



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	Dati inerziali	3
1.2	Classificazione	4
1.2.1	L'importanza dei dati	4
1.3	Obiettivo e panoramica del progetto	4
2	Server	6
2.1	RESTful Web API	6
2.1.1	Protocollo e formato	6
2.1.2	Endpoints	7
2.1.3	Implementazione	11
2.2	Ricevitore	11
2.2.1	Protocollo e formato	11
2.2.2	Messaggi	12
2.2.3	Azioni	13
3	Applicazione	14
3.1	Interfaccia	15
3.1.1	Sezione di analisi	15
3.1.2	Sezione di apprendimento	16
3.1.3	Sezione per l'inserimento di dati aggiuntivi	18
3.2	Feedback sonoro	20
3.3	Countdown Timers	21
3.4	Accesso alle API	21
3.5	Sensori di movimento	22
3.5.1	Attivazione dei sensori	22
3.5.2	Estrazione dei dati inerziali	22
3.6	Comunicazioni con il server	24
4	Classificazione	26
4.1	Apprendimento e Test	26
4.1.1	Caricamento dei dati	26
4.1.2	Valutazione della posizione del dispositivo	27
4.1.3	Visualizzazione grafica dei dati	28
4.1.4	Partizionamento dei dati	29
4.1.5	Preparazione dei dati	30
4.1.6	Creazione della rete neurale	32
4.1.7	Statistiche del modello	32

4.1.8	Salvataggio del modello	33
4.1.9	Testing della rete neurale	33
4.2	Predizioni	34
4.2.1	Scelta del modello	34
4.2.2	Preparazione dei dati	34
4.2.3	Ipotesi dell'etichetta	36
5	Conclusioni	37
	Riferimenti	38
	Bibliografia	38
	Siti	38

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.

Esistono diverse tecniche di acquisizione usate per il riconoscimento delle attività. Alcuni dati utili possono essere frame video e immagini oppure i dati inerziali ottenuti da sensori di movimento [1].

In questa trattazione ho scelto l'uso dei dati inerziali per il riconoscimento delle attività. Gli smartphone e gli indossabili (ad esempio braccialetti ed orologi intelligenti) sono sempre più comuni e perciò costituiscono una fonte di informazioni potenzialmente infinita.

1.1 Dati inerziali

Le informazioni relative al movimento di un corpo nello spazio tridimensionale sono raccolte dai cosiddetti **sensori inerziali**, principalmente accelerometro e giroscopio.

Accelerometro

L'accelerometro è un sensore capace di misurare l'accelerazione gravitazionale su un oggetto. Un tempo destinato ad usi scientifici e militari, è diventato di uso comune con l'avanzare della tecnologia.

La totalità degli smartphone e degli indossabili intelligenti sono oggi dotati di un *accelerometro a 3 assi* in grado di misurare l'accelerazione applicata su ognuno dei 3 assi dello spazio. Da questo deriva la possibilità di conoscere l'orientamento del dispositivo e di conseguenza i movimenti.

Giroscopio

Il giroscopio è un sensore capace di ricavare l'orientamento dell'oggetto sulla base di alcune proprietà fisiche che lo contraddistinguono.

Una buona parte di smartphone ed indossabili intelligenti sono dotati anche di un *giroscopio a 3 assi* che coopera con l'accelerometro nella raccolta di dati inerziali.

1.2 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 possa applicare in autonomia le stesse leggi anche a dati futuri.

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

1.2.1 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 [2].

1.3 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.

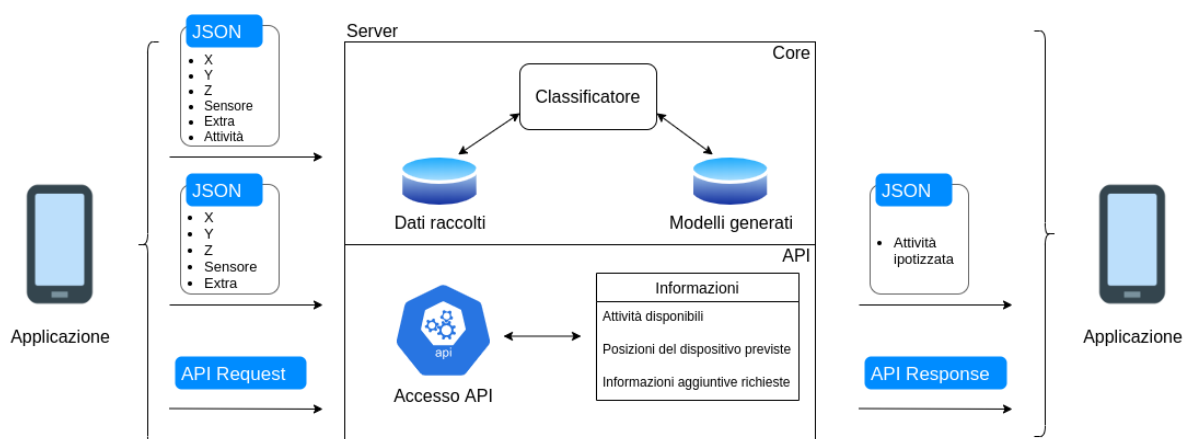


Figura 1.1: Panoramica del progetto

Applicazione

L'applicazione permette, mediante l'utilizzo dei principali sensori inerziali del dispositivo, la raccolta dei dati che saranno poi elaborati remotamente. Si occupa inoltre di fornire all'utente un riscontro dell'attività ipotizzata in fase di analisi.

Ricevitore

Chiameremo *ricevitore* il componente che si occupa di gestire ed immagazzinare i dati ricevuti dall'applicazione. Tra i suoi compiti anche quello di fornire eventuali risposte.

Classificatore

Il classificatore gestisce la mole di valori ottenuta dall'applicazione tentando di eseguire il riconoscimento vero e proprio delle attività per poi restituire l'ipotesi formulata.

Rest Web API

Sono state aggiunte delle RESTful Web API 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 tutti 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 [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 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 un'interfaccia composta da un insieme di *endpoints* pubblici che consentono di ottenere delle informazioni. Tutti i componenti interessati a tali informazioni sono in grado di ottenerle mediante un sistema di richiesta-risposta tra client e server.

