessario normalizzarli all'interno del range [0, 1] tramite l'utilizzo dell'operatore `MinMaxScaler()` di Scikit-learn, già utilizzato in precedenza per normalizzare le coordinate dai file di allenamento.

```
formatted_landmarks =
↪  scaler.fit_transform(np.array(landmarks_list).reshape(-1,
↪  1)).reshape(1, -1)
```

Le coordinate appena estratte e normalizzate possono ora essere utilizzate come valori di input per il relativo modello. Per facilitare il conteggio delle ripetizioni e migliorare la robustezza dei modelli, i test vengono eseguiti tramite l'utilizzo dei modelli allenati sugli esercizi singoli.

In Figura 16 viene mostrato qualche frame esemplificativo dei test effettuati sull'esercizio pushup, in Figura 17 quelli relativi all'esercizio situp e in Figura 18 quelli sullo squat.

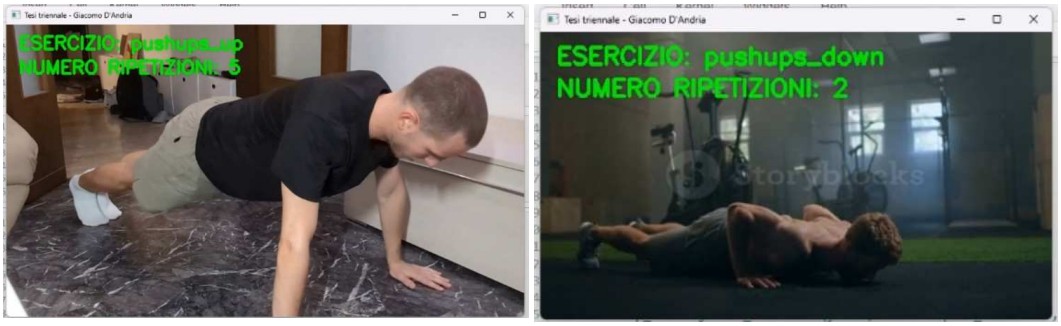

Figura 16: Test effettuati sull'esercizio pushup

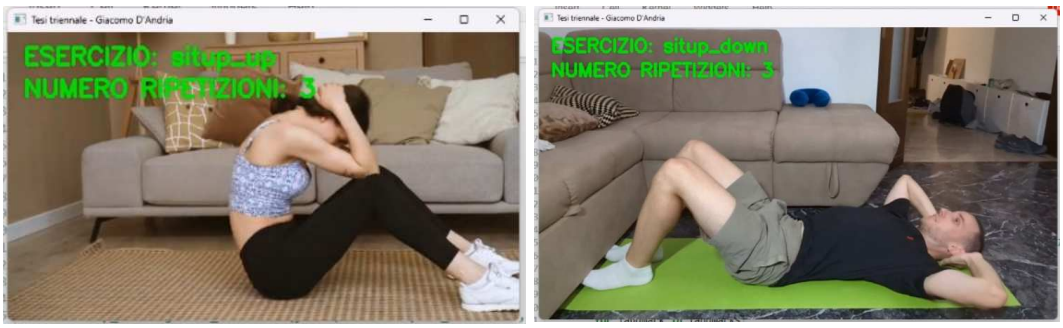

Figura 17: Test effettuati sull'esercizio situp

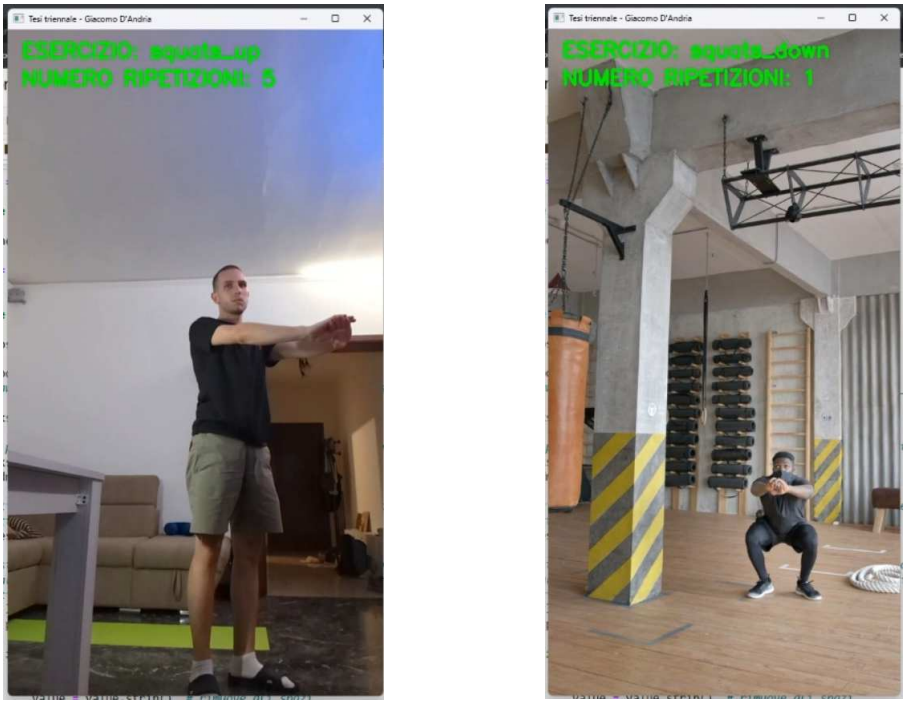

Figura 18: Test effettuati sull'esercizio squat

Per quanto riguarda i risultati ottenuti tramite la tecnica appena analizzata, è stata osservata una maggiore confidenza nel riconoscimento delle posizioni dell'estremo "up" rispetto a quello "down". Una debolezza di tale tipologia causa alle volte la presenza di previsioni errate nel riconoscimento dell'estremo "down", che viene alle volte scambiato con quello "up". In seguito a ciò, il conteggio delle ripetizioni non avviene sempre in maniera accurata, e alle volte viene conteggiato un numero superiore di ripetizioni rispetto a quello effettivamente svolto dall'utente.

5 Conclusione e lavoro futuro

Nella stesura di questa tesi è stata resa possibile la sperimentazione di varie tecniche di machine learning per tentare di effettuare il riconoscimento delle

posizioni durante l'esecuzione di diverse tipologie di esercizio fisico. Inoltre, è stato anche reso possibile l'utilizzo di modelli pre-allenati per estrarre le coordinate dei giunti del corpo umano partendo da una sequenza di immagini (*frame*) al fine di poter confrontare queste ultime con quelle utilizzate durante l'allenamento dei modelli di machine learning.

