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.sa 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

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. Quest'ultimo aspetto è troppo spesso sottovalutato [2].

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.

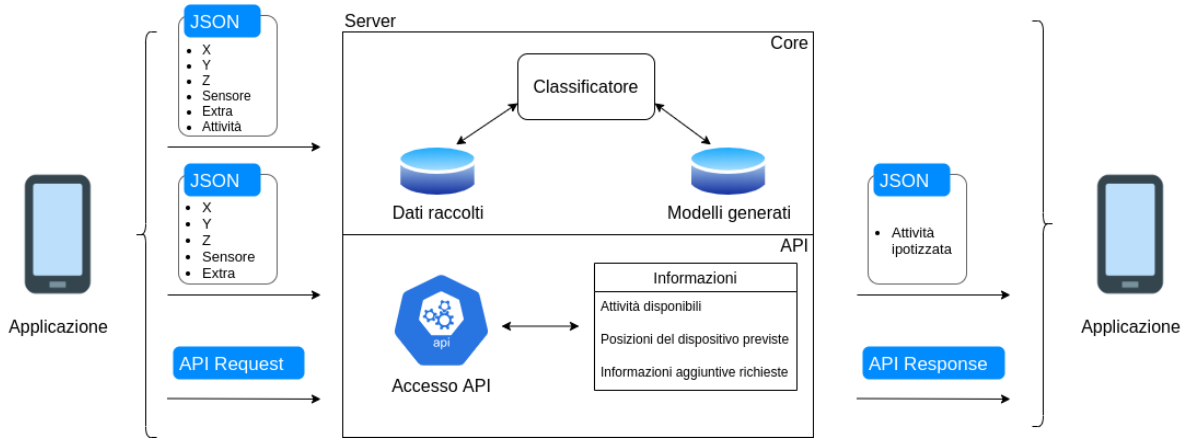

Figura 1.1: Panoramica del progetto

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 programmativo.

Docker

Docker [3] 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 rispondi 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
```

Listing 1: Endpoint per la lista delle attività

Lista delle posizioni del dispositivo

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

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

Listing 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
```

Listing 3: Endpoint per le informazioni sui dati aggiuntivi

2.1.3 Implementazione

L'implementazione del software è stata effettuata con l'utilizzo di Flask [4], un framework leggero per svilluppare 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)
```

Listing 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
        }
    }
}
```

Listing 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	Phone Position	Activity
8e9c147c-6fa0-466f-993b-f726a786933c	2	1.5047760009765625	4.826629638671875	7.0322265625	1592314676719	2	0
8e9c147c-6fa0-466f-993b-f726a786933c	3	1.76776123046875	5.1774139404296875	8.53094482421875	1592314676721	3	0
8e9c147c-6fa0-466f-993b-f726a786933c	4	1.7921142578125	5.622589111328125	9.804702758789062	1592314676724	4	0
8e9c147c-6fa0-466f-993b-f726a786933c	5	1.8303985595703125	5.4080047607421875	12.129501342773438	1592314676790	5	0
8e9c147c-6fa0-466f-993b-f726a786933c	7	3.29852294921875	3.7346649169921875	17.15325927734375	1592314676797	6	0
8e9c147c-6fa0-466f-993b-f726a786933c	8	3.4316558837890625	2.5201263427734375	16.86737060546875	1592314676799	7	0
8e9c147c-6fa0-466f-993b-f726a786933c	9	2.33697509765625	2.123687744140625	15.2672119140625	1592314676870	8	0
8e9c147c-6fa0-466f-993b-f726a786933c	10	1.0340576171875	2.21173095703125	12.205001831054688	1592314676873	9	0
8e9c147c-6fa0-466f-993b-f726a786933c	11	-0.002960205078125	2.9204254150390625	8.565078735351562	1592314676876	10	0
8e9c147c-6fa0-466f-993b-f726a786933c	12	-0.40704345703125	3.8975830078125	6.2041015625	1592314676879	11	0
8e9c147c-6fa0-466f-993b-f726a786933c	13	0.1982269287109375	4.698699951171875	5.07843017578125	1592314676950	12	0
8e9c147c-6fa0-466f-993b-f726a786933c	14	1.20355224609375	5.3319854736328125	5.075714111328125	1592314676956	13	0
8e9c147c-6fa0-466f-993b-f726a786933c	15	1.9846649169921875	5.592987060546875	6.3537445068359375	1592314676961	14	0