Si ha accesso ad informazioni sempre aggiornate. Un cambiamento dei valori in fase di amministrazione consente la distribuzione delle modifiche senza la necessità di rilasciare un aggiornamento dell'applicazione o riprogrammare il ricevitore.

2.1.1 Protocollo e formato

Tutte le richieste e le risposte utilizzano 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, uno 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 uno specifico URL che ci si aspetta risponda con i dati richiesti.

Gli endpoints che ho previsto sono riservati ad ottenere informazioni su

- la lista delle attività classificabili
- la lista delle posizioni previste
- un modello per la richiesta di informazioni aggiuntive

```
http://IP_ADDRESS:PORT/activities  
http://IP_ADDRESS:PORT/positions  
http://IP_ADDRESS:PORT/form
```

Code Snippet 1: Elenco degli endpoints disponibili

Lista delle attività

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

Come è mostrato nel ritaglio di codice 2, la risposta di una chiamata a questo endpoint fornisce una lista di tutte le attività classificabili, ognuna contenente

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

```

{
  "status": "success",
  "activities": [
    {
      "id": 0,
      "activity": "walk",
      "translations": {"en": "Walk", "it": "Camminata"},
      "time": 60,
      "sensors": [
        {"sensor": "accelerometer", "enabled": true},
        {"sensor": "gyroscope", "enabled": true}
      ]
    },
    {
      "id": 1,
      "activity": "run",
      "translations": {"en": "Run", "it": "Corsa"},
      "time": 60,
      "sensors": [
        {"sensor": "accelerometer", "enabled": true},
        {"sensor": "gyroscope", "enabled": true}
      ]
    },
    {
      "id": 2,
      "activity": "jumps",
      "translations": {"en": "Jumps", "it": "Salti"},
      "time": 60,
      "sensors": [
        {"sensor": "accelerometer", "enabled": true},
        {"sensor": "gyroscope", "enabled": true}
      ]
    }
  ]
}

```

Code Snippet 2: Esempio di risposta dell'endpoint delle attività

Lista delle posizioni del dispositivo

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

```
{
  "status": "success",
  "positions": [
    {
      "id": 0,
      "position": "left_hand",
      "translations": {
        "en": "In left hand",
        "it": "Nella mano sinistra"
      }
    },
    {
      "id": 1,
      "position": "right_hand",
      "translations": {
        "en": "In right hand",
        "it": "Nella mano destra"
      }
    },
    {
      "id": 2,
      "position": "front_left_pocket",
      "translations": {
        "en": "In the front left pocket",
        "it": "Nella tasca anteriore sinistra"
      }
    },
    {
      "id": 3,
      "position": "front_right_pocket",
      "translations": {
        "en": "In the front right pocket",
        "it": "Nella tasca anteriore destra"
      }
    }
  ]
}
```

Code Snippet 3: Esempio di risposta dell'endpoint delle posizioni

Modello per la richiesta di informazioni aggiuntive

Il terzo endpoint fornisce una struttura per la generazione sull'applicazione di un modulo per la richiesta di dati aggiuntivi. Ne discuteremo meglio nel capitolo 3.

```
{
  "status": "success",
  "groups": [ {
    "elements": [
      {
        "id": "name",
        "type": "input-text",
        "uploadable": false,
        "text": "Name",
        "translations": {
          "en": "Name",
          "it": "Nome"
        }
      }
    ]
  },
  {
    "elements": [ {
      "id": "stature",
      "type": "input-text",
      "uploadable": true,
      "text": "Stature",
      "translations": {
        "en": "Stature [m]", "it": "Altezza [m]"
      },
      "options": null
    }, {
      "id": "weight",
      "type": "input-text",
      "uploadable": true,
      "text": "Weight",
      "translations": {
        "en": "Weight [Kg]", "it": "Peso [Kg]"
      }
    }
  ]
}
}
```

Code Snippet 4: Esempio di risposta dell'endpoint sui dati aggiuntivi

2.1.3 Implementazione

L'implementazione del software è stata effettuata con l'utilizzo di Flask. Si occupa di esporre i file JSON contenenti le informazioni appena viste nei rispettivi endpoints.

