



UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

Integrazione di classificatori di ADL in App Android

Relatore: Prof.ssa Daniela Micucci

Correlatore: Prof. Marco Mobilio

Relazione della prova finale di:

Gabriele De Rosa

Matricola 829835

Anno Accademico 2019-2020

Sommario

Il riconoscimento dell'attività umana (Human Activity Recognition, HAR) è un campo molto attivo della ricerca e molte tecniche di *deep learning* sviluppate negli ultimi anni hanno dimostrato di essere affidabili per la classificazione delle attività di vita quotidiana (Activities of Daily Living, ADLs).

Malgrado le avanzate tecnologie messe in campo, spesso si incorre in limiti e problemi: alcuni di essi derivanti direttamente dalla non idealità del mondo reale in cui viviamo, altri invece da attribuire ad aspetti puramente organizzativi, di gestione e significato dei dati raccolti.

La seguente trattazione tenta di descrivere quanto svolto durante l'esperienza di stage presso l'Università degli Studi di Milano - Bicocca, durante il quale si è cercato di sviluppare un classificatore di ADLs in grado di apprendere ed analizzare i dati sensoriali ottenuti da un'applicazione Android.

Indice

1	Introduzione	3
1.1	Classificazione	3
1.1.1	L'importanza dei dati	3
1.2	Obiettivo e panoramica del progetto	4
2	Server	5
2.1	RESTful Web API	5
2.1.1	Protocollo e formato	5
2.1.2	Endpoints	6
2.1.3	Implementazione	7
2.2	Ricevitore	7
2.2.1	Protocollo e formato	7
2.2.2	Messaggi	8
2.2.3	Azioni	9
3	Applicazione	10
3.1	Interfaccia	11
3.1.1	Sezione di analisi	11
3.1.2	Sezione di apprendimento	13
3.1.3	Sezione per l'inserimento di dati aggiuntivi	14
3.2	Feedback sonoro	15
3.3	Countdown Timers	16
3.4	Accesso alle API	16
3.5	Sensori di movimento	17
3.6	Comunicazioni con il server	18
4	Classificazione	20
4.1	Apprendimento e Test	20
4.1.1	Caricamento dei dati	20
4.1.2	Valutazione della posizione del dispositivo	21
4.1.3	Visualizzazione grafica dei dati	22
4.1.4	Apprendimento	23
4.1.5	Preparazione dei dati	24
4.1.6	Creazione della rete neurale	25
4.1.7	Statistiche del modello	26
4.1.8	Testing della rete neurale	27
4.2	Predizioni	28
5	Conclusioni	29

Riferimenti	30
Bibliografia	30
Siti	30

Capitolo 1

Introduzione

Il riconoscimento dell'attività umana (Human Activity Recognition, HAR) è un campo molto attivo della ricerca e una quantità sempre maggiore di tecniche di *deep learning* sviluppate negli ultimi anni ha dimostrato di essere affidabile per la classificazione delle attività di vita quotidiana (Activities of Daily Living, ADLs).

Un aspetto che lega la quasi totalità delle tecniche messe in campo quando si parla di HAR è sicuramente l'ottenimento delle informazioni utili per le analisi. Per il riconoscimento di una attività sono indubbiamente necessari i dati ottenuti dai sensori di movimento posizionati sui nostri smartphone o sui sempre più comuni braccialetti ed orologi intelligenti.

1.1 Classificazione

Il riconoscimento delle attività si basa sulla *classificazione*, un problema statistico che ha l'obiettivo di ipotizzare quale tra un insieme di etichette meglio definisce un insieme di caratteristiche.

La classificazione, in informatica, è un ramo dell'apprendimento supervisionato (*supervised learning*), ovvero una branca dell'apprendimento automatico (*machine learning*) che punta ad insegnare ad un sistema informatico una regola generale di calcolo su un certo dominio di dati in modo che successivamente sia in grado di applicare in autonomia le stesse leggi anche a dati futuri.

Definiamo quindi **classificatore** un algoritmo in grado di risolvere il problema della classificazione, in grado di fornire in output l'etichetta che meglio identifica i dati ricevuti in input. Nel caso in esame il classificatore dovrà essere in grado di ipotizzare una attività ricevendo in input un set di dati sensoriali raccolti dall'applicazione sviluppata.

1.1.1 L'importanza dei dati

Ipotizzando che la qualità dei dati sia ottima (o almeno sufficiente), l'aspetto di cui bisogna assolutamente tener conto quando si parla di apprendimento automatico è la quantità di dati che si è in grado di raccogliere. L'efficienza e l'efficacia di un classificatore, in generale, si basano interamente sui valori precedentemente appresi.

La necessità di un set di dati ampio per l'apprendimento è principalmente conseguenza del fatto che non viviamo in un mondo ideale: le attività svolte nella vita reale non sono perfettamente suddivisibili per essere facilmente classificate e inoltre, dato che diverse

persone potrebbero svolgere una uguale attività in modi differenti, non si ha nemmeno una corrispondenza biunivoca tra un insieme di dati e l'attività [1].

Dati necessari per il riconoscimento

È già stato anticipato che i dati su cui solitamente si basa lo sviluppo di un classificatore per il riconoscimento di una attività sono forniti dai più comuni sensori di movimento.

Ad essi ne possono essere aggiunti di ulteriori, come ad esempio le informazioni sulle caratteristiche fisiche dell'individuo o sulla posizione in cui è situato il dispositivo di raccolta.

1.2 Obiettivo e panoramica del progetto

Lo scopo del progetto è quindi lo sviluppo di un classificatore di ADLs in grado di interagire con una applicazione Android.

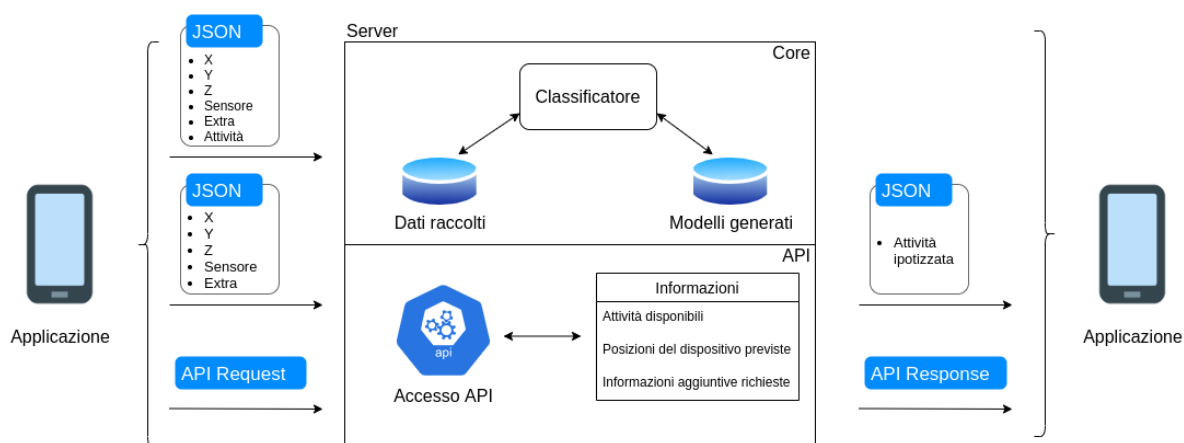


Figura 1.1: Panoramica del progetto

Applicazione

L'applicazione Android ha il compito di ottenere i dati relativi ai movimenti dell'utente e gestire lo scambio di tutte le informazioni raccolte con il server.

