

Elaborazione di segnali ECG trasformati in immagini RGB

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Informatica
Corso di laurea in Informatica

Gianluca Circelli
Matricola 1792550



Relatore

Maria De Marsico


Ringraziamenti

Prima di iniziare vorrei ringraziare chi mi ha permesso di essere qui e di raggiungere questo risultato. Ringrazio la relatrice Maria De Marsico che mi ha aiutato durante tutta la durata del tirocinio ed è sempre stata disponibile. Un ringraziamento speciale va poi ai miei genitori e a mio fratello, che mi hanno sempre sostenuto durante i momenti più difficili.

Questo traguardo non sarebbe stato possibile senza di voi.

Indice

| | | |
|----------|--|-----------|
| 1 | Introduzione | 5 |
| 2 | | 6 |
| 2.1 | Derivazioni e analisi delle onde e dei picchi del battito cardiaco | 6 |
| 3 | | 10 |
| 3.1 | Elaborazione dei dati | 11 |
| 3.2 | Pulizia del segnale e scelta delle derivazioni | 13 |
| 3.3 | Individuazione dei picchi R | 15 |
| 3.4 | Segmentazione in battiti | 16 |
| 3.5 | Creazione delle immagini caratteristiche | 20 |
| 3.6 | Organizzazione del dataset | 21 |
| 3.7 | Train, Validation e Test set | 23 |
| 3.8 | Caricamento delle immagini | 25 |
| 3.9 | Creazione della rete | 27 |
| 3.9.1 | Le reti neurali | 27 |
| 3.9.2 | La rete | 29 |
| 3.10 | Valutazione delle performance | 34 |
| 3.11 | Conclusioni | 41 |

Elenco delle figure

| | | |
|------|--|----|
| 2.1 | Triangolo di Einthoven | 7 |
| 2.2 | Derivazioni in 3D con i relativi angoli | 7 |
| 2.3 | Schema di un battito cardiaco | 9 |
| 3.1 | Distribuzione età dei pazienti | 12 |
| 3.2 | Distribuzione sesso dei pazienti | 12 |
| 3.3 | Segnale ECG filtrato dalle librerie Biosppy e Neurokit2 . . . | 13 |
| 3.4 | Ecg a 12 derivazioni dove si può osservare l'aumento del picco R nelle derivazioni precordiali | 14 |
| 3.5 | ECG con i picchi R in rosso rilevati | 16 |
| 3.6 | Rappresentazione dei margini di ogni battito rilevato | 17 |
| 3.7 | Battito 1 segmentato | 17 |
| 3.8 | Battito 2 segmentato | 17 |
| 3.9 | Battito 3 segmentato | 18 |
| 3.10 | Battito 4 segmentato | 18 |
| 3.11 | Battito 5 segmentato | 18 |
| 3.12 | Battito 6 segmentato | 18 |
| 3.13 | Battito 7 segmentato | 18 |
| 3.14 | Battito 8 segmentato | 18 |
| 3.15 | Battito 9 segmentato | 19 |
| 3.16 | Battito 10 segmentato | 19 |
| 3.17 | Immagine caratteristica ecg_id 00008 | 20 |
| 3.18 | Immagine caratteristica ecg_id 00018 | 20 |
| 3.19 | Immagine caratteristica ecg_id 00020 | 21 |
| 3.20 | Distribuzione degli ecg normali e anomali | 22 |
| 3.21 | L'overfit è un problema che può essere visualizzato su grafico sia sul validation set che sul test set. In questo esempio sul test set si può vedere che l'accuratezza del train è alta ma sul test set man mano diminuisce mentre il grafico dell'errore tra la predizione e la classe corretta è esattamente al contrario. | 24 |
| 3.22 | Immagine dell'ECG 00008 ridimensionata alle dimensioni usate nella rete, 110 x 110. Questa immagine ha le dimensioni reali. L'immagine nella rete ha i valori scalati tra 0 e 1 (questa ha ancora i valori tra 0 e 255 per i 3 canali) | 26 |

| | | |
|------|--|----|
| 3.23 | Immagine dell'ECG 00017 ridimensionata alle dimensioni usate nella rete, 110 x 110. Questa immagine ha le dimensioni reali. L'immagine nella rete ha i valori scalati tra 0 e 1 (questa ha ancora i valori tra 0 e 255 per i 3 canali) | 26 |
| 3.24 | Esempio di rete neurale classica. Ogni cerchio rappresenta un neurone | 28 |
| 3.25 | Esempio di rete neurale CNN per la classificazione di immagini in input | 29 |
| 3.26 | Esempio di funzionamento del MaxPooling2D | 30 |
| 3.27 | Funzione ReLU usata nella rete | 31 |
| 3.28 | Funzione sigmoide usata nell'ultimo strato della rete per la classificazione | 31 |
| 3.29 | Architettura della rete, con output shape e numero di parametri per ogni strato | 32 |
| 3.30 | Plot dell'andamento dell'accuratezza e della loss durante il training | 33 |
| 3.31 | Matrice di confusione prodotta dal test della rete | 34 |
| 3.32 | Curva ROC prodotta (in blu). La retta rossa invece mostra la la ROC di un classificatore casuale | 36 |
| 3.33 | Confronto dell'accuratezza media per il rilevamento di 2 (celeste), 5 (arancione) e 20 (verde) classi | 37 |
| 3.34 | Confronto del punteggio F1 medio nel rilevamento di 2 (celeste), 5 (arancione) e 20 (verde) classi | 37 |
| 3.35 | Valori di accuratezza, F1 e AUC per la classificazione a 2 classi | 38 |
| 3.36 | Matrice di confusione per classificazione binaria con softmax | 39 |
| 3.37 | Matrice di confusione per classificazione binaria con Few-Shot | 39 |