Flask

Flask [4] è uno dei più famosi framework per lo sviluppo di applicazioni web con Python. Le sue caratteristiche principali sono leggerezza e semplicità, pur permettendo anche implementazioni più avanzate.

```
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 5: Flask App per una RESTful Web API con 3 endpoints

2.2 Ricevitore

Sempre da figura 1.1 è possibile intuire che 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 a gestire la ricezione dei dati e fornire le risposte. 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

Avendo la necessità di trasmettere i dati con il minimo ritardo, la connessione al ricevitore avviene via socket mediante TCP. Essendo un protocollo di rete a livello di trasporto ci consente di avere una velocità maggiore rispetto a HTTP.

Per comodità si continua ad utilizzare il formato JSON anche in questo frangente per organizzare le informazioni durante lo scambio dati.

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 il risultato di una classificazione. Malgrado ciò la lettura dei messaggi ricevuti resta la medesima ed il valore per identificare la tipologia di richiesta è disponibile internamente alle informazioni.

Come mostrato nell'esempio 6, oltre alle informazioni sul tipo di richiesta, un singolo messaggio contiene

- un codice identificante un gruppo di messaggi
- un indice indicante la progressione dei dati
- i valori dei 3 assi (x, y, z)
- il sensore di movimento che ha generato i valori ottenuti
- un 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 6: Esempio di messaggio ricevuto per l'apprendimento

2.2.3 Azioni

Le azioni da intraprendere per le diverse tipologie di richiesta sono differenti.

Messaggi di Apprendimento

Nel caso della ricezione di dati per l'apprendimento è necessario procedere al salvataggio degli stessi.

I record vengono suddivisi sulla base del relativo sensore. Per ogni sensore si genera un file CSV contenente i record con tutte le informazioni rimanenti.

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	20.509.910.590.078.100	77.220.234.051.171.800	1592314677117	right_hand	walk

Figura 2.1: Esempio del dataset CSV contenente dati accelerometrici

Ogni volta che la base di dati subisce delle modifiche si avvia il processo di *train* del classificatore, trattato nel capitolo 4.

Messaggi di Analisi

Nel caso di ricezione di dati per l'analisi non si deve immagazzinare le informazioni ricevute. L'obiettivo di questa fase è quello di fornire in risposta al client l'informazione che si aspetta.

Alla ricezione di un numero minimo di record si avvia il processo di classificazione, trattato sempre nel capitolo 4. L'ipotesi ottenuta dal classificatore sarà inviata come risposta.

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

Code Snippet 7: Esempio del messaggio di risposta con l'ipotesi formulata

Capitolo 3

Applicazione

Un ulteriore aspetto chiave del progetto è l'applicazione Android. Si tratta di un livello di intermezzo tra l'utente e il classificatore.

Attraverso l'uso dei sensori inerziali del dispositivo è possibile richiedere al sistema di ipotizzare l'attività che si sta svolgendo oppure addestrare lo stesso sistema con una precisa attività tra quelle selezionabili.

Compatibilità

Si offre una compatibilità con tutte le versioni di Android che supportano almeno la versione delle API 16, nella pratica tutte le versioni maggiori o uguali ad Android 4.1 rilasciato nel 2012.

Al momento della stesura di questa relazione corrisponde ad assicurare il funzionamento dell'app sul 99,8% dei dispositivi con questo sistema operativo.

Linguaggi di sviluppo

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

Traduzioni

L'intero applicativo è dotato di supporto multilingua: italiano e inglese.

3.1 Interfaccia

L'applicazione presenta un'interfaccia minimale sviluppata secondo le linee guida dell'ormai noto Material Design.

Material Design

Con Material Design [5] 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 di "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

La suddivisione delle sezioni dell'applicazione segue perfettamente le funzionalità da essa offerte: l'analisi dell'attività, l'apprendimento di una attività e l'inserimento di informazioni aggiuntive.

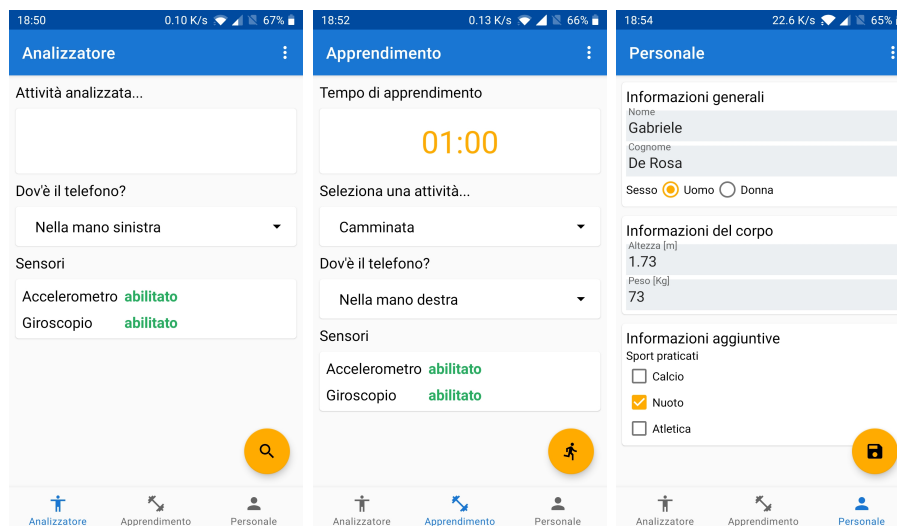


Figura 3.1: Le 3 sezioni principali dell'applicazione

3.1.1 Sezione di analisi

La sezione di analisi consente di testare l'efficacia del classificatore. Si vuole ottenere in risposta un riscontro sull'attività che si sta svolgendo.

La raccolta dei dati inerziali procede con l'invio delle informazioni al server ricevitore che, come abbiamo visto nella sezione 2.2, dopo l'elaborazione restituirà in risposta l'ipotesi.

Fase 1: Inizializzazione

In una prima fase l'applicazione richiede in autonomia alle API le informazioni relative alle posizioni del dispositivo che sono disponibili.