Classificatore

Il componente che effettuerà la classificazione, che come abbiamo detto, gestirà la mole di valori ottenuta tentando di eseguire il riconoscimento vero e proprio delle attività.

Rest API

Sono poi state aggiunte delle API Rest utilizzate come fonte iniziale di informazioni. Vedremo essere indispensabili per rendere l'intero sistema programmabile in modo dinamico, almeno per quanto riguarda tutto ciò che sarà impostabile da un amministratore in fase di avvio.

Capitolo 2

Server

Il passaggio principale che ho dovuto affrontare per l'inizio del progetto è stata l'inizializzazione di un server che potesse gestire lo scambio di informazioni con l'applicazione ed occuparsi di svolgere tutte i compiti del caso.

Il lavoro del server è facilmente scomponibile in due parti, la gestione dei dati utili per la classificazione (che sarà poi affrontata nel capitolo 4) e l'erogazione di dati informativi che vedremo consentire ad un amministratore l'interazione con il sistema.

Si è scelto di sviluppare il tutto con il linguaggio Python e di inserire per comodità il software in due differenti contenitori Docker, in modo da mantenere differenziate queste due parti anche a livello programmatico.

Docker

Docker [2] è un progetto open-source in grado di effettuare il deploy di applicazioni all'interno di contenitori software che grazie alla virtualizzazione si trovano ad un livello di astrazione differente dal sistema host. Questa tecnica è molto utile quando si ha vuole mantenere separati l'installazione di un'applicativo dal sistema ospitante.

2.1 RESTful Web API

Come si intuisce in figura 1.1, si è pensato alla creazione di una RESTful Web API, ovvero interfaccia composta da un insieme di *endpoints* pubblici che consentono di ottenere delle informazioni mediante un sistema di richiesta-risposta tra client e server.

Tutti i componenti dell'app e del classificatore interessati a tali informazioni sono in grado di ricavarle con richieste a queste API, il che consente di avere accesso ad informazioni sempre aggiornate. Un cambiamento dei valori in fase di amministrazione consentirà la distribuzione delle modifiche senza la necessità del rilascio di un'aggiornamento dell'applicazione o riprogrammazione del ricevitore.

2.1.1 Protocollo e formato

Tutte le richieste e le risposte utilizzando per lo scambio dati il protocollo HTTP, il comune protocollo di livello applicativo utilizzato nel World Wide Web.

I dati sono trasmessi con il formato JSON, una formato standard che sfrutta una notazione chiave-valore facilmente leggibile.

2.1.2 Endpoints

Gli endpoints sono i punti in cui un software esterno accede a determinate informazioni. L'accesso avviene mediante la chiamata ad un URL web che ci si aspetta risponda con i dati richiesti.

Lista delle attività

Il primo degli endpoints impostato riguarda la lista di tutte le attività che il sistema imparerà a riconoscere. Ogni attività conterrà

- l'identificativo numerico
- il nome
- le traduzioni nelle lingue ammesse
- il tempo (in secondi) indicante la durata necessaria per l'addestramento dell'attività

```
http://IP_ADDRESS:PORT/activities
```

Code Snippet 1: Endpoint per la lista delle attività

Lista delle posizioni del dispositivo

Il secondo endpoint fornisce la lista di tutte le luoghi in cui può essere posizionato il dispositivo durante l'esecuzione di una analisi o di un apprendimento.

```
http://IP_ADDRESS:PORT/positions
```

Code Snippet 2: Endpoint per la lista delle posizioni del dispositivo

Informazioni sui dati aggiuntivi

Il terzo endpoint fornisce il modulo di dati aggiuntivi da richiedere all'utente per mezzo dell'applicazione.

```
http://IP_ADDRESS:PORT/form
```

Code Snippet 3: Endpoint per le informazioni sui dati aggiuntivi

2.1.3 Implementazione

L'implementazione del software è stata effettuata con l'utilizzo di Flask [3], un framework leggero per sviluppare con facilità semplici applicazioni web.

Il codice realizzato si occupa semplicemente di esporre i 3 file JSON contenenti i contenuti nei rispettivi endpoints.

```
from flask import Flask, json

api = Flask(__name__)

@api.route('/activities', methods=['GET'])
def get_activities():
    return json_activities_data

@api.route('/positions', methods=['GET'])
def get_positions():
    return json_positions_data

@api.route('/form', methods=['GET'])
def get_form():
    return json_form_data

if __name__ == '__main__':
    api.run(host='0.0.0.0', port=80)
```

Code Snippet 4: Flask App per una RESTful Web API con 3 endpoints

2.2 Ricevitore

Come è sempre possibile intuire dalla figura 1.1, sul server trovano luogo anche il classificatore, i dati che sono stati raccolti e tutti i modelli generati. In questa sezione ci occuperemo di ciò che riguarda l'applicativo riservato allo scambio dei dati con i client. In particolare la parte del software che si occupa di accettare le connessioni dai client (idealmente l'applicazione realizzata, di cui discuteremo nel capitolo 3) e di immagazzinare i dati ottenuti.

2.2.1 Protocollo e formato

Al contrario di quanto accade con la API, la connessione al *ricevitore* avviene mediante una connessione TCP diretta via Socket. Essendo TCP un protocollo di rete a livello di trasporto, ci consente di avere una velocità maggiore a quanto avremmo avuto utilizzando il protocollo HTTP. Questa scelta è stata effettuata in seguito alla necessità di trasmettere i dati con il minimo ritardo. Malgrado questa scelta si mantiene, per comodità, l'utilizzo del formato JSON anche in questo frangente.

2.2.2 Messaggi

Il ricevitore deve gestire due tipi di richieste, quelle che inviano i dati per effettuarne l'apprendimento e quelle che inviano i dati in attesa di ricevere una predizione. Tuttavia la lettura dei messaggi ricevuti resta la medesima ed il valore relativo alla tipologia sarà un valore interno alle informazioni.

Per concludere, i dati ricevuti in un singolo messaggio sono

- il codice identificante un set di messaggi
- il indice indicante la progressione dei dati
- i valori degli assi (x,y,z)
- il sensore di movimento che ha generato i valori ottenuti
- il valore temporale (*timestamp*)
- la posizione del telefono selezionata durante l'attività in corso

e, in aggiunta, nel caso dell'apprendimento...

- l'attività che si sta svolgendo.

```
{
  "status": "OK",
  "mode": "learning",
  "data": {
    "archive": "bea38ae7-ff27-47eb-b907-de63eeb0772a",
    "type": "data",
    "info": {
      "index": 153,
      "activity": "walk",
      "sensor": "accelerometer",
      "position": "right_hand"
    },
    "values": {
      "x": 1.23456,
      "y": 1.23456,
      "z": 1.23456,
      "t": 1234567
    }
  }
}
```

Code Snippet 5: Esempio di messaggio ricevuto per l'apprendimento

2.2.3 Azioni

Come logico pensare, le azioni da intraprendere per le diverse tipologie di richiesta sono differenti.