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"
}
```

Listing 6: Esempio di messaggio di risposta con l'ipotesi formulata

Capitolo 3

Applicazione

L'applicazione sviluppata svolge principalmente la funzione di raccoglimento di qualsiasi dato utile alla classificazione. È possibile racchiudere le funzionalità offerte in

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

L'applicazione offre la compatibilità con il sistema operativo *Android*, in particolar modo con tutte le versioni di Android che supportano la versione delle API 16 o superiore (ovvero Android 4.1, del 2012, o versioni successive). Nella pratica, al momento della stesura di questa relazione, ciò assicura il funzionamento dell'app sul 99,8% dei dispositivi Android.

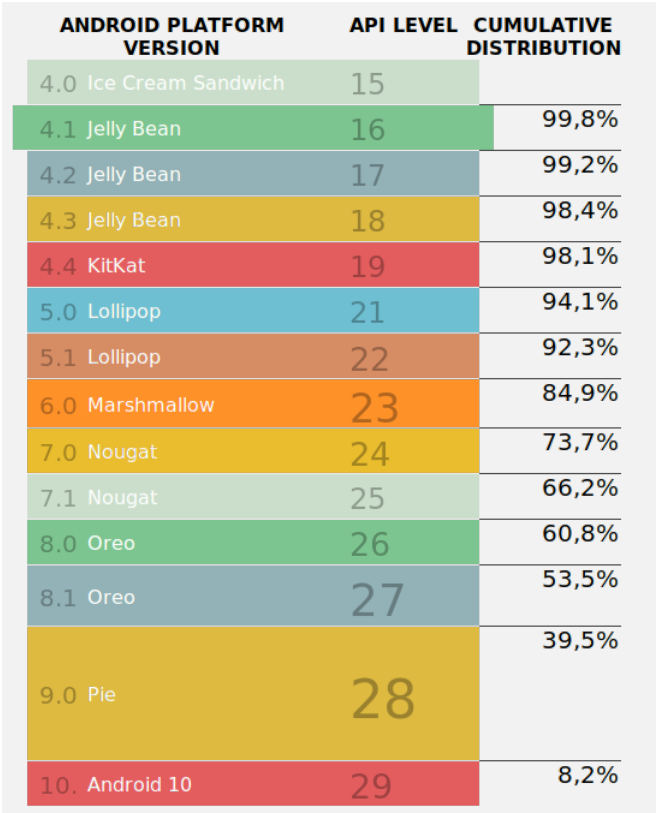

Figura 3.1: Distribuzione cumulativa delle versioni di Android

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

3.1 Interfaccia

Si è voluta dare un'interfaccia minimale sviluppata secondo le linee guida dell'ormai conosciuto Material Design [5], un linguaggio visivo che sintetizza i principi classici del buon design ispirandosi al mondo fisico.

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.

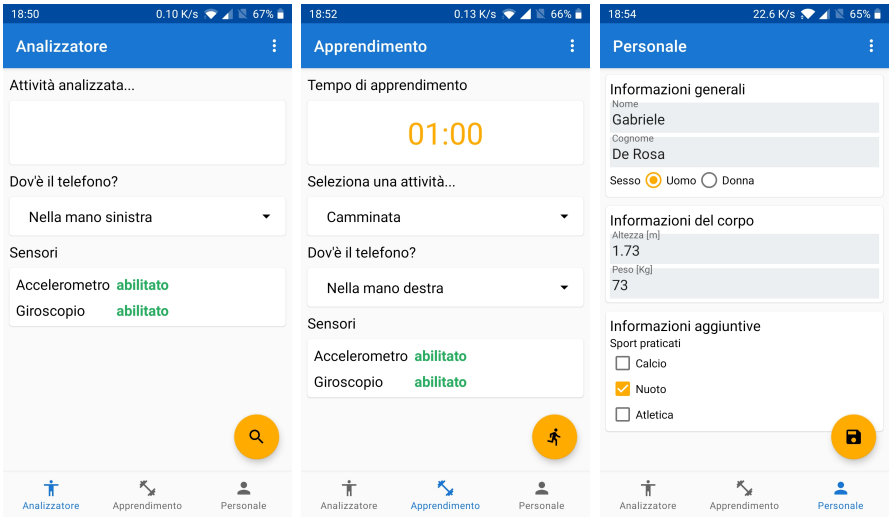

Figura 3.2: Le 3 sezioni principali dell'applicazione

3.1.1 Sezione di analisi

Nella sezione di analisi è possibile avviare un processo che sarà in grado di ipotizzare l'attività che si sta svolgendo. L'insieme delle azioni svolte in questa sezione sono divisibili in 4 fasi.

Fase 1: Inizializzazione

In questa prima fase l'applicazione scarica tramite l'utilizzo delle API (vedi sezione 2.1) le informazioni relative alle posizioni del dispositivo che sono disponibili e selezionabili. L'utente ha la possibilità di selezionare la posizione dove tenere il telefono durante l'attività ed avviare l'analisi.

Fase 2: Preparazione

In seguito all'avvio dell'analisi sarà avviato un servizio in foreground [6] dell'applicazione 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 service appena avviato si occuperà di cercare un contatto con il server per stabilire una connessione TCP.

Qualora la connessione avvenisse con successo sarà avviato un conto alla rovescia pensato appositamente in modo da consentire all'utente di prepararsi posizionando il dispositivo nella posizione selezionata precedentemente.