Dopo aver ottenuto una risposta simile a quella mostrata nel ritaglio di codice 3 l'utente ha la possibilità di selezionare la posizione, tra quelle scaricate, in cui desidera

tenere il dispositivo durante l'esecuzione dell'attività. In seguito a questa scelta l'analisi può essere avviata.

Fase 2: Preparazione

All'avvio dell'analisi sarà inizializzato un servizio in foreground [6] 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. Questo *tempo di preparazione* è pensato appositamente in modo da consentire all'utente di prepararsi posizionando il dispositivo nella posizione selezionata. Nella UI è mostrato un countdown.

Fase 3: Analisi

Allo scadere del conto alla rovescia i sensori sono abilitati. Ogni informazione raccolta è inserita in un messaggio simile a quello visto nell'esempio 6 e subito inviata.

Fase 4: Predizione

Durante la fase di analisi, e ininterrottamente fino allo stop dell'utente, il processo resta in ascolto in attesa di messaggi di risposta dal server. La maggioranza delle risposte sono messaggi di conferma di avvenuta ricezione delle informazioni e messaggi contenenti le ipotesi stipulate dal classificatore sull'attività in corso di esecuzione. In questa seconda eventualità i risultati vengono mostrati all'utente.

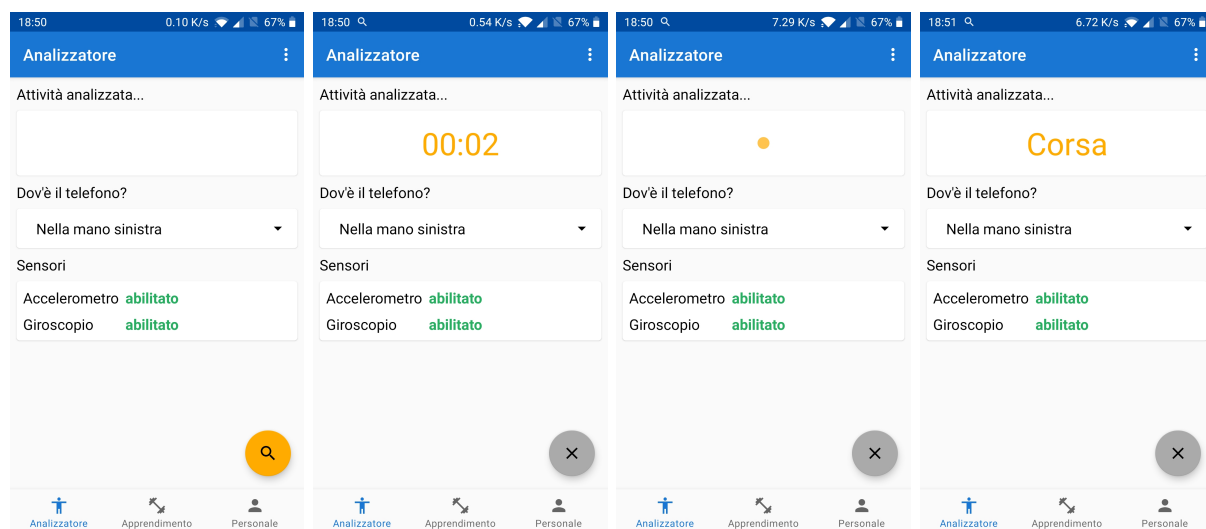


Figura 3.2: Le 4 fasi dell'analisi

3.1.2 Sezione di apprendimento

La sezione di apprendimento consente di incrementare l'insieme di informazioni a disposizione del classificatore. Si vogliono raccogliere dati inerziali di precise attività su cui,

come vedremo nel capitolo 4, la rete neurale si baserà per la generazione di modelli sempre più accurati.

Il processo è simile a quanto appena visto. La differenza consiste nel fatto che alle informazioni ottenute dai sensori è aggiunta l'attività selezionata manualmente dall'utente.

In questa fase si dà piena fiducia all'utilizzatore sulla correttezza dei dati inseriti.

Fase 1: Inizializzazione

In una prima fase l'applicazione richiede in autonomia alle API le informazioni di cui necessita. Si cercano

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

È immediato ottenere tali informazioni mediante richieste ai due endpoint relativi, i cui esempi di risposta sono mostrati nei codici 2 e 3.

Una volta avuta le risposte l'utente ha la possibilità di selezionare

- la posizione in cui desidera tenere il dispositivo durante l'esecuzione dell'attività.
- l'attività che ha intenzione di addestrare

In seguito a queste scelte l'apprendimento può essere avviato.

Fase 2: Preparazione

Anche in questo frangente sarà inizializzato un servizio in foreground [6] 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 utile come *tempo di preparazione*.

Fase 3: Apprendimento

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

Il secondo conto alla rovescia indica il *tempo di esecuzione* necessario. I secondi previsti potrebbero differire tra le diverse attività, in base alle informazioni ottenute nella prima fase.

Fino allo scadere del tempo tutti i dati raccolti dai sensori sono inviati al *ricevitore* di cui abbiamo precedentemente discusso.

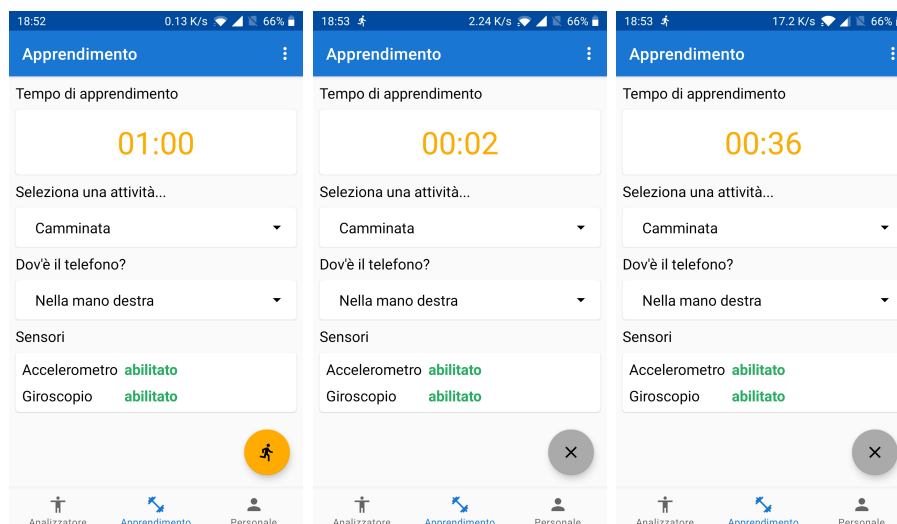


Figura 3.3: Le 3 fasi dell'apprendimento

3.1.3 Sezione per l'inserimento di dati aggiuntivi

La sezione per l'inserimento di dati aggiuntivi permette l'aggiunta di ulteriori informazioni che potrebbero tornare utili durante il riconoscimento dell'attività. Un esempio pratico potrebbe essere le informazioni riguardanti i dati fisici dell'utente (altezza, peso, etc.) qualora si ritenesse rilevante il loro valore per la classificazione.

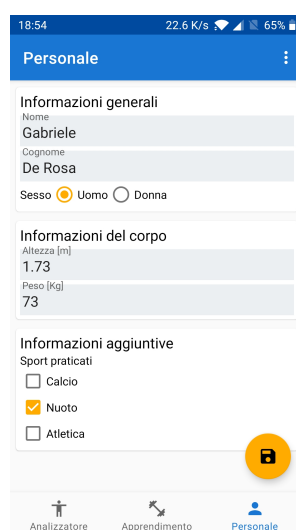


Figura 3.4: Il modulo di inserimento dati generato programmaticamente

Generazione del layout

Ritengo opportuno soffermarmi maggiormente sulla generazione del layout di questa sezione.

Malgrado agli occhi dell'utilizzatore appaia come un semplice modulo di inserimento dati, la generazione di ogni componente dello stesso avviene in modo dinamico. L'intero modulo è generato programmaticamente a partire dalle informazioni ricevute in risposta da una richiesta al relativo endpoint delle API.

Questa opportunità permette ad un amministratore di sistema di variare le informazioni richieste senza dover rilasciare un'aggiornamento dell'applicazione.

Per facilitarne la comprensione è possibile vederne un'applicazione in un esempio. La risposta di esempio mostrata nel codice 4 è in grado di generare il layout di figura 3.5.

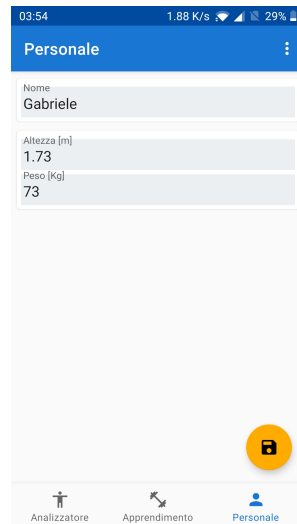


Figura 3.5: Esempio di modulo generato dalle informazioni del codice 4

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* [7] 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 8: 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* [8] offerta direttamente da Android.

