POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo di un'applicazione Android con l'integrazione di una rete neurale per il supporto a dispositivi medicali indossabili

Relatori

Prof. Eros PASERO

Ing. Vincenzo RANDAZZO

Candidato

Giacomo GORGA

Ottobre 2021

Sommario

Il mondo che ci circonda è in costante evoluzione, e la tecnologia si sta proponendo sempre più come soluzione alla quasi totalità dei problemi. Moltissime persone oramai possiedono uno smartphone, i quali, tramite gli assistenti vocali, l'intelligenza artificiale e la vastità di funzionalità messe a disposizione dell'utente, risultano indispensabili. Questo progresso può essere applicato anche in ambito medico. Il progetto descritto in questa tesi si pone nel campo dei dispositivi medici indossabili per il monitoraggio della salute.
La prima fase del progetto ha riguardato l'aggiornamento con nuove funzionalità dell'app Android relativa al dispositivo EcgWatch sviluppato negli anni passati dal Laboratorio di Neuronica per acquisire un elettrocardiogramma con due dita.
Successivamente, in un lavoro di tesi parallelo al presente, l'hardware dell'EcgWatch è stato rivisto, aggiornato ed esteso per misurare, anche, la saturazione dell'ossigeno nel sangue, stimare la pressione arteriosa e trasmettere i dati allo smartphone. Tale dispositivo, chiamato PulsEcg, è un dispositivo indossabile come un orologio che presenta due elettrodi, uno nella parte inferiore che rimane a contatto con il polso, e un altro nella parte superiore, su cui è possibile appoggiare un dito della mano opposta: in questo modo si genera una differenza di potenziale per misurare l'elettrocardiogramma. Inoltre, al centro dell'elettrodo superiore, vi è anche il sensore per la misurazione della fotopletismografia.
Entrambi i dispositivi sono comandabili tramite l'omonima app per smartphone con la quale comunicano tramite il protocollo Bluetooth: l'EcgWatch tramite Bluetooth 2.0, il PulsEcg tramite Bluetooth 4.0 (Low Energy). L'acquisizione dei campioni avviene con una frequenza di 1000 Hz per EcgWatch e di 500 Hz per PulsEcg, e una volta terminato l'invio di tali campioni all'app, quest'ultima effettua delle operazioni di filtraggio ed elaborazione del segnale tramite degli algoritmi studiati ad hoc. Vengono calcolate la frequenza cardiaca, la percentuale di ossigeno nel sangue, e, tramite una rete neurale integrata, la pressione sistolica e diastolica.
Le due app, sviluppate nel corso di questa tesi, risultano pertanto una tecnologia abilitante per i dispositivi wearable. Infatti, per consentire l'adozione degli stessi su una fetta quanto più grande possibile di popolazione, è stata prestata particolare enfasi alla semplicità dell'interfaccia utente ed, più in generale, alla user experience.

Ringraziamenti

Un grazie al Prof. Eros Pasero, che segue questo progetto da tempo con dedizione e passione, redendosi sempre disponibile e puntuale.

Un grazie all'Ing. Vincenzo Randazzo, anch'egli sempre disponbile a fornire il proprio supporto, il cui aiuto è stato fondamentale in alcuni punti cruciali dello sviluppo.

Un grazie a tutti i miei amici, per aver sempre mostrato interesse durante il proseguo della mia carriera universitaria e per essere sempre presenti.

Il grazie più grande alla mia famiglia, il mio faro in un mare di difficoltà e insicurezze. Spesso non sono riuscito a mostrare la mia gratitudine nei vostri confronti, un po' a causa del mio essere e un po' a causa dei risultati non sempre ottimi, ma spero che questo lavoro come conclusione di un lungo cammino possa rendervi orgogliosi di me.

Con affetto

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

ECG
Elettrocardiogramma

PPG
Fotopletismografia

AI
Artificial Intelligence

UI
User Interface

UX
User Experience

IDE
Integrated Development Environment

VM
Virtual Machine

JVM
Java Virtual Machine

ART
Android RunTime

API
Application Programming Interface

SDK

Software Development Kit

NDK

Native Development Kit

GPS

Global Positioning System

RFCOMM

Radio Frequency Communication

ISM

Industrial Scientific Medical

BLE

Bluetooth Low Energy

GATT

Generic Attribute

CPU

Central Processing Unit

GPU

Graphic Processing Unit

PDF

Portable Document Format

ML

Machine Learning

NN

Neural Network

ANN

Artificial Neural Network

PWV

Pulse Wave Velocity

PTT

Pulse Transit Time

SBP

Systolic Blood Pressure

DBP

Diastolic Blood Pressure

BPM

Battiti per minuto

SpO2

Saturazione dell'ossigeno

IoT

Internet of Things

RMSE

Root Mean Squared Error

MAE

Mean Absolute Error

MTU

Maximum Transmission Unit

Capitolo 1

Introduzione

1.1 Premessa

Il mondo che ci circonda è in costante evoluzione, e la tecnologia si sta proponendo sempre più come soluzione alla quasi totalità dei problemi. Moltissime persone oramai possiedono uno smartphone, il quale, tramite gli assistenti vocali, l'intelligenza artificiale e la vastità di funzionalità messe a disposizione dell'utente, risulta indispensabile. Non solo: gli elettrodomestici sono sempre più intelligenti, le case, le automobili, ecc. Questo progresso può essere applicato anche in ambito medico: da dispositivi complessissimi e costosi come braccia e gambe robotiche, fino ad arrivare a comuni oggetti di elettronica per il monitoraggio della salute, in primis smartwatch.
Non da sottovalutare, l'impatto avuto dalla pandemia di COVID-19: l'affidarsi a soluzioni informatiche per poter continuare a svolgere le proprie attività quotidiane è divenuta una necessità. Si vuole però porre l'attenzione su un altro aspetto: la notevole sensibilizzazione della popolazione nei confronti del tema salute e prevenzione delle malattie.
Questo è il contesto in cui nasce l'idea di creare un dispositivo medico indossabile per il monitoraggio della salute e controllabile tramite smartphone, dedicato a persone con patologie cardiache, a pazienti COVID, soggetti ipertesi, ma anche a coloro che semplicemente desiderano ogni giorno avere una panoramica su alcuni aspetti fondamentali del proprio benessere.

1.2 Stato dell'arte

Nel marzo 2014, Google presenta la prima versione di Android Wear [1], il sistema operativo per smartwatch dedicato all'interfacciamento con dispositivi Android, e a settembre dello stesso anno Apple presenta la prima versione di Apple Watch [2],

l'orologio intelligente della casa di Cupertino pioniere di questa fascia di mercato. Da allora, tutte le aziende produttrici di smartphone hanno lanciato i propri smartwatch, e nuove aziende sono nate con il solo intento di produrre orologi intelligenti, come ad esempio Huami, dedicati al controllo delle notifiche, al monitoraggio della salute oppure per lo sport, sia a livello amatoriale che agonistico. Tuttavia, pochi sono i dispositivi con certificazione medica in grado di fornire informazioni dettagliate e valide, e di questi il prezzo è non sempre accessibile a tutti, senza contare la difficoltà di utilizzo per utenti inesperti.

1.3 Obiettivi della tesi

Questa tesi vuole raccogliere tutte le premesse e le considerazioni fatte sino ad ora, ergo studiare la possibilità di offrire sul mercato un prodotto innovativo sotto diversi punti di vista. Si parla di un dispositivo indossabile, adatto a tutte le fasce d'età, di qualità, ad un prezzo accessibile, con risultati affidabili dal punto di vista medico, e controllabile tramite smartphone, sia Android che iOS.
I dispositivi sviluppati per adempire agli obiettivi sopra citati sono:

- EcgWatch, in grado di monitorare elettrocardiogramma e frequenza cardiaca;
- PulsEcg, in grado di monitorare elettrocardiogramma e frequenza cardiaca, fotopletismografia e saturazione sanguigna, pressione arteriosa sistolica e diastolica.

Il secondo in particolare risulta unico nel suo genere, perché attualmente non esiste un dispositivo indossabile (nemmeno tra i più costosi) in grado di eseguire contemporaneamente tutte le funzioni elencate.
Il lavoro svolto che sarà minuziosamente descritto riguarda lo sviluppo delle applicazioni Android a supporto dei due dispositivi, e come sia possibile ottenere tramite una rete neurale la pressione sistolica e diastolica senza l'ausilio di uno sfigmomanometro.
La tabella 1.1 riassume le caratteristiche delle due applicazioni.

	Android	iOS	Bluetooth	ECG	PPG	Pressione
EcgWatch	✓	X	✓	✓	X	X
PulsEcg	✓	✓	✓	✓	✓	✓

Tabella 1.1: Confronto applicazioni EcgWatch e PulsEcg

Capitolo 2

Nozioni di medicina

Lo sviluppo di dispositivi per scopi medici prevede una conoscenza basilare sugli argomenti interessati, e in questo capitolo si vogliono illustrare alcune nozioni di medicina, in particolare cardiologia, che sono indispensabili per comprendere dove traggano le fondamenta gli algoritmi sviluppati.

2.1 Il cuore

Il cuore è l'organo centrale dell'apparato circolatorio degli animali superiori. Nell'uomo, il cuore è situato nella parte mediana della cavità toracica, subito dietro lo sterno, tra i 2 polmoni. Ha la forma di un cono appiattito in senso antero-posteriore, con la base disposta verso l'alto a destra e all'indietro, e la punta diretta in basso a sinistra e in avanti.
Il cuore consta essenzialmente di una massa contrattile, il *miocardio*, di un rivestimento esterno, l'*epicardio*, e di uno interno, l'*endocardio*. Altri elementi morfologici sono lo scheletro del cuore, costituito dagli anelli fibrosi che circondano gli orifizi atrioventricolari e arteriosi, e il sistema di conduzione specifico, costituito da un tessuto miocardico particolare nel quale si originano gli stimoli automatici del cuore, che vengono poi trasmessi a tutto il miocardio.
Dal punto di vista funzionale, il cuore è un muscolo cavo con funzione di pompa aspirante e premente che, durante la rivoluzione cardiaca, riceve nelle sue cavità il sangue che, per mezzo delle vene, gli giunge dalla periferia e lo invia poi nelle due arterie, aorta e polmonare [3].

2.1.1 Atri e ventricoli

Il cuore presenta 4 cavità: 2 superiori (*atri* od *orecchiette*), separate fra loro dal setto interatriale, e 2 inferiori (*ventricoli*), separate fra loro dal setto interventricolare.

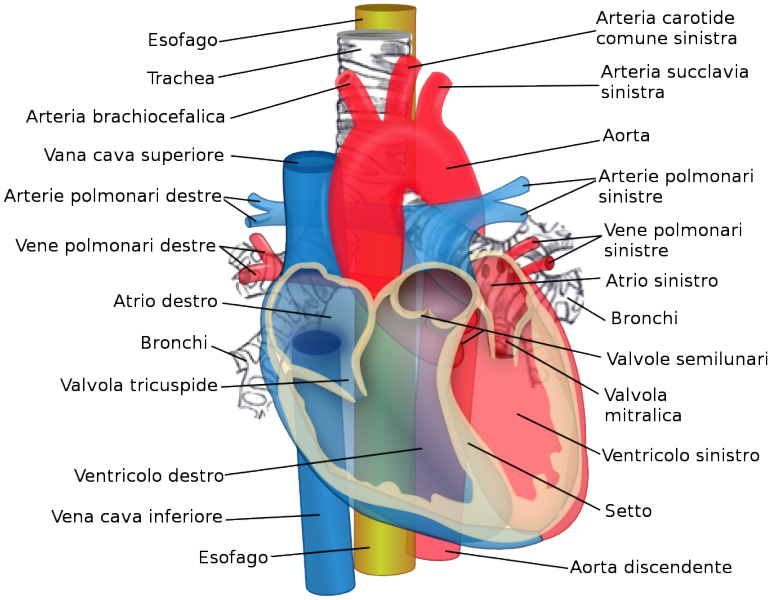


Figura 2.1: Struttura anatomica del cuore

Nell'atrio destro sboccano le vene cava superiore, cava inferiore e il seno coronario; nell'atrio sinistro sboccano le vene polmonari e alcune piccole vene cardiache.
I ventricoli hanno la forma di coni con la base in alto; la base di ciascun ventricolo presenta due orifizi muniti di valvole: l'orifizio atrioventricolare sul quale si impiantano le valvole atrioventricolari (*bicuspide* per il ventricolo sinistro, *tricuspide* per il destro) e che mette in comunicazione il ventricolo con il tronco arterioso che si origina da esso (l'arteria polmonare dal ventricolo destro, l'aorta per il sinistro); le valvole degli orifizi arteriosi sono chiamate valvole *semilunari*. Le valvole atrioventricolari e quelle semilunari consentono il passaggio del sangue in una sola direzione e regolano in tal modo la circolazione attraverso le cavità cardiache [4]. La figura 2.1 illustra quanto spiegato.

2.1.2 Fibrillazione atriale

Sulle patologie legate al cuore si potrebbero scrivere dei libri, ma quella su cui si vuole porre l'interesse è la fibrillazione atriale, perché è l'unica in grado di essere riconosciuta come possibile patologia dagli algoritmi presenti nelle applicazioni.
La fibrillazione atriale si verifica quando gli atri non si contraggono in maniera sincrona e pertanto *fibrillano*, ovvero battono in modo molto rapido e irregolare. Il sangue non viene pompato in modo efficiente al resto del corpo, e di conseguenza il paziente affetto può avvertire debolezza e riscontrare sensazioni cardiache fastidiose

come un battito accelerato o irregolare.
La fibrillazione atriale può essere:

- parossistica (occasionale): di durata variabile, da pochi minuti a diversi giorni, ma che si risolve in maniera spontanea;
- persistente: risolvibile tramite la somministrazione di terapia farmacologica o l'erogazione di una particolare scarica elettrica (cardioversione) per ripristinare il normale ritmo cardiaco;
- permanente: continuamente presente e non curabile.

Le cause di questa patologia non sono sempre chiare: talvolta è congenita, può svilupparsi conseguentemente ad un infarto, ma può anche presentarsi senza problemi cardiaci pregressi [5].

2.2 Sistema elettrico

Parallelamente al sistema o apparato circolatorio, esiste anche un sistema elettrico che vede sempre il cuore come protagonista: infatti, il cuore è un generatore di impulsi elettrici, che nascono nel ventricolo destro. Il sistema elettrico coordina l'attività meccanica del cuore che genera l'eiezione cardiaca, cioè l'espulsione di sangue.
Un sistema elettrico invia al miocardio di lavoro impulsi ritmici, o stimoli, che determinano la contrazione cardiaca [6]. Il sistema elettrico è composto da tre parti principali:

1. i centri principali per la formazione dello stimolo (o impulso), cioè il nodo del seno e il nodo atrioventricolare;
2. i principali fasci per la conduzione dello stimolo, quindi fascio di His, branca sinistra e branca destra;
3. il sistema di Purkinje, che si infiltra nel miocardio ventricolare.

Tutte queste tre parti sono capaci di generare spontaneamente uno stimolo con frequenza decrescente, procedendo dalla base del cuore verso l'apice.
Il sistema elettrico può essere in qualche modo paragonato ad un circuito, pertanto valgono leggi analoghe a quelle descritte dalla teoria dei circuiti. Se consideriamo il cuore come un generatore di corrente e i vasi sanguigni come delle resistenze, tra le estremità del corpo (mani e piedi) si genera una differenza di potenziale.
La figura 2.2 indica le parti principali del sistema elettrico nel cuore.

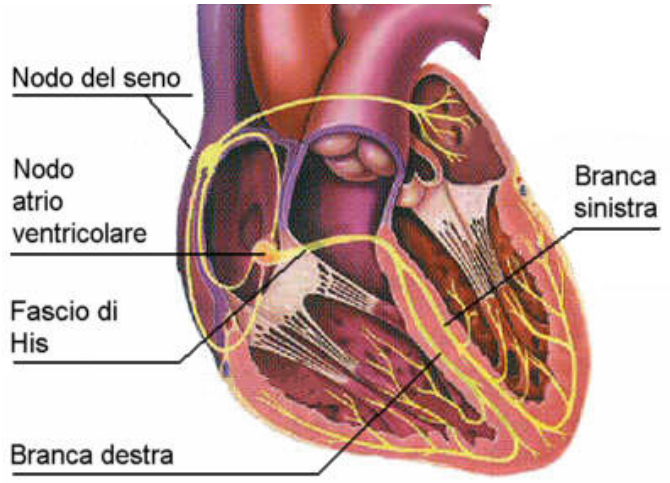


Figura 2.2: Sistema elettrico del cuore

2.3 Elettrocardiogramma

L'elettrocardiogramma (ECG) è la registrazione grafica delle correnti elettriche e delle variazioni di differenza di potenziale che si verificano sulle estremità e sulla superficie del corpo. Ogni battito del cuore è costituito da una contrazione degli atri, dalla propagazione dell'eccitazione ai ventricoli, da una contrazione dei ventricoli e quindi dall'immissione di sangue in circolo; la corrente elettrica generata in queste fasi compare sul tracciato dell'elettrocardiogramma sotto forma di una linea continua a tratti ascendente, discendente o orizzontale.
Questo esame, peraltro poco costoso, viene realizzato sia per verificare se una persona è adatta a svolgere una determinata attività (ad esempio sportiva), sia per diagnosticare e studiare particolari disturbi, come aritmie, blocchi cardiaci, ischemie e infarto; la lettura dell'elettrocardiogramma permette inoltre di esaminare le dimensioni e le funzionalità del cuore.
In alcuni casi particolari viene verificato il funzionamento del cuore sotto sforzo, oppure registrato per un giorno intero (12-24 ore): si tratta dell'elettrografia dinamica (o Holter), che permette di effettuare una registrazione continua dell'elettrocardiogramma [7].

Considerando l'ecg come un grafico a tutti gli effetti con ascisse e ordinate, l'unità di misura dell'asse orizzontale sono i secondi, mentre per quello verticale i millivolt (mV). L'elettrocardiogramma è rappresentato convenzionalmente su carta millimetrata, come si vede in figura 2.3: un quadratino rappresenta 40 ms se visto sulle ascisse, 0,1 mV se visto sulle ordinate. La curva spiega quindi come varia la differenza di potenziale nel tempo, generando delle onde.

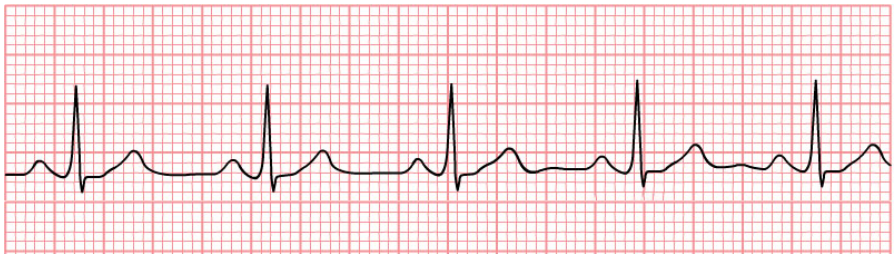


Figura 2.3: Tratto di un elettrocardiogramma

2.3.1 Derivazioni

Le derivazioni elettrocardiografiche possono essere considerate degli schermi di proiezione per i vettori corrispondenti alle forze elettromotrici presenti nel cuore; in altre parole, le derivazioni consentono di esplorare le diverse porzioni cardiache nella loro attivazione elettrica.

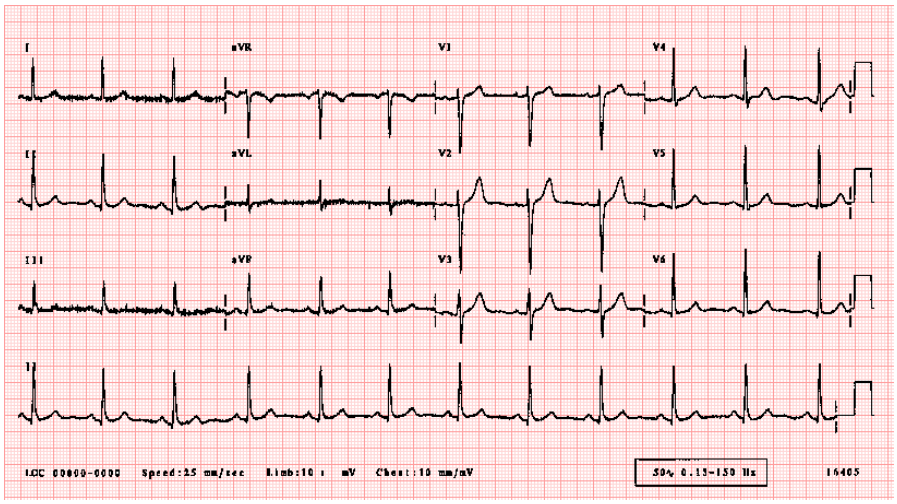


Figura 2.4: Esempio di elettrocardiogramma completo

L'esecuzione di un elettrocardiogramma completo prevede 12 derivazioni: 6 periferiche svolte da 4 elettrodi posti su caviglie e polsi, e 6 precordiali svolte da 6 elettrodi posti sul torace; la figura 2.4 mostra un esempio di ECG completo. Più in particolare, le derivazioni periferiche (verticali) si suddividono in derivazioni bipolari degli arti I, II e III (dette derivazioni standard), e derivazioni unipolari degli arti aumentate aVR, aVL e aVF (dette derivazioni di Goldberger); le derivazioni precordiali (orizzontali) unipolari o della parete toracica sono da V1 a V6 o V9 (dette derivazioni di Wilson) [8].

Esistono pertanto due tipi di derivazioni in questa modalità di registrazione: unipolari e bipolari. Le derivazioni unipolari vengono registrate accoppiando un singolo elettrodo "esplorante" con un terminale centrale, mentre le derivazioni bipolari registrano la differenza di potenziale fra due elettrodi.
I dispositivi che verranno descritti successivamente, possedendo 2 elettrodi, sono in grado di misurare solo una derivazione. Per comprendere il motivo per cui è possibile ottenere solo una derivazione e quali delle 12 si possono misurare, si illustra il triangolo di Einthoven.

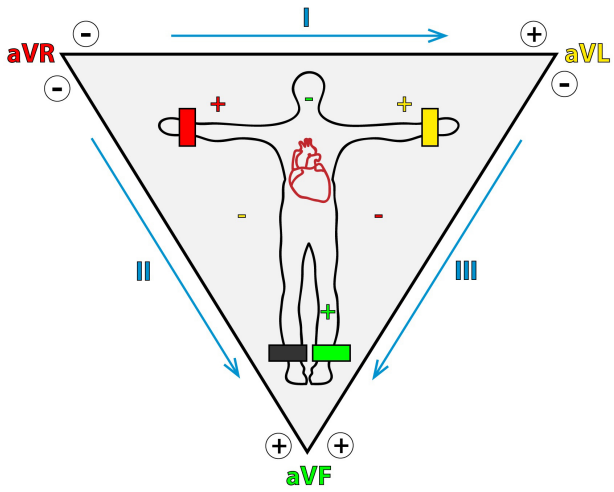


Figura 2.5: Il triangolo di Einthoven

Il Triangolo di Einthoven si basa sulla disposizione immaginaria di un triangolo equilatero rovesciato sul torace, il cui centro coincide con il cuore. Ogni angolo della figura geometrica è elettricamente coincidente con un punto di un arto specifico a cui viene assegnato un nome: VL (sinistra), VR (destra) e VF (piede sinistro); il piede destro è definito neutro e non partecipa alla formazione del triangolo (corrisponde alla massa in un circuito elettrico).
Dal calcolo delle differenze di potenziale tra due elettrodi periferici alla volta, si ottiene una derivazione bipolare:

- I derivazione (D1): misurata tra l'elettrodo positivo sul braccio sinistro e quello negativo sul braccio destro;
- II derivazione (D2): misurata tra l'elettrodo positivo sulla gamba sinistra e quello negativo sul braccio destro;
- III derivazione (D3): misurata tra l'elettrodo positivo sulla gamba sinistra e quello negativo sul braccio sinistro.

La figura 2.5 schematizza il triangolo di Einthoven sovrapponendolo ad una persona e mostrando la posizione degli elettrodi.
La derivazione a cui si farà riferimento nei prossimi capitoli senza specificare è la I derivazione; dal punto di vista medico, essa consente di capire se il ritmo cardiaco sinusale, cioè positivo in D1 e negativo in aVR. Dal punto di vista delle patologie, è possibile riconoscere solo una possibile fibrillazione atriale; non è possibile fare nessun'altra assunzione che abbia validità medica.

2.3.2 Significato delle onde

L'elettrocardiogramma, in condizioni normali, presenta una sequenza di onde particolari che rappresentano l'eccitabilità contrattile sia degli atri che dei ventricoli; in condizioni patologiche, la registrazione presenta anomalie caratteristiche tali da consentire una eventuale diagnosi di malattia cardiaca.
La linea di base di un tracciato ECG è chiamata linea isoelettrica e indica i potenziali a riposo. Le deviazioni da questo punto sono indicate in ordine alfabetico e, dopo ciascuna, il tracciato ritorna normalmente al punto isoelettrico [9].