Fase 3: Analisi

Allo scadere del countdown riservato alla preparazione è avviata l'analisi vera e propria ed inizia anche lo scambio dati con il server.

I sensori abilitati iniziano la raccolta dei dati ed ogni informazione ottenuta viene inviata.

Fase 4: Predizione

Durante l'analisi il server risponde con diversi messaggi. Tra i principali troviamo il messaggio di conferma di ricezione di un dato, ma soprattutto è rilevante il messaggio contenente l'attività che è stata ipotizzata.

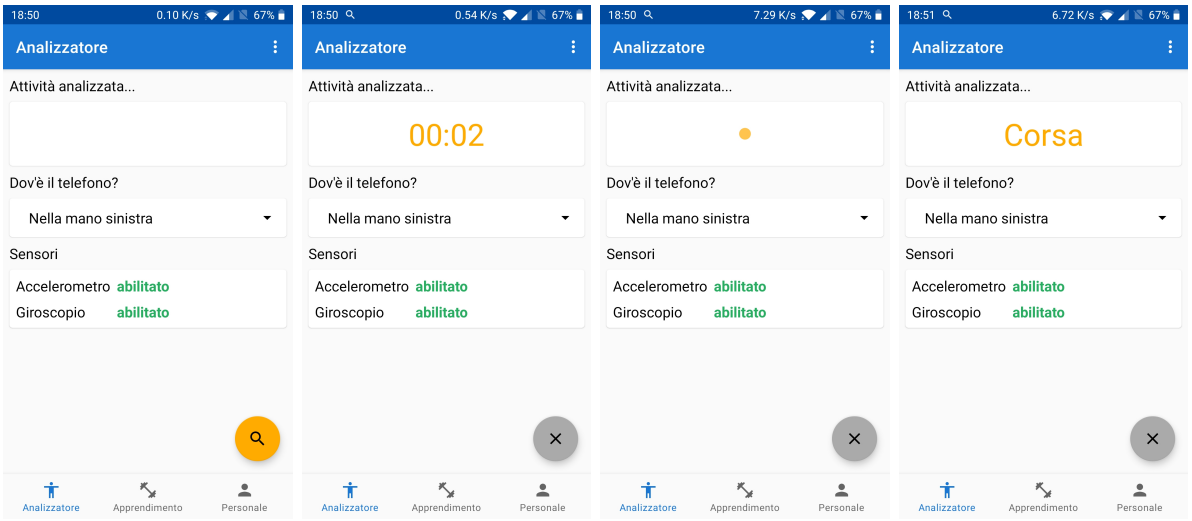

Figura 3.3: Le 4 fasi dell'analisi

3.1.2 Sezione di apprendimento

Nella sezione di apprendimento è possibile avviare un processo in grado di raccogliere dei dati sensoriali associandoli ad una determinata attività.

L'obiettivo sarà poi utilizzare questi dati per effettuare il *train* della dati per la creazione di un modello. In questa fase si da 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 tramite l'utilizzo delle API (vedi sezione 2.1) diverse informazioni:

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

L'utente dovrà quindi selezionare la posizione dove tenere il telefono durante l'attività e l'attività che andrà a svolgere prima di avviare l'apprendimento.

Fase 2: Preparazione

In seguito all'avvio dell'apprendimento sarà avviato un servizio in foreground [6] dell'applicazione 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 service appena avviato si occuperà di cercare un contatto con il server per stabilire una connessione TCP.

Qualora la connessione avvenisse con successo sarà avviato un conto alla rovescia pensato appositamente in modo da consentire all'utente di prepararsi posizionando il dispositivo nella posizione selezionata precedentemente.

Fase 3: Apprendimento

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

I dati raccolti dai sensori sono inviati costantemente al server. Una volta scaduto il conto alla rovescia la raccolta dei dati sarà terminata.

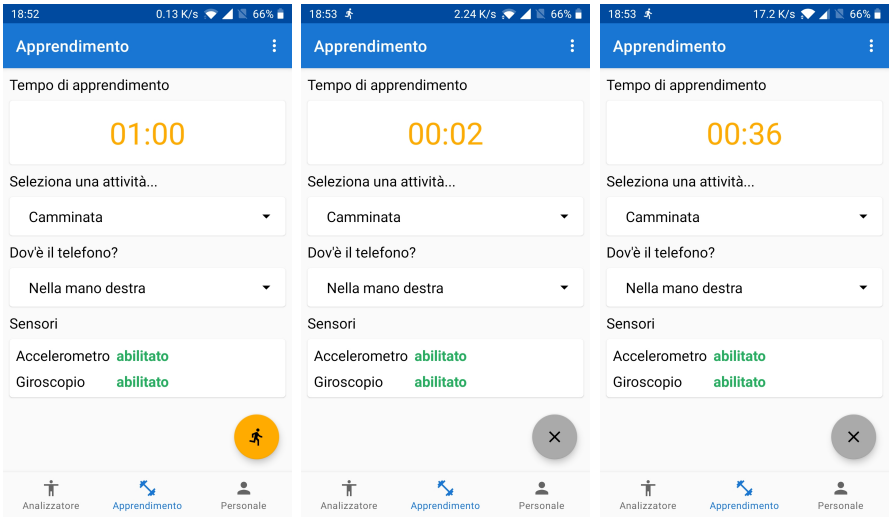

Figura 3.4: Le 3 fasi dell'apprendimento

3.1.3 Sezione per l'inserimento di dati aggiuntivi

Nella sezione di apprendimento è presente un modulo di inserimento dati utile per permettere all'utilizzatore di inserire dati aggiuntivi relativi alla sua persona. Potenzialmente si potrebbe pensare che questi dati saranno poi utilizzabili per una classificazione più approfondita.