```
// 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 9: Implementazione di un conto alla rovescia

Entrambi i timer implementati (*timer di preparazione* e *timer di apprendimento*) non hanno un tempo stabilito in partenza.

I secondi utili per la preparazione sono settabili dall'utente tra le impostazioni dell'applicazione, invece il tempo di apprendimento è invece ottenuto tramite la relativa richiesta alle API e può dipendere dall'attività selezionata.

3.4 Accesso alle API

Le richieste agli endpoint delle API, di cui abbiamo discusso le varie applicazioni per l'ottenimento delle informazioni utili, sono fatte utilizzando la libreria Retrofit.

Con l'opportuna implementazione di alcune classi associabili 1 : 1 ai dati che ci si aspetta dalle risposte delle API questa libreria permette di ottenere facilmente le informazioni desiderate.

Retrofit

Retrofit [9] 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.

3.5.1 Attivazione dei sensori

I sensori del dispositivo sono gestiti direttamente dal sistema operativo. Pertanto Android rende disponibile un gestore (*SensorManager* [10]) per permettere di richiederne l'attivazione e la disattivazione.

```
// Get sensors type
Sensor acc = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
Sensor gyro = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);

// Start accelerometer if the device has one
sensorManager.registerListener(sensorEventListener, acc, 50);

// Start gyroscope if the device has one
sensorManager.registerListener(sensorEventListener, gyro, 50);
```

Code Snippet 10: Attivazione dei sensori

```
// Stop active sensors
sensorManager.unregisterListener(sensorEventListener);
```

Code Snippet 11: Disattivazione dei sensori

Frequenza di campionamento

Ho scelto una frequenza di campionamento di 20 Hz che corrisponde ad avere una acquisizione ogni

$$T = \frac{1}{20} = 0.05s = 50ms$$

Si può notare come 50ms sia proprio il periodo di campionamento impostato nel codice 10 all'attivazione dei sensori.

3.5.2 Estrazione dei dati inerziali

Dopo l'attivazione dei sensori il sistema operativo richiama una specifica funzione di *callback* ogni volta che si realizza un evento, ovvero ad ogni nuova acquisizione.

È quindi necessario implementare l'interfaccia *SensorEventListener* [11] definendo il callback con l'opportuna gestione dei dati inerziali ottenuti.