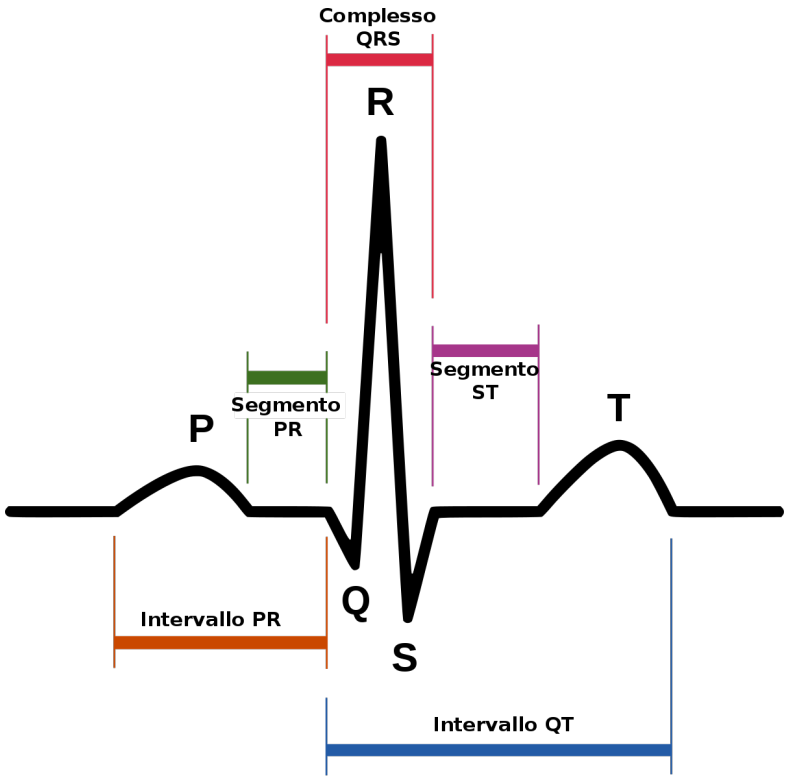


Figura 2.6: Le onde dell'elettrocardiogramma

Dopo aver osservato la figura 2.6 che schematizza le onde presenti in un tratto elettrocardiografico (cioè le deviazioni dalla linea isoelettrica), si riporta una descrizione di ciascuna:

- onda P: è l'onda dell'attivazione atriale (depolarizzazione) e consente di capire se l'impulso nasce correttamente, quindi rilevare eventuali aritmie. La dimensione dell'onda P può indicare anomalie sull'atrio, come ad esempio un ingrossamento. La distanza tra l'onda P e il complesso QRS consente anche di evidenziare alterazioni sulla conduzione dell'impulso: maggiore distanza indica un rallentamento della conduzione cardiaca (blocco di primo grado, vedere figura 2.7), minore distanza trattasi di anomalie spesso congenite, come ad esempio pre-eccitazione cardiaca. La durata tipica di quest'onda varia dai 50 ai 100 ms;

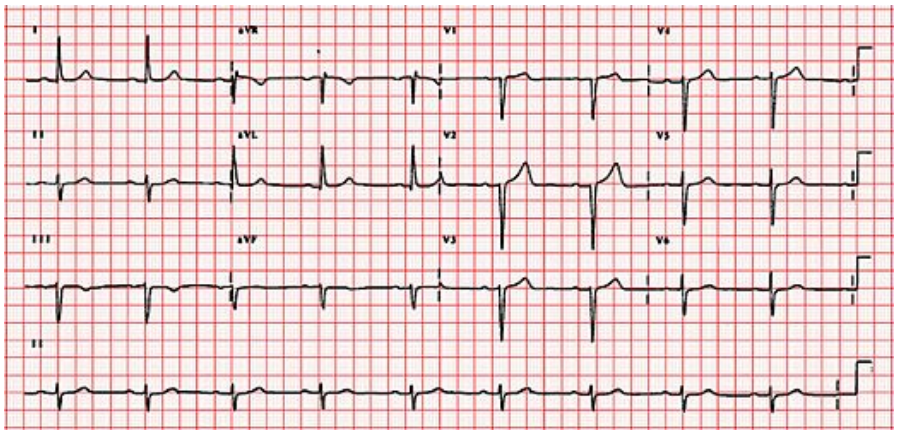


Figura 2.7: Elettrocardiogramma di un paziente con blocco atrioventricolare di I grado

- tratto P-Q: passaggio dell'impulso dall'atrio al ventricolo, la durata tipica è tra i 120 e i 200 ms;
- complesso QRS: fornisce informazioni circa l'attivazione del ventricolo e può essere indice di disturbi circolatori sia pregressi che acuti. Anche sul QRS si possono valutare disturbi di conduzione cardiaca, come ad esempio il blocco di Branca, che è un impedimento o un ritardo nella trasmissione degli impulsi. La figura 2.8 mostra un esempio di ECG con blocco di Branca.

  I voltaggi dell'onda possono essere dipendenti da dati costituzionali e/o patologie: ad esempio, un paziente obeso presenta voltaggi bassi; un paziente soggetto a cuore ipertrofico, quindi con le pareti inspessite, presenta un voltaggio maggiore. La durata normale del complesso è di 60 - 80 ms, mentre l'ampiezza tipica è di 0,6 - 0,8 mV;

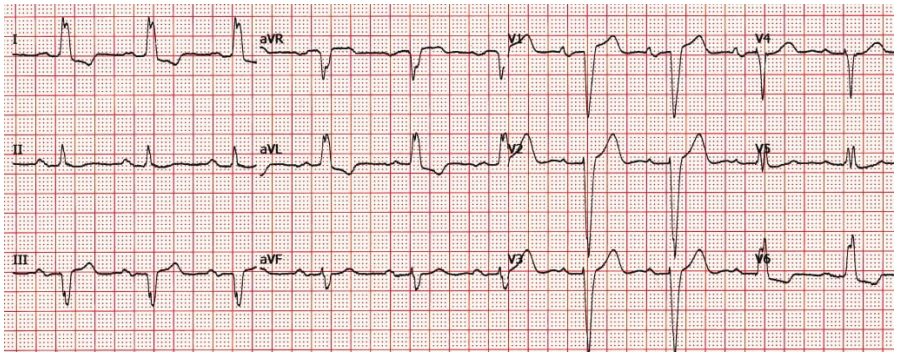


Figura 2.8: Elettrocardiogramma di un paziente con blocco di Branca sinistro

- tratto S-T: in questo tratto avviene la ripolarizzazione ventricolare, cioè il recupero di eccitabilità delle cellule cardiache, e si valutano alterazioni morfodinamiche rispetto alla linea isoelettrica dell'ecg, cioè il tratto T-P in cui non vi è attività cardiaca: una eccessiva convessità (detta *sopraslivellamento*) o concavità (detta *sottoslivellamento*) sono indice di un'ischemia cardiaca [10]. La figura 2.9 illustra un esempio di sottoslivellamento.

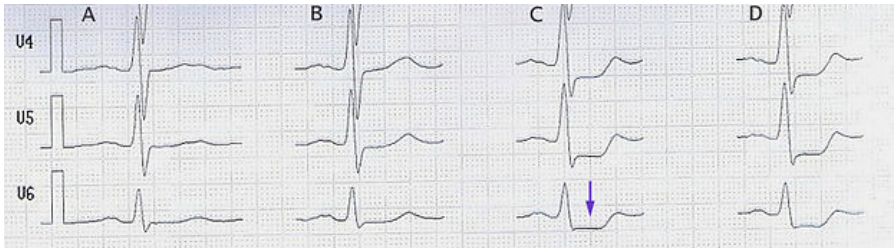


Figura 2.9: Esempio di sottoslivellamento

- onda T: rappresenta la ripolarizzazione ventricolare, ed è normalmente positiva: una negatività è indice di un'ischemia.

Tutte le derivazioni presentano l'onda P seguita dal complesso QRS, dal tratto S-T e dall'onda T; l'aspetto di ciascuna onda è differente in ciascuna derivazione.

2.4 Fotopletismografia

Il termine "pletismografia" deriva dal greco ed è la combinazione delle parole *plethysmos*, che significa aumento, e *grafia*, che indica qualcosa di scritto. La pletismografia è una tecnica diagnostica che permette di registrare graficamente le variazioni di volume di un organo o di una parte del corpo, indotta da variazioni del rispettivo contenuto di sangue. La fotopletismografia (PPG) è un tipo di pletismografia che studia la diffusione dei raggi infrarossi nei tessuti per capirne l'irrorazione.
Si esegue mediante l'uso di un rilevatore periferico da posizionare sui polpastrelli delle dita delle mani o dei piedi, comprendente una fotocellula e una sorgente luminosa. La fotocellula viene colpita da stimoli riflessi in rapporto al grado di vascolarizzazione del sistema esaminato. Poiché l'indicatore luminoso è assorbito più forte da sangue che i tessuti circostanti, i cambiamenti in flusso sanguigno possono essere individuati dai sensori di PPG come cambiamenti nell'intensità di indicatore luminoso [11].
Il grafico ottenuto è costituito da una linea continua che rappresenta un movimento ondulatorio come in figura 2.10; sull'asse delle ascisse vi è il tempo, mentre quello delle ordinate non ha unità di misura.

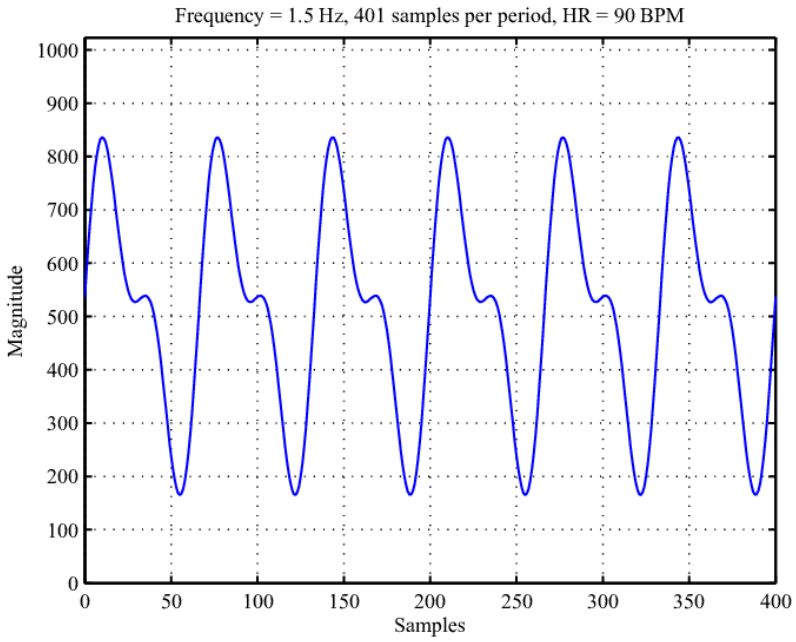


Figura 2.10: Esempio di PPG

2.4.1 Analisi dell'onda

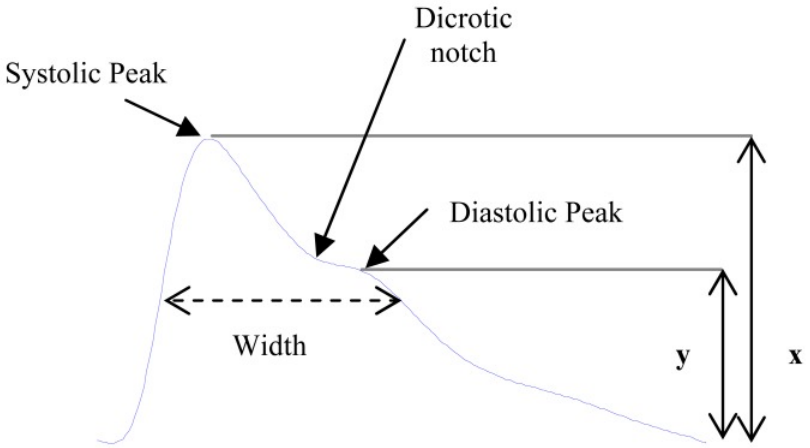


Figura 2.11: Onda del PPG

Si osservi la figura 2.11, che mostra la forma tipo di una singola onda del PPG. Essa è solitamente divisa in 2 fasi: la fase anacrotica che è data dal fronte di salita, e la fase catacrotica che è data dal fronte di discesa. La prima fase costituisce la sistole e presenta un picco sistolico, la cui ampiezza (x) indica la variazione del volume di sangue causato dal flusso nelle arterie. Il punto che separa la parte sistolica da quella diastolica prende il nome di *dicrotic notch*, e indica la chiusura della valvola aortica e l'inizio della diastole (vedere sezione 2.5).

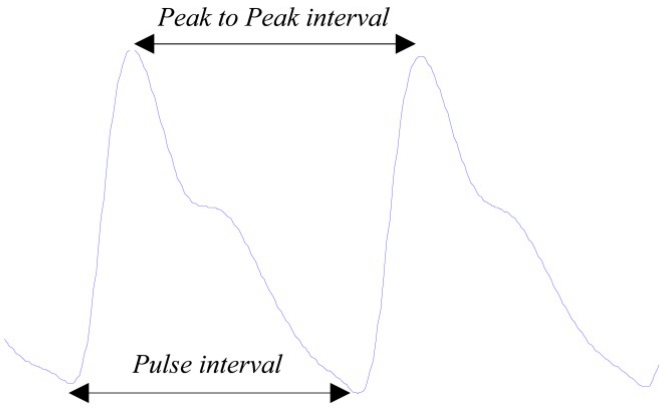


Figura 2.12: Distanza picco picco tra due onde

La distanza tra i picchi sistolici di 2 onde consecutive, come illustrata in figura 2.12, può essere paragonata agli intervalli R-R nell'ECG, e rappresentano un intero ciclo cardiaco.

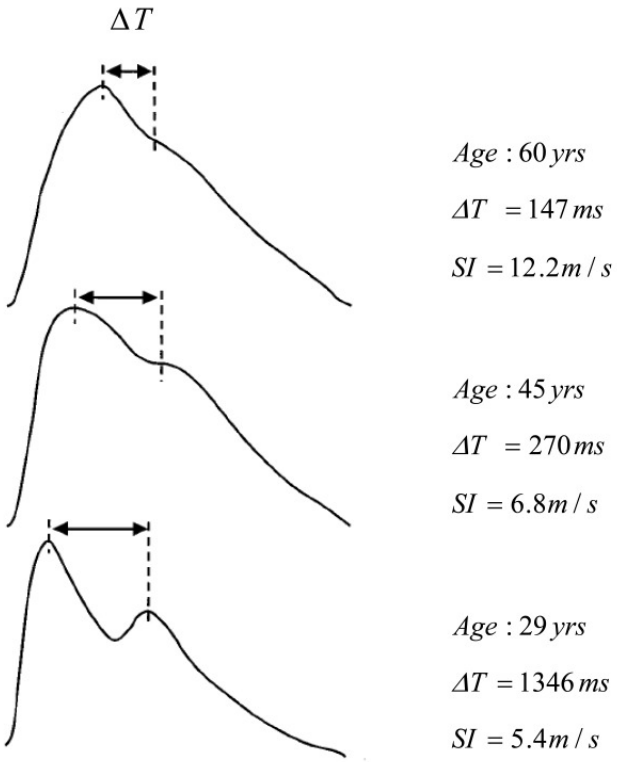


Figura 2.13: Differenza delle distanze tra picco sistolico e diastolico per età

Osservando invece la figura 2.13, si trae una considerazione interessante inerente la distanza tra il picco sistolico e quello diastolico di un'onda, che tende a diminuire con il passare degli anni [12].

2.4.2 Saturazione

La saturazione dell' ossigeno (SpO2) è un indice ematico che riflette la percentuale di emoglobina satura di ossigeno rispetto alla quantità totale di emoglobina presente nel sangue. La saturazione dell'ossigeno è un parametro vitale per definire il tenore di ossigeno di sangue e la consegna dell'ossigeno. A livello degli alveoli polmonari avviene uno scambio gassoso: l'ossigeno contenuto nell'aria inspirata entra nel circolo sanguigno, mentre l'anidride carbonica viene rilasciata. L'anidride carbonica ripercorre in senso inverso le vie respiratorie fino all'esterno del corpo (espirazione), mentre l'ossigeno viene trasportato nel sangue verso tutti gli organi e i tessuti

del corpo grazie all'emoglobina, una proteina con una struttura chimica adatta al legame con l'ossigeno. Le molecole di ossigeno raggiungono le cellule dell'organismo e al loro interno si verifica la cosiddetta "respirazione cellulare" che consente la produzione di energia.
L'SpO2 si indica in percentuale: il valore tipico di un soggetto in salute è compreso tra 95 e 100% (anche se il 100% è teoricamente impossibile, per cui spesso è limitato a 99). Un valore inferiore al 90-92% è considerato troppo basso e richiede il completamento esterno dell'ossigeno, come ad esempio capita a molti pazienti affetti dal COVID-19 che sviluppano sintomi gravi.
L'SpO2 si misura con uno strumento chiamato saturimetro o pulsiossimetro, come quello in figura 2.14, dalla forma simile a quella di una molletta che si "pinza" al dito. Possiede sensori che emettono raggi luminosi di lunghezze d'onda diverse fra loro e comunicano con una fotocellula; normalmente vi è un display su cui viene mostrata la percentuale (alcuni misurano anche i battiti). Dal punto di vista pratico, l'SpO2 si ottiene tramite dei calcoli matematici a partire dal grafico del PPG.

Figura 2.14: Un saturimetro

2.5 Pressione arteriosa

La pressione arteriosa è la pressione che il cuore esercita per far circolare il sangue nel corpo; si misura in millimetri di mercurio (mmHg). Il valore della pressione è dato da due numeri:

- pressione arteriosa sistolica: si misura al momento in cui il cuore si contrae e pompa il sangue nelle arterie;
- pressione arteriosa diastolica: si misura tra due contrazioni, mentre il cuore si rilassa e si riempie di sangue.

Il valore della pressione varia normalmente nel corso della giornata: aumenta con lo sforzo, le emozioni, il freddo o il dolore, mentre diminuisce con il riposo e il sonno. Può essere molto variabile tra le persone, ma in condizioni normali per la sistolica si riscontrano valori compresi tra i 100 e i 140 mmHg, mentre per la diastolica si sta tra i 60 e i 90 mmHg. La pressione sistolica è sempre maggiore di quella diastolica di almeno una ventina di punti. I soggetti che non stanno in questi intervalli vengono detti ipertesi, se li superano, o ipotesi, se stanno al di sotto.
La tabella 2.1 riassume tutti i range di valori di pressione possibili.

Categoria	Sistolica	Diastolica
Ipotensione	<100	<60
Ottimale	<120	<80
Normale	120-129	80-84
Normale-alta	130-139	85-89
Ipertensione di grado 1	140-159	90-99
Ipertensione di grado 2	160-179	100-109
Ipertensione di grado 3	>180	>110
Ipertensione sistolica isolata	>140	<90

Tabella 2.1: Valori pressione sistolica e diastolica divisi per categoria

Per mantenere la pressione arteriosa a valori ottimali, è importante avere uno stile di vita sano che preveda un'alimentazione equilibrata, attività fisica, niente fumo e un peso adeguato.
Per poter misurare la pressione in maniera esatta, bisognerebbe ricorrere a metodi invasivi che prevedono la presenza di un catetare nell'arteria; tuttavia, è una pratica applicata solo a pazienti ricoverati con particolari patologie. Nella norma, si ricorre alla misurazione non invasiva utilizzando uno strumento chiamato sfigmomanometro. La tipologia più utilizzata ad oggi è lo sfigmomanometro elettronico, che

è caratterizzato da una macchinetta con un display e una cuffia da allacciare sul braccio sinistro, come quello in figura 2.15. Il prezzo di acquisto è alla portata di tutti ed è estremamente semplice da utilizzare.

Figura 2.15: Uno sfigmomanometro

Capitolo 3

Ambiente Android e protocollo Bluetooth

In questo capitolo si vuole fare una panoramica di quelli che sono il sistema operativo Android e il protocollo Bluetooth, illustrando tutte le principali caratteristiche e i concetti chiave che saranno utili per comprendere i criteri con cui sono state sviluppate le applicazioni EcgWatch e soprattutto PulsEcg.

3.1 Il mondo Android

Figura 3.1: Il robottino verde simbolo di Android

Android è il sistema operativo mobile più diffuso al mondo, sviluppato da Google e basato su kernel Linux. Quello in figura 3.1 è l'iconico logo rappresentato da un robottino verde. Android è disponibile per smartphone, tablet, TV box (Android

TV) e smartwatch (Wear OS); tramite Android Auto, è possibile interfacciarsi con le principali funzionalità (telefono, mappe, musica e molto altro) dello smartphone Android dallo schermo della propria automobile.
Il codice sorgente è open source, e questo fa sì che i produttori di smartphone (ma anche semplici appassionati) possano apportare delle modifiche e dare vita a firmware (cioè un programma in grado di interporsi tra l'hardware e il sistema operativo) personalizzati. Infatti, se si vanno a leggere le specifiche di uno smartphone con Android, è riportato il nome del sistema dato dal produttore. Le modifiche possono essere sia dal punto di vista dell'interfaccia utente, sia dal punto di vista dell'implementazione di basso livello, aggiungendo/modificando funzionalità e impattando quindi su prestazioni ed esperienza d'uso. Questo chiaramente comporta vantaggi e svantaggi: la possibilità di modificare il codice fa sì che ogni modello di smartphone Android sia singolare con le proprie caratteristiche, offrendo agli utenti un'ampia scelta; inoltre, un firmware ottimizzato a dovere permette di ottenere prestazioni elevate anche in presenza di un hardware mediocre. Tuttavia, la differenza tra le implementazioni genera non pochi problemi agli sviluppatori: spesso capita che la stessa applicazione non funzioni ugualmente su tutti gli smartphone/tablet, e per ovviare a ciò è necessario creare soluzioni dedicate e workaround per garantire a tutti gli utenti una buona esperienza.
La maggior parte dei sistemi Android è in grado di ricevere aggiornamenti via OTA (over-the-air), che possono essere sia avanzamenti della versione di sistema (attualmente siamo alla 12), sia aggiornamenti minori e/o di sicurezza. Anche in questo caso, il supporto software è strettamente demandato al produttore del dispositivo: normalmente, per i prodotti di fascia alta, vengono garantiti aggiornamenti per almeno 2 anni e almeno 1 avanzamento di versione di Android.

3.2 Android RunTime

3.2.1 Dalvik

Il sistema Android è progettato per girare le applicazioni su macchina virtuale. Fino alla versione 4.4, la VM utilizzata si chiamava Dalvik, sviluppata da un dipendente Google e ottimizzata per dispositivi mobili. Essa si basa sul paradigma JIT (just-in time) che prevede una compilazione in tempo reale delle app: in fase di installazione l'app viene compilata parzialmente, e il resto del lavoro fatto ad ogni apertura. La Dalvik viene spesso associata a una JVM, anche se di fatto non funziona con bytecode Java: infatti, quest'ultimo viene convertito in un set di istruzioni compatibile.

3.2.2 ART

A partire da Android 5.0, la Dalvik VM è stata sostituita con la Android RunTime (ART), introducendo una novità fondamentale, cioè il passaggio dal paradigma JIT alla tecnologia AOT (ahead-of-time), cioè compilazione completa in fase di installazione. ART è retrocompatibile.
È chiaro come la seconda soluzione porti notevoli vantaggi in termini di prestazioni e gestione delle risorse: infatti, ART è fino al 59% più performante in relazione al suo predecessore [13]. La presenza di una VM comporta vantaggi in termini di sicurezza e compatibilità, ma rimane sempre il "tallone di Achille" delle prestazioni, perché per ottenere prestazioni molto elevate occorre inevitabilmente un hardware molto potente.
Ogni applicazione ha un suo processo dedicato con una propria istanza della VM.

3.3 Sviluppo Android

3.3.1 SDK

Le applicazioni Android sono normalmente scritte in Java, anche se dal 2019 il nuovo linguaggio ufficiale è il Kotlin (logo in figura 3.2). Quest'ultimo è open source, sviluppato dall'azienda JetBrains, ed è completamente interoperabile con Java, in quanto produce lo stesso bytecode; infatti, in uno stesso progetto possono coesistere classi Java e classi Kotlin, e chiamarsi vicendevolmente. Kotlin si propone di superare tutte le limitazioni di Java, come ad esempio gestione delle eccezioni o dichiarazione di costanti/variabili, dando comunque agli sviluppatori una buona ragione per migrare dal Java vista la totale compatibilità. [14]

Figura 3.2: Il logo di Kotlin

Per sviluppare in Android, a prescindere dal linguaggio, occorre un SDK (Software Development Kit), ovvero uno strumento comprendente tutto l'occorrente per la realizzazione di un'app: librerie, API, un emulatore, un debugger,

documentazione, tutorial e stringhe di esempio.

3.3.2 NDK

Per gli sviluppatori più temerari, Google mette a disposizione l'NDK (Native Development Kit): anche questo è un toolset completo di tutto, ma che dà la possibilità di sviluppare in linguaggio nativo, quindi C/C++. Il vantaggio di questa soluzione è il poter sfruttare al 100% le risorse hardware, superando le limitazioni di prestazioni dovute alla VM.

3.3.3 Android Studio

L'IDE ufficiale a supporto dello sviluppo Android si chiama Android Studio, sviluppato da JetBrains (come Kotlin), e integra tutto il necessario per fornire al programmatore uno strumento completo, ricco di funzionalità e semplice da usare. Da Android Studio è infatti possibile scaricare e installare in pochi click SDK, NDK e plugin vari, disegnare efficacemente l'interfaccia grafica delle varie schermate, importare immagini e molto altro.