Elenco delle tabelle

| | |
|---|----|
| 3.1 Metriche prodotte durante il test del modello | 35 |
|---|----|

Capitolo 1

Introduzione

Questa relazione riporta il lavoro svolto durante il tirocinio sulla classificazione di segnali ECG. L'obiettivo è quello di riuscire a distinguere segnali di elettrocardiogramma anomali da quelli normali con una buona precisione. Per farlo però ci si è discostati dall'analisi del segnale vero e proprio così com'è: infatti si è voluto procedere analizzando le immagini, codificanti ognuna un preciso ECG, e usare un modello di machine learning per riconoscerne quelle anomale e quelle normali. Nello specifico sono state selezionate 3 tra le 12 derivazioni, le quali sono state usate per la creazione delle immagini. Queste sono state scelte dopo aver visto che le anomalie sul ritmo cardiaco sono più presenti tra le derivazioni 1, 2 e 3 mentre le altre patologie principalmente sulle derivazioni precordiali. A seguire sono state quindi selezionate le 3 con i picchi R più alti, in modo da riconoscere correttamente ogni singolo battito. Per quanto riguarda invece i segnali usati per creare tali immagini è stato usato un dataset dove ogni tracciato ha associato la propria etichetta (cioè il **ground truth**) che indica se è normale o meno. Questo lavoro ha portato ad un'accuratezza di oltre l'80%. In seguito vengono spiegati in dettaglio tutti gli step con le relative scelte.

Capitolo 2

2.1 Derivazioni e analisi delle onde e dei picchi del battito cardiaco

L'elettrocardiogramma permette di avere un tracciato dell'attività elettrica del cuore al fine di individuare eventuali anomalie, come alterazioni del ritmo cardiaco, della conduzione elettrica del cuore e della morfologia dell'onda, applicando 10 elettrodi sugli arti e sul costato. Infatti l'onda del battito è la cartina tornasole del corretto funzionamento degli atri e ventricoli ed in condizioni sane ha l'aspetto come riportato in Immagine 3. La linea isoelettrica è la linea orizzontale che rappresenta l'assenza di attività. Tuttavia dal punto di vista elettrico il cuore viene rappresentato come un vettore, il quale ha una direzione che in condizioni normali va in media in basso e a sinistra. La sua direzione, così come l'intensità, varia durante il singolo battito e per questo si hanno 12 derivazioni che permettono di misurarle. Ogni derivazione è un vettore con una direzione, che si riferisce alla direzione e all'orientamento del flusso di corrente elettrica dal polo negativo al positivo ed è indicata in gradi, e permettono insieme di misurare il vettore cardiaco (detto asse cardiaco) da più punti di osservazione. Vengono misurate sfruttando gli elettrodi appoggiati sul corpo del paziente e si dividono in 6 derivazioni degli arti sul piano frontale (I, II, III, aVR, aVL, aVF) e 6 derivazioni precordiali sul piano orizzontale (V1, V2, V3, V4, V5, V6). Le derivazioni degli arti, anche dette periferiche, vengono divise in:

- Derivazioni bipolari standard (I, II, III): misurano la differenza di potenziale tra due elettrodi sugli arti: I misura la differenza di potenziale tra braccio destro (polo negativo) e sinistro (polo positivo), II misura la differenza di potenziale tra braccio destro (polo negativo) e gamba sinistra (polo positivo) e III tra braccio sinistro (polo negativo) e gamba sinistra (polo positivo). Il vettore I ha direzione 0° , II ha direzione di 60° e III ha direzione di 120° ;
- Derivazioni monopolar aumentate (aVR, aVL, aVF): misurano la differenza di potenziale tra un punto a potenziale 0 (polo negativo, di solito viene usata la gamba destra) e i 3 arti (polo positivo) e si amplificano: aVR è la rappresentazione quindi del potenziale assoluto del braccio destro, aVL è la rappresentazione quindi del potenziale

assoluto del braccio sinistro e aVF è la rappresentazione quindi del potenziale assoluto della gamba sinistra. Il vettore aVR ha direzione -150° , aVL ha direzione di -30° e aVF ha direzione di 90° .