Callback

Una callback è una determinata funzione richiamata dal sistema operativo al verificarsi di un determinato evento. Consente di implementare una gestione personalizzata dell'evento che l'ha invocata.

```
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 12: Implementazione del callback dei sensori

3.6 Comunicazioni con il server

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

Connessione al server

Per prima cosa 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 mediante la classe *Socket* [12].

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

Code Snippet 13: Implementazione della connessione via socket

Invio di un messaggio

Una volta attiva la connessione è attiva, si può procedere all'invio dei dati ogni qual volta 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 14: 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 15: Implementazione della ricezione di un messaggio

Ogni messaggio ricevuto viene elaborato in base alla tipologia. Alcuni di essi rappresentano solo una conferma di avvenuta connessione o di corretta ricezione dei messaggi. Altri invece contengono informazioni da mostrare direttamente all'utente.

Nel codice 7 di esempio è mostrato uno dei messaggi ricevibili dall'applicazione in questa fase.

Capitolo 4

Classificazione

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

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

Keras

Keras [13] 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 [14]) 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 ricevuti dall'applicazione.

4.1.1 Caricamento dei dati

I file CSV 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 semplificare la trattazione nelle procedure seguenti sarà considerato 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 si può notare i valori contenuti in un singolo record corrispondono a quelli inviati dall'applicazione in un messaggio durante la raccolta dei dati di apprendimento, eccezion fatta per il tipo di sensore già utilizzato 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 16: Creazione del dataframe a partire dal file CSV