3.3.4 Cross-platform development

Sebbene lo sviluppo di app native sia tendenzialmente preferibile, esistono molti framework che permettono lo sviluppo di app cross-platform (cioè Android e iOS), quali React Native, Xamarin e Flutter.

3.3.5 Gestione dei permessi

I dispositivi Android possiedono molti componenti hardware, come fotocamera, microfono e GPS, e per il loro utilizzo è necessario richiedere all'utente il permesso esplicito. I permessi sono anche necessari quando si vuole leggere/scrivere dati nella memoria di massa. Le funzionalità di un'applicazione sono incapsulate nel *manfiest*, un file xml che contiene l'elenco dei componenti e dei permessi richiesti.

3.3.6 Ciclo di vita di un'applicazione

Un'applicazione è costituita da uno o più componenti, i quali possono essere;

- Activity: componente essenziale che possiede una sua interfaccia grafica e può eseguire task nell'applicazione;
- Service: non ha un'interfaccia in quanto gira in backgroung, e serve per eseguire task lunghi senza bloccare l'utilizzo del dispositivo;

- Broadcast Receiver: serve per rimanere in attesa di un messaggio particolare proveniente dall'applicazione o dal sistema ed eseguire un'operazione;
- Content Provider; permette la gestione dei dati dell'applicazione corrente interagendo con le altre applicazioni.

La figura 3.3 schematizza quanto appena spiegato.

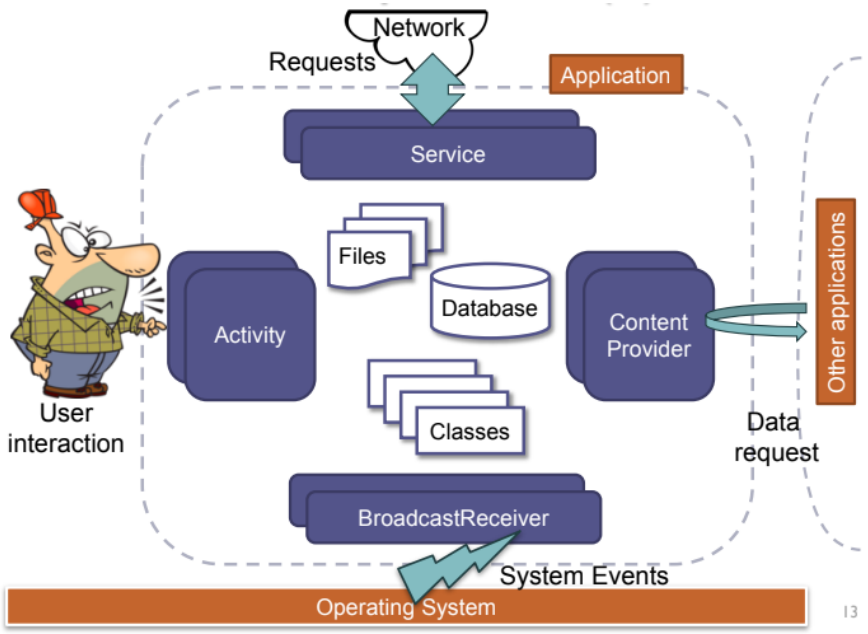


Figura 3.3: Schema componenti di un'app Android

Il sistema operativo crea il processo per l'applicazione quando richiesto sulla base delle informazioni contenute nel manifest. Nel caso l'avvio avvenga tramite click esplicito dell'utente sulla corrispondente icona, viene creato un *Intent* che incapsula le informazioni, e viene mandato al processo di sistema chiamato *Zygote* che crea un nuovo processo tramite una fork; il thread principale (*main thread*) del nuovo processo istanzia l'app avviando l'activity segnata come principale nel manifest. Ogni componente può essere presente zero o più volte, in base allo scopo e alla complessità dell'applicazione, ad eccezione dell'activity, che deve essercene almeno una. Per essere più precisi, le linee guida suggeriscono una Single Activity Application, cioè un'unica activity in tutto il progetto. Essendo l'activity l'unico componente in grado di mostrare un'interfaccia grafica (in generale, qualsiasi operazione grafica deve essere fatta sul main thread), come gestire un certo numero di schermate diverse? Tramite i fragment, ovvero classi a cui è associata un'interfaccia e che vengono affiliate ad una activity. I vantaggi principali

dell'utilizzo dei fragment sono la modularità, la riusabilità e l'adattamento, cioè nella stessa schermata possono esserci più fragment (o più istanze dello stesso fragment) posti in qualsiasi maniera e ciascuno atto a svolgere un compito preciso.

3.3.7 Layout e views

Un layout è un oggetto grafico in grado di incapsulare delle viste e disporle in un certo modo. I tipi sono: Linear Layout, Relative Layout, Grid Layout, Constraint Layout e Frame Layout. Non ce n'è uno migliore rispetto ad un altro, semplicemente permettono disposizioni differenti delle viste figlie e vanno utilizzati opportunamente.
È possibile creare le viste tramite XML oppure tramite codice; si può anche utilizzare un approccio misto (che spesso risulta il più conveniente). Con *vista* si intendono la classe *View* e tutte le sue sottoclassi, che insieme generano una vera e propria gerarchia mostrata in figura 3.4. Figlie dirette di *View* sono la classe *ViewGroup*, da cui ereditano tutti i layout, *AnalogClock* (deprecato), *ProgressBar*, che serve per mostrare un indicatore di progresso, *TextView*, che serve per mostrare un testo a schermo, e *ImageView*, che serve per mostrare un'immagine. Le viste sono quindi gli elementi grafici con cui l'utente può in qualche modo interagire.

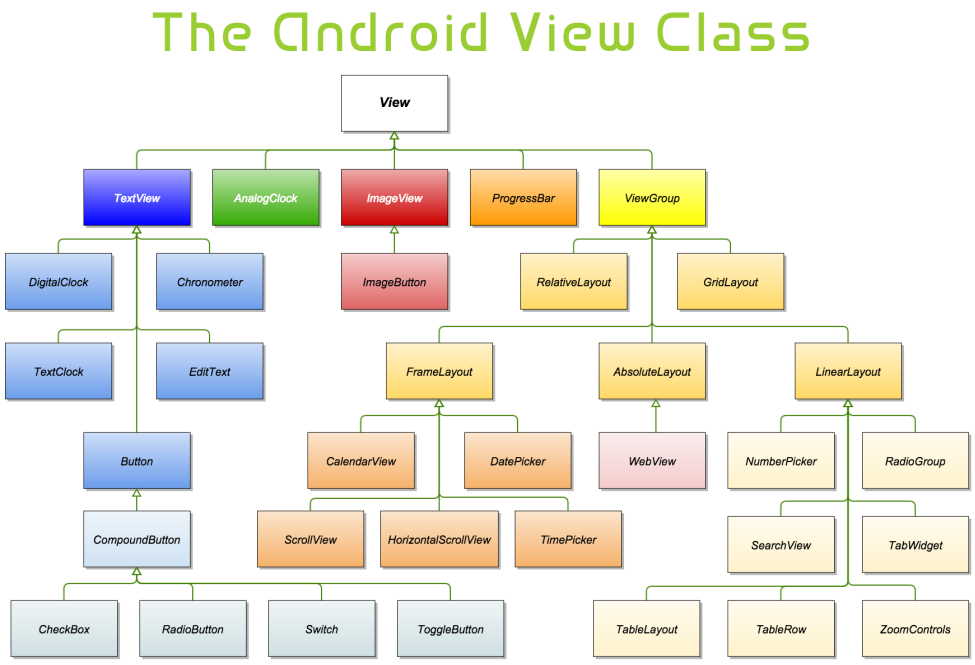


Figura 3.4: Gerarchia delle viste in Android

3.4 UI/UX

Nella realizzazione di un'applicazione bisogna tenere conto di un aspetto fondamentale, ovvero chi andrà ad utilizzarla. Nella maggior parte dei casi, l'utente finale è un utente inesperto, che ha bisogno di interfacciarsi con qualcosa di estremamente intuitivo. Anche per le applicazioni realizzate in questo progetto di tesi gli utilizzatori pensiamo essere principalmente persone non più giovani, che quindi a maggior ragione necessitano di qualcosa di facile. L'interfaccia deve cercare di soddisfare le aspettative dell'utente, evitando di mostrare schermate complicate, suddividendo le operazioni richieste in semplici task e dandone la giusta priorità; può essere utile introdurre un tutorial che la prima volta che si apre l'applicazione guidi passo passo nelle funzionalità.

3.4.1 Material Design

Nel giugno del 2014, Google presenta un nuovo modo di pensare la grafica delle applicazioni mobile e non solo: Material Design, che come si evince dal nome la parola chiave è appunto *Material*, cioè dare al materiale digitale le sembianze di quello reale, con superfici fisiche, bordi e ombre. Le caratteristiche fondamentali del material design sono 4 [15]:

1. superfici tattili (*quantum paper*): ogni elemento di un'interfaccia grafica diventa una superficie reale e tangibile, acquisendo delle ombre realistiche perché posto in modo gerarchico rispetto agli alti elementi;
2. animazioni intelligenti: le animazioni devono essere una conseguenza di un'interazione dell'utente, e deve partire esattamente dal punto in cui viene premuto lo schermo;
3. adattabilità: l'interfaccia deve adattarsi in modo dinamico ed automatico su ogni dispositivo;
4. inchiostro digitale: Google ha indicato un solo font, il Roboto, del quale però esistono diversi stili.

La figura 3.5 illustra un esempio di due schermate realizzate con Material Design.

Sulla documentazione ufficiale vengono anche suggerite linee guida per il tema dell'applicazione, quindi stabilire delle forme per le superfici e combinare correttamente i colori, da selezionare da delle palette standard. Nello specifico di Android, importando material design si possono quindi utilizzare tutti gli oggetti grafici della libreria che di fatto ereditano dalle viste standard di Android (ad esempio, la *TextView* diventa *MaterialTextView*), ma hanno degli attributi aggiuntivi; inoltre, è possibile definire degli stili "di default", nel senso che si possono stabilire attributi standard per una specifica view in tutto il progetto.

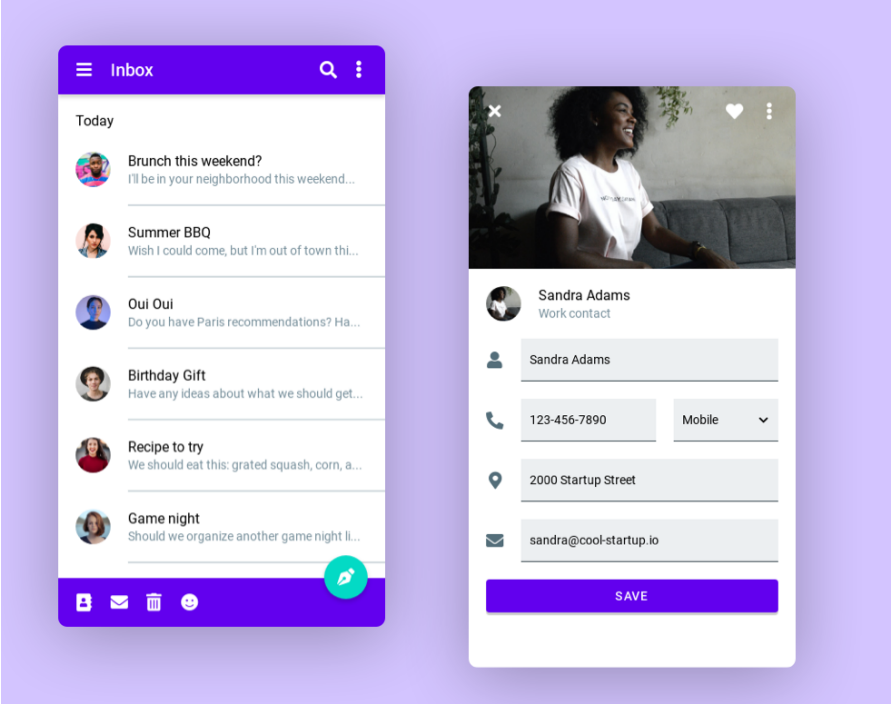


Figura 3.5: Interfaccia realizzata con Material Design

3.4.2 UX design

La progettazione dell'esperienza utente prevede di seguire un modello che si basa su 5 step, o meglio, 5 "piani" sovrapposti [16]:

- Strategia: risponde alla domanda "perché lo stai facendo?", cioè analizza le necessità e gli obiettivi dell'utente;
- Scopo: risponde alla domanda "che cosa farà?", cioè include tutte le funzionalità dell'app;
- Struttura: definisce l'organizzazione della struttura logica;
- Scheletro: definisce lo schema inteso come la posizione degli elementi grafici sullo schermo;
- Superficie: include tutto quello che l'utente può vedere e con cui può interagire.

3.5 Servizi Google

I servizi di Google Play (Google Play Services) sono un insieme di API che offrono agli sviluppatori e agli utenti una migliore esperienza. Sono indispensabili per il

funzionamento di moltissime app, e talvolta migliorano i servizi base presenti tra la API standard di Android. Tra i servizi principali forniti citiamo Google Maps API, che permette di sfruttare le funzionalità di Google Maps, Location API, che permette di ottenere la geolocalizzazione, e Google Drive API, per l'interfacciamento con il cloud di Google. I servizi Google sono disponibili di default su tutti gli smartphone Android, ad eccezione dei Huawei a seguito del ban del governo statunitense [17].

3.6 Librerie

Le librerie sono un insieme di classi e di funzioni che possono essere integrate all'interno di un'app (più in generale, di un software) che permettono allo sviluppatore di avere a disposizione del codice già scritto da qualcun altro per svolgere specifiche azioni.
In un progetto Android alcune librerie sono già presenti di default (AndroidX [18]) che contengono tutto l'indispensabile per la creazione di un'interfaccia. Le librerie vengono dichiarate nel file *build.gradle*.
Per la realizzazione delle applicazioni EcgWatch e PulsEcg è stato necessario integrare le seguenti librerie:

- Gson (solo PulsEcg): libreria di Google per serializzazione/deserializzazione di file json [19];
- CircularImageView: libreria per creare facilmente ImageView rotonde [20];
- Play Services Location: il servizio di Google dedicato alla geolocalizzazione [21];
- Tensowflow lite (solo PulsEcg): libreria per l'integrazione e l'interpretazione di file tflite (vedere 5) [22];
- MPAndroidChart: la libreria essenziale per lo scopo delle applicazioni, in quanto serve per mostrare i grafici delle acquisizioni [23].

3.6.1 MPAndroidChart

L'integrazione di questa libreria ha richiesto uno sforzo maggiore dal momento che sono state apportate modifiche al codice stesso della libreria al fine di adattarla alle esigenze delle applicazioni EcgWatch e PulsEcg.
Questa libreria è in grado di *renderizzare* qualsiasi tipo di grafico (torta, candele, istogramma, ecc.), e nel nostro caso si parla di elettrocardiogramma e fotopletismografia (solo PulsEcg). La difficoltà è stata adattare la libreria in modo tale da ottenere la carta millimetrata come mostrato in figura: quadratini precisi per

garantire leggibilità e valenza medica, etichette sugli assi, spessori diversi per la griglia in corrispondenza delle etichette e unità di misura.
La libreria non è importata nel file *build.gradle* come le altre, ma è copiata localmente per permettere la modifica del codice.

3.7 Bluetooth

Il bluetooth è un protocollo radio per la trasmissione di dati nato a fine anni Novanta. Sfrutta la frequenza ISM 2.4GHz, e nasce con l'obiettivo di ottenere bassi consumi, un corto raggio di azione e dispositivi compatibili economici. Un dispositivo bluetooth può effettuare la scansione per trovare uno o più dispositivi che si trovano nello stato *discoverable*. Successivamente avviene il *pairing*, dove il dispositivo che scansiona invia una richiesta di accoppiamento al dispositivo scoperto; quest'ultimo può accettare la richiesta e permettere il *bonding*, cioè lo scambio delle chiavi di sicurezza, per poi procedere alla connessione vera e propria con scambio di informazioni. La connessione tra due o più dispositivi prende il nome di *piconet*. In una piconet, un dispositivo può essere *master*, se si occupa della sincronizzazione degli altri dispositivi, o *slave*. Un dispositivo bluetooth può trovarsi in 2 stati: connesso, quindi coinvolto nelle attività di scambio dati, o in standby, quindi non connesso o non coinvolto in scambio dati (rimane comunque in ascolto per eventuali messaggi dal master della piconet). L'architettura del bluetooth è costituita da uno stack, cioè una struttura a livelli, in cui ciascun livello di occupa di un compito preciso. Il protocollo ha subito diversi aggiornamenti negli anni, e attualmente si trova nella versione 5.1. La versione di nostro interesse è la 2.0, che è la prima ad integrare l'*Enhanced Data Rate* (EDR) con velocità di trasmissione fino a 3 Mbps, crittografia sui canali, e potenza dei segnali radio ridotta. Su Android, l'interazione con il bluetooth può avvenire grazie all'utilizzo di API standard, oppure integrando delle librerie specifiche con una diversa gestione dello stack.

3.8 Bluetooth Low Energy

Il BLE è una tecnologia pensata per il risparmio energetico e costo ridotto per i dispositivi. Il BLE utilizza le stesse frequenze radio, e la prima versione del bluetooth a consentirne l'implementazione è la 4.0; tuttavia, il BLE non è compatibile con il bluetooth "classico". Gli attuali dispositivi mobili vengono solitamente rilasciati con supporto hardware e software sia per il Bluetooth classico che per il Bluetooth Low Energy. Il BLE prevede un'architettura diversa e ruoli diversi dei dispositivi, i quali utilizzano tutti il Profilo di Attributo Generico (GATT). Il GATT può essere:

- Client: invia comandi e riceve risposte;

- Server: riceve comandi e invia risposte;
- Characteristic: dato scambiato tra client e server, può essere in lettura o scrittura;
- Service: insieme di caratteristiche correlate;
- Descriptor: informazioni aggiuntive alla caratteristica.

Un GATT client può essere connesso ad un solo GATT server alla volta. Per quanto concerne Android, il BLE è disponibile a partire dalla versione 4.3 del sistema, e anche qua vi sono delle API standard per la gestione. La cosa interessante da osservare è che su Android il BLE necessita del GPS attivo per il corretto funzionamento. Vi è anche una sorta di aneddoto legato a codesta questione, in quanto inizialmente gli sviluppatori lo segnalarono come una issue; Google rispose che è il comportamento voluto e che non c'è la volontà di fixarlo [24], perché sono richiesti i permessi di accesso alla posizione (anche se di fatto non viene effettuata alcuna geolocalizzazione).

Capitolo 4

EcgWatch

In questo capitolo si parlerà del dispositivo EcgWatch e dell'omonima applicazione Android per l'interfacciamento con esso, descrivendone i dettagli implementativi.

4.1 Panoramica del dispositivo

Il dispositivo EcgWatch si presenta come in figura 4.1, uno scatolotto di dimensioni contenute allacciabile al polso con un cinturino. Sulla sua superficie sono presenti 2 elettrodi, che serviranno per l'acquisizione del segnale elettrocardiografico: in particolare, uno degli elettrodi rimane a contatto con un polso del paziente, mentre sull'altro è possibile appoggiare il dito indice dell'altra mano.

Figura 4.1: Il dispositivo EcgWatch

Dal punto di vista hardware, EcgWatch presenta i seguenti componenti [25]:

- un microcontrollore Texas Instrument (MSP430G2955) che tiene conto dei vincoli quali una quantità sufficiente di memoria per permettere la memorizzazione dell'acquisizione (in gergo *bufferizzazione* dei campioni), a cui è collegato il modulo Bluetooth e un ADC;
- due elettrodi di argento, perché riutilizzabili e per il loro deterioramento dall'uso quasi assente;
- un modulo Bluetooth 2.0 F2M03GLA, che supporta RFCOMM e permette la comunicazione via UART, GPIO o USB;
- una batteria al litio, ricaricabile e in grado di fornire l'alto picco di corrente richiesto dal modulo Bluetooth trasmittente.

4.2 Applicazione Android

L'applicazione EcgWatch è indispensabile per controllare il dispositivo dal momento che quest'ultimo non presenta alcuna interfaccia per l'utente. Collegandosi tramite bluetooth, è possibile effettuare delle scansioni e ottenere l'elettrocardiogramma, che si può salvare per essere consultato e condiviso anche successivamente.
L'applicazione memorizza in locale le misurazioni e alcune informazioni basilari dell'utente, come nome e cognome, e non vengono in alcun modo condivise con terzi o salvate su database online.
L'app è disponibile solo per Android e scaricabile dal Play Store.

4.2.1 Permessi

È indispensabile che l'utente accetti il permesso per la lettura/scrittura sul file system, che viene richiesto non appena l'app viene aperta la prima volta; se rifiutato, l'app si chiude.
Se si vuole includere la posizione nella condivisione di una misurazione (vedere paragrafo 4.2.6), è necessario accettare i permessi per la geolocalizzazione.

4.2.2 Struttura

L'applicazione è composta di 8 activity, una per ciascuna schermata visualizzabile, in un certo senso in disallineamento con le linee guida di app single activity, ma il corretto funzionamento dell'app non ha mai incentivato modifiche strutturali.
Sono presenti altre classi di supporto, alcune delle quali verranno descritte nei paragrafi successivi, che contengono metodi e costanti utili e indipendenti dalle

activity.

4.2.3 Interfacce ed esperienza utente

Alla prima apertura, l'applicazione presenta un breve tutorial (che è possibile saltare ed è riconsultabile successivamente), la FirstTimeActivity, dove vengono illustrate le principali funzionalità.
Successivamente, tramite la FirstConfigurationActivity vengono richiesti alcuni dati basilari del paziente, quali nome, cognome, email, email del dottore (opzionale) e braccio preferito su cui indossare il dispositivo (figura 4.2).

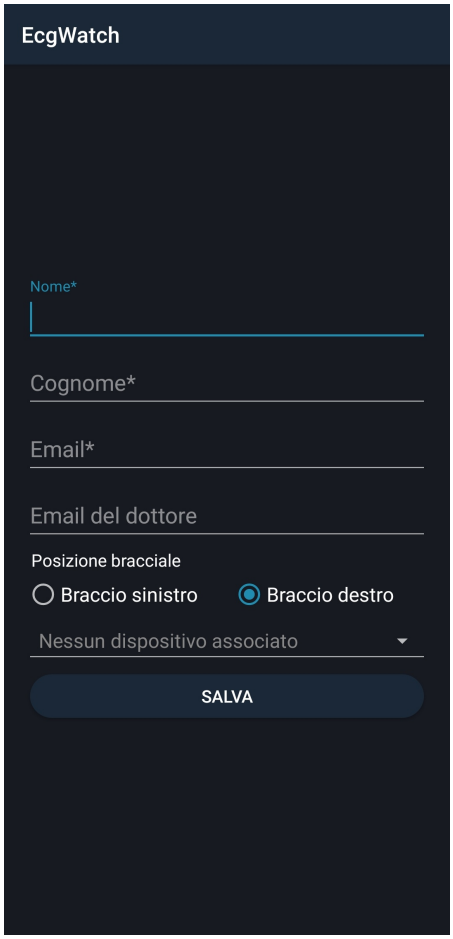


Figura 4.2: La schermata di configurazione iniziale di EcgWatch

Premendo sul tasto "SALVA", ci si ritrova nella schermata principale, cioè la MainActivity, mostrata in figura 4.3, dove l'utente ha a disposizione l'accesso a

tutte le funzionalità in maniera intuitiva. In alto a destra, i pulsanti per tutorial, impostazioni e chiusura app; al centro dello schermo, le card per collegamento con il dispositivo, acquisizione ECG e archivio delle acquisizioni.

Figura 4.3: La schermata principale di EcgWatch

Impostazioni

Nella schermata in figura 4.4, che corrisponde alla SettingActivity, si possono modificare tutte le informazioni personali inserite precedentemente; inoltre, è possibile impostare l'autosalvataggio delle acquisizioni, la condivisione della posizione e la durata delle acquisizioni (10 o 20 secondi).
Al fondo, c'è la versione dell'app installata.

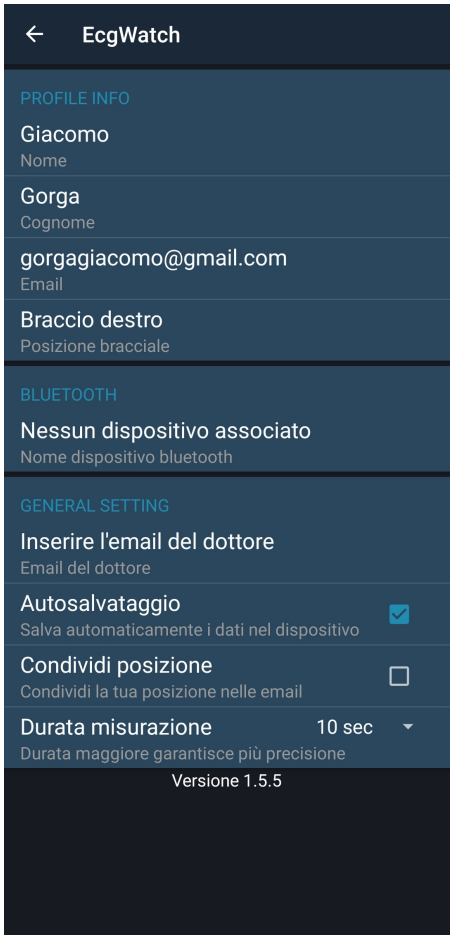


Figura 4.4: La schermata delle impostazioni di EcgWatch

Connessione al dispositivo

Come già detto, la connessione al dispositivo avviene tramite bluetooth. Premendo sull'apposito tasto, si aprirà una dialog come in figura **??** con l'elenco degli EcgWatch accoppiati; se vuoto (o se non è presente il dispositivo d'interesse), premendo su "ASSOCIA UN DISPOSITIVO" si viene rediretti alle impostazioni bluetooth del telefono, ove è possibile scansionare ed accoppiare.