Messaggi di Apprendimento

Nel caso della ricezione di dati per l'apprendimento la prima azione da intraprendere è il salvataggio dei dati ricevuti. Si utilizza il formato CSV grazie al quale i vari record, già suddivisi in base al sensore corrispondente, sono organizzati in modo ordinato.

In secondo luogo sarà avviato il processo di *train* del classificatore, che sarà trattato nel capitolo 4.

Archive	Index	X Axis	Y Axis	Z Axis	Timestamp	Positions	Activity
8e9c147c-6fa0-466f-993b-f726a786933c	1	1.325.225.830.078.120	4.625.091.552.734.370	611.468.505.859.375	1592314676717	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	2	15.047.760.009.765.600	4.826.629.638.671.870	5371100.45.00	1592314676719	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	3	176.776.123.046.875	5.177.413.940.429.680	853.094.482.421.875	1592314676721	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	4	17.921.142.578.125	5.622.589.111.328.120	9.804.702.758.789.060	1592314676724	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	5	18.303.985.595.703.100	5.408.004.760.742.180	12.129.501.342.773.400	1592314676790	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	7	329.852.294.921.875	3.734.664.916.992.180	1.715.325.927.734.370	1592314676797	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	8	34.316.558.837.890.600	2.520.126.342.773.430	1.686.737.060.546.870	1592314676799	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	9	233.697.509.765.625	2.123.687.744.140.620	152.672.119.140.625	1592314676870	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	10	10.340.576.171.875	221.173.095.703.125	12.205.001.831.054.600	1592314676873	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	11	-2.960.205.078.125	29.204.254.150.390.600	8.565.078.735.351.560	1592314676876	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	12	-40.704.345.703.125	38.975.830.078.125	34016933.05.00	1592314676879	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	13	1.982.269.287.109.370	4.698.699.951.171.870	507.843.017.578.125	1592314676950	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	14	120.355.224.609.375	53.319.854.736.328.100	5.075.714.111.328.120	1592314676956	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	15	19.846.649.169.921.800	5.592.987.060.546.870	63.537.445.068.359.300	1592314676961	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	16	24.078.521.728.515.600	5.583.526.611.328.120	7.306.808.471.679.680	1592314676968	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	17	22.655.181.884.765.600	54.467.315.673.828.100	783.056.640.625	1592314677031	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	19	115.960.693.359.375	4.864.990.234.375	644538.15.00	1592314677043	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	20	3.939.361.572.265.620	42.266.693.115.234.300	72.860.107.421.875	1592314677049	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	21	-67.889.404.296.875	34.715.118.408.203.100	75.244.293.212.890.600	1592314677111	right_hand	walk
8e9c147c-6fa0-466f-993b-f726a786933c	22	1.704.432.929.125	28.508.810.580.078.100	77.238.234.051.171.800	1592314677117	right_hand	walk

Figura 2.1: Esempio del dataset CSV contenente dati accelerometrici

Messaggi di Analisi

Nel caso di ricezione di dati per l'analisi non si deve immagazzinare passivamente tutte le informazioni ricevute, ma è necessario fornire al client connesso le risposte che si aspetta.

Alla ricezione del numero minimo di record si avvia il processo di predizione del classificatore, trattato sempre nel capitolo 4. Una volta ottenuta l'ipotesi, questa sarà inviata come risposta in formato JSON.

```
{
  "status": "OK",
  "type": "prediction",
  "activity": "run"
}
```

Code Snippet 6: Esempio di messaggio di risposta con l'ipotesi formulata

Capitolo 3

Applicazione

Un altro aspetto chiave del progetto è l'applicazione Android, sviluppata per essere fonte per la raccolta di qualsiasi dato utile alla classificazione. Sintetizzando l'intero funzionamento è possibile raccogliere le funzionalità offerte nella

- raccolta di dati per l'analisi dell'attività.
- raccolta di dati per l'apprendimento.
- raccolta di dati aggiuntivi.

Compatibilità

La compatibilità offerta è con tutte le versioni di Android che supportano la versione delle API 16 o superiore, nella pratica tutte le versioni maggiori o uguali ad Android 4.1 rilasciato nel 2012. Nella pratica, al momento della stesura di questa relazione, ciò assicura il funzionamento dell'app sul 99,8% dei dispositivi con questo sistema operativo.

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0 Ice Cream Sandwich	15	
4.1 Jelly Bean	16	99,8%
4.2 Jelly Bean	17	99,2%
4.3 Jelly Bean	18	98,4%
4.4 KitKat	19	98,1%
5.0 Lollipop	21	94,1%
5.1 Lollipop	22	92,3%
6.0 Marshmallow	23	84,9%
7.0 Nougat	24	73,7%
7.1 Nougat	25	66,2%
8.0 Oreo	26	60,8%
8.1 Oreo	27	53,5%
9.0 Pie	28	39,5%
10. Android 10	29	8,2%

Figura 3.1: Distribuzione cumulativa delle versioni di Android

Linguaggi di sviluppo

I linguaggi utilizzati sono stati *Java* per quanto riguarda l'aspetto programmatico, mentre *XML* per tutti i layout.

Traduzioni

L'intero applicativo, di cui segue la spiegazione, è stato interamente sviluppato con un supporto multilingua: italiano e inglese.

3.1 Interfaccia

Si è voluta dare un'interfaccia minimale sviluppata secondo le linee guida dell'ormai conosciuto Material Design.

Material Design

Con Material Design [4] si intende un linguaggio visivo che *sintetizza i principi classici del buon design utilizzando le nuove innovazioni della tecnologia e della scienza.*

L'intero linguaggio si basa sul concetto fisico "Materiale" di cui si vuole effettuare una trasposizione nel design di tutte le caratteristiche (luci, ombre, etc.) che lo definiscono nel mondo reale.

Suddivisione per funzionalità

Le 3 sezioni presenti suddividono con esattezza le 3 funzionalità offerte: l'analisi dell'attività, l'apprendimento di una attività e l'inserimento di informazioni aggiuntive.

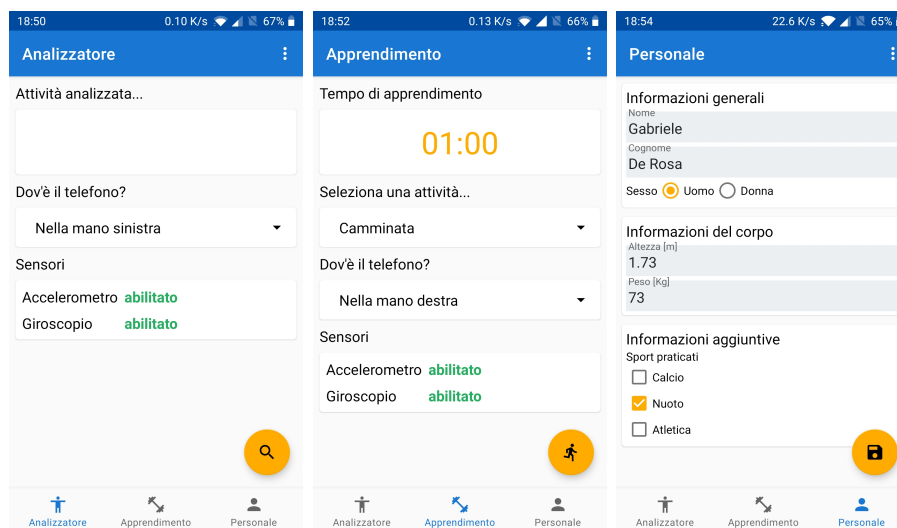


Figura 3.2: Le 3 sezioni principali dell'applicazione

3.1.1 Sezione di analisi

La sezione di analisi consente di testare l'efficacia del classificatore. Per meglio dire, è possibile avviare il processo di invio dei dati sensoriali al server ricevitore che, come abbiamo visto nella sezione 2.2, restituirà in risposta l'ipotesi.