4.1.2 Valutazione della posizione del dispositivo

Comunemente durante lo sviluppo di tecniche per il riconoscimento delle attività la posizione del dispositivo di raccolta dati viene spesso sottovalutata. Tuttavia si tratta di un aspetto molto importante per una miglior classificazione [15].

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.

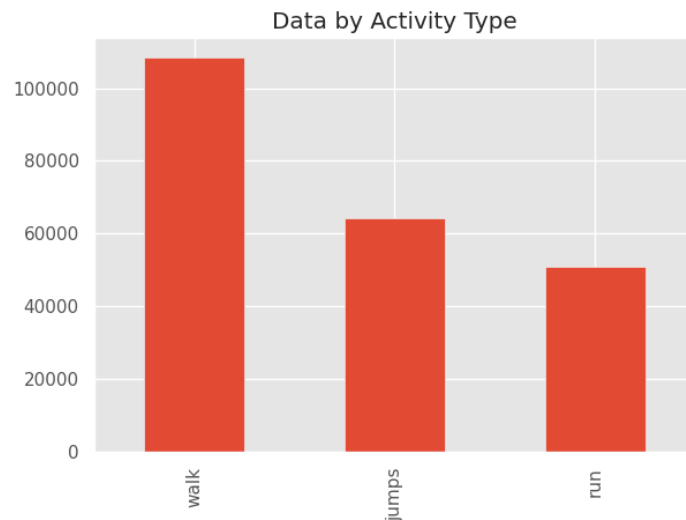


Figura 4.1: Visualizzazione della suddivisione per attività

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.

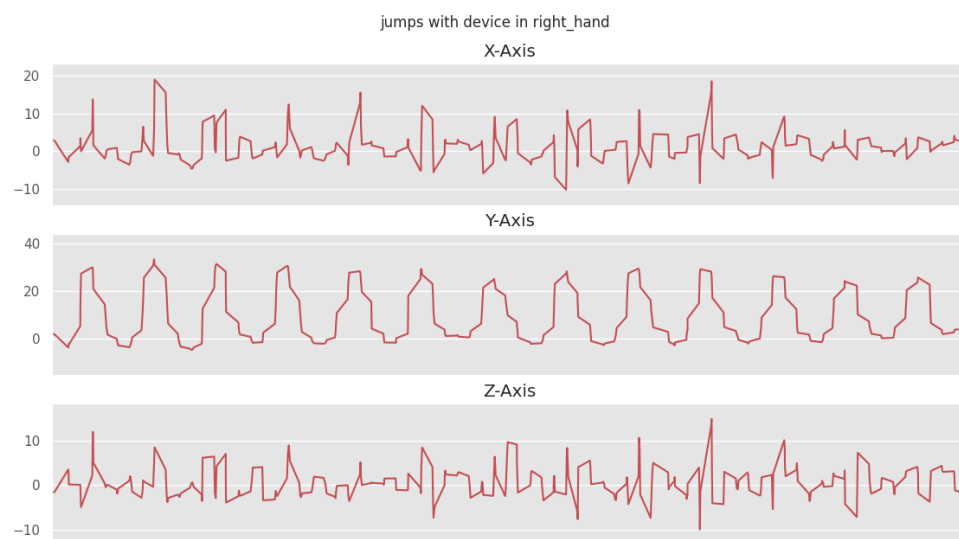


Figura 4.2: Dati accelerometrici durante l'attività "Salti" con il dispositivo nella mano destra



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 Partizionamento dei dati

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

A questo punto è necessario organizzare i dati in nostro possesso in modo che ad una serie di caratteristiche (i tre assi x, y, z ed il valore temporale) sia associabile un'etichetta rappresentante l'attività.

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 17: 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

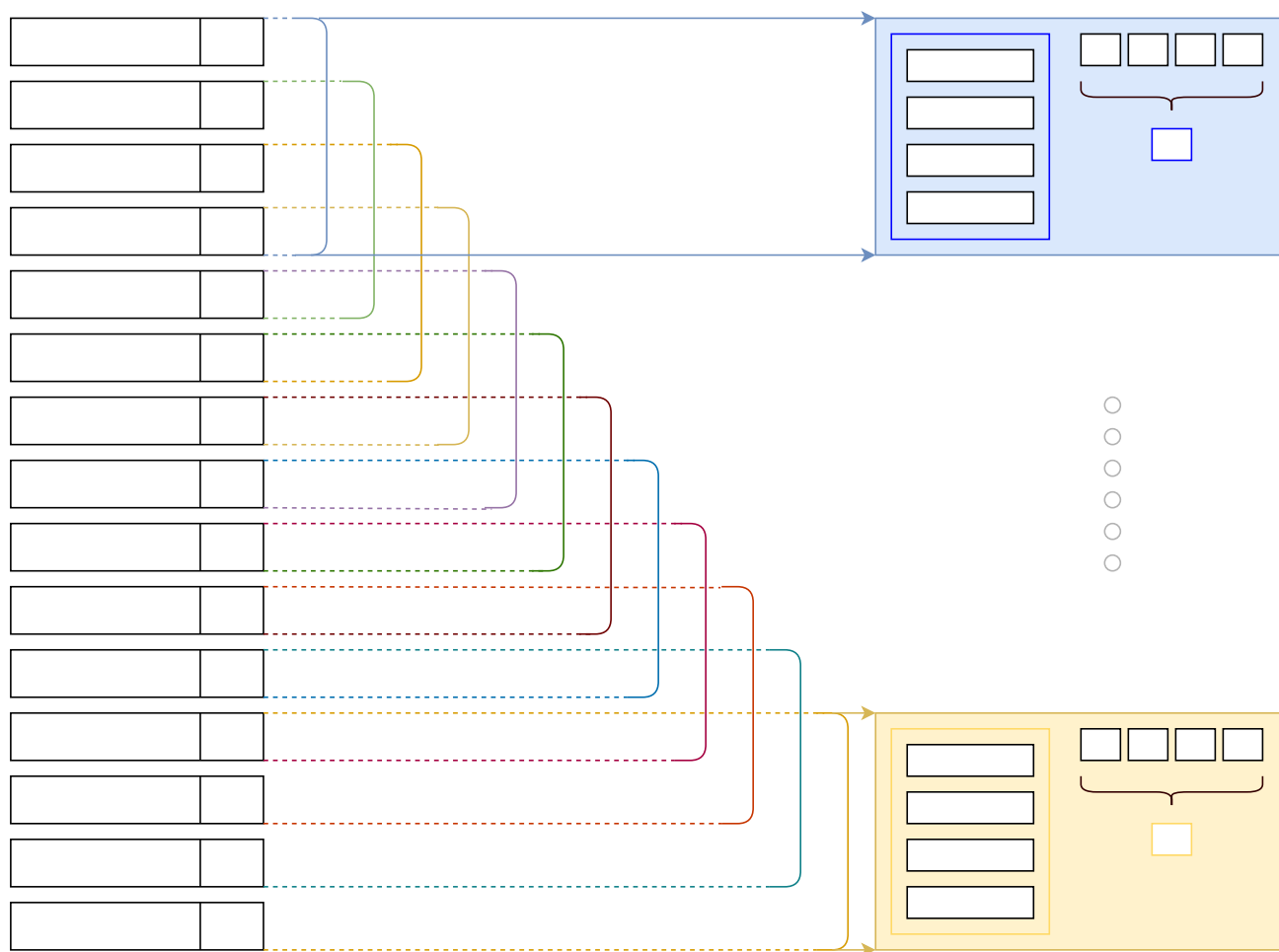


Figura 4.5: Creazione dei segmenti e delle etichette

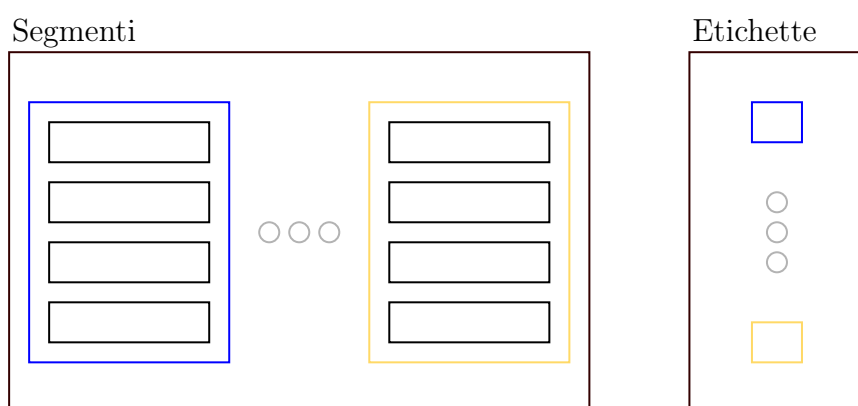


Figura 4.6: Risultato dopo la creazione dei segmenti e delle etichette

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 18: 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 19: 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.

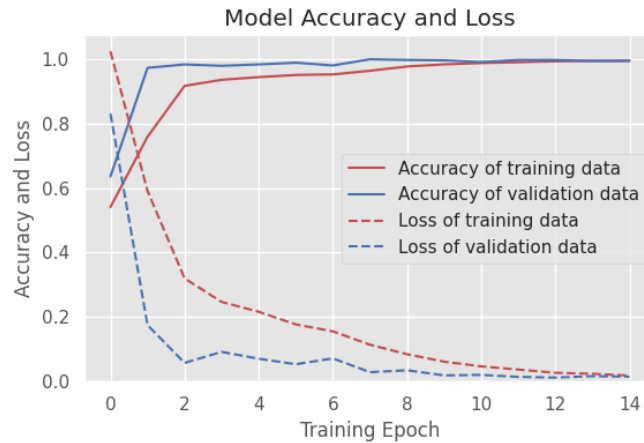


Figura 4.7: Statistiche del modello ottenuto

4.1.8 Salvataggio del modello

Il modello generato è facilmente salvato in un file con estensione `.h5` che mi permetterà in seguito di ricaricarlo senza dover riefettuare l'intero processo sui dati.