Tornando all'app, nell'elenco nella dialog ora comparirà il dispositivo, e con un tap si esegue la connessione: in particolare, si tenta di stabilire una socket bluetooth con lo smartphone sulla base dello standard RFCOMM. Se la creazione del socket viene eseguita senza eccezioni, vengono salvate tramite la classe java *SingletonSocket* gli oggetti *BluetoothSocket* e *BluetoothDevice*, che corrispondono rispettivamente alla socket appena creata e al dispositivo [26]. Infine, tramite il metodo *BluetoothSocket#connect* si va ad effettuare la connessione vera e propria.

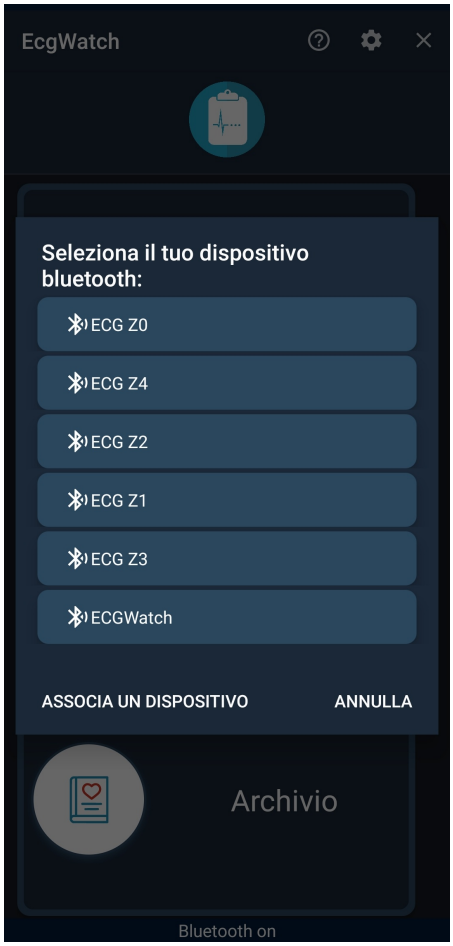


Figura 4.5: La dialog con i dispositivi accoppiati in EcgWatch

Se la connessione va a buon fine, la dialog si chiude e sarà visibile il nome del dispositivo connesso.
Per capire lo stato di connessione (connesso/disconnesso) e, prima ancora, lo stato del modulo Bluetooth del telefono (acceso/spento) si utilizza un BroadcastReceiver, che si attiva tramite la callback *onReceive*, andando a discriminare opportunamente il tipo di messaggio ricevuto, quindi capire se proviene dall'hardware del telefono (*BluetoothAdapter*) o dal dispositivo EcgWatch (*BluetoothDevice*).

Acquisizione

Se si è connessi ad un dispositivo, è possibile effettuare una scansione elettrocardiografica. Con un tap sulla card "Acquisisci" si apre la TakeEcgActivity: una animazione (non reale) e un conto alla rovescia saranno visibili sullo schermo, e

non appena terminato si aprirà una nuova schermata analoga a quella in figura 4.6, la ShowEcgActivity, mostrando l'elettrocardiogramma, i battiti, e un avviso di allerta nel caso si sia riscontrata una possibile fibrillazione atriale.

Figura 4.6: Esempio di elettrocardiogramma in EcgWatch

Le sbarrette in rosso indicano i picchi R. Il grafico si può scorrere e zoomare. Dal punto di vista implementativo, la libreria utilizzata è la precedentemente descritta MPAndroidChart; di seguito, una porzione del codice per ottenere il risultato illustrato:

```
1   LineChart lineChart = findViewById(R.id.chartToPdf);
2
3   lineChart.getAxisLeft().setTextColor(isPdf ? Color.BLACK : Color.
    WHITE);
4   lineChart.getAxisLeft().setAxisMinimum(0);
5   lineChart.getAxisLeft().setAxisMaximum(3);
6
7   lineChart.getAxisRight().setTextColor(isPdf ? Color.BLACK : Color
    .WHITE);
8   lineChart.getAxisRight().setAxisMinimum(0);
9   lineChart.getAxisRight().setAxisMaximum(3);
10
11  lineChart.getXAxis().setGridColor(isPdf ? Color.RED : Color.GRAY)
    ;
12  lineChart.getAxisRight().setGridColor(isPdf ? Color.RED : Color.
    GRAY);
13  lineChart.getAxisLeft().setGridColor(isPdf ? Color.RED : Color.
    GRAY);
14
15  lineChart.setDrawBorders(true);
16  lineChart.setBorderColor(isPdf ? Color.BLACK : Color.WHITE);
17  lineChart.getViewPortHandler().setMinMaxScaleY(1f, 25f);
18
19  lineChart.setScaleXEnabled(true);
```

```
20      lineChart.setScaleYEnabled(true);
21      lineChart.enableScroll();
22      lineChart.setDragEnabled(true);
23
24      lineChart.getXAxis().setTextColor(isPdf ? Color.BLACK : Color.
    WHITE);
25      lineChart.getXAxis().setGranularity(40);
26
27      lineChart.getAxisRight().setLabelCount(30);
28      lineChart.getAxisLeft().setLabelCount(30);
29      lineChart.getXAxis().setLabelCount(250);
30
31      lineChart.setData(new LineData(sets));
```

LineChart è la classe che corrisponde all'intero grafico, alla cui istanza si impostano tutti i parametri desiderati. Per quanto concerne i campioni in input (quelli passati tramite il metodo *LineChart#setData*), vengono creati 2 set diversi, uno per i picchi e uno per l'ecg, poi aggiunti alla stessa lista di *ILineDataSet* (un oggetto della libreria), e infine renderizzati insieme sullo stesso grafico; *isPdf* è un booleano per discriminare la generazione del grafico in app o su PDF (vedere 4.2.6).
Tornando alla schermata mostrata in figura, in altro a destra i 4 pulsanti servono rispettivamente per generare il PDF, condividere il file ecg, eliminare la misurazione e leggere alcune informazioni generiche sull'interpretazione dell'elettrocardiogramma.

Archivio

Nell'archivio (EcgActivity, figura 4.7) i file delle misurazioni sono elencati dal più recente al meno recente e separati per mese. Ogni file quando viene salvato è nominato così: <nome> <cognome> <data e ora>.ecg, dove ".ecg" rappresenta un'estensione personalizzata, ma di fatto si tratta di un file testuale contenente i campioni dell'elettrocardiogramma.
Nel menu in altro a destra vi sono 2 pulsanti: uno con la ricerca testuale per nome del file, e un altro per l'importazione di file ecg, utile quindi per visualizzare misurazioni altrui condivise ad esempio tramite email o app di messaggistica. Cliccando su ciascuna entry, è possibile visualizzare la ShowEcgActivity.
Tenendo invece premuto su una entry, si attiva la modalità di selezione multipla. Il menu avrà ora 3 pulsanti: uno per la condivisione del file ecg (vedere 4.2.6), uno per la cancellazione dei file selezionati, e uno per selezionare contemporaneamente tutti i file. Premendo sul fab (*floating action button*) in basso a destra, si apre una dialog (figura 4.8) dove è possibile filtrare la ricerca dei file per nome paziente, cognome paziente e range di date.

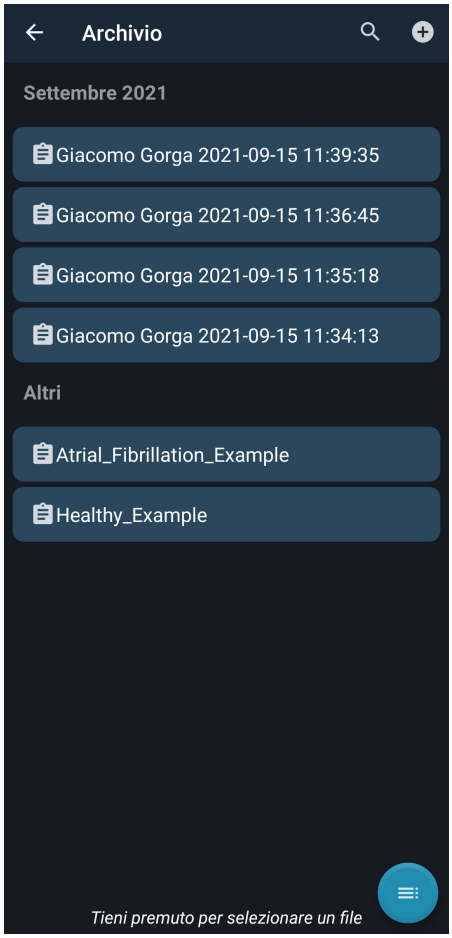


Figura 4.7: La lista dei file nell'archivio di EcgWatch

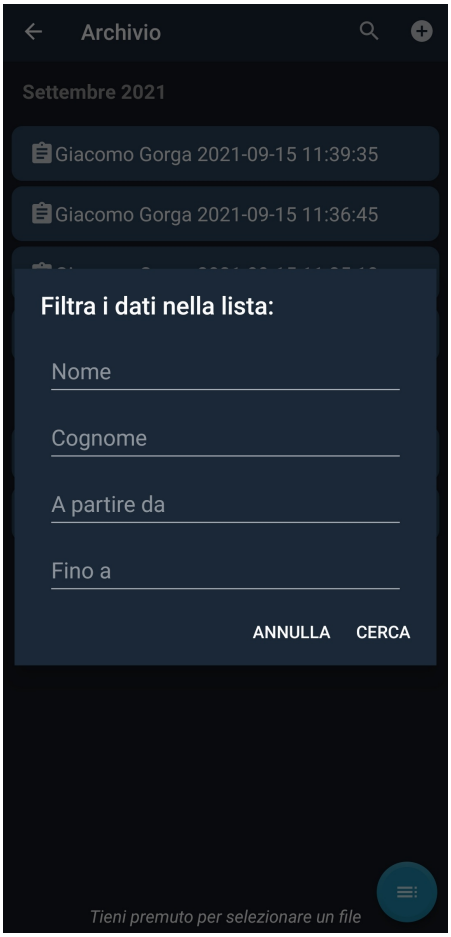


Figura 4.8: La dialog per filtrare i file nell'archivio

4.2.4 Comunicazione con il dispositivo

La classe java *BtCommunication* contiene alcune costanti utili per la comunicazione e una sottoclasse *ConnectedThread* che estende la classe *Thread*, e ne viene creata un'istanza. Il nuovo thread si occupa quindi di inviare al dispositivo (più rigorosamente, esegue un'operazione di *write*) una stringa per indicare di voler iniziare un'acquisizione e quale deve essere la durata. Il dispositivo immagazzina in una memoria tutti i samples della misurazione in corso, e al termine del conto alla rovescia si esegue una operazione di lettura (*read*), sempre inviando una stringa, così l'EcgWatch invia all'app tutti i campioni acquisiti che vengono salvati nel file ecg non filtrati.

4.2.5 Algoritmi per l'elaborazione del segnale

Filtri

Il segnale, quando viene acquisito, è soggetto a rumore, cioè valori imprevisti e indesiderati che si sovrappongono al segnale utile. Il rumore è sempre presente, e può essere più o meno intenso a seconda delle influenze elettromagnetiche del luogo in cui si effettua la misurazione. Per eliminarlo, si ricorre a filtri implementati in app: nella teoria dei segnali, un filtro (digitale) è un insieme di funzioni matematiche su campioni di segnali discreti nel tempo in ingresso che modifica alcuni aspetti del segnale stesso.
Il primo filtro applicato è un filtro per il *bias*, ovvero la tendenza del segnale a deviare dal valore medio. La funzione riceve come parametro un intero *window* che va ad indicare un finestra di valori con cui si applica l'attenuazione: in altre parole, per ciascun elemento del vettore in input la finestra stabilisce un intorno pari a 2**window* su cui fare la media. Naturalmente, una finestra maggiore significa attenuazione maggiore, perché in un intorno grande ci sono più variazioni. Questo filtro viene utilizzato due volte, la prima con una finestra pari a 500 e la seconda pari a 7.
Il secondo filtro è invece un filtro di Butterworth, cioè un filtro passa-basso standard di primo ordine, applicato con una frequenza di 15 Hz.

Determinazione dei picchi

Una volta ottenuto un dataset di valori filtrato (nonché i punti del grafico), si può andare a ricercare i picchi R, che sono indispensabili per il calcolo della frequenza cardiaca. Uno degli algoritmi più utilizzati per il riconoscimento real-time del complesso QRS dal segnale ECG e inizialmente integrato nell'app (in una versione rielaborata) è il Pan-Tompkins. L'algoritmo, basato sulla pendenza (slope), l'ampiezza e la larghezza del complesso QRS, include una serie di filtri e metodi. Il filtro passa banda, ottenuto dalla cascata di un filtro passa basso con frequenza di campionamento 11Hz e da un filtro passa alto con frequenza di campionamento 5Hz, seleziona l'intervallo di frequenze allo scopo di ridurre le componenti estranee al QRS, cioè le altre onde (P, T). Il filtro derivatore evidenzia la rapida variazione che caratterizza il complesso QRS, mentre il filtro quadratore rende il segnale positivo ed enfatizza le componenti del complesso QRS. Infine, il filtro a media mobile (moving average) opera lo *smoothing* (letteralmente, levigatura) del segnale in uscita dal precedente filtro, che potrebbe presentare picchi multipli in corrispondenza del QRS. Dall'output così ottenuto si possono determinare i picchi R verificando i massimi locali.
Nonostante l'efficacia di questo algoritmo sia dimostrata essere oltre il 99% [27], su EcgWatch si trattava di percentuali molto più basse, certamente non accettabili.

Dopo alcune prove di soluzione, quella che ha dato i risultati migliori, nonché al momento adottata, è un algoritmo relativamente banale, che non esegue nessuna operazione di filtraggio, ma soltanto alcune considerazioni e calcoli prettamente matematici.

```
1     public static ArrayList<Integer> findPeakJack(double[] input) {
2         int length = input.length;
3         int k = 0, i;
4         ArrayList<Integer> peaks = new ArrayList<>();
5
6         ArrayList<Integer> localMaxs = new ArrayList<>();    //x
    coordinates of local maxs
7         ArrayList<Integer> localMins = new ArrayList<>();    //x
    coordinates of local mins
8
9         for (i = 1; i < length - 1; i++) {
10            if (input[i] > input[i + 1] && input[i] > input[i - 1])
11                localMaxs.add(i);
12        }
13
14        for (i = 1; i < length - 1; i++) {
15            if (input[i] < input[i + 1] && input[i] < input[i - 1])
16                localMins.add(i);
17        }
18
19        for (int local_max : localMaxs) {
20            //check if local max could be a peak
21            if (local_max < 35 || local_max > length - 35)
22                continue;
23
24            int local_min = -1;
25
26            for (int min : localMins) {
27                if (min <= local_max)
28                    continue;
29
30                if (min > local_max + 160)
31                    break;
32
33                //find local min following current local max
34                if (min <= local_max + 80 && input[local_max] - input
    [min] >= 0.3) {
35                    local_min = min;
36                    break;
37                }
38                else if (min > local_max + 80) {
```

```
39                      int minTmp = local_max + ((min - local_max) / 2);
      //average value between max and min
40                      double slope1 = input[local_max] - input[minTmp];
41                      double slope2 = input[minTmp] - input[min];
42
43                      if (slope1 > slope2 && slope1 >= 0.25) {
44                          local_min = min;
45                          break;
46                      }
47                  }
48              }
49
50              if (local_min != -1) {
51                  if (k > 0 && local_max - peaks.get(k - 1) < 180) {
52                      //peaks cannot be too close each other
53                      double slope_temp = (input[local_max] - input[
    local_max + 40]);
54                      double slope_old = (input[peaks.get(k - 1)] -
    input[peaks.get(k - 1) + 40]);
55
56                      if (slope_temp > slope_old && input[local_min] <
    input[peaks.get(k - 1) + 40]) //if the slope after current local
    max is major than previous, so replace previous peak
57                          peaks.set(k- 1, local_max);
58                  }
59                  else {
60                      //peak valid
61                      peaks.add(local_max);
62                      k++;
63                  }
64              }
65          }
66
67          return peaks;
68      }
```

Il metodo riceve in input l'array contenente i campioni acquisiti già filtrati. Innanzitutto, si trovano tutti i massimi e i minimi locali, poi si itera su tutti i massimi per capire se ciascuno possa essere o meno un picco. L'idea è quella di confrontare il massimo corrente con il minimo seguente, e viene fatto in due modi:

- la distanza R-S è solitamente inferiore a 80 ms, vale a dire 2 mm sulla carta millimetrata, quindi si verifica se nella lista dei minimi è presente un valore tale per cui si trova in un range opportuno di ascisse successive al massimo (entro gli 80 campioni, 1 per millisecondo) e la differenza sull'asse delle ordinate è maggiore o uguale a 0,3 mV;

- per distanze superiori agli 80 ms, ma in ogni caso inferiori ai 160, si fa un confronto con la pendenza tra il primo tratto della discesa, cioè quello tra il massimo locale e il valor medio tra massimo e minimo, e il secondo tratto, cioè tra valor medio e minimo; la pendenza del primo tratto deve essere maggiore e l'ampiezza sulle ordinate di almeno 0,25 mV.

Le casistiche da considerare per confermare o scartare il picco individuato sono:

- Se non ci sono altri picchi, oppure è opportunamente distante dal precedente, allora è un picco;
- Se è vicino ad un altro picco, allora si effettua una verifica sulla pendenza, cioè se la pendenza è maggiore rispetto al picco rilevato in precedenza, allora il picco precedente era falso, e viene sostituito con il corrente.

Il metodo ritorna la lista dei picchi.

Calcolo della frequenza cardiaca e fibrillazione atriale

Esistono alcuni metodi banali per il calcolo immediato dei battiti dato un ECG: basta contare il numero di picchi R presenti in 10 secondi di misurazione e moltiplicare per 6, oppure dividere 300 per il numero di quadratini presenti tra due picchi R sulla carta millimetrata. Questi calcoli ovviamente sono molto approssimativi e soggetti ad errori, pertanto l'algoritmo utilizzato si basa sulla distanza tra i picchi R tenendo conto però della durata effettiva della misurazione e del numero di picchi R rilevati.

```
1    public static int[] heartBeatCalculation(ArrayList<Integer> peaks
     , Context context) {
2        int sum = 0, RR, j, FAbpmCounter = 0, previousBpm = 0;
3        int[] heart_rate = {0, 0};
4
5        for (j = 1; j < peaks.size(); j++) {
6            RR = peaks.get(j) - peaks.get(j - 1);
7            sum += RR;
8
9            if (((HEARTBEAT_CONSTANT / RR) < previousBpm - 5) || ((
     HEARTBEAT_CONSTANT / RR) > previousBpm + 5)) {
10               FAbpmCounter += 1;
11           }
12           previousBpm = HEARTBEAT_CONSTANT / RR;
13       }
14
15       sum = sum / --j;
```

```
16
17          int seconds = LocalData.getSharedPreferenceInt(context,
        LocalData.SECONDS, SECONDS_OF_AQUISITION);
18          if (j < 5 || j > seconds * 4)  // heart rate not possible
        maxBPM = 240
19          {
20              heart_rate[0] = -1;// signaling error
21          } else {
22              heart_rate[0] = HEARTBEAT_CONSTANT / sum;
23
24              //added for FA detection
25              if (FAbpmCounter > (j - FAbpmCounter)) {
26                  heart_rate[1] = 1;
27              }
28              // end FA
29          }
30
31          return heart_rate;
32      }
```

Il codice riportato è una versione ridotta dell'algoritmo utilizzato in app, ma ugualmente funzionante.
Per ciascun picco, si calcola la distanza dal precedente (*peak* è una lista contenente le ascisse dei picchi), e si somma alla variabile *sum*. *HEARTBEAT_COSTANT* è una costante dal valore di 60000, tramite cui si calcola un valore puntuale dei battiti: se la divisione con la distanza RR dell'iterazione corrente fornisce un valore che differisce dal precedente di oltre 5 punti, viene incrementato un contatore (*FAbpmCounter*) che serve per determinare la presenza di una eventuale fibrillazione atriale. Al termine dell'iterazione, si recupera la durata della misurazione dalle shared preferencies e si calcola il battito medio dividendo *HEARTBEAT_COSTANT* con la media delle distanze. Infine, se *FAbpmCounter* è maggiore della differenza tra il numero di picchi e *FAbpmCounter* stesso, si presume la presenza di fibrillazione atriale, perché significa che su oltre la metà dei picchi R vi è una distanza non costante. Il metodo ritorna un array di 2 elementi (4 nella versione completa) contenente la frequenza cardiaca media e la presenza di fibrillazione atriale (1 se vero, 0 se falso).

4.2.6 Altre funzionalità

Condivisione file

Sia dalla ShowEcgActivity che tramite la modalità selezione della EcgActivity, è possibile condividere il file *raw*, cioè il file con estensione *ecg* contenente tutti i samples. L'utilità di questa funzione è, attualmente, prevalentemente per scopi

di debug, ma permette anche ad un utente di visualizzare il proprio elettrocardiogramma su un altro smartphone o tablet.
All'atto pratico, la condivisione avviene creando un intent con all'interno il file, un testo predefinito e un oggetto (utile solo se si seleziona la mail come metodo di condivisione).

PDF

La generazione automatica del PDF inerente ad una misurazione è sicuramente una delle funzionalità più utili, per non dire indispensabili, in quanto il PDF fornisce un riepilogo completo tale da dare alla misurazione un documento leggibile e valutabile da un medico. Vengono infatti riportate in maniera ordinata nome e cognome del paziente, data e durata della misurazione, la geolocalizzazione se richiesta, bpm e stato di salute [1].
La pagina del file è in formato A4 disposta orizzontalmente, e l'elettrocardiogramma è disposto su 1 riga per ogni 10 secondi; in altri termini, per misurazioni da 10 secondi avrà solo 1 riga, per misurazioni da 20 secondi avrà 2 righe.
La figura 4.9 riporta un esempio di PDF per una misurazione di 20 secondi.

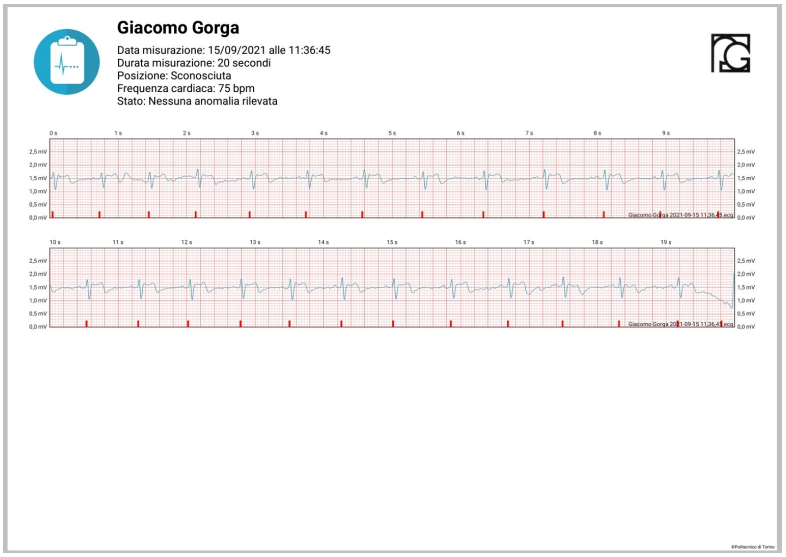


Figura 4.9: Il PDF di una misurazione di 20 secondi

[1]Importante: lo stato di salute indica soltanto se sia stata rilevata una possibile fibrillazione atriale, ma in ogni caso non sostituisce il parere di un medico

Geolocalizzazione

La geolocalizzazione è inseribile sia all'interno del documento PDF, che allegabile testualmente nella condivisione del file .ecg. L'implementazione si trova nella classe Java *LocationClass*, e prevede l'utilizzo dei servizi Google come metodo principale; qualora il dispositivo Android non li integrasse, vengono utilizzate le API standard che sfruttano semplicemente l'hardware a disposizione: è proprio a tal proposito che è stata necessaria l'integrazione dei servizi Google come metodo preferito, perché su smartphone di gamma medio/bassa il sensore GPS possiede una potenza e una precisione inferiori, tali che spesso la posizione non è reperibile o è errata.
A prescindere dal metodo utilizzato, il GPS deve essere abilitato, pertanto nel caso non lo fosse ne viene richiesta l'attivazione come mostrato nelle figure 4.10 e 4.11.