L'insieme delle azioni svolte in questa sezione sono divisibili in 4 fasi.

Fase 1: Inizializzazione

In una prima fase l'applicazione scarica tramite l'utilizzo delle API (viste nella sezione 2.1) le informazioni relative alle posizioni del dispositivo che sono disponibili e selezionabili. L'utente ha quindi la possibilità di selezionare la posizione in cui desidera tenere il telefono durante l'esecuzione della attività ed in seguito avviare l'analisi.

Fase 2: Preparazione

All'avvio dell'analisi sarà inizializzato un servizio in foreground [5] che si occuperà di svolgere tutte le azioni necessarie per le fasi seguenti. La UI continuerà ad essere aggiornata con le informazioni ricevute dal servizio.

Il primo obiettivo è quello di contattare il server per stabilire una connessione TCP ed iniziare lo scambio dati. Qualora la connessione avvenisse con successo passerà qualche altro secondo prima che i sensori inizino la raccolta dei dati. Durante questo *tempo di preparazione*, pensato appositamente in modo da consentire all'utente di prepararsi posizionando il dispositivo nella posizione selezionata, sarà mostrato un countdown.

Fase 3: Analisi

Allo scadere del countdown i sensori vengono abilitati. Ogni informazione raccolta è organizzata in un formato JSON organizzato in modo simile a quanto visto nell'esempio 5 ed inviata come messaggio.

Fase 4: Predizione

Durante il processo di analisi si ottengono numerose risposte dal server che manda sia messaggi di conferme che messaggi contenenti le ipotesi sulla attività. La predizione ottenuta è mostrata all'utilizzatore.

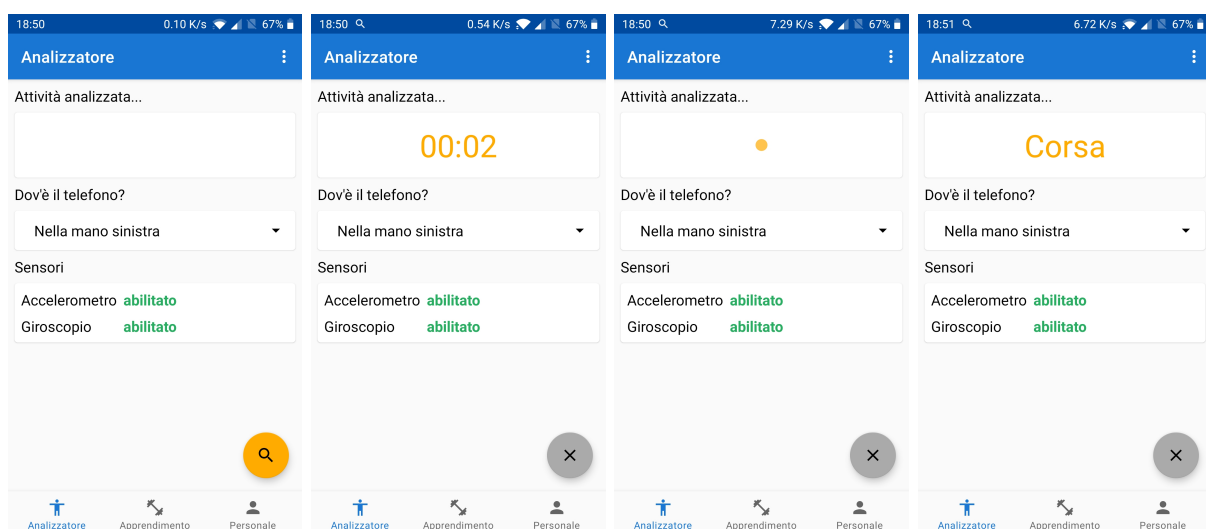


Figura 3.3: Le 4 fasi dell'analisi

3.1.2 Sezione di apprendimento

Nella sezione di apprendimento è possibile avviare un processo di raccolta dati del tutto simile a quanto appena visto, ma al quale si associa anche una attività tra quelle selezionabili.

L'obiettivo sarà poi quello di utilizzare questi dati sensoriali di cui si conosce la corrispondenza pratica per effettuare il *train* della rete neurale da cui si genererà il modello. In questa fase si dà piena fiducia all'utente sulla correttezza dei dati inseriti.

L'insieme delle azioni svolte in questa sezione sono divisibili in 3 fasi.

Fase 1: Inizializzazione

In questa prima fase l'applicazione scarica diverse informazioni tramite l'utilizzo delle API (viste nella sezione 2.1):

- le informazioni relative alle posizioni del dispositivo selezionabili
- la lista di attività addestrabili e il tempo necessario per il loro apprendimento

Prima dell'avvio dell'apprendimento l'utilizzatore dovrà quindi selezionare sia la posizione in cui tenere il telefono durante l'esecuzione che l'attività da allenare.

Fase 2: Preparazione

Anche in questo frangente sarà inizializzato un servizio in foreground [5] che si occuperà di svolgere tutte le azioni necessarie per le fasi seguenti e la UI continuerà ad essere aggiornata con le informazioni ricevute da questo servizio.

Nuovamente come prima cosa viene contattato il server per stabilire una connessione TCP ed una volta ricevuta risposta affermativa sarà avviato il conto alla rovescia per la preparazione.

Fase 3: Apprendimento

Allo scadere del countdown sono avviati in parallelo un ulteriore conto alla rovescia e tutti i sensori per la raccolta dei dati.

Il secondo countdown indica il tempo di esecuzione necessario e potrebbe differire tra le diverse attività. Fino allo scadere del tempo tutti i dati raccolti dai sensori sono inviati al server.