```
model_m.save(model_file, overwrite=True)
```

Code Snippet 20: Salvataggio del modello ottenuto

4.1.9 Testing della rete neurale

Per testare il modello appena creato utilizzo la partizione precedentemente separata.

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.

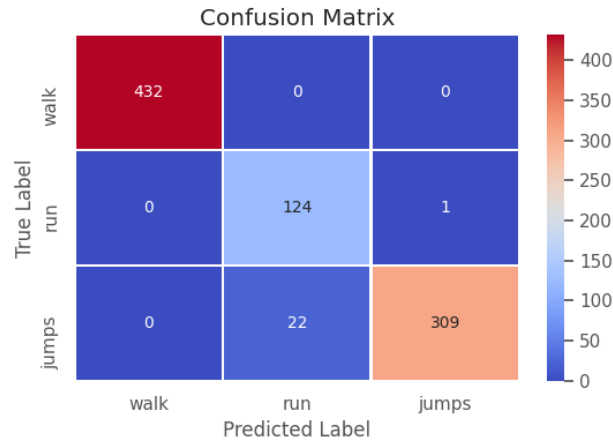


Figura 4.8: Matrice di confusione del modello ottenuto

4.2 Predizioni

Il processo di predizione consiste nell’ipotizzare, grazie ad un modello tra quelli generati, l’etichetta più opportuna da assegnare ad un insieme di record contenenti

- i valori ottenuti dal sensore
- il dato temporale
- il tipo di sensore a cui si riferiscono
- la posizione del dispositivo durante l’esecuzione

4.2.1 Scelta del modello

Per prima cosa, tra le informazioni devo differenziare i dati che mi permetteranno di identificare il modello e le caratteristiche su cui avvierò la predizione.

Ricordo che durante l’apprendimento l’informazione che identificava il tipo di sensore e quella che differenziava la posizione del dispositivo sono state utilizzate per la generazione di differenti modelli. Disponendo di queste informazioni è quindi possibile ricavare il modello corretto.

Si nota che le informazioni rimanenti (valori degli assi e valore temporale) corrispondono alle caratteristiche di analisi.

4.2.2 Preparazione dei dati

Fornita una serie di caratteristiche (i tre assi x, y, z ed il valore temporale), ci si aspetta di ottenere in risposta un’etichetta rappresentante l’attività.

Perchè tutto funzioni è indispensabile che i dati forniti alla rete neurale durante la fase di ipotesi abbiano la stessa struttura di quelli utilizzati durante la generazione dei modelli. Si attuano in questa fase le stesse procedure viste in precedenza nella corrispondente sezione 4.1.5 di apprendimento.

Operazioni preliminari

Per prima cosa si attua la trasformazione del valore temporale dal significato assoluto a quello relativo. Ed in seguito si normalizzano le 4 caratteristiche.

Creazione dei segmenti

L'insieme dei record è nuovamente preso a gruppi, anche sovrapposti, per la creazione dei segmenti.

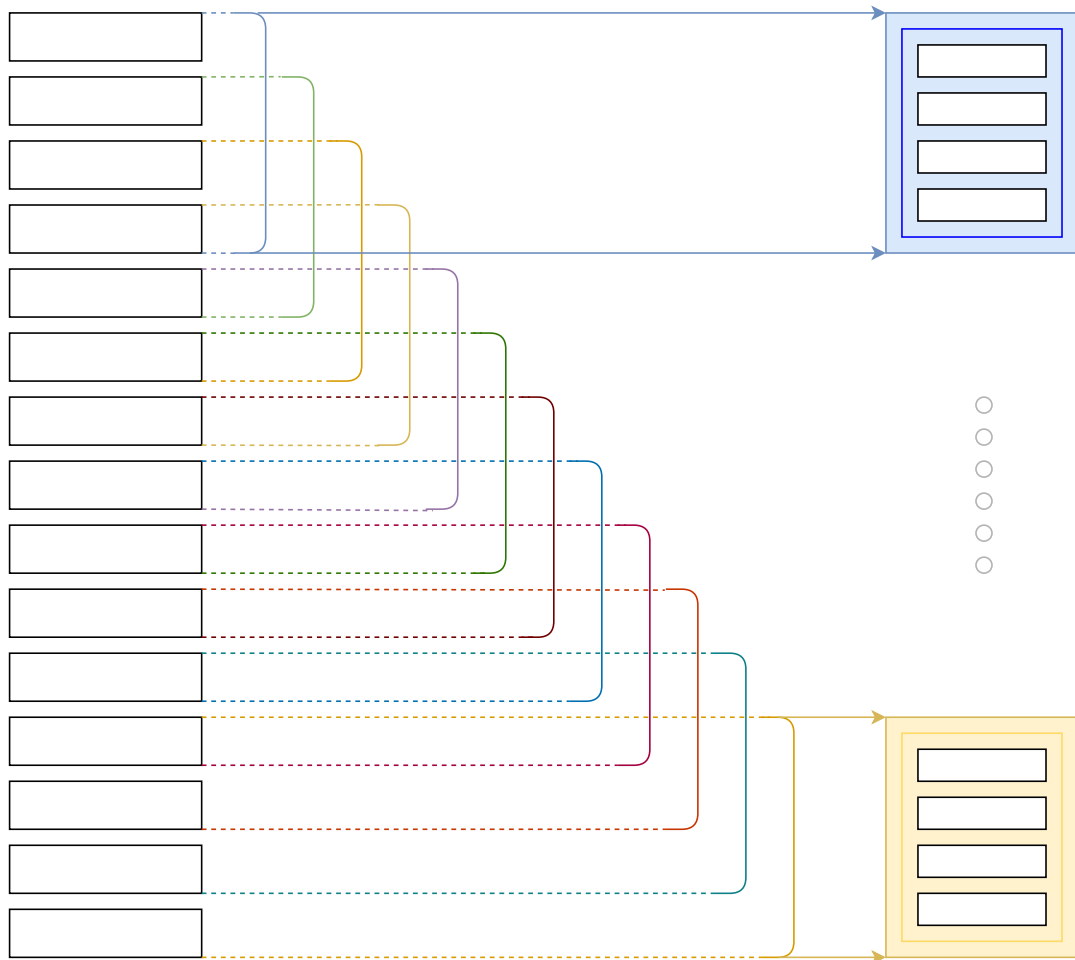


Figura 4.9: Creazione dei segmenti

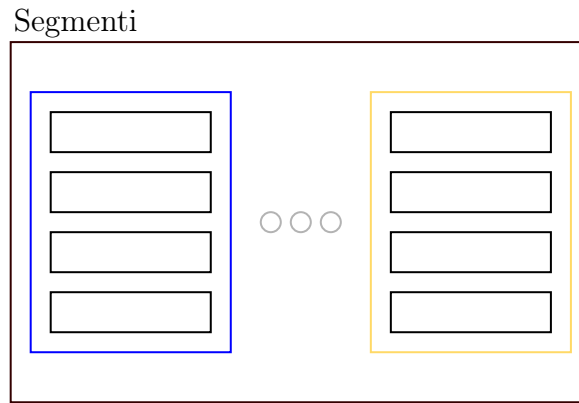


Figura 4.10: Risultato dopo la creazione dei segmenti

4.2.3 Ipotesi dell'etichetta

A questo punto il lavoro è delegato alla rete neurale che ricevendo in input l'insieme dei segmenti e con l'ausilio del modello selezionato fornisce in output l'insieme delle etichette ipotizzate. Il risultato ottenuto è un'insieme di etichette, ognuna associata ad un segmento.

Per ricavare l'etichetta assoluta associata alle caratteristiche date in input procedo banalmente considerando quella che compare più volte.

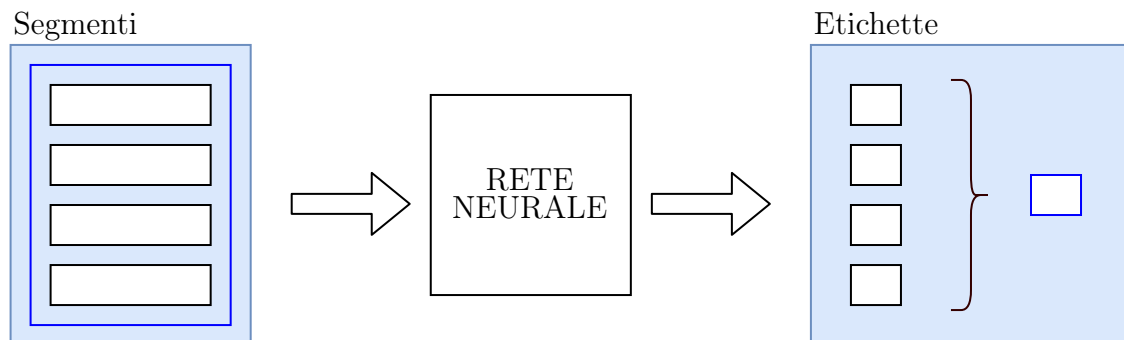


Figura 4.11: Processo di predizione

Capitolo 5

Conclusioni

Riferimenti

Bibliografia

- [1] C. Jobanputra, J. Bavishi e N. Doshi. «Human Activity Recognition: A Survey». In: 155 (2019), pp. 698–703. ISSN: 1877-0509. DOI: 10.1016/j.procs.2019.08.100 (cit. a p. 3).
- [2] 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).
- [15] 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. 27).

Siti

- [3] *Docker*. URL: <https://github.com/docker> (cit. a p. 6).
- [4] *Flask*. URL: <https://github.com/pallets/flask/> (cit. a p. 11).
- [5] *Material Design*. URL: <https://material.io/> (cit. a p. 15).
- [6] *Android Services*. URL: <https://developer.android.com/guide/components/services> (cit. alle pp. 16, 17).
- [7] *TextToSpeech*. URL: <https://developer.android.com/reference/android/speech/tts/TextToSpeech?hl=en> (cit. a p. 20).
- [8] *CountDownTimer*. URL: <https://developer.android.com/reference/android/os/CountDownTimer> (cit. a p. 21).
- [9] *Retrofit*. URL: <https://square.github.io/retrofit/> (cit. a p. 21).
- [10] *SensorManager*. URL: <https://developer.android.com/reference/android/hardware/SensorManager> (cit. a p. 22).
- [11] *SensorEventListener*. URL: <https://developer.android.com/reference/android/hardware/SensorEventListener?hl=en> (cit. a p. 22).
- [12] *Socket*. URL: <https://developer.android.com/reference/java/net/Socket> (cit. a p. 24).
- [13] *Keras*. URL: <https://github.com/keras-team/keras> (cit. a p. 26).
- [14] *Tensorflow*. URL: <https://github.com/tensorflow/tensorflow> (cit. a p. 26).