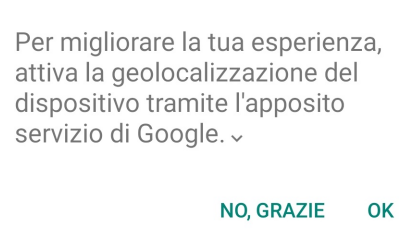


Figura 4.10: La dialog per l'attivazione del GPS con i servizi Google

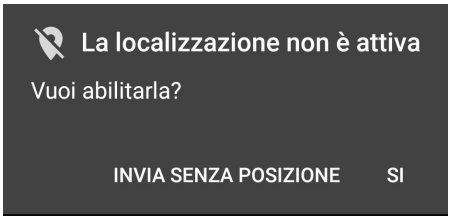


Figura 4.11: La dialog per l'attivazione del GPS senza i servizi Google

Il processo di geolocalizzazione è chiaramente asincrono, cioè la risposta giunge in un tempo non prevedibile (normalmente varia tra i 2 e i 10 secondi); questa risposta è un oggetto di tipo *Location* che viene ricevuto come parametro in una callback.
Per poter fornire dei feedback visivi all'utente e tenere traccia dei vari step da compiere, si utilizza un'interfaccia Java creata ad hoc, chiamata *LocationProcessing*, che mette a disposizione i seguenti metodi:

- onGpsActivationRequired: dalle dialog mostrate in figura, se l'utente risponde positivamente viene invocata questa callback, che riceve in input 2 parametri: un boolean che indica l'utilizzo o meno dei servizi Google, e un oggetto di tipo ApiException; quest'ultimo serve solo in presenza dei servizi e permette l'abilitazione del GPS senza passare dalle impostazioni, viceversa questo parametro è *null* e l'utente viene rediretto alle impostazioni per l'attivazione manuale;

- onSendWithoutPosition: se l'utente rifiuta l'abilitazione del GPS, questa callback permette il proseguo dell'operazione di condivisione interrompendo quella di geolocalizzazione;
- onLocationNotFound: il processo di geolocalizzazione ha un timeout di 60 secondi, cioè se non vengono trovate coordinate entro il tempo limite, viene invocata questa callback e mostrata la dialog in figura 4.12.

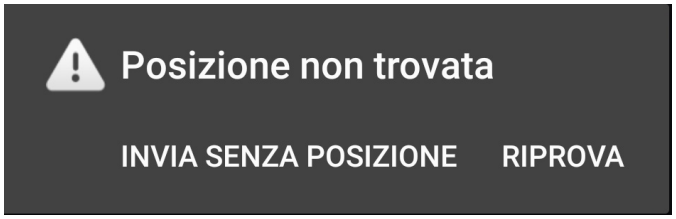


Figura 4.12: Dialog di posizione non trovata

  Premendo su "Riprova", si ricomincia l'operazione da capo; premendo su "Invia senza posizione", viene chiamata la callback onSendWithoutPosition;
- onLocationRetreived: quando giunge la risposta dal satellite sul posizionamento, viene invocata questa callback che riceve come parametro un oggetto di tipo *Location* contenente le coordinate. Affinché siano leggibili da un utente umano, è necessario risolverle in un indirizzo testuale, e viene fatto tramite il *Geocoder* [28]. Anche in questo caso si tratta di un'operazione asincrona, ma che viene risolta tramite una banale tecnica di polling;
- onLocationProcessed: quando la risoluzione delle coordinate in indirizzo termina, viene invocata questa callback che riceve come parametro una stringa (appunto, l'indirizzo appena calcolato). Se la risoluzione non va a buon fine, ci si accontenta delle coordinate, tramite le quali è comunque possibile facilmente ottenere manualmente una posizione di più facile lettura. In ogni caso, si procede con la creazione dell'intent per la condivisione, e la posizione è salvata in una variabile globale.

L'interfaccia java è implementata sia nella EcgActivity che nella ShowEcgActivity, perché per mostrare le dialog all'utente è necessario avere un riferimento preciso della schermata visualizzata.

Capitolo 5

Rete neurale per la pressione arteriosa

5.1 Correlazione tra ECG, PPG e pressione

Nel capitolo 2 sono stati analizzati diversi aspetti dei segnali ECG e PPG, come si possano ricavare dei parametri fondamentali, e introdotto la pressione arteriosa e l'importanza dell'avere valori nella norma.
La pressione è correlata alla velocità dell'onda pulsante (PWV, Pulse Wave Velocity). L'arteria si assuma essere un tubo cilindrico elastico con raggio r e spessore h. L'equazione di Moens–Korteweg mette così in relazione questi parametri:

$$PWV = \sqrt{\frac{Eh}{2r\rho}} \tag{5.1}$$

dove E indica il modulo elastico della parete dell'arteria (modulo di Young) e ρ la densità del sangue.
La pressione e la velocità del sangue nell'arteria sono direttamente proporzionali. Se si immagina di porre sulle estremità dell'arteria due sensori ad una distanza D, si può calcolare il tempo impiegato dal sangue a percorrere tale distanza, chiamato *Pulse Transit Time* (PTT):

$$PTT = \frac{D}{PWV} = D\sqrt{\frac{2r\rho}{Eh}} \tag{5.2}$$

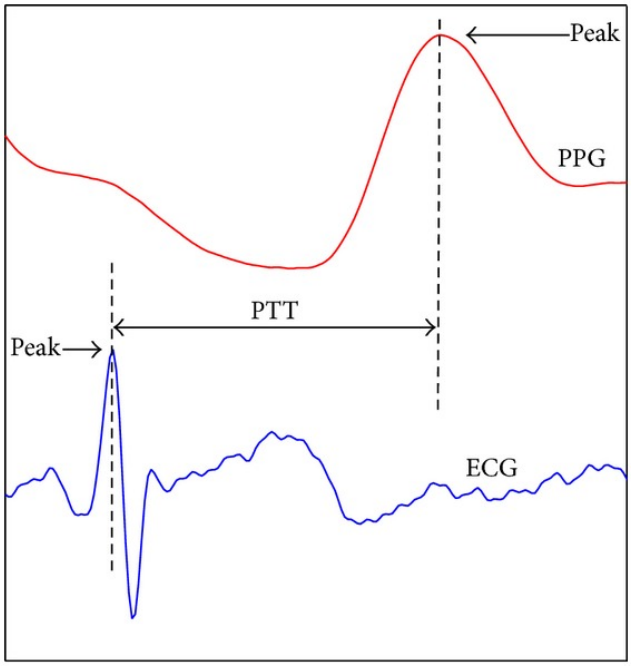


Figura 5.1: Sovrapposizione dei grafici ECG e PPG con indicazione del PTT

Nella figura 5.1 si evidenzia il significato grafico del pulse transit time, cioè la distanza tra il picco R dell'onda dell'ECG e il picco sistolico dell'onda del PPG. Nelle precedenti equazioni E viene assunta come costante, anche se di fatto non lo è, perché il modulo elastico tende ad aumentare con l'aumento della pressione. La relazione viene espressa tramite la seguente equazione:

$$E(P) = E_0 e^{\alpha P} \tag{5.3}$$

dove P è la pressione continua, E_0 è il modulo elastico a 0 mmHg, e α è un parametro strettamente legato alla rigidità dell'arteria che va valutato puntualmente. Sostituendo E in 5.2 con 5.3, si può ottenere la seguente relazione per esprimere la pressione continua [29]:

$$P = -\frac{2}{\alpha} \ln (PTT) + \frac{1}{\alpha} \ln (\frac{2r\rho}{E_0 h} D^2) \tag{5.4}$$

La relazione tra pressione continua, sistolica e diastolica è:

$$P = \frac{1}{3} SBP + \frac{2}{3} DBP \tag{5.5}$$

dove *SBP* è la pressione sistolica e *DBP* la diastolica, entrambe ottenibili tramite la misurazione con uno sfigmomanometro certificato.
Semplificando 5.4, si può scrivere:

$$P = A \ln(PTT) + B \tag{5.6}$$

dove A e B sono dei coefficienti strettamente dipendenti dalle caratteristiche fisiche dell'arteria in considerazione [30]. È naturale pensare che sia praticamente impossibile ottenere dei valori esatti per dimensioni e coefficiente elastico; approssimazioni porterebbero alla presenza di un errore che su molte persone potrebbe essere eccessivamente grande: nonostante le formule sopra riportate, un algoritmo generico è irrealizzabile a meno che non venga fatta una calibrazione iniziale individuale. Si genera un nuovo problema, ovvero il rischio nel demandare all'utente finale la calibrazione personalizzata di un algoritmo sofisticato, senza contare che non vi è la garanzia che la strumentazione utilizzata per farla possieda certificazioni mediche.
Sapendo comunque della correlazione matematica tra i grafici, la soluzione consiste nel creare una rete neurale, a cui è possibile "insegnare" come predire dei valori di pressione sistolica e diastolica dando in input ECG e PPG.

5.2 Cenni sulle reti neurali

Una rete neurale artificiale (ANN, Artificial Neural Network), più semplicemente rete neurale (NN), è un modello di apprendimento che tenta di emulare una rete neurale biologica, quindi un cervello umano. Le reti neurali sono in continua evoluzione, e tutt'ora rimangono per certi aspetti una materia da esplorare; tuttavia, oggi trovano un largo impiego per la risoluzione di problemi complessi e per affrontare le nuove sfide dell'intelligenza artificiale e dell'IoT.
Come in un cervello, in una rete neurale vi sono dei neuroni (chiamati anche nodi) che costituiscono le unità computazionali, e sono collegati tra loro tramite sinapsi. Una rete neurale può essere immaginata come composta di diversi *layer* (strati) di nodi, ciascuno dei quali è collegato ai nodi del layer successivo. Tutte le reti neurali sono composte da almeno tre layer:

- Un input layer, ovvero i dati di ingresso;
- Uno o più hidden layer, dove avviene l'elaborazione vera e propria;
- Un output layer, contenente il risultato finale.

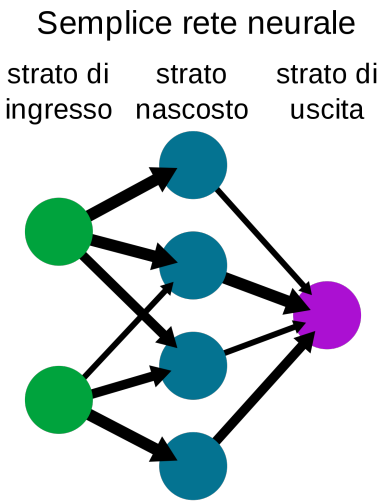


Figura 5.2: Schema di una banale rete neurale con tre strati

L'input di ciascun layer (tranne il primo) è costituito dall'output del layer precedente. Nell'algoritmo, le connessioni tra i nodi dei layer vengono "pesate" tramite fattori moltiplicativi, che rappresentano la "forza" della connessione stessa. Come per i neuroni biologici, i nodi delle reti neurali integrano i segnali in ingresso secondo una apposita funzione, chiamata funzione di attivazione. Se il valore di questa operazione supera la soglia prevista, il nodo sollecitato trasmette questo valore al layer successivo, altrimenti il valore trasmesso sarà zero.
Esistono essenzialmente 2 tipi di reti:

- convoluzionali: sono utilizzate per il riconoscimento delle immagini, accettano input di dimensioni fisse e genera output di dimensioni fisse, e usano un modello di connettività tra i suoi neuroni ed è ispirata dall'organizzazione della corteccia visiva degli animali, i cui singoli neuroni sono disposti in modo tale da rispondere alle regioni sovrapposte piastrellando il campo visivo [31];

- ricorrenti: ideali per analisi di testi e sequenze, accettano dimensioni arbitrarie in input/output, e hanno una propria memoria interna [32].

L'output prodotto da una rete neurale viene chiamato *predizione*: più la predizione si avvicina alla realtà, più si dice che la rete è *accurata*. La predizione non è frutto di un calcolo matematico puntuale, ma è il risultato di un "ragionamento" fatto dalla rete: affinché le predizioni siano accurate, occorre insegnare alla rete a "pensare" tramite una fase di *training* (addestramento). Durante questa fase, vengono stabiliti i pesi delle connessioni tra i nodi. Per addestrare una rete, occorrono un *training set* e un *validation set*, ovvero dei dataset contenenti le informazioni

che la rete deve apprendere. L'addestramento dura un certo numero di *epoche*, vale a dire gli step da eseguire perché la rete apprenda. A tal proposito, si utilizza una funzione di *loss*, tipicamente una RMSE (*root mean squared error*) o una MAE (*mean absolute error*), che in sostanza indicano tramite un valore di quanto il risultato predetto si discosta da quello corretto. Quando la loss si stabilizza, significa che l'addestramento è terminato.
Solitamente, l'addestramento è un'operazione estremamente onerosa dal punto di vista delle risorse e anche del tempo che impiega. Dal momento che le GPU possiedono una potenza di calcolo molto maggiore rispetto alle CPU, vengono normalmente impiegate per l'esecuzione di una rete: esitono infatti delle GPU create appositamente per il machine learning.

5.3 Realizzazione della rete

Lo strumento utilizzato per la scrittura del codice è Google Colab, una specie di IDE online che permette di eseguire la compilazione e l'esecuzione su una macchina remota dotata di una GPU molto potente.
La rete è realizzata in linguaggio Python con Tensorflow, una piattaforma open source end-to-end sviluppata da Google per l'apprendimento automatico. Dispone di un ecosistema completo e flessibile di strumenti, librerie e risorse della community che consente ai ricercatori di promuovere lo stato dell'arte in ML e gli sviluppatori di creare e distribuire facilmente applicazioni basate su ML.
In Tensorflow, gli input e gli output sono costituiti dai tensori: sono array multidimensionali con un tipo unificato (chiamato *dtype*). Tutti i tensori sono immutabili come i numeri e le stringhe Python: non si può mai aggiornare il contenuto di un tensore, solo crearne uno nuovo [33]. I tensori possono essere multidimensionali.
La rete neurale architetturata è una versione modificata della ResNet, ossia una rete originariamente utilizzata per la classificazione costituita da layer convoluzionali e layer fully connected (cioè con tutti i nodi collegati tra loro), a cui vengono aggiunti tre layer LSTM (*Long Short-Term Memory*), ovvero uno strato in grado di apprendere da lunghe sequenze temporali e conservarne la memoria [34]. In particolare, il primo di questi LSTM è *bidirectional* (BLSTM), perché hanno accesso a una sequenza di input su scala molto più ampia: infatti, un BLSTM cerca caratteristiche contestuali nella sequenza sia prima che dopo, ed è utile perché non si conosce a priori dove sia la caratteristica che la rete non deve dimenticare [35].
Il codice Python riportato di seguito è importante perché serve per creare il modello appena descritto. I layer utilizzati sono quelli forniti da Keras, una libreria open source per l'apprendimento automatico che offre API semplici e consistenti [36].

```
1    def resnet_model(input_shape, num_classes=10):
```

```
2
3       filters = [64, 128, 256, 512]
4       kernels = [3, 3, 3, 3]
5       strides = [1, 2, 2, 2]
6
7       tf1 = tf.compat.v1
8
9       # Inizio della rete
10      sign = tf.keras.layers.Input(shape=input_shape, dtype='float32')
11      x = tf.keras.layers.Conv1D(64, 3, strides=1, padding='same')(sign
        )
12      # tf.keras.layers.Conv1D(filters=32, kernel_size=10, activation='
        relu', padding='same')
13
14
15      for i in range(len(kernels)):
16          x = _resnet_block(
17              x,
18              filters[i],
19              kernels[i],
20              strides[i])
21
22
23      x = tf.keras.layers.BatchNormalization()(x)
24      x = tf.keras.layers.Activation('relu')(x)
25      x = tf.keras.layers.AveragePooling1D(4, 1)(x)
26
27      x = tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128,
        return_sequences=True))(x)
28      x = tf.keras.layers.LSTM(128, return_sequences=True)(x)
29      x = tf.keras.layers.LSTM(128)(x)
30      x = tf.keras.layers.Dense(num_classes)(x)
31
32      model = tf.keras.Model(inputs=sign, outputs=x, name='resnet18')
33
34      return model
```

Come accennato in precedenza, la rete riceve in input i campioni di ECG e PPG e restituisce i corrispondenti valori di pressione sistolica e diastolica. In particolare, l'input deve essere costituito da 5 secondi di misurazione con campionamento a 125 Hz, quindi 625 campioni di ECG e altri 625 di PPG. Pertanto, il tensore in input ha dimensione [n 625 2]: significa che è tridimensionale, e possono esserci n matrici 625 x 2, vale a dire n misurazioni da 5 secondi di ECG e PPG, più in particolare nelle celle con indice 0 della terza dimensione risiedono i campioni di PPG, mentre in quelle con indice 1 risiedono i campioni di ECG. Il tensore in output ha dimensione [n 2], quindi una coppia di valori di pressione per ogni 5 secondi di campioni.

I set di valori sia per l'addestramento, che da passare in input per verificare i valori, sono racchiusi dentro dei file .mat generati tramite uno script Matlab. I dati per l'addestramento sono elaborati a partire dal database MIMIC, una raccolta contenente ECG, PPG e pressioni di pazienti ricoverati in ospedale misurati anche con tecniche invasive, ergo soggetti ad un errore molto basso.
Quanto spiegato e ulteriori dettagli sono illustrati sul paper [37].

5.4 Avanzamento versione di Tensorflow

La rete inizialmente è stata scritta utilizzando Tensorflow in versione 1.15. L'obiettivo finale è quello di eseguire la rete su smartphone: per farlo, occorre convertire il modello con i relativi pesi in un file Tensorflow Lite (tflite). Quest'ultimo è creato appositamente per ottimizzare l'esecuzione su un dispositivo che non può vantare grande potenza di calcolo se messo vicino a un computer. Il problema dei modelli tflite convertiti a partire da Tensorflow 1.x è dato dal mancato supporto ai layer convoluzionali, causando quindi un'eccezione nella fase di istanziamento del modello.
Dopo aver testato librerie differenti sia per la conversione che per l'esecuzione, l'unica soluzione possibile è stata la migrazione a Tensorflow 2.x, che fornisce supporto per la conversione in Tensorflow Lite per i modelli contenenti layer convoluzionali, come nel caso della rete in esame.
La maggior parte delle funzioni sono rimaste invariate tra le due versioni dal punto di vista di implementazione; il problema maggiore è dato dalla differenza dei layer LSTM, che ha reso incompatibili i pesi dell'addestramento eseguito precedentemente con il modello aggiornato. Si è reso quindi indispensabile non solo eseguire nuovamente l'addestramento, ma anche riscriverne il codice, seguendo le istruzioni presenti sulla documentazione ufficiale [38].
Di seguito il codice Python dell'addestramento:

```
1     loss_object = tf.keras.losses.Huber()
2     optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate,
      name='adam')
3
4     train_loss = tf.keras.metrics.Mean(name='train_loss')
5     train_accuracy = tf.keras.metrics.MeanAbsoluteError(name='
    train_accuracy')
6
7     test_loss = tf.keras.metrics.Mean(name='test_loss')
8     test_accuracy = tf.keras.metrics.MeanAbsoluteError(name='
    test_accuracy')
9
10
```

```
    def train_step(x, y):
        with tf.GradientTape() as tape:
        predictions = model(x, training=True)
        loss = loss_object(y_true=y, y_pred=predictions)
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(grads_and_vars=zip(gradients, model
    .trainable_variables))
        train_loss(loss)
        train_accuracy(y, predictions)

    def test_step(x, y):
        predictions = model(x, training=False)
        v_loss = loss_object(y, predictions)

        test_loss(v_loss)
        test_accuracy(y, predictions)


    # start training
    for epoch in range(epochs):
        train_loss.reset_states()
        train_accuracy.reset_states()
        test_loss.reset_states()
        test_accuracy.reset_states()

        for x, y in train_ds:
            train_step(x, y)

        for test_x, test_y in test_ds:
            test_step(test_x, test_y)
```

Le metriche e l'optimizer utilizzati sono gli stessi applicati nella prima versione della rete, ovviamente con le librerie aggiornate per Tensorflow 2. Il numero di epoche stabilito per l'addestramento è pari a 100, che è il valore tipicamente scelto. Per capire quale sia il valore ottimale, bisogna osservare per ciascuna epoca quanto vale la loss: in particolare, essa tende a scendere fino a stabilizzarsi, cioè dopo un certo numero di epoche smette di variare. Nel caso in esame, alla prima epoca il valore della loss è pari a 66.45325469970703, mentre alla centesima è pari a 0.5899175405502319; nelle epoche 98 e 99 vale circa 0.60. Questi risultati suggeriscono come 100 epoche siano sufficienti per raggiungere un buon livello di accuratezza.
Il tempo impiegato per questo addestramento su Colab utilizzando la CPU è di circa 4 ore.

5.5 Modello Tensorflow Lite

L'avanzamento di versione con il nuovo addestramento ha finalmente risolto i problemi di inferenza del file tflite, apportando anche miglioramenti alle predizioni. Di seguito il codice per generare un modello tflite:

```
converter = tf.lite.TFLiteConverter.from_saved_model("/content/drive/My Drive/Checkpoint") # path to the SavedModel directory
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS, tf.lite.OpsSet.SELECT_TF_OPS]

converter.allow_custom_ops=True
tflite_model = converter.convert()
```

Come si può notare, la libreria di Tensorflow rende l'operazione molto semplice; è necessario specificare alcune opzioni al converter perché alcune operazioni presenti nel codice Python della rete non sono supportate di default da Tensorflow Lite.
Il modello tflite accetta in input tensori dimensionati [1 625 2], ergo per testare ad esempio 10 secondi di misurazione occorrerà spezzarla a metà e passare l'input alla rete due volte. Di seguito il codice in Python per testare 5 secondi di misurazione:
.

```
interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()
# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

input_data = np.array(test, dtype=np.float32)
interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()
output_data = interpreter.get_tensor(output_details[0]['index'])
print('\n\n\n\n\n')
print(output_data)
```

Viene creato un oggetto di tipo *interpreter* a cui viene passato il modello tflite creato precedentemente. Successivamente, tramite $np.array(test, dtype = np.float32)$ (dove *np* sta per *numpy*) viene creato il tensore da passare in input tramite $interpreter.set\_tensor(input\_details[0]['index'], input\_data)$.
Si esegue la rete tramite $interpreter.invoke()$ e si ottiene il tensore in output tramite $interpreter.get_tensor(output\_details[0]['index'])$ che avrà dimensione [1 2].

5.5.1 On-device training

Il modello Tensorflow Lite, quando viene convertito, eredita i pesi definiti dall'addestramento in Python. Tuttavia, una delle caratteristiche più interessanti di un'intelligenza artificiale è il poter continuare ad apprendere durante il suo utilizzo. Tensorflow Lite permette di eseguire questa operazione: in Java, è possibile memorizzare il peso addestrato come formato checkpoint nella memoria interna dell'applicazione. Anche in questo caso, l'addestramento è un'operazione lunga e costosa, per cui è opportuno che venga eseguito tramite un servizio in background che giri occasionalmente duranti i periodi di inattività [39].

Capitolo 6

PulsEcg

6.1 Panoramica del dispositivo

Il dispositivo PulsEcg si presenta in forma sperimentale come in figura 6.1, un mockup rosso stampato in 3D su cui sono incollati 2 elettrodi e la circuiteria. In particolare, si trovano due circuiti, uno dedicato agli elettrodi e uno dedicato al sensore infrarosso.
Il primo circuito è costituito da:

- connettori: il connettore degli elettrodi, il connettore USB, il connettore piatto e il connettore della batteria formano l'interazione tra il PCB e il mondo esterno;
- modulo Bluetooth 4.0: usato per la comunicazione dei dati ad altri dispositivi, in particolare per inviare i dati all'applicazione sviluppata da questa tesi;
- blocchi di alimentazione: Battery Charger, Battery Gauge sono usati per controllare lo stato della batteria e la sua modalità di carica; Voltage Regulator e Analog Power Domain sono due componenti che generano una tensione di alimentazione stabile e una tensione di riferimento usata da tutti gli altri componenti che devono avere un'alimentazione stabile;
- filtri: questo blocco è utilizzato per il condizionamento del segnale elettrocardiografico;
- microcontrollore Texas Instruments CC2640R2F: controlla tutti gli altri blocchi ed è usato per raccogliere i dati dell'elettrocardiogramma attraverso il suo convertitore analogico-digitale (ADC) interno mentre il valore di carica della batteria e il valore della fotopletismografia attraverso la comunicazione I2C.

Per quanto concerne il secondo circuito si ha:

- connettore piatto: usato per collegare il circuito stampato al circuito principale attraverso il cavo piatto che porta le linee di alimentazione e le linee I2C;
- sensore Maxim Integrated MAXM86161: impiegato per ricavare i dati utili per la fotopletismografia, utilizza la luce rossa e infrarossa.

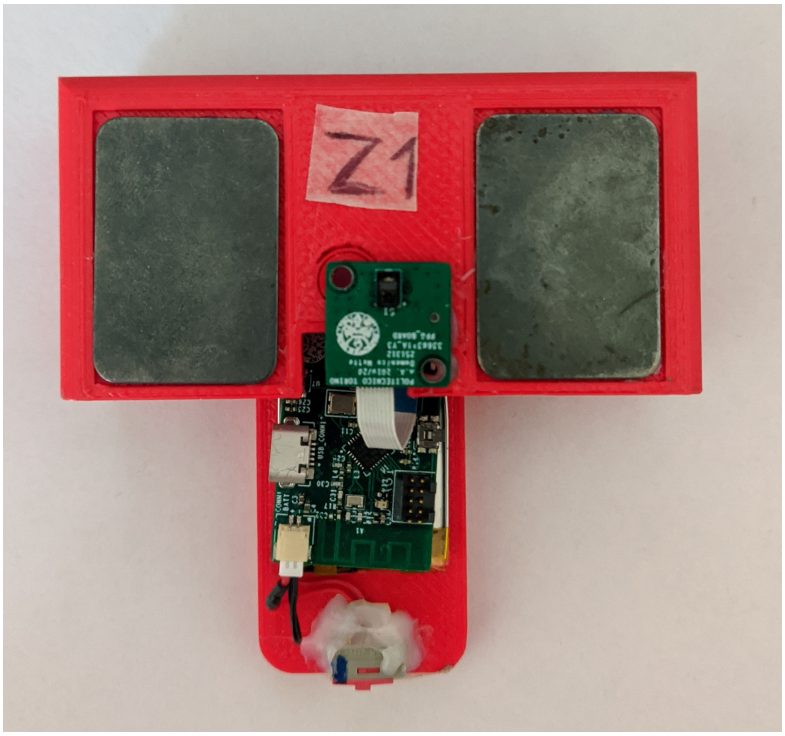


Figura 6.1: Versione sperimentale del PulsEcg con circuito montato su mockup stampato in 3D

Il firmware, scritto in linguaggio C e in continuo sviluppo in progetti di tesi paralleli, gestisce la connessione Bluetooth, un primo processamento del segnale proveniente direttamente dai sensori, e le funzionalità di spegnimento automatico in caso di inutilizzo.