Un aspetto particolare di questo modulo consiste nella sua dinamicità. Tutte la sua intera struttura è generata programmativamente tramite le informazioni ottenute in fase di creazione tramite una chiamata alle API (vedi sezione delle 2.1).

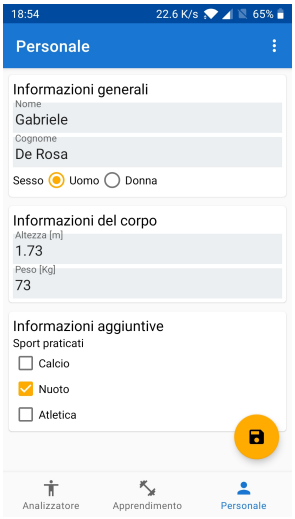

Figura 3.5: Il modulo di inserimento dati generato programmativamente

3.1.4 Feedback sonoro

Oltre a quanto si vede sull'interfaccia dell'applicazione, essa fornisce un feedback sonoro mediante il sintetizzatore vocale del dispositivo che fornisce tutte le informazioni visualizzate sulla UI. Questa necessità dipende dal fatto che l'utilizzo potrebbe avvenire dopo aver posizionato il dispositivo in modo da impedirne la visualizzazione delle informazioni (ad esempio in tasca).

3.2 Implementazione

Gli aspetti programmativi che più interessano sono indubbiamente quelli legati all'accesso alle API, alla gestione dei conti alla rovescia, alle comunicazioni con il server, al feedback sonoro e alla raccolta dei dati con i principali sensori di movimento.

3.2.1 Accesso alle API

Le chiamate alle API sono fatte utilizzando la librerie Retrofit [7].

3.2.2 Countdown Timers

Per l'implementazione dei conti alla rovescia ho implementato la classe *CountDownTimer* [8] offerta direttamente da Android.

3.2.3 Connessione con il server

Per la connessione al server è stata implementata con l'utilizzo della classe *Socket* [9].

3.2.4 Feedback sonoro

La sintetizzazione vocale è stata implementata con le librerie di *text to speech* [10] offerte da Android.

3.2.5 Sensori di movimento

I sensori di movimento implementati sono accelerometro e giroscopio.

Capitolo 4

Classificazione

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

Ho optato per l'utilizzo di Keras [11]. Si tratta di 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 Caricamento dei dati

Il passaggio che precede l'apprendimento è il recupero dei dati collezionati nei file CSV.

Archive	Index	X Axis	Y Axis	Z Axis	Timestamp	Phone Position	Activity
8e9c147c-6fa0-466f-993b-f726a786933c	2	1.5047760009765625	4.826629638671875	7.0322265625	1592314676719	2	0
8e9c147c-6fa0-466f-993b-f726a786933c	3	1.76776123046875	5.1774139404296875	8.53094482421875	1592314676721	3	0
8e9c147c-6fa0-466f-993b-f726a786933c	4	1.7921142578125	5.622589111328125	9.804702758789062	1592314676724	4	0