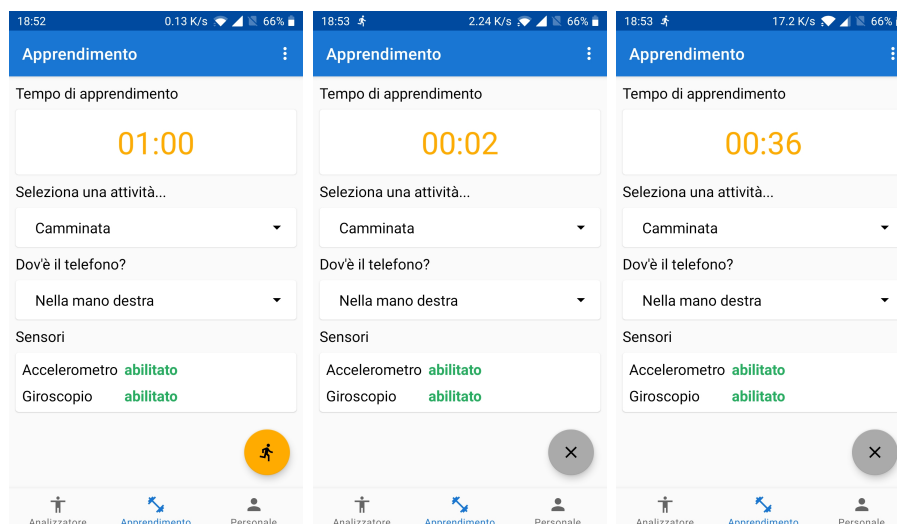


Figura 3.4: Le 3 fasi dell'apprendimento

3.1.3 Sezione per l'inserimento di dati aggiuntivi

La sezione è utile per dare la possibilità di inserire dati aggiuntivi che potrebbero essere utilizzati in qualche modo dal classificatore. Un esempio possibile è la richiesta di dati fisici dell'utente (altezza, peso, etc.) qualora in futuro si decidesse dar loro una rilevanza nella classificazione.

Malgrado agli occhi dell'utente appaia come un semplice modulo di inserimento dati, la sezione in questione è la più dinamica di tutti per quanto riguarda l'interfaccia. Il modulo che richiede all'utente i dati personali è generato programmaticamente per intero a partire da quanto scaricato dal relativo endpoint delle API. Questa opportunità permette ad un amministratore di sistema di variare le richieste senza dover rilasciare un'aggiornamento dell'applicazione.

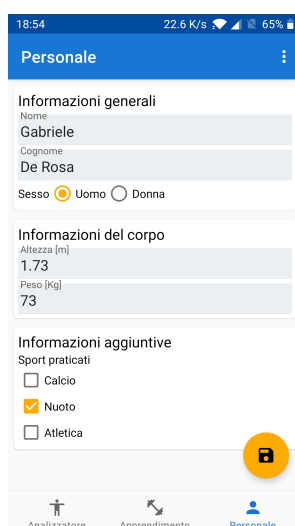


Figura 3.5: Il modulo di inserimento dati generato programmaticamente

3.2 Feedback sonoro

Dal funzionamento che abbiamo appena visto è intuibile che l'app possa essere utilizzata con il dispositivo situato in posizioni che non permetterebbero un collegamento visivo diretto. Quando la posizione selezionata è, per esempio, nelle tasche.

Tale situazione renderebbe impossibile l'accesso alle informazioni relative al progresso dell'attività in corso di esecuzione, che sia un'analisi o un apprendimento. Questa eventualità ha reso necessaria l'implementazione di un feedback sonoro realizzato mediante le librerie di sintetizzazione vocale per il *text to speech* [6] offerte da Android.

Oltre al feedback visivo è offerto quindi un feedback sonoro per tutte le informazioni rilevanti, come l'inizio e la fine dell'attività, il conto alla rovescia ed i risultati dell'attività ipotizzata.

```
// Set TTS object
TextToSpeech textToSpeech = new TextToSpeech(this,
    new TextToSpeech.OnInitListener() {

        @Override
        public void onInit(int status) {
            if (status == TextToSpeech.SUCCESS) {

                Locale curr = new Locale(getString(R.string.current_lang));
                int ttsLang = textToSpeech.setLanguage(curr);

                if (ttsLang == TextToSpeech.LANG_MISSING_DATA
                    || ttsLang == TextToSpeech.LANG_NOT_SUPPORTED) {
                    Log.e("[TTS]", "The Language is not supported!");
                } else {
                    Log.i("[TTS]", "Language Supported.");
                }
                Log.i("[TTS]", "Initialization success.");
            } else {
                Log.e("[TTS]", "Initialization failed!");
            }
        }
    });

// and then
int speechStatus
    = textToSpeech.speak(tts, TextToSpeech.QUEUE_FLUSH, null);

if (speechStatus == TextToSpeech.ERROR) {
    Log.e("[TTS]", "Error in converting Text to Speech!");
}
```

Code Snippet 7: Implementazione del text to speech in Android

3.3 Countdown Timers

I countdown che regolano il tempo di svolgimento delle attività sono implementati utilizzando la classe *CountDownTimer* [7] offerta direttamente da Android.

I due timer implementati non hanno un tempo stabilito in partenza. I secondi utili per la preparazione possono essere facilmente impostati dall'utente mediante un input nelle impostazioni. Il tempo per l'apprendimento dell'attività è invece scaricato dalle API tramite la medesima richiesta delle informazioni sulle attività.

```
// Set the countdown
myCountdown = new CountDownTimer(n * 1000, 1000) { // n seconds
    @Override
    public void onTick(long millisUntilFinished) {
        // callback each second
    }

    @Override
    public void onFinish() {
        // callback on finish
    }
};

myCountdown.start();
```

Code Snippet 8: Implementazione di un conto alla rovescia

3.4 Accesso alle API

Le chiamate alle API, di cui abbiamo visto le varie applicazioni per l'ottenimento di dati informativi, sono fatte utilizzando la libreria Retrofit. Questa libreria, implementando le classi opportune associate ai dati che ci si aspetta dalle risposte (*API Response*), permette di ottenere facilmente le informazioni discusse.

Retrofit

Retrofit [8] è una libreria open source nata per trasformare una richiesta ad una REST API in una Java Interface.

3.5 Sensori di movimento

I sensori di movimento implementati sono **accelerometro** e **giroscopio**.

Entrambi presentano gli stessi valori informativi (i tre assi x, y, z), pertanto durante lo sviluppo sono stati gestiti in modo analogo.

I sensori sono gestiti dal sistema operativo stesso. Per richiedere l'accesso è necessario fare richiesta al gestore (*SensorManager*) per abilitare i sensori necessari ed implementare la classe *SensorEventListener* [9] insieme ai *callback* che saranno chiamati dal sistema ad ogni evento.

Callback

Una callback è una funzione richiamata dal sistema operativo quando si verifica un determinato evento, in modo da implementarne una gestione personalizzata.

```
public void onSensorChanged(SensorEvent event) {

    if(event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {

        // Get values
        float x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];
        // and send data
        sendGyroscopeData(x, y, z);

    }
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {

        // Get values
        float x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];
        // and send data
        sendAccelerometerData(x, y, z);

    }

}
```

Code Snippet 9: Implementazione del callback dei sensori

3.6 Comunicazioni con il server

Oltre a ricavare le informazioni sensoriali è ovviamente necessario procedere al loro invio al server dove verranno processate come visto nella sezione 2.2.

Connessione al server

Per prima cosa però è indispensabile avere una connessione attiva. Tale connessione è richiesta all'avvio dei servizi di analisi e apprendimento.