6.2 Applicazione Android

Come per EcgWatch, anche il dispositivo PulsEcg è comandabile esclusivamente dall'omonima app, disponibile sia per Android che per iOS. L'interfacciamento avviene tramite Bluetooth Low Energy ed è possibile misurare elettrocardiogramma e battiti, fotopletismografia e saturazione, e ottenere i valori di pressione sistolica e diastolica.
Anche per l'applicazione PulsEcg i dati rimangono salvati esclusivamente in locale e in alcun modo vengono condivisi con terzi.

6.2.1 Permessi

È indispensabile accettare sia il permesso per la lettura/scrittura sul filesystem, sia quello di accesso alla posizione, perché il Bluetooth Low Energy su Android funziona esclusivamente con il GPS attivo (vedere paragrafo 3.8).

6.2.2 Struttura

L'applicazione è composta di 3 Activity, che servono per visualizzare la schermata principale, il riepilogo e i dettagli di una misurazione; tutte le altre schermate sono Fragment. Vi sono, oltre alle classi di supporto per costanti e metodi statici di utility, due *manager*, ovvero classi singleton (cioè a singola istanza) atte a facilitare la gestione delle impostazioni (SettingsManager) e della connessione bluetooth (BleConnectionManager).

6.2.3 Interfacce ed esperienza utente

Come in EcgWatch, alla prima apertura l'applicazione presenta un breve tutorial, il TutorialFragment. Dopo di che, tramite il FirstConfigurationFragment, vengono richiesti nome, cognome, email, email del dottore (opzionale) e braccio preferito su cui indossare il dispositivo. Premendo sul tasto "SALVA", compare la schermata principale, cioè la MainActivity (i fragment prima citati sono figli di questa Activity).

Figura 6.2: L'interfaccia della MainActivity

Come si può vedere in figura 6.2, tale schermata è analoga a quella proposta da EcgWatch, con un miglioramento nell'aspetto grafico. Infatti, i componenti grafici utilizzati per realizzare le schermate sono quelli di Material Design.

Impostazioni

Nelle impostazioni, ovvero il SettingsFragment (figura 6.3), si possono modificare le informazioni personali inserite in configurazione, impostare l'autosalvataggio delle acquisizioni, la condivisione della posizione e la durata delle acquisizioni (10 o 30 secondi). L'opzione "Reset calibrazione" sarà presente solo nelle versioni di sviluppo, e non nelle versioni scaricabili dal Play Store (vedere 6.2.6).

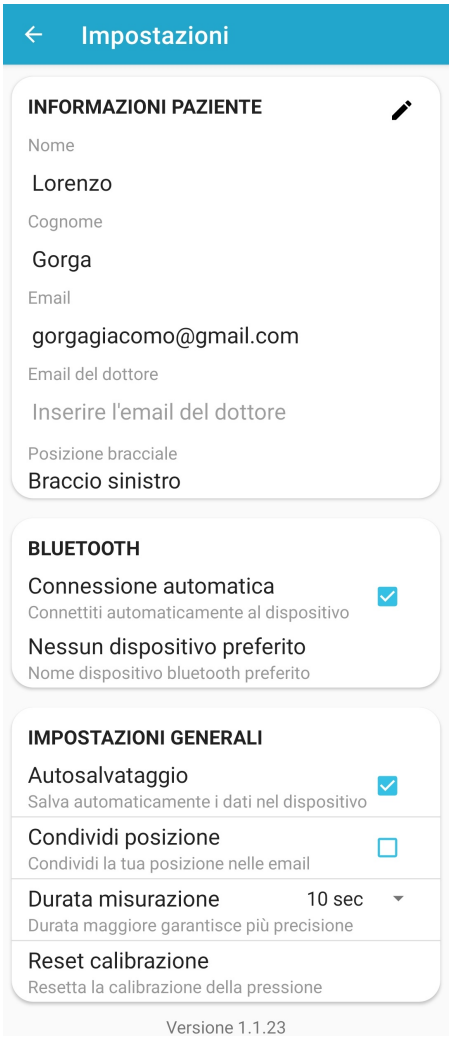


Figura 6.3: La schermata delle impostazioni di PulsEcg

Connessione al dispositivo

Dalla schermata principale, premendo sulla prima card, il codice esegue dapprima un controllo sullo stato di bluetooth e GPS: qualora uno dei due o entrambi non fossero attivi, se ne richiede l'abilitazione tramite delle dialog di sistema. Quando entrambi sono accesi, si apre il BleDevicesFragment (figura 6.4), che contiene nella parte alta una lista di tutti i dispositivi compatibili accoppiati. Premendo su una entry, è possibile tentare di connettersi con tale dispositivo; tenendo premuto, possibile invece disaccoppiarsi.
Quando si è connessi, un check compare nella entry del dispositivo corrispondente: premendoci, è possibile disconnettersi; si può essere connessi a un solo dispositivo

Figura 6.4: Il BleDevicesFragment con la lista dei dispositivi compatibili associati

Figura 6.5: La MainActivity con un dispositivo connesso

per volta. La figura 6.5 mostra la schermata principale quando vi è un dispositivo connesso.
A codice, la connessione avviene chiamando il metodo *BluetoothDevice#connectGatt* che restituisce un oggetto di tipo *BluetoothGatt* e che riceve tra i parametri un oggetto di tipo *BluetoothGattCallback*, una classe astratta la cui implementazione è analizzata nella sezione 6.2.4.
Nella parte bassa del fragment, invece, è possibile eseguire una scansione dei dispositivi compatibili nelle vicinanze (*discovery*), e quindi effettuare la procedura di accoppiamento direttamente dall'app senza passare dalle impostazioni.
Per implementare un meccanismo di scansione, occorre innanzitutto creare un

oggetto di tipo *ScanCallback*, una classe astratta di cui si implementano i metodi *onScanResult*, *onBatchResult* e on *onScanFailed*. Quello di interesse è il primo, che ha come parametro un oggetto di tipo *ScanResult* tramite il quale è possibile ottenere il *BluetoothDevice*, il cui nome viene mostrato a schermo.
L'oggetto *ScanCallback*, affinché possa effettivamente ricevere delle callback dalla scansione, viene passato al metodo *BluetoothLeScannerCompat#startScan*, che va a comunicare con l'hardware per ottenere le informazioni. La scansione è interrompibile manualmente, altrimenti si ferma automaticamente dopo 15 secondi. I dispositivi rilevati vengono listati, e premendo su ogni entry della lista si procede con l'accoppiamento. A codice, viene invocato il metodo *BluetoothDevice#createBond*, le cui callback vengono intercettate tramite un BroadcastReceiver: infatti, la callback *onReceive* del receiver ha come parametro un *Intent* dove dagli extra si ricava lo stato del *bonding*: quando è completo, il dispositivo interessato viene aggiunto alla prima lista dei dispositivi accoppiati con cui è possibile connettersi.

Acquisizione

Figura 6.6: Il TakeEcgFragment con l'animazione e il contatore

Figura 6.7: La RecapActivity di un'acquisizione da 10 secondi

A dispositivo connesso, con un tap sulla card "Acquisisci", si apre il TakeEcgFragment, mostrato in figura 6.6. Una scritta richiede all'utente di appoggiare il dito

sull'elettrodo, e quando viene fatto un led rosso sul dispositivo inizia a lampeggiare; sull'applicazione vengono mostrati un conto alla rovescia e un'animazione stilizzata di un ECG.
Terminata l'acquisizione si apre una nuova schermata, la RecapActivity, che mostra il riepilogo della misurazione appena completata, come in figura 6.7.
Sulla toolbar vi sono le opzioni per eliminare e salvare la misurazione (il secondo è visibile solo se l'impostazione di salvataggio automatico è disabilitata). Subito sotto ci sono le informazioni generali sulla misurazione, scendendo si trovano una preview dell'ECG, i battiti, la saturazione e la pressione; infine, una scritta per la calibrazione della pressione (vedere 6.2.6). Premendo su ciascuna card, si apre la GraphsActivity: quest'ultima è costituita da una toolbar (le cui opzioni sono eliminazione, condivisione file e PDF, e informazioni) e da un ViewPager, cioè uno strumento che permette di creare delle tab attraverso cui navigare tra diverse schermate ciascuna rappresentata da un fragment.

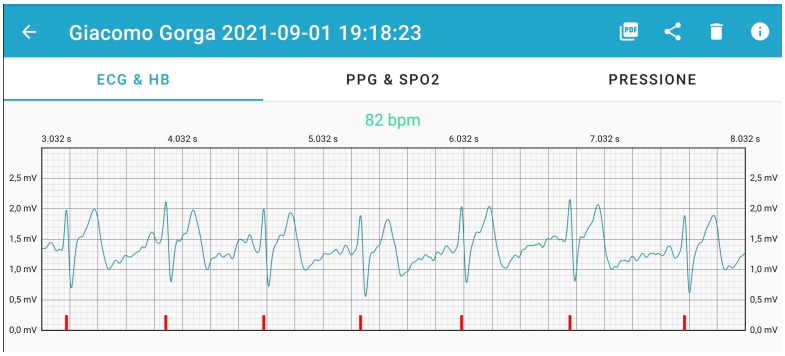


Figura 6.8: La GraphsActivity con la tab del ShowEcgFragment che mostra l'elettrocardiogramma

Il primo, il ShowEcgFragment, mostra l'ECG con picchi e battiti, come si può osservare nella figura 6.8; nella parte bassa, in caso di possibile fibrillazione atriale rilevata, compare una opportuna scritta a segnalarlo. L'implementazione di questo grafico è uguale a quella per EcgWatch.

Il secondo, il ShowPpgFragment, mostra il PPG e la saturazione come in figura 6.9. Anche questo grafico si può scorrere e zoomare, e la sua implementazione è sovrapponibile a quella del grafico dell'ECG, con l'unica differenza che non ci sono assi orizzontali e nessuna etichetta sull'asse Y.

La figura 6.10 mostra il terzo e ultimo fragment, il PressureFragment, il quale presenta banalmente una card come nella schermata di riepilogo con pressione sistolica e diastolica.

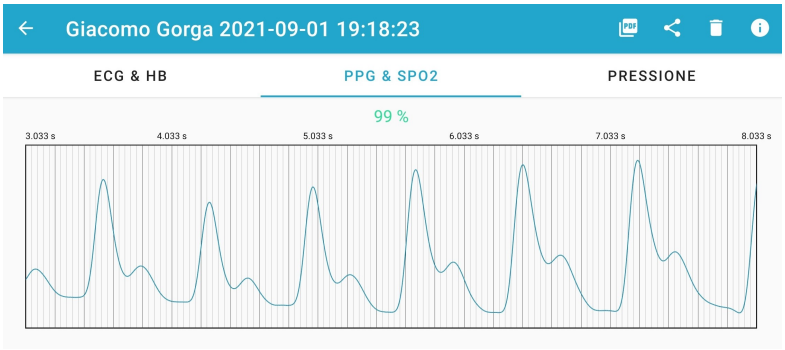


Figura 6.9: La GraphsActivity con la tab del ShowPpgFragment che mostra la fotopletismografia

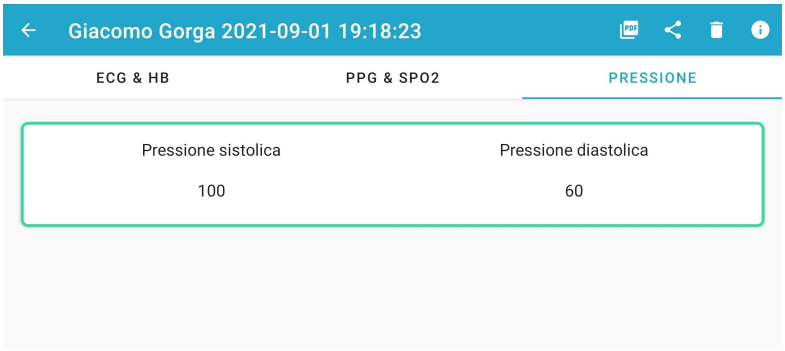


Figura 6.10: La GraphsActivity con la tab del PressureFragment che mostra la pressione

Archivio

L'archivio, l'EcgListFragment, elenca i file contenenti i dati delle acquisizioni dal più recente al meno recente separatamente per mese e anno, come si osserva in figura 6.11. Di default, il nome del file è strutturato come <nome> <cognome> <data e ora>.json: si tratta quindi di un normalissimo file JSON che contiene i samples del PPG e dell'ECG, battiti, saturazione, pressione, stato (se possibile fibrillazione atriale o nessuna anomalia), le informazioni sull'utente che ha eseguito la misurazione, data, ora e durata della misurazione. In questo modo, quando si vuole visualizzare nuovamente i dettagli di una misurazione, viene fatta una normale lettura da file evitando quindi di applicare ogni volta gli algoritmi sui dati. Cliccando su ciascuna entry viene aperta la RecapActivity, da cui è possibile poi accedere ai grafici e dettagli ulteriori.
Per il resto, i pulsanti e le relative funzionalità sono uguali a quelle di EcgWatch. La figura 6.12 mostra la dialog per filtrare le misurazioni nella lista per nome,

cognome e data.

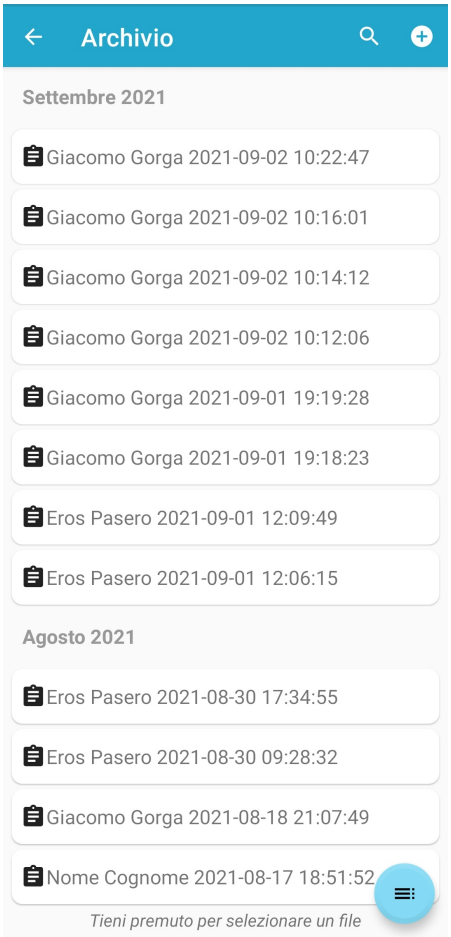


Figura 6.11: La schermata EcgListFragment con la lista delle misurazioni

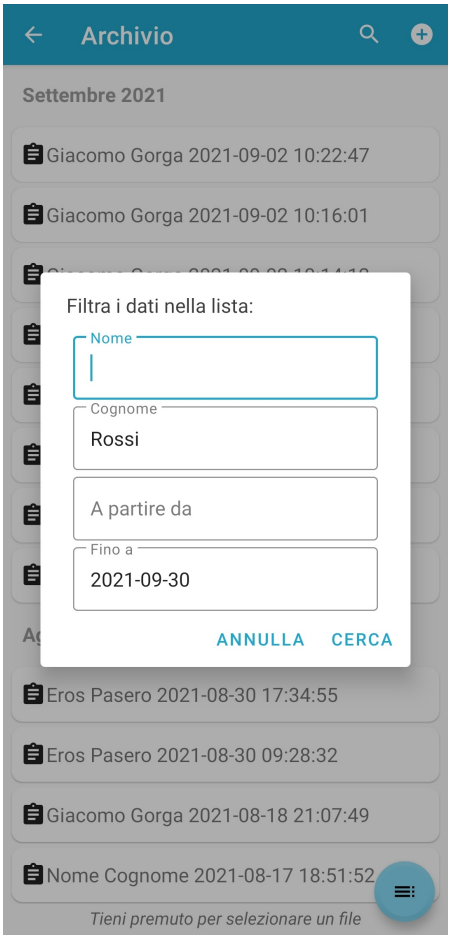


Figura 6.12: La dialog per filtrare la lista delle misurazioni

6.2.4 Comunicazione con il dispositivo

Quando l'utente sceglie un dispositivo dal *BleDevicesFragment* per effettuare la connessione, viene innanzitutto istanziato il *BleService*, che è un servizio creato come *bound service* [40], e che è necessario per mantenere viva la connessione BLE con il dispositivo PulsEcg. L'oggetto di tipo *BluetoothGattCallback*, che viene passato al metodo *BluetoothDevice#connectGatt* come precedentemente descritto nel paragrafo 6.2.3, è una variabile globale della classe *BleService* e sono ivi implementati i suoi metodi. Di seguito la descrizione:

- *onConnectionStateChange*: questa callback viene invocata quando cambia lo stato di connessione con il GATT, che può essere connesso o disconnesso;

- *onDescriptorWrite*: quando viene richiesta una nuova acquisizione, a codice è necessario richiedere le notifiche per le caratteristiche del dispositivo, che sono batteria, payload ECG e payload PPG. Per fare ciò, per ciascuna caratteristica si invoca il metodo *BluetoothGatt#writeDrescriptor* a cui viene passato come parametro il descriptor della caratteristica. Per ciascuna invocazione, viene chiamata questa callback che verifica siano state richieste le notifiche per tutte le caratteristiche; se sì, si invoca il metodo *BluetoothGatt#writeCharacteristic* al quale viene passato come parametro un oggetto di tipo *BluetoothGattCharacteristic*, contenente l'informazione sulla durata richiesta dell'acquisizione. Detto in altri termini, il client invia un dato in scrittura che serve per segnalare l'inizio dell'acquisizione;

- *onMtuChanged*: callback invocata quando viene cambiato l'MTU (Maximum Transmission Unit), cioè la dimensione massima in byte di un singolo pacchetto. Di fatto, l'MTU viene impostato una sola volta al valore di 204 per via della limitazione hardware. In questa implementazione viene chiamato il metodo *BluetoothGatt#discoverServices()* per scoprire i servizi offerti dal GATT e capire se è un dispositivo realmente compatibile;

- *onServicesDiscovered*: callback in risposta a *BluetoothGatt#discoverServices()* in cui si controlla che il GATT effettivamente possieda i servizi richiesti;

- *onCharacteristicRead*: callback quando avviene la lettura di una caratteristica, di fatto non utilizzata nell'implementazione in esame;

- *onCharacteristicWrite*: callback in risposta a *BluetoothGatt#writeCharacteristic*, è l'inizio dell'acquisizione, vengono inizializzate alcune variabili che serviranno successivamente;

- *onCharacteristicChanged*: questa callback costituisce il fulcro dell'elaborazione dati. Essa viene invocata quando cambia la caratteristica ricevuta in lettura, quindi di fatto il campione acquisito. La prima caratteristica ricevuta corrisponde sempre alla batteria del dispositivo, e dopo iniziano ad arrivare i campioni di ECG e PPG. Ciascun pacchetto è di dimensione 4 byte, e viene opportunamente elaborato con *and bit a bit* e salvato nel corrispondente array inizializzato precedentemente, e ad ogni iterazione si incrementa un contatore per tenere traccia dei dati ricevuti.

A differenza di EcgWatch, il PulsEcg non prevede una memoria per bufferizzare i campioni, ma vengono trasmessi non appena acquisiti; questo può generare

problematiche di desincronizzazione e/o perdita di pacchetti (vedere sezione 7.1). Si vuole far notare che ogni acquisizione in realtà dura 4 secondi in più del valore impostato principalmente perché, a causa dei filtri hardware implementati, i valori iniziali e finali dell'acquisizione spesso sono soggetti ad un rumore troppo forte. Pertanto, il timer visualizzato a schermo con i secondi rimanenti è fittizio, perché il conto viene fatto tramite un offset.

6.2.5 Algoritmi per l'elaborazione del segnale

Se durante l'acquisizione la soglia dei pacchetti non venisse raggiunta, cioè vi è stata una perdita di pacchetti durante la trasmissione, il TakeEcgFragment si chiude automaticamente e viene mostrato un messaggio che segnala all'utente che l'acquisizione è fallita. Se invece non ci sono perdite, dai vettori contenenti i campioni di ECG e PPG vengono tagliati i primi 1500 elementi (3 secondi) e gli ultimi 500 (1 secondo).
Quando il timer mostrato a schermo nel TakeEcgFragment termina, avviene la sincronizzazione con quello che è stato realmente acquisito tramite un'interfaccia dichiara nel *BleService* chiamata *BluetoothAcquisition* che comprende un unico metodo *onAcquisitionFinished*, implementato nella MainActivity: questo serve non solo per allineare due operazioni parallele, ma anche per riconvengere sul thread principale.
Successivamente, si applicano una normalizzazione sui segnali, i filtri, gli algoritmi per calcolo della frequenza cardiaca, SpO2 e pressione, e infine si salva il tutto nel file JSON.
Gli algoritmi per la determinazione dei picchi dell'ECG e dei battiti sono uguali a quelli descritti per EcgWatch (vedere sezione 4.2.5).

Filtri ECG

Il segnale elettrocardiografico viene sottoposto ad un pre-processamento che prevede una traslazione verso il basso di 1650 (mV), e poi una divisione per 750. I filtri a cui viene poi sottoposto il segnale così ottenuto sono due:

- *notch filter* (filtro elimina-banda): è un filtro che non fa passare le frequenze in un dato intervallo, ma attenua quelle in una gamma specifica a livelli molto bassi; è l'opposto di un filtro passa-banda. L'implementazione in app prevede l'utilizzo di una funzione biquadratica ripetuta quattro volte con parametri diversi;

```
1        public static double[] notchFilter(double[] input) {
2            double[] output = Arrays.copyOf(input, input.length);
```

```
3
4               bquad(output, 0.9622, 0, -0.9622, 0, -0.9844);
5               bquad(output, 1, 1.618, 1, 1.6054, 0.9844);
6               bquad(output, 1, -1.618, 1, -1.6054, 0.9844);
7               bquad(output, 1, 0.618, 1, 0.6132, 0.9844);
8               bquad(output, 1, -0.618, 1, -0.6132, 0.9844);
9
10              return output;
11          }
12
13          public static void bquad(double[] input, double b0,
    double b1, double b2, double a1, double a2) {
14              double x1 = 0, x2 = 0, y1 = 0, y2 = 0;
15              double part1, part2;
16              for (int i = 0; i < input.length; i++) {
17                  part1 = b0 * input[i] + b1 * x1 + b2 * x2;
18                  part2 = a1 * y1 + a2 * y2;
19                  x2 = x1;
20                  x1 = input[i];
21                  input[i] = part1 - part2;
22                  y2 = y1;
23                  y1 = input[i];
24              }
25          }
26
```

- *moving average* (media mobile): è un filtro che consiste nell'applicare per ciascun elemento presente la media tra i valori di un intorno centrato in tale elemento e la cui ampiezza pari alla finestra ricevuta come parametro.

```
1           public static double[] movingAverage(double[] vector, int
    window) {
2               if (window <= 0)
3                   return vector;
4
5               int j = 0;
6               double temp = 0;
7               int medium = window / 2, count = 0;
8               double[] output = new double[vector.length];
9
10              for (int i = 0; i < vector.length; i++) {
11                  for (j = i - medium; j <= i + medium; j++) {
12                      if (j >= 0 && j < vector.length) {
13                          temp += vector[j];
14                          count++;
15                      }
16                  }
```

```
                output[i] = temp / count;
                temp = 0;
                count = 0;
            }
            return output;
        }

```

Il metodo *wrap* per l'applicazione dei filtri al segnale esegue innanzitutto il filtro notch, poi la media mobile per 22 volte, con finestre che variano da 10 a 2. Infine, Il segnale viene amplificato moltiplicando ciascun valore per un fattore 1,8 e alzato sommando 1,4.