8e9c147c-6fa0-466f-993b-f726a786933c	5	1.8303985595703125	5.4080047607421875	12.129501342773438	1592314676790	5	0
8e9c147c-6fa0-466f-993b-f726a786933c	7	3.29852294921875	3.7346649169921875	17.15325927734375	1592314676797	6	0
8e9c147c-6fa0-466f-993b-f726a786933c	8	3.4316558837890625	2.5201263427734375	16.86737060546875	1592314676799	7	0
8e9c147c-6fa0-466f-993b-f726a786933c	9	2.33697509765625	2.123687744140625	15.2672119140625	1592314676870	8	0
8e9c147c-6fa0-466f-993b-f726a786933c	10	1.0340576171875	2.21173095703125	12.205001831054688	1592314676873	9	0
8e9c147c-6fa0-466f-993b-f726a786933c	11	-0.002960205078125	2.9204254150390625	8.565078735351562	1592314676876	10	0
8e9c147c-6fa0-466f-993b-f726a786933c	12	-0.40704345703125	3.8975830078125	6.2041015625	1592314676879	11	0
8e9c147c-6fa0-466f-993b-f726a786933c	13	0.1982269287109375	4.698699951171875	5.07843017578125	1592314676950	12	0
8e9c147c-6fa0-466f-993b-f726a786933c	14	1.20355224609375	5.3319854736328125	5.075714111328125	1592314676956	13	0
8e9c147c-6fa0-466f-993b-f726a786933c	15	1.9846649169921875	5.592987060546875	6.3537445068359375	1592314676961	14	0

Figura 4.1: Esempio di dati contenuti nel file CSV

Dopo la sola lettura è già possibile ottenere una chiara visualizzazione grafica della suddivisione per attività dei dati presenti.

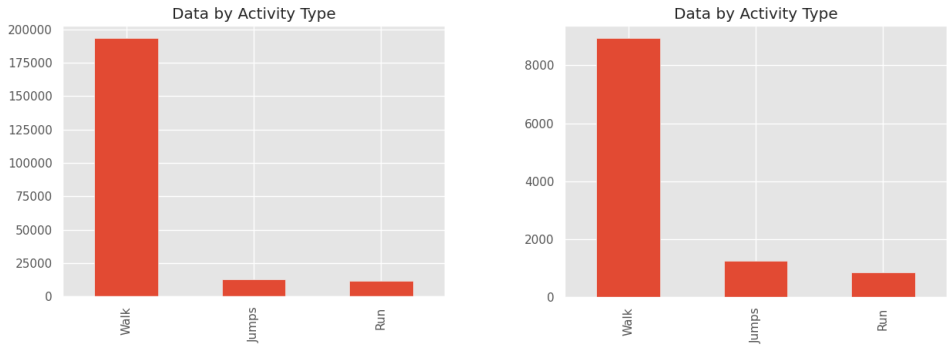

Figura 4.2: Visualizzazione della suddivisione dei dati per i due sensori

I dataset sono relativi ai sensori attivi sull'applicazione, ovvero accelerometro e giroscopio. Entrambi contengono la stessa tipologia di informazioni (i valori dei tre assi, il timestamp e il posizionamento del dispositivo). Nelle procedure seguenti considererò quindi il singolo dataset, ricordando però di dover applicare i passaggi indistintamente ad entrambi.

4.2 Apprendimento e Test

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

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*.

È indispensabile non sovrapporre queste frazioni se non si vuole ottenere una valutazione dell'efficienza falsata.

4.2.1 Preparazione dei dati

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

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

Trasformazione del valore temporale

Tra le caratteristiche abbiamo i *timestamp*s che 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()
df['phone-position'] = df['phone-position'] / number_of_phone_positions
```