Come abbiamo precedentemente discusso si è scelta una connessione TCP via socket. La corrispondente implementazione Java avviene proprio mediante la classe *Socket* [10].

```
socket = new Socket(DESTINATION, PORT);
```

Code Snippet 10: Implementazione della connessione via socket

Invio di un messaggio

Una volta attiva la connessione è attiva, si può procedere con l'invio dei dati ogni qual volta che il sistema operativo notifica un cambiamento di stato dei sensori mediante l'apposito callback di cui abbiamo visto l'implementazione.

```
// Buffer to send the message to the server
OutputStreamWriter osw
    = new OutputStreamWriter(socket.getOutputStream());

BufferedWriter bw = new BufferedWriter(osw);
PrintWriter bufferOut = new PrintWriter(bw, true);

// Send message
try {
    bufferOut.println(MESSAGE_TO_SEND);
    bufferOut.flush();
}
catch (Exception e) {
    Log.e(TAG, "Error: " + e);
}
```

Code Snippet 11: Implementazione dell'invio di un messaggio

Ricezione di un messaggio

La ricezione delle risposte da parte del server avviene invece rimanendo in ascolto dei messaggi in ricezione.

```

try {

    // Receives the message which the server sends back
    InputStreamReader isr
        = new InputStreamReader(socket.getInputStream());

    BufferedReader bufferIn = new BufferedReader(isr);

    // In this while the client listens
    // for the messages sent by the server
    while (mRun) {

        serverMessage = bufferIn.readLine();

        Log.d(TAG, "Message received: " + serverMessage);

    }

} catch (Exception e) {
    Log.e(TAG, "Error: " + e);
}

```

Code Snippet 12: Implementazione della ricezione di un messaggio

Capitolo 4

Classificazione

Il cuore del progetto riguarda la classificazione delle attività mediante i dati ottenuti.

Ho optato per l'utilizzo di Keras di cui vediamo le fasi di *training* e di *prediction*.

Keras

Keras [11] è una libreria open-source per le reti neurali che astrae lo sviluppo rendendolo più comprensibile, pur mantenendo pieno supporto alle librerie di più basso livello (es. Tensorflow [12]) su cui si basa.

4.1 Apprendimento e Test

Un classificatore basa le sue predizioni sui modelli che riesce a ricavare dall'insieme di informazioni che ha a disposizione. Nel nostro caso l'insieme di queste informazioni è contenuto nei file con i valori generati e ricevuti dall'applicazione.

4.1.1 Caricamento dei dati

I file CSV presentati sono organizzati come nell'esempio visto in figura 2.1.

È necessario ricordare che i dati ottenuti da differenti sensori sono stati immagazzinati dal *ricevitore* in diversi file CSV. Nel caso in esame sono quindi presenti due file, uno per l'accelerometro ed un secondo per il giroscopio.

Entrambi contengono la stessa tipologia di informazioni e sono strutturati in modo equivalente. Per questo motivo nelle procedure seguenti considererò un singolo dataset, ricordando però di dover applicare tutti i passaggi indistintamente ad entrambi.

Lettura del file

L'intero dataset contiene un ampio numero di record, ognuno dei quali è composto dalle seguenti informazioni:

- un identificativo che raggruppa i valori ottenuti da una singola esecuzione
- un indice crescente che ordina i record di un archivio
- i valori acquisiti dal sensore
- l'istante temporale di acquisizione

- la posizione del dispositivo
- la relativa attività

Come è possibile notate i valori contenuti in un singolo record corrispondono a quelli inviati dall'applicazione durante la raccolta dei dati sensoriali di apprendimento, eccezion fatta per il tipo di sensore che è già stato usato per la divisione preliminare.

```
# Name the columns of the file
column_names_list = ['archive',
                     'index',
                     'x-axis',
                     'y-axis',
                     'z-axis',
                     'timestamp',
                     'phone-position',
                     'activity']

# Read the file
df = pd.read_csv(file_path, header=None, names=column_names_list)
```

Code Snippet 13: Creazione del dataframe a partire dal file CSV

4.1.2 Valutazione della posizione del dispositivo

Comunemente la posizione del dispositivo di raccolta dati viene spesso sottovalutata durante lo sviluppo di tecniche per il riconoscimento delle attività. Tuttavia si tratta di un aspetto molto importante per la valutazione dei dati [13].

Dopo una prima valutazione che prevedeva l'utilizzo di questo dato come una semplice caratteristica informativa, ho deciso di dare ad esso un'importanza maggiore.

Il maggior valore è derivato dalla decisione di partizionare i record presenti nel dataset in base alla relativa posizione. Tale scelta comporta la necessità di eseguire le procedure di *train* e *test* seguenti su tutte le diverse partizioni e conseguentemente la creazione di un modello indipendente per ognuna di esse.

Ai fini della trattazione proseguirò considerando solamente il gruppo di dati relativo ad una posizione tra quelle che ho personalmente impostato.

Ricordando che i dati sono già stati in precedenza partizionati in base al sensore di riferimento, è importante tener presente che tutte le seguenti procedure dovranno essere ripetute per ogni posizione e per ogni sensore.

Consideriamo quindi solo i dati *accelerometrici* e *"nella mano destra"*.

4.1.3 Visualizzazione grafica dei dati

Suddivisione grafica

Per ogni partizione di dati che si va a considerare è possibile ottenere una chiara visualizzazione grafica della suddivisione per attività dei dati presenti.

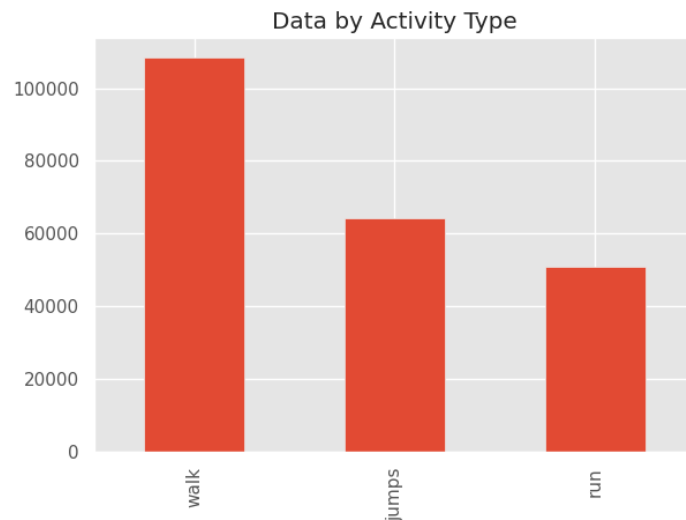


Figura 4.1: Visualizzazione della suddivisione per attività

Grafici delle attività

Inoltre, di ogni attività è possibile utilizzare una visualizzazione grafica per meglio comprendere le differenze tra le attività anziché basarsi solamente sulla lettura dei valori.