Filtri PPG

Il segnale PPG viene pre-processato andando ad applicare una normalizzazione sottraendo ciascun campione al doppio della media dei campioni. Per procedere con il filtraggio, l'algoritmo principale utilizzato è la media mobile, la cui implementazione è riportata nel paragrafo precedente. Questo algoritmo viene applicato 3 volte di seguito, con finestre rispettivamente di 30, 20 e 10.
Poi, viene applicato un algoritmo di *detrending*, il cui obiettivo è portare tutti i valori a un certo "livello": infatti, l'onda filtrata tende ad aumentare gradualmente i valori nel tempo, il detrendig pone i massimi e i minimi locali del segnale circa sulla stessa linea. Per farlo, si calcola la media mobile sul segnale con finestra 1200 e la media complessiva di tutti i valori. Infine, per ciascun elemento del vettore dei campioni, si sottrae il corrispondente nel vettore della media mobile e si aggiunge il valore della media complessiva. Si riporta l'implementazione;

```
1   public static double[] detrending(double[] ppgR) {
2       double[] output = new double[ppgR.length];
3       double meanR = 0;
4       double[] meanedSig = movingAverage(ppgR, 1200);
5
6       for (double d : ppgR) meanR += d;
7       meanR = meanR / ppgR.length;
8       for (int i = 0; i < ppgR.length; i++) {
9           output[i] = ppgR[i] - meanedSig[i] + meanR;
10      }
11      return output;
12  }
```

Infine, opzionalmente, si può applicare un filtro di Butterworth per rendere il segnale più smussato, anche se influisce poco sul risultato e nulla sul calcolo della saturazione. L'implementazione di tale filtro è riportata nella sezione 4.2.5.

Calcolo della saturazione

Il sensore PPG raccoglie 2 segnali, il *PPG Red* e il *PPG Infrared*. Infatti, la luce rossa e la luce infrarossa hanno due frequenze differenti, per cui penetrano diversamente sotto la pelle del polpastrello, generando quindi due segnali diversi; quello che viene mostrato nel grafico è il *PPG Red*. Entrambi i segnali devono essere filtrati, e dopo di che si vanno a calcolare i massimi e minimi locali di entrambi salvandoli in 4 liste separate. Poi, ciclando sul numero di elementi delle liste, si determina il rapporto tra la differenza massimo-minimo e il minimo per ciascun indice della lista, e ogni quoziente salvato in una nuova lista. Le 2 liste dei rapporti servono per ottenere un valore di SpO2 per ciascun rapporto tra i corrispondenti item delle due liste.
Questo procedimento, di cui viene riportato il codice Java, viene eseguito più volte, andando ad ogni iterazione a tenere traccia con delle mappe i valori di SpO2 e la loro deviazione standard, e ad incrementare il valore prefissato per la distanza minima tra due massimi/minimi. L'iterazione migliore è quella che produce una deviazione standard minore tra i valori di SpO2 calcolati, di cui infine si va a fare la media per ottenere il risultato finale. Se quest'ultimo è maggiore di 99, viene troncato a tale valore, perché il 100% è di fatto un valore impossibile.

```
1     for (int minSeparation = MIN_SEPARATION_START; minSeparation <=
      MIN_SEPARATION_END; minSeparation += 10) {
2         RMaxFirst = findLocalMaxMin(ppgR, minSeparation, maxR, minR);
3         IRMaxFirst = findLocalMaxMin(ppgIR, minSeparation, maxIr,
      minIr);
4
5         if (RMaxFirst) {
6             for (int i = 0; i < minR.size() && i < maxR.size(); i++)
      {
7                 temp = (maxR.get(i) - minR.get(i));
8                 temp = temp / minR.get(i);
9                 ratioR.add(temp);
10            }
11        } else {
12            for (int i = 0; i < minR.size() - 1 && i < maxR.size(); i
      ++) {
13                temp = (maxR.get(i) - minR.get(i + 1));
14                temp = temp / minR.get(i + 1);
15                ratioR.add(temp);
16            }
17        }
18
19        if (IRMaxFirst) {
```

```
20              for (int i = 0; i < minIr.size() && i < maxIr.size(); i
    ++) {
21                  temp = (maxIr.get(i) - minIr.get(i));
22                  temp = temp / minIr.get(i);
23                  ratioIr.add(temp);
24              }
25          } else {
26              for (int i = 0; i < minIr.size() - 1 && i < maxIr.size();
     i++) {
27                  temp = (maxIr.get(i) - minIr.get(i + 1));
28                  temp = temp / minIr.get(i + 1);
29                  ratioIr.add(temp);
30              }
31          }
32
33          double localSpO2;
34
35          for (int i = 0; i < ratioIr.size() && i < ratioR.size(); i++)
     {
36              temp = ratioR.get(i) / ratioIr.get(i);
37              localSpO2 = 104 - (17 * temp);
38              localSpO2s.add(localSpO2);
39          }
40
41          int index = (minSeparation - MIN_SEPARATION_START) / 10;
42          allLocals.put(index, localSpO2s);
43          stdDevs.put(index, Utility.calculateStandardDeviation(
    localSpO2s));
44      }
```

Le costanti *MIN_SEPARATION_START* e *MIN_SEPARATION_END* valgono rispettivamente 200 e 300; *maxR*, *minR*, *maxIr*, *minIr*, *ratioR*, *ratioIr*, *localSpO2s* sono tutti *ArrayList<Double>*, *IRMaxFirst* sono *RMaxFirst* sono booleani, mentre *allLocals* e *stdDevs* sono delle mappe. Il metodo *findLocalMaxMin* riceve come parametri il vettore in input, la separazione minima tra massimi minimi, e le liste vuote di massimi e minimi; restituisce un valore booleano: true se viene trovato prima un massimo rispetto ad un minimo, false viceversa.

6.2.6 Predizione della pressione

Nel capitolo precedente è stato illustrato in maniera riassuntiva l'implementazione della rete neurale per la predizione della pressione e come il modello possa essere trasferito in un unico file in grado di essere interpretato da un'applicazione mobile. Il file nominato *model.tflite* viene salvato tra gli asset dell'applicazione, e viene istanziato tramite una classe chiamata *Interpreter* di cui si crea un oggetto, che riceve il file come parametro nel costruttore; la funzione chiamata subito dopo

Interpreter#allocateTensors è opzionale su Android.
Successivamente, gli input (quindi i vettori contenenti i segnali ECG e PPG) sono soggetti a ricampionamento da 500 Hz a 125 Hz e a normalizzazione sottraendo a ciascun campione la media dei campioni e dividendo per la deviazione standard. I tensori in input non sono null'altro che dei vettori tridimensionali 1 x 625 x 2, dove nelle celle con indice 0 della terza dimensione risiedono i valori del PPG e in quelle con indice 1 i valori dell'ECG, mentre il tensore in output è una matrice 1 x 2. I due tensori vengono passati come parametri a *Interpreter#run* che esegue l'algoritmo della rete neurale; le predizioni effettuate sono due per misurazioni da 10 secondi e sei per 30 secondi. I valori finali di pressione sistolica e diastolica sono dati dalla media delle rispettive pressioni predette.
Di seguito il codice che esegue quanto spiegato.

```
1     public static int[] useNeuralNetwork(double[] ppg, double[] ecg,
      Context context) {
2         int[] pressure = new int[2];
3
4         try {
5             AssetManager am = context.getAssets();
6             InputStream is = am.open("model.tflite");
7             File tempFile = File.createTempFile("tmp", "tflite");
8             OutputStream out = new FileOutputStream(tempFile);
9             copy(is, out);
10
11            try (Interpreter interpreter = new Interpreter(tempFile))
       {
12                interpreter.allocateTensors();
13
14                float[][] output = new float[1][2];
15                float[][][][] test = new float[1][625][2];
16
17                ppg = resampling500to125(ppg);
18                ecg = resampling500to125(ecg);
19
20                float meanPpg = Utility.meanEvaluation(ppg);
21                float meanEcg = Utility.meanEvaluation(ecg);
22                float stdPpg = Utility.calculateStandardDeviation(ppg
      );
23                float stdEcg = Utility.calculateStandardDeviation(ecg
      );
24
25                int dim1 = (int)(ppg.length / 625);
26                float[][] result = new float[dim1][2];
27
28                for (int d = 0; d < dim1; d++) {
29                    int start = d * 625;
```

```
30                      for (int i = 0; i < 625; i++) {
31                          float value = (ppg[i + start] - meanPpg) /
    stdPpg;
32                          test[0][i][0] = value;
33                          value = (ecg[i + start] - meanEcg) / stdEcg;
34                          test[0][i][1] = value;
35                      }
36
37                      interpreter.run(test, output);
38
39                      result[d][0] = output[0][0];
40                      result[d][1] = output[0][1];
41                  }
42
43                  float sumSist = 0, sumDiast = 0;
44                  for (int d = 0; d < dim1; d++) {
45                      sumSist += result[d][0];
46                      sumDiast += result[d][1];
47                  }
48
49                  pressure[0] = Math.round(sumSist / dim1);
50                  pressure[1] = Math.round(sumDiast / dim1);
51                  tempFile.delete();
52              }
53          } catch (IOException e) {
54              e.printStackTrace();
55          }
56
57          return pressure;
58      }
```

Calibrazione della predizione

Come spiegato nella sezione 5.3, i valori utilizzati per l'addestramento della rete provengono dal database MIMIC, che contiene dati di pazienti patologici. Questo fattore, mischiato a una precisione minore della strumentazione utilizzata per elaborare gli input, comporta un errore sulla predizione dei valori di pressione che talvolta si discosta di molto dalla realtà. La pressione diastolica in particolare è soggetta a una calibrazione fissa di 15, cioè il valore mostrato all'utente è dato dal valore predetto incrementato di 15 unità. Tuttavia, i valori sia di sistolica che di diastolica necessitano di una ricalibrazione ulteriore che non può essere fissa, ma deve essere personalizzata per ciascun paziente.
Nella schermata di riepilogo della misurazione, la scritta *"I valori di pressione non sono corretti?"* se premuta causa l'apertura della dialog mostrata in figura 6.13 che permette di inserire manualmente dei valori di pressione sistolica e diastolica: naturalmente, questo preclude che la misurazione della pressione sia stata effettuata

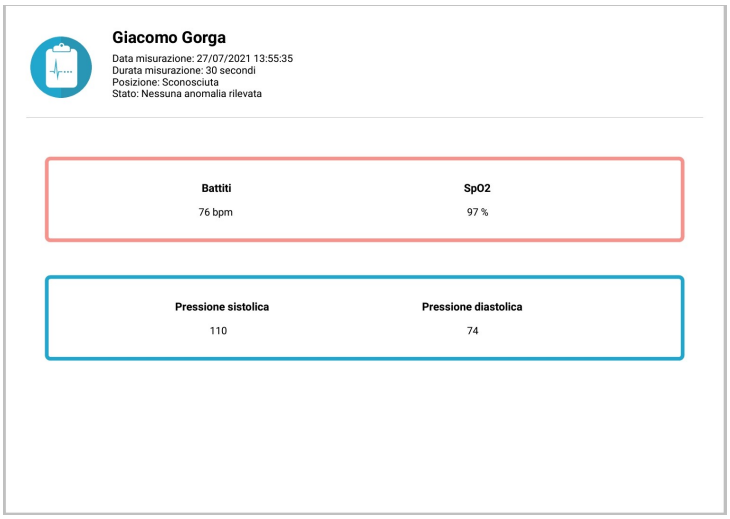
con uno sfigmomanometro certificato durante o a pochi secondi di differenza dalla misurazione eseguita con il PulsEcg (altrimenti i risultati non sarebbero paragonabili).

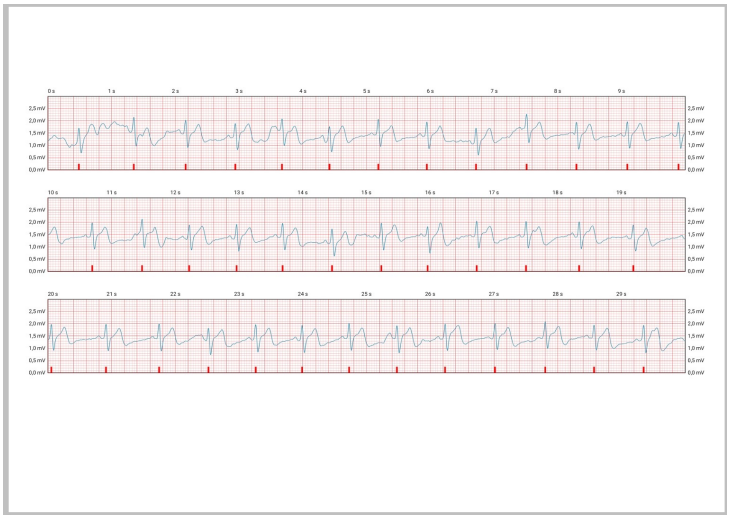
Figura 6.13: La dialog per la calibrazione della pressione

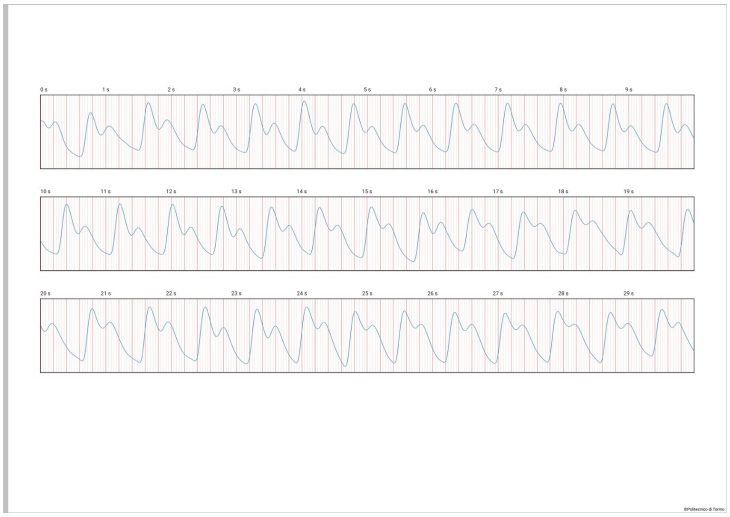
Inserendo i dati negli appositi spazi e premendo sulla conferma, vengono memorizzati i rispettivi offset calcolati come differenza tra valore reale e valore predetto. L'offset calcolato sarà automaticamente sommato algebricamente a tutte le successive predizioni: si tratta quindi di una calibrazione lineare. Eventuali calibrazioni ulteriori sono calcolate come *(calibrazione attuale * 2 + nuovo offset) / 2*.
L'opzione "Reset calibrazione" nelle impostazioni serve ad azzerare gli offset da sommare alle predizioni.

6.2.7 Altre funzionalità

Anche PulsEcg come EcgWatch fornisce la possibilità di condividere il file con i dati e il PDF ordinato con il riepilogo della misurazione, con la possibilità di includere la geolocalizzazione. L'implementazione di tali funzionalità è analoga a quella svolta per EcgWatch.
Di seguito si vuole riportare per completezza l'esempio di un PDF di una misurazione dalla durata di 30 secondi. Il documento è strutturato con un riepilogo nella prima pagina con informazioni su paziente, durata e data misurazione, i valori calcolati per battiti e saturazione e la predizione della pressione. In seconda e terza pagina vi sono rispettivamente i grafici di ECG e PPG, suddivisi su 3 righe da 10 secondi ciascuna.






0 s
1 s
2 s
3 s
4 s
5 s
6 s
7 s
8 s
9 s
10 s
11 s
12 s
13 s
14 s
15 s
16 s
17 s
18 s
19 s
20 s
21 s
22 s
23 s
24 s
25 s
26 s
27 s
28 s
29 s

Capitolo 7

Test e analisi dei risultati

7.1 Test su dispositivi Android

Come spiegato nella sezione 3.1, la quasi totalità dei produttori di smartphone Android installa una versione del sistema modificata, cioè con una propria gestione dell'interfaccia, delle proprie funzionalità, e spesso un'implementazione differente sulla gestione del ciclo di vita delle app e dell'interazione con i moduli hardware. Queste discrepanze richiedono una fase di test che verifichi la coerenza del comportamento previsto dallo sviluppatore tra più dispositivi Android, sia dal punto di vista grafico che funzionale.
Le applicazioni EcgWatch e PulsEcg sono state testate su due modelli differenti di tablet Samsung e Huawei, e su smartphone Samsung, Huawei/Honor, Xiaomi, OnePlus e LG; le versioni di Android variano da 7 a 11.
Alle versioni attuali (Ottobre 2021) delle app caricate sul Play Store, 1.5.6 per EcgWatch e 1.2 per PulsEcg, non vi sono discrepanze significative grafiche e o funzionali tra i dispositivi testati, tranne 2, i quali con PulsEcg soffrono di una problematica piuttosto grave inerente la perdita di pacchetti. I 2 smartphone che presentano questo fenomeno sono entrambi Huawei di fascia bassa: dal momento che i campioni acquisiti non vengono memorizzati nel dispositivo PulsEcg, ma inviati in tempo reale, serve una certa velocità di elaborazione da parte del Bluetooth dello smartphone, aspetto che su un telefono economico potrebbe venire trascurato, e questo fatto costituirebbe la motivazione più probabile del malfunzionamento, il quale appunto non si verifica su dispositivi di fascia medio/alta.
Un'altra analisi da considerare è la tendenza alla perdita dei pacchetti quando la batteria del dispositivo PulsEcg è quasi scarica: smartphone e tablet Android ne soffrono indistintamente dalla fascia di prezzo. Si tratta comunque di un'indagine prettamente statistica, e le cause possono essere ricercate sia nell'applicazione che nel firmware del dispositivo.

7.2 Confronto con dispositivi medici certificati

Nei capitoli precedenti sono stati mostrati alcuni esempi di misurazioni effettuate con i dispositivi EcgWatch e PulsEcg. I risultati sono certamente verosimili, ma è fondamentale capire se sono anche reali, e per questo è necessario un confronto con un *golden standard*, vale a dire un dispositivo medico certificato che sia in grado di emettere le stesse informazioni in maniera affidabile.
Sono stati esaminati i risultati di 3 dispositivi diversi:

- GE Healthcare B105 (figura 7.1), uno strumento professionale al quale è possibile collegare gli elettrodi per l'elettrocardiogramma, un pulsossimetro e una cuffia per la pressione;

Figura 7.1: Il GE Healthcare B105

- KardiaMobile 6L (figura 7.2), un dispositivo compatto dotato di 3 elettrodi in grado di misurare elettrocardiogramma fino a 6 derivazioni comunicando con l'apposita app per smartphone tramite BLE;

Figura 7.2: Il KardiaMobile 6L

- Withings ScanWatch (figura 7.3), un orologio dall'aspetto elegante con cui si possono acquisire ECG e SpO2, consultando i risultati sempre tramite app.

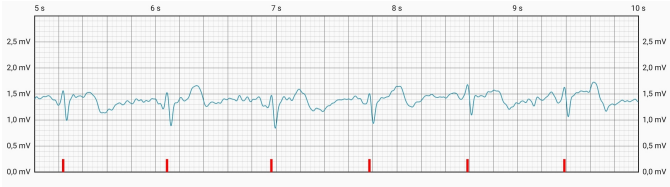
Figura 7.3: Il Withings ScanWatch

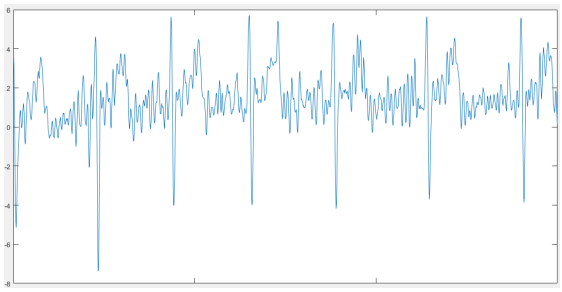
Nei paragrafi successivi viene proposto il confronto tra i dispositivi sopra citati e PulsEcg. Nel corso della tesi sono state testate in tutto più di 20 persone, ma nelle prossime sezioni il campione statistico è dato da 4 persone, più altre 4 solo per la pressione, tutte senza patologie.

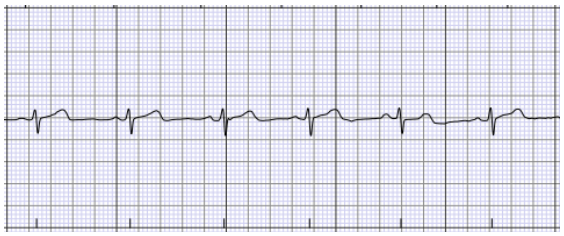
7.2.1 Analisi risultati ECG e frequenza cardiaca

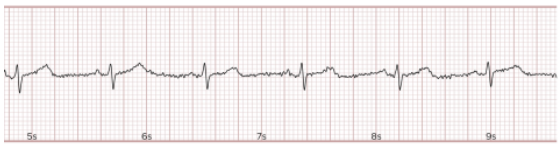
L'ECG è sicuramente la parte più delicata da validare, perché essendo che gli elettrodi sono soggetti a rumore la sfida concreta sta nel realizzare un filtraggio efficace.
Per ciascuna delle 4 persone esaminate, si riportano in ordine le misurazioni fornite da PulsEcg, GE, Kardia e Withings. In particolare, PulsEcg e GE sono state eseguite contemporaneamente, mentre le altre due in un istante immediatamente successivo.

- Primo paziente, donna di 27 anni:

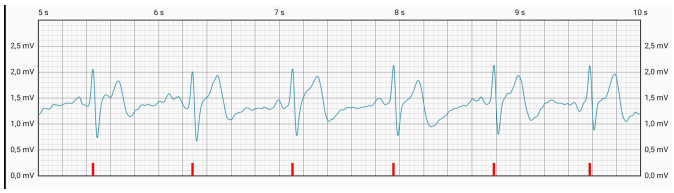


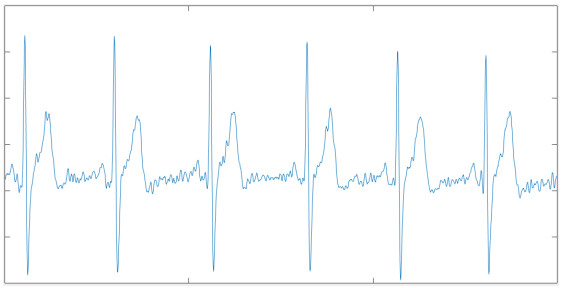


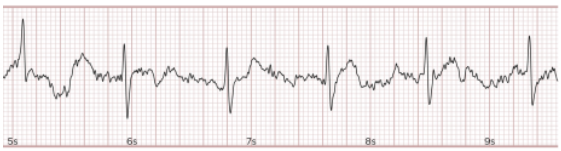




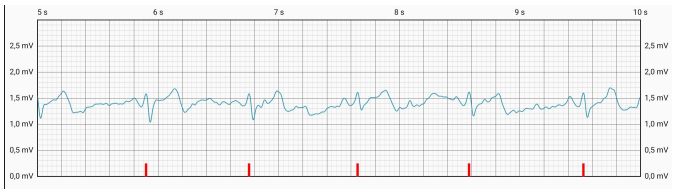
- Secondo paziente, uomo di 24 anni:

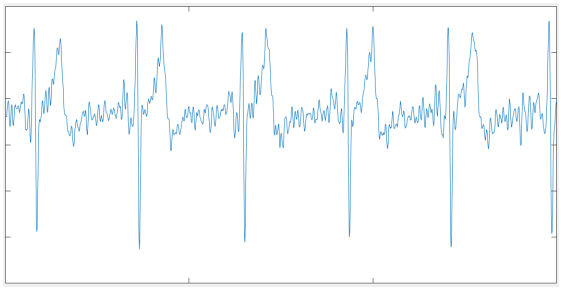


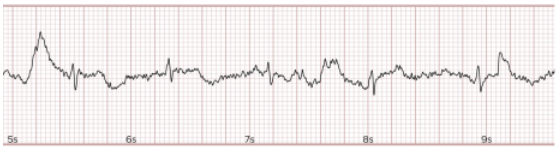




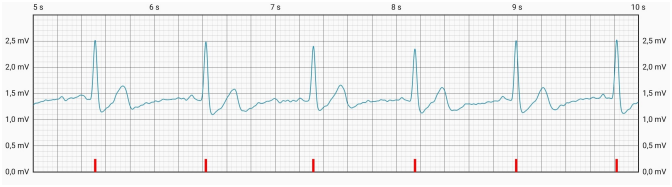
- Terzo paziente, uomo di 25 anni:

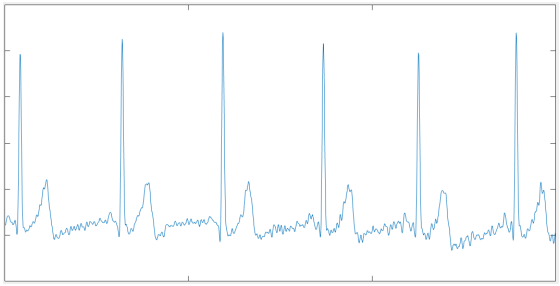


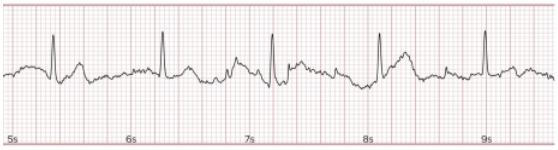




- Quarto paziente, uomo di 32 anni:







I confronti illustrano come l'andamento dei vari tracciati sia simile: ad esempio, nel quarto paziente i picchi R sono tutti molto alti e in generale il grafico è lineare, mentre per il terzo paziente il tracciato rimane rumoroso e i complessi QRS non sono particolarmente accentuati. Questi confronti attribuiscono dunque una veridicità

del risultato prodotto da PulsEcg viste le analogie con dispositivi certificati; lo spazio di miglioramento è ancora molto ampio, specialmente per pazienti che per loro natura hanno un ECG più "piatto" e soggetto ad artefatti.
Tuttavia, i dati racchiusi nella tabella 7.1 dimostrano come il calcolo della frequenza cardiaca sia affidabile, anche perché strettamente correlato al numero di picchi R individuati, il quale a sua volta dipende da un tracciato adeguatamente leggibile tale per cui l'algoritmo descritto nella sezione 4.2.5 produca il risultato corretto.
La tabella 7.2 riporta invece la media (in valore assoluto) e la varianza dello scostamento dei risultati tra i 2 dispositivi a confronto.