Listing 7: 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 4.3). Ogni raggruppamento sarà caratterizzato da

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

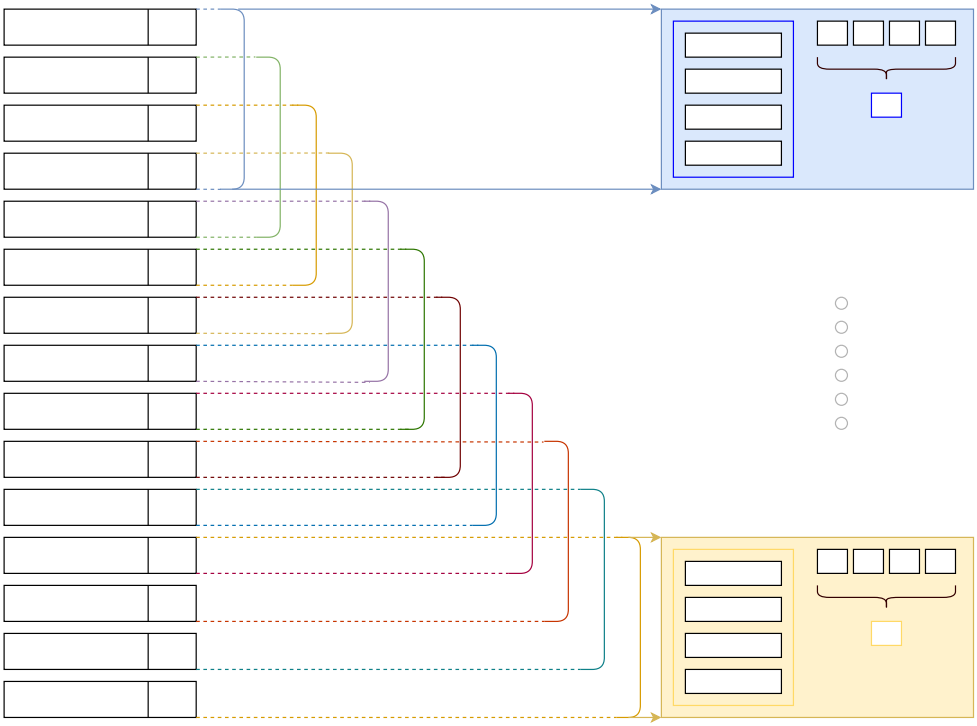

Figura 4.3: Creazione dei segmenti e delle etichette

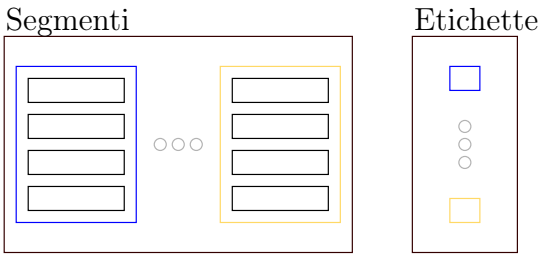

Figura 4.4: Risultato dopo la creazione dei segmenti e delle etichette

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

4.2.2 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 propabilità 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'))
```

Listing 8: 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)
```