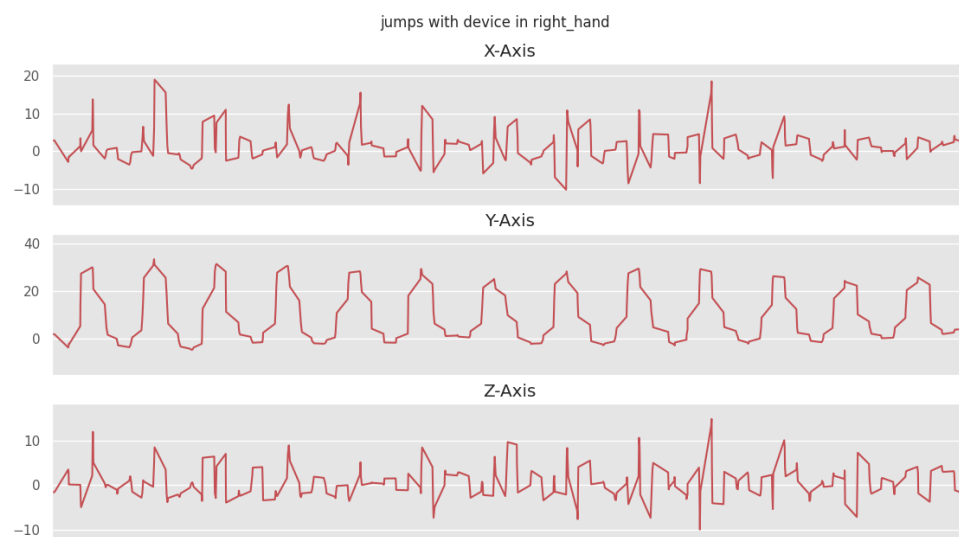


Figura 4.2: Dati accelerometrici durante l'attività "Salti" con il dispositivo nella mano destra



Figura 4.3: Dati accelerometrici durante l'attività "Corsa" con il dispositivo nella mano destra



Figura 4.4: Dati accelerometrici durante l'attività "Camminata" con il dispositivo nella mano destra

4.1.4 Apprendimento

La mole di dati considerata necessita un partizionamento per differenziare i dati che saranno utilizzati per l'*apprendimento* e quelli che saranno utilizzati per il *test*.

È indispensabile verificare che le partizioni non abbiano sovrapposizioni se non si vuole ottenere una valutazione dell'efficienza falsata.

Personalmente ho scelto una semplice suddivisione che prevede l'utilizzo di $\frac{4}{5}$ dei dati per il *train* e la parte restante ($\frac{1}{5}$) per il *test*.

4.1.5 Preparazione dei dati

Ci si aspetta che, fornita una serie di caratteristiche (i tre assi x, y, z ed il valore temporale, la rete neurale dia in risposta un'etichetta rappresentante l'attività associata.

Dobbiamo quindi organizzare i dati in nostro possesso in modo da renderlo possibile.

Trasformazione del valore temporale

Una delle quattro caratteristiche che intendiamo utilizzare è il valore temporale. I *timestamps* però rappresentano il tempo assoluto, ovvero il momento esatto di svolgimento dell'attività durante la raccolta dei dati.

Il momento esatto non è in alcun modo rilevante nella classificazione, ma a partire da questo è possibile ricavare il tempo trascorso tra l'acquisizione di una tripla di dati (x, y, z) e l'acquisizione di quella immediatamente successiva. Posso presumere una somiglianza in tali distanze temporali durante lo svolgimento di una uguale attività.

Normalizzazione dei dati

La rete neurale accetta in ingresso valori compresi tra 0 e 1. Eseguo una semplice normalizzazione sui dati delle caratteristiche.

```
# Normalize features for data set (values between 0 and 1)
df['x-axis'] = df['x-axis'] / df['x-axis'].max()
df['y-axis'] = df['y-axis'] / df['y-axis'].max()
df['z-axis'] = df['z-axis'] / df['z-axis'].max()
df['timestamp'] = df['timestamp'] / df['timestamp'].max()
```

Code Snippet 14: Banale normalizzazione dei dati

Creazione dei segmenti e delle etichette

La parte principale dell'intero adattamento risulta essere quella che suddivide i dati in un formato che possa realizzare l'associazione tra una serie di caratteristiche e una etichetta.

Per realizzare ciò i record sono presi a gruppi, anche sovrapposti (così come si vede in figura 4.5). Ogni raggruppamento sarà caratterizzato da

- un segmento contenente i record con le sole caratteristiche
- l'etichetta più frequente