Paziente	Sesso	Età	BPM PulsEcg	BPM GE B105
I	Donna	27	68	70
			67	70
			73	72
II	Uomo	24	73	74
			74	74
			80	79
III	Uomo	25	65	65
			65	69
			70	72
			65	68
			68	69
			68	71
IV	Uomo	32	67	70
			71	67
			70	69
			74	74

Tabella 7.1: Risultati test calcolo della frequenza cardiaca con GE B105

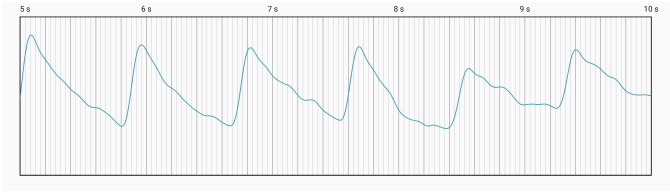
Paziente	Media	Varianza
I	1,33	2,89
II	0	0,67
III	2,17	1,81
IV	0,5	6,25
Totale	0,94	4,18

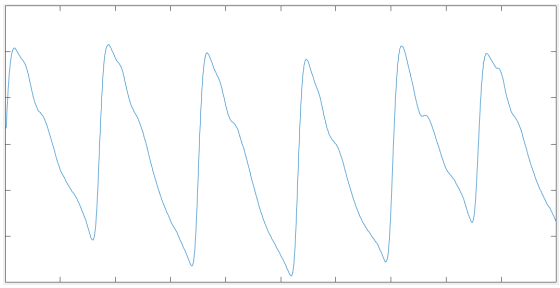
Tabella 7.2: Media e varianza degli scostamenti di BPM per paziente e totali tra PulsEcg e GE B105

7.2.2 Analisi risultati PPG e saturazione

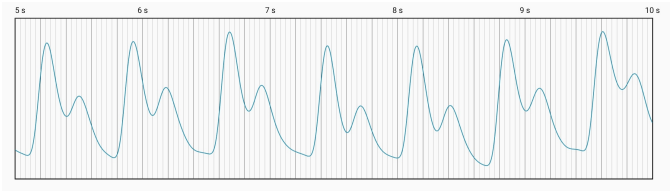
Il campione di persone in esame è lo stesso del paragrafo precedente, e per ciascuno il confronto è tra i PPG misurati rispettivamente da PulsEcg e dal GE B105.

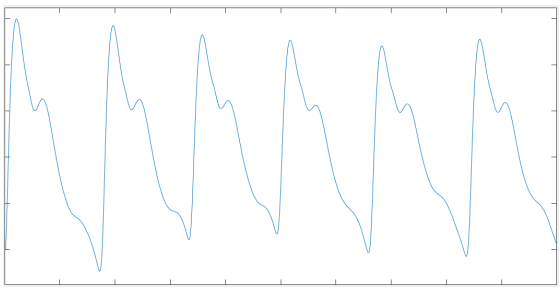
- Primo paziente, donna di 27 anni:



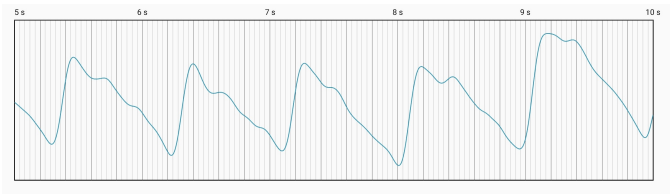


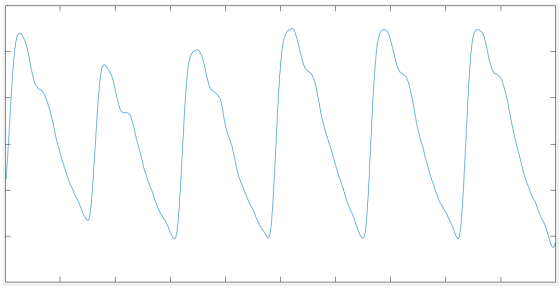
- Secondo paziente, uomo di 24 anni:



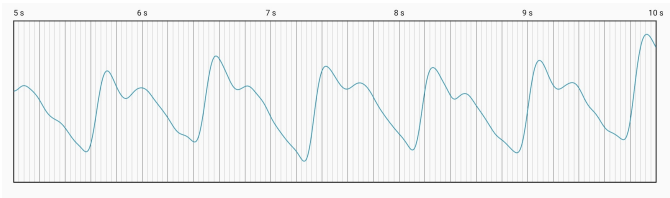


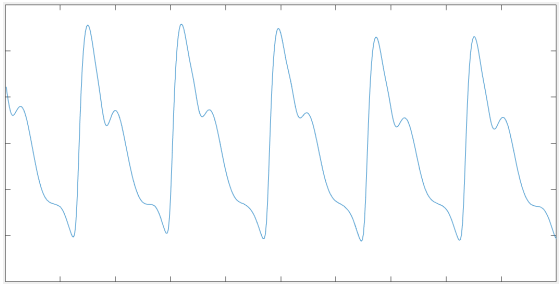
- Terzo paziente, uomo di 25 anni:





- Quarto paziente, uomo di 32 anni:





I tracciati prodotti hanno un andamento analogo: se si osserva ad esempio il picco diastolico delle onde, si nota che sul primo e sul terzo paziente è poco accentuato, mentre per secondo e quarto paziente è ben visibile, e questa caratteristica è rispettivamente concorde per tutti i pazienti su entrambi i grafici. Una differenza invece è l'ampiezza delle onde, che per PulsEcg è minore su tutti i tracciati se comparati con il GE. Questo può chiaramente influire in qualche modo sulla stima della saturazione, i cui risultati sono riportati in tabella 7.3, mentre in tabella 7.4 vi sono media e varianza degli scostamenti tra i dispositivi.

Paziente	Sesso	Età	SpO2 PulsEcg	SpO2 GE B105
I	Donna	27	98%	98%
			97%	98%
			96%	98%
II	Uomo	24	97%	97%
			97%	97%
			97%	98%
III	Uomo	25	95%	97%
			99%	98%
			93%	98%
			96%	99%
			97%	98%
			97%	98%
IV	Uomo	32	97%	98%
			96%	97%
			97%	98%
			98%	98%

Tabella 7.3: Risultati test calcolo SpO2 con GE B105

Paziente	Media	Varianza
I	1	0,67
II	0,33	0,22
III	1,83	3,47
IV	0,75	0,19
Totale	1,13	1,86

Tabella 7.4: Media e varianza degli scostamenti di SpO2 per paziente e totali tra PulsEcg e GE B105

Per il secondo e il quarto paziente, che come appena visto presentano un PPG maggiormente accentuato, le stime della saturazione sono perfettamente concordi, al più sottostimate di un punto percentuale da PulsEcg. Anche per il primo paziente la stima è accettabile, tant'è che si sta nel margine di errore del 2%. Discorso diverso per il terzo paziente, in cui si sono riscontrati errori anche del 3% e del 5%. Le ragioni vanno cercate nel tracciato: infatti, per la misurazione del terzo paziente che ha prodotto una saturazione del 93%, si presenta come in figura 7.4.

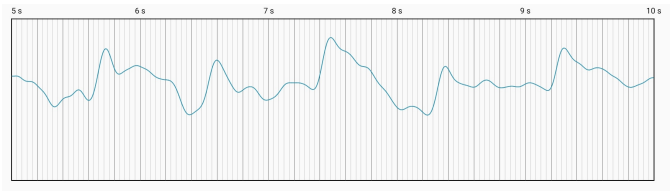


Figura 7.4: Tracciato PPG con onde poco rinoscibili

In generale, non è atipico trovare soggetti in cui il sensore per il PPG non riesce ad acquisire un buon segnale, a prescindere dal sesso e dall'età; le motivazioni non sono state prese approfonditamente in esame e potrebbero essere oggetto di sviluppi futuri.

7.2.3 Analisi risultati pressione

Al fine di attribuire una validità alle predizioni della pressione prodotte dalla rete neurale in app, sono state eseguite numerose prove su pazienti diversi.
Il primo campione è di 4 persone senza patologie, dove il confronto è stato effettuato con uno sfigmomanometro OMRON M6 COMFORT clinicamente validato; i risultati sono riportati in tabella 7.5.
Il secondo campione è analogo a quello per ECG e PPG, dove il dispositivo di confronto è sempre il GE V105; i risultati sono riportati in tabella 7.7. Tutte le misurazioni sono della durata di 10 secondi.
Gli output elaborati dalla rete sono chiaramente dipendenti dagli input forniti: le misurazioni che maggiormente si discostano dalla realtà presentano un tracciato ECG e/o PPG con un andamento non corrispondente al reale. Tuttavia, i risultati riportati sono frutto di una previa calibrazione personalizzata su ciascun paziente, confermando l'importanza di questa funzionalità per garantire, in situazioni di relax, informazioni attendibili; per il paziente I persistono comunque differenze importanti.
Le tabelle 7.6 e 7.8 riportano media e varianza della differenza dei risultati rispettivamente tra PulsEcg e OMRON e tra PulsEcg e GE.

Paziente	Sesso	Età	Durata misurazione	Predizione	Sfigmomanometro
V	Uomo	24	10 secondi	101 / 68	103 / 69
			10 secondi	108 / 68	107 / 67
			30 secondi	101 / 63	109 / 70
			30 secondi	111 / 70	111 / 72
VI	Uomo	20	10 secondi	96 / 60	95 / 64
			10 secondi	95 / 56	96 / 60
			30 secondi	95 / 65	94 / 63
			30 secondi	95 / 60	91 / 59
VII	Donna	55	10 secondi	93 / 66	99 / 73
			10 secondi	98 / 69	100 / 72
			30 secondi	94 / 69	97 / 72
			30 secondi	96 / 73	102 / 73
VIII	Uomo	61	10 secondi	108 / 67	105 / 62
			10 secondi	101 / 61	104 / 65
			30 secondi	102 / 64	103 / 62
			30 secondi	105 / 64	103 / 62

Tabella 7.5: Risultati test pressione con sfigmomanometro OMRON M6 COMFORT

Paziente	Media	Varianza
V	2,25 / 2,25	12,19 / 8,69
VI	1,25 / 1,25	3,19 / 7,69
VII	4,25 / 3,25	3,19 / 6,19
VIII	0,25 / 1,25	5,69 / 10,69
Totale	1,25 / 1,13	10,69 / 11,73

Tabella 7.6: Media e varianza degli scostamenti di pressione per paziente e totali tra PulsEcg e OMRON M6 COMFORT

Paziente	Sesso	Età	Predizione	GE B105
I	Donna	27	103 / 61	92 / 58
			107 / 57	91 / 59
II	Uomo	24	110 / 60	110 / 63
			112 / 64	111 / 61
III	Uomo	25	113 / 64	117 / 74
			115 / 77	112 / 72
			122 / 73	119 / 74
			114 / 72	120 / 74
			110 / 76	111 / 69
IV	Uomo	32	117 / 79	124 / 79
			129 / 86	126 / 78
			118 / 77	131 / 77
			126 / 72	123 / 76

Tabella 7.7: Risultati test pressione con GE B105

Paziente	Media	Varianza
I	13,5 / 0,5	6,25 / 6,25
II	0,5 / 0	0,25 / 9
III	1 / 0,2	13,2 / 35,76
IV	3,5 / 1	46,75 / 19
Totale	0,69 / 0,31	52,21 / 22,21

Tabella 7.8: Media e varianza degli scostamenti di pressione per paziente e totali tra PulsEcg e GE B105

Capitolo 8

Conclusioni

A valle delle considerazioni effettuate e del lavoro descritto nei capitoli precedenti, si può suddividere quanto svolto nel corso della tesi in 4 parti principali:

1. Sviluppo e test dell'applicazione Android: sebbene le applicazioni non includano parti decisamente complesse da realizzare, richiedono comunque una conoscenza abbastanza ampia e approfondita inerenti il mondo Android, il protocollo Bluetooth, le librerie e le metodologie da utilizzare;

2. Studio degli algoritmi: sono da intendersi gli algoritmi per filtraggi, rilevamento dei picchi nell'ECG, conta battiti e stima saturazione, ma anche tentativi non integrati in app di proporre un calcolo puntuale in sostituzione alla rete neurale per la determinazione della pressione basandosi prettamente sull'elaborazione matematica dei tracciati ECG e PPG;

3. Integrazione della rete neurale: il mondo del deep learning è distante dalla programmazione "tradizionale", e ha richiesto uno sforzo per comprendere innanzitutto le fondamenta teoriche, seguito da una analisi di quanto svolto nella tesi passata che aveva l'obiettivo di studiare una struttura per la rete neurale, per poi procedere con il riaddestramento in Python, e infine conversione in Tensorflow Lite e inferenza del modello con ricampionamento e normalizzazione dei segnali in input;

4. Validazione dei risultati: certamente la parte più interessante e fondamentale per attribuire un valore al progetto, dal momento che l'obiettivo finale dello studio è quello di proporre un prodotto innovativo, ma allo stesso tempo affidabile dal punto di vista medico, senza però sostituire in alcun modo il parere di un dottore e senza fornire soluzioni di natura alcuna all'eventuale problema rilevato.

8.1 Sviluppi futuri

Per quanto concerne EcgWatch, il dispositivo si trova verosimilmente in una versione definitiva o quasi, e la stessa cosa si può dire per l'app Android: infatti, per quest'ultima le mancanze non sono tanto funzionali, bensì strutturali, come ad esempio la ridondanza di activity.
La questione cambia per PulsEcg, dove l'eleganza del codice è una questione marginale, ma le problematiche e le mancanze principali sono da affrontare con rigore e perlopiù a fianco di un hardwarista. In ordine si sono:

1. perdita dei pacchetti nella trasmissione Bluetooth, come spiegato nella sezione 7.1;
2. la percentuale della batteria del dispositivo viene trasmessa solo quando si fanno acquisizioni, dovrebbe essere inviata automaticamente all'app non appena viene eseguita la connessione;
3. non è possibile interrompere un'acquisizione, o meglio, è possibile farlo dal punto di vista dell'interfaccia per l'utente, ma allo stato attuale il dispositivo continua ad inviare indipendentemente i pacchetti;
4. i filtri dell'ECG devono essere modificati e migliorati;
5. il tracciato del PPG è talvolta distante dalla realtà risultando troppo piatto e con un andamento atipico (verificare di conseguenza l'efficacia dell'algoritmo per la determinazione dell'SpO2);
6. l'algoritmo conta picchi per l'ECG funziona bene, ma ha ancora delle debolezze. Anche in questo contesto potrebbe essere interessante prendere in considerazione lo sviluppo di una rete neurale dedicata, in modo da contemplare anche i casi particolari che l'algoritmo fatica a riconoscere [41];
7. la calibrazione lineare della stima della pressione è una buona soluzione seppur non definitiva. Ciò che dovrebbe essere implementato è un apprendimento in-app, ossia il modello Tensorflow Lite deve proseguire con l'apprendimento ogni volta che l'utente inserisce dei nuovi valori di calibrazione della pressione. La realizzazione di tale meccanismo non è complessa, ma richiede necessariamente un nuovo addestramento della rete che preveda nel dataset di training esclusivamente la presenza di campioni acquisiti direttamente con il PulsEcg e opportunamente filtrati: questo per evitare che i dati in input siano soggetti a un bias.

Queste sono le problematiche su cui i futuri sviluppi dovranno concentrarsi in primis. Non è stata citata l'assenza di una memoria per il buffering dei campioni,

perché ciò richiederebbe la riprogettazione dell'hardware, ma è comunque indispensabile in futuro la numerazione dei pacchetti trasmessi via Bluetooth, questo perché in caso di perdita o desincronizzazione (assolutamente possibile) si possa agire di conseguenza.
Un'implementazione nuova interessante sarebbe l'opportunità per l'utente di registrare più derivazioni: con soli 2 elettrodi è ovviamente impossibile acquisirne più di una alla volta, ma spostando il dispositivo dal polso alla caviglia sinistra è possibile ottenere le derivazioni periferiche come schematizzato dal triangolo di Einthoven (vedere sezione 2.3.1). Altre proposte che riguardano strettamente il lato applicativo sono la creazione di un sistema di accounting con differenziazione dei ruoli medico/paziente e la possibilità per quest'ultimo di inviare direttamente dall'app (senza quindi utilizzare email o altro per la condivisione) l'esito di una misurazione e ricevere un commento; ancora, con la creazione dell'account, la possibilità di salvare in cloud le proprie misurazioni per averle sempre disponibili su qualsiasi smartphone o tablet, ma ciò richiederebbe la creazione di un servizio con costi sia in termini di sforzo che economici, che varrà la pena fare quando (e soprattutto se) il progetto non sarà più soltanto una proposta di tesi, ma una startup di successo.

Bibliografia

[1] Williams Pelegrin. *Android Wear: Google reveals new watch details at I/O*. Giugno 2014. URL: https://www.digitaltrends.com/mobile/android-wear-os-news-release-features/ (cit. a p. 1).

[2] Vlad Savov. *APPLE WATCH ANNOUNCED: AVAILABLE FOR $349 EARLY NEXT YEAR*. Settembre 2014. URL: https://www.theverge.com/2014/9/9/6125873/apple-watch-smartwatch-announced (cit. a p. 1).

[3] Garzanti. *l'Universale*. In: vol. 11, p. 438 (cit. a p. 3).

[4] Garzanti. *l'Universale*. In: vol. 11, pp. 438–439 (cit. a p. 4).

[5] Br Heart J. «Pathology of atrial fibrillation in man». In: *British heart journal* 34 (5) (Maggio 1972), pp. 520–525. DOI: https://doi.org/10.1136/hrt.34.5.520 (cit. a p. 5).

[6] Vincent M. Christoffels e Antoon F.M. Moorman. «Development of the Cardiac Conduction System». In: *Circulation: Arrhythmia and Electrophysiology* 2 (2) (apr. 2019). DOI: https://doi.org/10.1161/CIRCEP.108.829341 (cit. a p. 5).

[7] Eric B. Bass, Edward I. Curtiss e Vincent C. Arena. «The Duration of Holter Monitoring in Patients With Syncope». In: *Arch Intern Med. 1990* 150 (5) (Maggio 1990), pp. 1073–1078. DOI: https://doi.org/10.1001/archinte.1990.00390170103022 (cit. a p. 6).

[8] Ary L. Goldberger, Zachary D. Goldberger e Alexei Shvilkin. *Chapter 3 - ECG Leads*. In: vol. Goldberger's Clinical Electrocardiography (Eighth Edition). 2013, pp. 16–25. DOI: https://doi.org/10.1016/B978-0-323-08786-5.00003-8 (cit. a p. 7).

[9] Daniel E. Becker. «Fundamentals of electrocardiography interpretation». In: *Anesthesia progress* 53, 2 (2006), pp. 53–63. DOI: https://doi.org/10.2344/0003-3006(2006)53[53:FOEI]2.0.CO;2 (cit. a p. 9).

[10] Thinn Hlaing M.D., Tara DiMino M.D., Peter R. Kowey M.D., F.A.C.C., Gan-Xin Yan M.D. e Ph.D. «ECG Repolarization Waves: Their Genesis and Clinical Implications». In: *Noninvasive Electrocardiology* 10 (2) (apr. 2005), pp. 211–223. DOI: https://doi.org/10.1111/j.1542-474X.2005.05588.x (cit. a p. 11).

[11] D. Castaneda, A. Esparza, M. Ghamari, C. Soltanpur e H. Nazeran. «A review on wearable photoplethysmography sensors and their potential future applications in health care». In: *Int J Biosens Bioelectron* 4(4) (Agosto 2018), pp. 195–202. DOI: https://doi.org/10.15406/ijbsbe.2018.04.00125 (cit. a p. 12).

[12] Elgendi Mohamed. «On the analysis of fingertip photoplethysmogram signals». In: Current cardiology reviews vol. 8,1 (2012). DOI: https://doi.org/10.2174/157340312801215782 (cit. a p. 14).

[13] Ahmad Fikri, Alfan Presekal, Ruki Harwahyu e Riri Fitri Sari. *Performance Comparison of Dalvik and ART on Different Android-Based Mobile Devices*. In: vol. 2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI). 2018. DOI: https://doi.org/10.1109/ISRITI.2018.8864290 (cit. a p. 20).

[14] JetBrains. *Kotlin for Android*. English. JetBrains. 11 Feb. 2021 (cit. a p. 20).

[15] Lorenzo Miglietta. *Che cos'è il material design*. Mar. 2015 (cit. a p. 24).

[16] Jesse James Garrett. *The Elements of User Experience*. New Riders Publishing, 2011 (cit. a p. 25).

[17] Md Sajjad Hosain. «Huawei ban in the US: Projected consequences forinternational trade». In: *International Journal of Commerce and Economics* 1 (2) (mar. 2019), pp. 22–25 (cit. a p. 26).

[18] Google. *AndroidX Overview*. URL: https://developer.android.com/jetpack/androidx (cit. a p. 26).

[19] *Gson*. URL: https://github.com/google/gson (cit. a p. 26).

[20] *CircularImageView*. URL: https://github.com/lopspower/CircularImageView (cit. a p. 26).

[21] Google. *Set up Google Play services*. English (cit. a p. 26).

[22] Google. *Tensorflow Lite*. URL: https://www.tensorflow.org/lite (cit. a p. 26).

[23] *MPAndroidChart*. URL: https://github.com/PhilJay/MPAndroidChart (cit. a p. 26).

[24] *Get BLE scan result using 'BluetoothLeScanner.startScan' must open Location Service in Android 6.0*. 2015. URL: https://issuetracker.google.com/issues/37065090 (cit. a p. 28).

[25] Simone Valvo. «Sviluppo di un'interfaccia Android per device medicali per la misurazione non-invasiva e senza cuffia della pressione arteriosa». Tesi di laurea mag. Politecnico di Torino, apr. 2021 (cit. a p. 30).

[26] Google. *BluetoothDevice*. URL: https://developer.android.com/reference/android/bluetooth/BluetoothDevice (cit. a p. 33).

[27] Jiapu Pan e Willis J. Tompkins. «A Real-Time QRS Detection Algorithm». In: *IEEE Transactions on Biomedical Engineering* BME-32 (3) (mar. 1985), pp. 230–236. DOI: https://doi.org/10.1109/TBME.1985.325532 (cit. a p. 38).

[28] Google. *Geocoder*. DOI: https://developer.android.com/reference/android/location/Geocoder (cit. a p. 45).

[29] R. Wang, W. Jia, Z.H. Mao, R.J. Sclabassi e M. Sun. «Cuff-Free Blood Pressure Estimation Using Pulse Transit Time and Heart Rate». In: *2014 12th International Conference on Signal Processing (ICSP)*. Ottobre 2014. DOI: https://doi.org/10.1109/ICOSP.2014.7014980 (cit. a p. 47).

[30] D. Oreggia e S. Guarino. «Physiological parameters measurements in a cardiac cycle via a combo PPG-ECG system». In: *2015 AEIT International Annual Conference (AEIT)*. Ottobre 2015. DOI: https://doi.org/10.1109/AEIT.2015.7415214 (cit. a p. 48).

[31] Saad Albawi, Tareq Abed Mohammed e Saad Al-Zawi. «Understanding of a convolutional neural network». In: *2017 International Conference on Engineering and Technology (ICET)*. Agosto 2017. DOI: 10.1109/ICEngTechnol.2017.8308186 (cit. a p. 49).

[32] D. Mandic e Jonathon Chambers. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. Wiley, 2001 (cit. a p. 49).

[33] Google. *Introduzione ai tensori*. URL: https://www.tensorflow.org/guide/tensor (cit. a p. 50).

[34] Yong Yu, Xiaosheng Si, Changhua Hu e Jianxun Zhang. «A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures». In: *Neural Computation (2019)* 31 (7) (Luglio 2019), pp. 1235–1270. DOI: https://doi.org/10.1162/neco_a_01199 (cit. a p. 50).

[35] Stefano Villata. «Cuffless Blood Pressure Measurement». Tesi di laurea mag. Politecnico di Torino, Settembre 2020 (cit. a p. 50).

[36] Keras. *Keras*. URL: https://keras.io/ (cit. a p. 50).

[37] A. Paviglianiti, V. Randazzo, S. Villata, G. Cirrincione e E. Pasero. «A comparison of Deep Learning techniques for Arterial Blood Pressure prediction». In: *Cognitive Computation (2021)* (Agosto 2021). DOI: `https://doi.org/10.1007/s12559-021-09910-0` (cit. a p. 52).

[38] Google. *Migrare da TensorFlow 1.x a TensorFlow 2*. URL: `https://www.tensorflow.org/guide/migrate` (cit. a p. 52).

[39] Google. *Formazione sul dispositivo in TensorFlow Lite*. URL: `https://www.tensorflow.org/lite/examples/on_device_training/overview` (cit. a p. 55).

[40] Google. *Bound Services Overview*. URL: `https://developer.android.com/guide/components/bound-services#java` (cit. a p. 65).

[41] H.M. Rai e A. Trivediand K. Chatterjee. «R-Peak Detection using Daubechies Wavelet and ECG Signal Classification using Radial Basis Function Neural Network». In: *Journal of The Institution of Engineers (India): Series B* 95 (Maggio 2014), pp. 63–71. DOI: `https://doi.org/10.1007/s40031-014-0073-4` (cit. a p. 91).