Al termine di tale procedimento è stata resa possibile l'implementazione di un sistema di conteggio del numero di ripetizioni svolte dagli utenti di fronte a un dispositivo di registrazione video—sia per un'analisi in modalità *live* che per un successivo studio in differita.

Il riconoscimento delle posizioni "up" e "down" si è rivelata una valida strategia per effettuare il conteggio delle ripetizioni, tenendo naturalmente in considerazione le limitazioni dovute all'utilizzo di un *dataset* di modeste dimensioni che, talvolta, non permette una corretta identificazione delle pose, specialmente nel caso in cui il dispositivo di registrazione venga posto in prospettive vastamente differenti rispetto a quelle presenti tra i dati di allenamento.

Per rendere il conteggio più accurato è stato inoltre osservato che sarebbe necessario focalizzare delle attenzioni in più sul riconoscimento delle posizioni "down", che talvolta introducono incertezza nel modello e finiscono per sviare il corretto conto del numero delle ripetizioni effettuate dagli utenti.

Nondimeno, le tecniche utilizzate hanno suggerito la presenza di un'efficace metodologia che può essere ulteriormente espansa per permettere il riconoscimento di una più vasta gamma di prospettive di registrazione, nonché di un maggior numero di esercizi fisici. Per raggiungere tale risultato è però necessaria la creazione di un *dataset* di dimensioni ben maggiori che riesca a catalogare correttamente una più grande quantità di angolazioni di ripresa, permettendone così il successivo riconoscimento.

Per rendere ancor più efficace l'utilizzo di applicazioni di allenamento assistito come quella presentata in questa tesi si può inoltre implementare un ulteriore modello di machine learning che riesca a riconoscere in maniera più

puntuale l'effettiva correttezza dei movimenti eseguiti dagli utenti, oltre al mero conteggio delle ripetizioni eseguite, fornendo di conseguenza il feedback necessario per permettere la correzione delle eventuali posizioni errate assunte durante l'allenamento fisico degli utenti.