Listing 9: Apprendimento della rete neurale

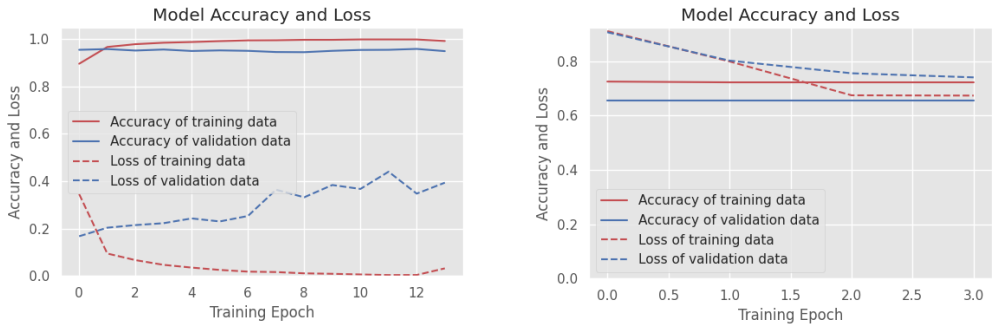

Figura 4.5: Statistiche del modello ottenuto

4.2.3 Testing della rete neurale

Il test della rete neurale creata avviene mediante l'uso dei dati che avevamo precedentemente separato dal dataset iniziale.

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.

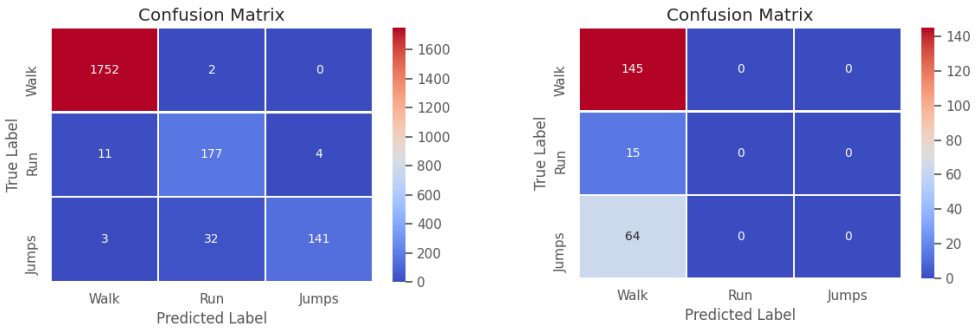

Figura 4.6: Statistiche del modello ottenuto

4.3 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).

[2] 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. 4).

Siti

[3] *Docker*. URL: https://github.com/docker (cit. a p. 5).

[4] *Flask*. URL: https://github.com/pallets/flask/ (cit. a p. 7).

[5] *Material Design*. URL: https://material.io/ (cit. a p. 11).

[6] *Android Services*. URL: https://developer.android.com/guide/components/services (cit. alle pp. 11, 13).

[7] *Retrofit*. URL: https://square.github.io/retrofit/ (cit. a p. 14).

[8] *CountDownTimer*. URL: https://developer.android.com/reference/android/os/CountDownTimer (cit. a p. 14).

[9] *Socket*. URL: https://developer.android.com/reference/java/net/Socket (cit. a p. 14).

[10] *TextToSpeech*. URL: https://developer.android.com/reference/android/speech/tts/TextToSpeech?hl=en (cit. a p. 14).

[11] *Keras*. URL: https://github.com/keras-team/keras (cit. a p. 16).

[12] *Tensorflow*. URL: https://github.com/tensorflow/tensorflow (cit. a p. 16).