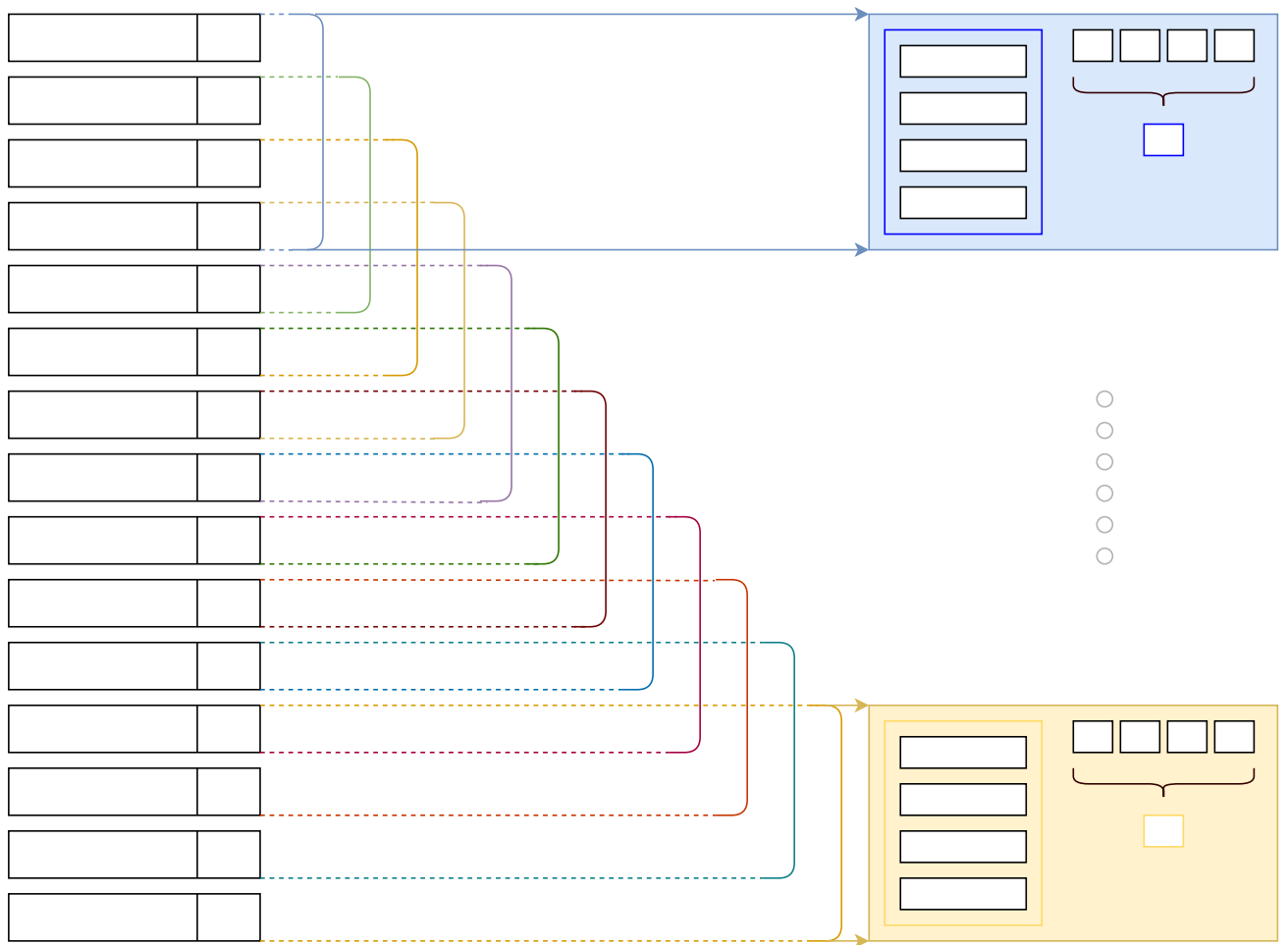


Figura 4.5: Creazione dei segmenti e delle etichette

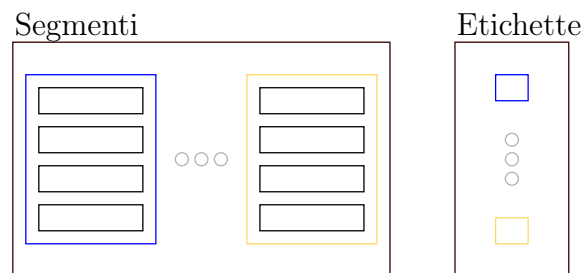


Figura 4.6: Risultato dopo la creazione dei segmenti e delle etichette

Alla fine del processo ho quindi ottenuto una serie di segmenti di dati a cui posso singolarmente associare una determinata etichetta identificativa.

4.1.6 Creazione della rete neurale

Una volta generati i dati nel formato supportato da *keras* procedo alla creazione di una rete neurale che abbia

- in input il formato dei dati appena generato

- 5 strati di 100 nodi connessi
- in output il calcolo di probabilità per ogni classe

```
# Create DNN
model_m = Sequential()
model_m.add(Dense(100, activation='relu')) # Layer 1
model_m.add(Dense(100, activation='relu')) # Layer 2
model_m.add(Dense(100, activation='relu')) # Layer 3
model_m.add(Dense(100, activation='relu')) # Layer 4
model_m.add(Dense(100, activation='relu')) # Layer 5
model_m.add(Flatten())
model_m.add(Dense(num_classes, activation='softmax'))
```

Code Snippet 15: Creazione della DNN

Per poi procedere all'apprendimento.

```
# Fit the model
model_m.fit(segments_train,
            labels_train,
            batch_size=BATCH_SIZE,
            epochs=EPOCH,
            validation_split=0.20,
            verbose=1)
```

Code Snippet 16: Apprendimento della rete neurale

4.1.7 Statistiche del modello

Al termine dell'attività di apprendimento il modello è stato generato. Ed è possibile vedere un grafico riassuntivo dei risultati ottenuti.

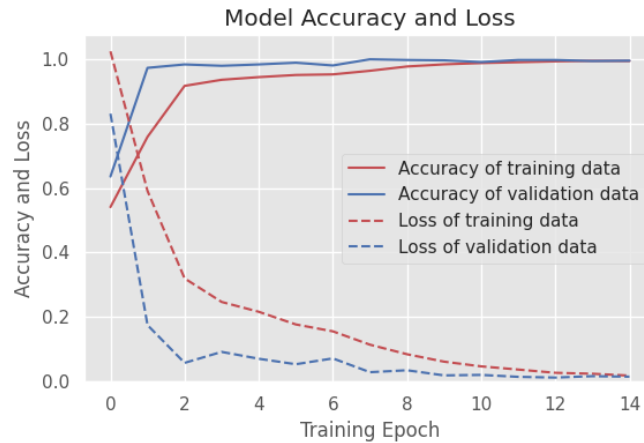


Figura 4.7: Statistiche del modello ottenuto

4.1.8 Testing della rete neurale

Il test della rete neurale creata avviene mediante l'uso della partizione di dati che avevamo precedentemente separato.

L'intento è quello di effettuare una predizione con i dati di test, di cui però si conoscono già i risultati. Sarà quindi immediato trovare l'efficienza del modello creato mediante un banale confronto tra i dati ipotizzati dalla rete neurale e i risultati corretti.

La qualità dei dati può essere visualizzata mediante una *matrice di confusione* che fornisce una rappresentazione grafica del confronto appena descritto.

Matrice di confusione

La matrice di confusione è una tabella di rappresentazione dell'accuratezza di un modello di classificazione. In una matrice di confusione sono contrapposti i valori ipotizzati da un modello di classificazione e i valori reali di cui si conosceva il corretto risultato.

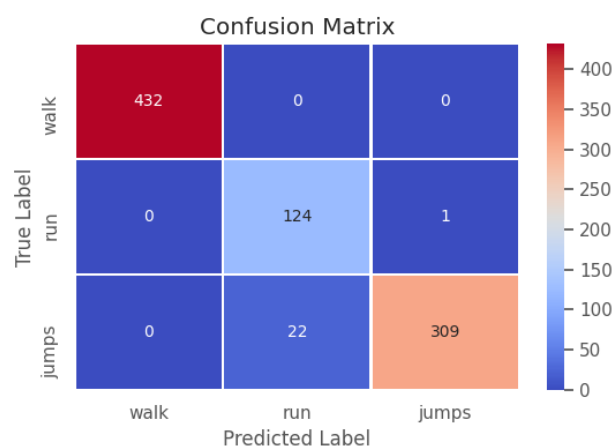


Figura 4.8: Matrice di confusione del modello ottenuto

4.2 Predizioni

In maniera del tutto equivalente a quanto già visto durante l'apprendimento, per effettuare una predizione sfruttando la rete neurale allenata forniremo in input una serie di segmenti e attenderemo che il classificatore ci fornisca in risposta una serie di etichette ipotizzate.

Capitolo 5

Conclusioni

Riferimenti

Bibliografia

- [1] A. Ferrari et al. «A Framework for Long-Term Data Collection to Support Automatic Human Activity Recognition». In: (2019). DOI: 10.3233/AISE190067 (cit. a p. 4).
- [13] E. Casilari, J. A. Santoyo-Ramón e J. M. Cano-García. «UMAFall: A Multisensor Dataset for the Research on Automatic Fall Detection». In: (2017). DOI: 10.1016/j.procs.2017.06.110 (cit. a p. 21).

Siti

- [2] *Docker*. URL: <https://github.com/docker> (cit. a p. 5).
- [3] *Flask*. URL: <https://github.com/pallets/flask/> (cit. a p. 7).
- [4] *Material Design*. URL: <https://material.io/> (cit. a p. 11).
- [5] *Android Services*. URL: <https://developer.android.com/guide/components/services> (cit. alle pp. 12, 13).
- [6] *TextToSpeech*. URL: <https://developer.android.com/reference/android/speech/tts/TextToSpeech?hl=en> (cit. a p. 15).
- [7] *CountDownTimer*. URL: <https://developer.android.com/reference/android/os/CountDownTimer> (cit. a p. 16).
- [8] *Retrofit*. URL: <https://square.github.io/retrofit/> (cit. a p. 16).
- [9] *SensorEventListener*. URL: <https://developer.android.com/reference/android/hardware/SensorEventListener?hl=en> (cit. a p. 17).
- [10] *Socket*. URL: <https://developer.android.com/reference/java/net/Socket> (cit. a p. 18).
- [11] *Keras*. URL: <https://github.com/keras-team/keras> (cit. a p. 20).
- [12] *Tensorflow*. URL: <https://github.com/tensorflow/tensorflow> (cit. a p. 20).