Si tratta, naturalmente, di un compito di notevole difficoltà che esula dagli obiettivi della presente tesi, ma rappresenta comunque un'efficace risposta alla sfida che la società contemporanea si trova a combattere, ovvero quella legata al tentativo di diminuire le patologie legate all'avanzamento d'età e all'inattività motoria per mantenere una buona salute fisica, mentale e sociale—ovvero, come è stato definito dall'OMS, la promozione all'*active ageing*.

6 Bibliografia

[1] Ce Zheng et al. *Deep Learning-Based Human Pose Estimation: A Survey*. 2023. arXiv: 2012.13392 [cs.CV].

[2] Óscar G Hernández et al. "Human pose detection for robotic-assisted and rehabilitation environments". In: *Applied Sciences* 11.9 (2021), p. 4183.

[3] Andrea Avogaro et al. "Markerless human pose estimation for biomedical applications: a survey". In: *Frontiers in Computer Science* 5 (July 2023). DOI: 10.3389/fcomp.2023.1153160. URL: https://doi.org/10.3389%2Ffcomp.2023.1153160.

[4] Milan Kresovic and Thong Duy Nguyen. "Bottom-up approaches for multi-person pose estimation and it's applications: A brief review". In: *CoRR* abs/2112.11834 (2021). arXiv: 2112.11834. URL: https://arxiv.org/abs/2112.11834.

[5] Scott Drew Pendleton et al. “Perception, Planning, Control, and Coordination for Autonomous Vehicles”. In: *Machines* 5.1 (2017). ISSN: 2075-1702. URL: https://www.mdpi.com/2075-1702/5/1/6.

[6] Hayet Belghit et al. “Vision-based Pose Estimation for Augmented Reality : A Comparison Study”. In: *CoRR* abs/1806.09316 (2018). arXiv: 1806.09316. URL: http://arxiv.org/abs/1806.09316.

[7] Tiago Guedes Russomanno et al. “Drone-Based Position Detection in Sports—Validation and Applications”. In: *Frontiers in Physiology* 13 (2022), p. 850512.

[8] Alexander Toshev and Christian Szegedy. “DeepPose: Human Pose Estimation via Deep Neural Networks”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition* (June 2014). DOI: 10.1109/cvpr.2014.214. URL: http://dx.doi.org/10.1109/CVPR.2014.214.

[9] Sungheon Park, Jihye Hwang, and Nojun Kwak. “3d human pose estimation using convolutional neural networks with 2d pose information”. In: *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*. Springer. 2016, pp. 156–169.

[10] Z. Cao et al. “OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[11] George Papandreou et al. “Towards accurate multi-person pose estimation in the wild”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4903–4911.

[12] Nikolas Adaloglou. “Intuitive explanation of skip connections in deep learning”. In: *AI Summer* (2020).

[13] Romeo Šajina and Marina Ivašić-Kos. "3D Pose Estimation and Tracking in Handball Actions Using a Monocular Camera". In: *Journal of Imaging* 8.11 (2022). ISSN: 2313-433X. URL: https://www.mdpi.com/2313-433X/8/11/308.

[14] Xiaowei Zhou et al. "Sparseness Meets Deepness: 3D Human Pose Estimation From Monocular Video". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[15] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. "Unsupervised Monocular Depth Estimation with Left-Right Consistency". In: *CVPR*. 2017.

[16] Fuyang Huang et al. "DeepFuse: An IMU-Aware Network for Real-Time 3D Human Pose Estimation from Multi-View Image". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Mar. 2020.

[17] Pierre Moulon et al. "OpenMVG: Open multiple view geometry". In: *International Workshop on Reproducible Research in Pattern Recognition*. Springer. 2016, pp. 60–74.

[18] Camillo Lugaresi et al. "MediaPipe: A Framework for Perceiving and Processing Reality". In: *Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR) 2019*. 2019. URL: https://mixedreality.cs.cornell.edu/s/NewTitle_May1_MediaPipe_CVPR_CV4ARVR_Workshop_2019.pdf.