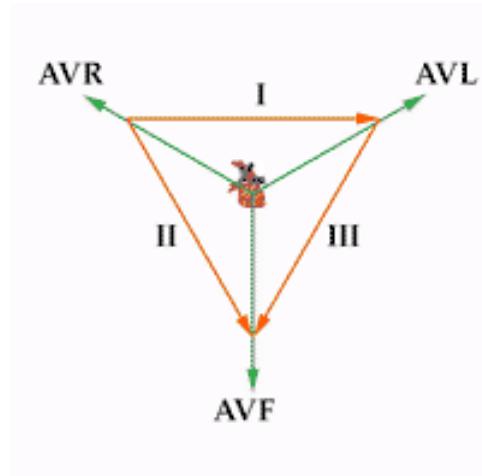


Figura 2.1: Triangolo di Einthoven

Le derivazioni periferiche formano il triangolo di Einthoven, Figura 2.1. Le derivazioni precordiali invece sono sul piano orizzontale e calcolano il potenziale assoluto in 6 punti, disposti sul torace del paziente. I 6 elettrodi fungono da poli positivi e anche queste quindi si rappresentano come vettori, da V₁ a V₆.

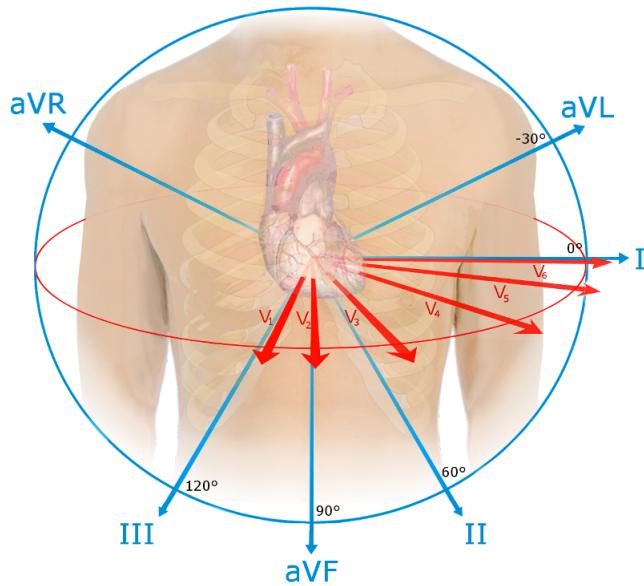


Figura 2.2: Derivazioni in 3D con i relativi angoli

Come detto ognuna delle 12 derivazioni misura l'intensità (e direzione) del vettore medio cardiaco, detto asse cardiaco, dal suo punto di vista. Nello specifico più questo è parallelo al vettore di una derivazione e più sarà

marcata la deflessione positiva sul tracciato di quella derivazione. Si ricorda infatti che ogni derivazione, essendo un vettore, ha una direzione, come visibile in Figura 2.2 e più il vettore dell'asse cardiaco è parallelo al vettore di una specifica derivazione e più sarà visibile in quella derivazione un'onda positiva. Pertanto, in condizioni ottimali si ha un'onda come riportato in Figura 2.3. Si possono distinguere diverse onde e intervalli, nello specifico:

- Un'onda P: è una piccola deflessione positiva che rappresenta la depolarizzazione delle camere superiori del cuore, chiamate atri. Questo è il momento in cui gli atri si contraggono per pompare il sangue nelle camere inferiori, chiamate ventricoli.
- Segmento PR: è la linea isoelettrica dove avviene il passaggio dell'attività elettrica dagli atri ai ventricoli;
- Complesso QRS: è visibile come una piccola deflessione negativa Q, seguita da una positiva R e una negativa S. Rappresenta la depolarizzazione delle camere inferiori del cuore, chiamate ventricoli. Questo è il momento in cui i ventricoli si contraggono per pompare il sangue fuori dal cuore e nel resto del corpo.
- Segmento ST: è la linea isoelettrica tra S e T e indica l'intervalle tra la depolarizzazione ventricolare e la ripolarizzazione ventricolare;
- Onda T: è l'ultima deflessione positiva che rappresenta la ripolarizzazione delle camere inferiori del cuore, chiamate ventricoli. Questo è il momento in cui i ventricoli si rilassano e si riempiono di sangue per prepararsi al prossimo battito cardiaco.
- Intervallo PR: è intervallo tra l'inizio dell'onda P e l'inizio del complesso QRS e rappresenta l'attività elettrica atriale;
- Intervallo QT: è intervallo tra l'inizio del complesso QRS e la fine dell'onda T e rappresenta l'attività elettrica ventricolare;
- Onda U: è subito successiva all'onda T ma non sempre presente. Rappresenta anch'essa la ripolarizzazione ventricolare.

Da un ECG è possibile ricavare un gran numero di aspetti. La più semplice è la frequenza cardiaca. Indica il numero di volte che il muscolo cardiaco esercita la propria attività nell'arco di un minuto. Nell'adulto in condizioni di riposo, la frequenza cardiaca normale è generalmente compresa tra i 60 e i 100 battiti al minuto. In un ECG si può calcolare prendendo i picchi R presenti e calcolare proporzionalmente quanti ce ne sono in 60 secondi.

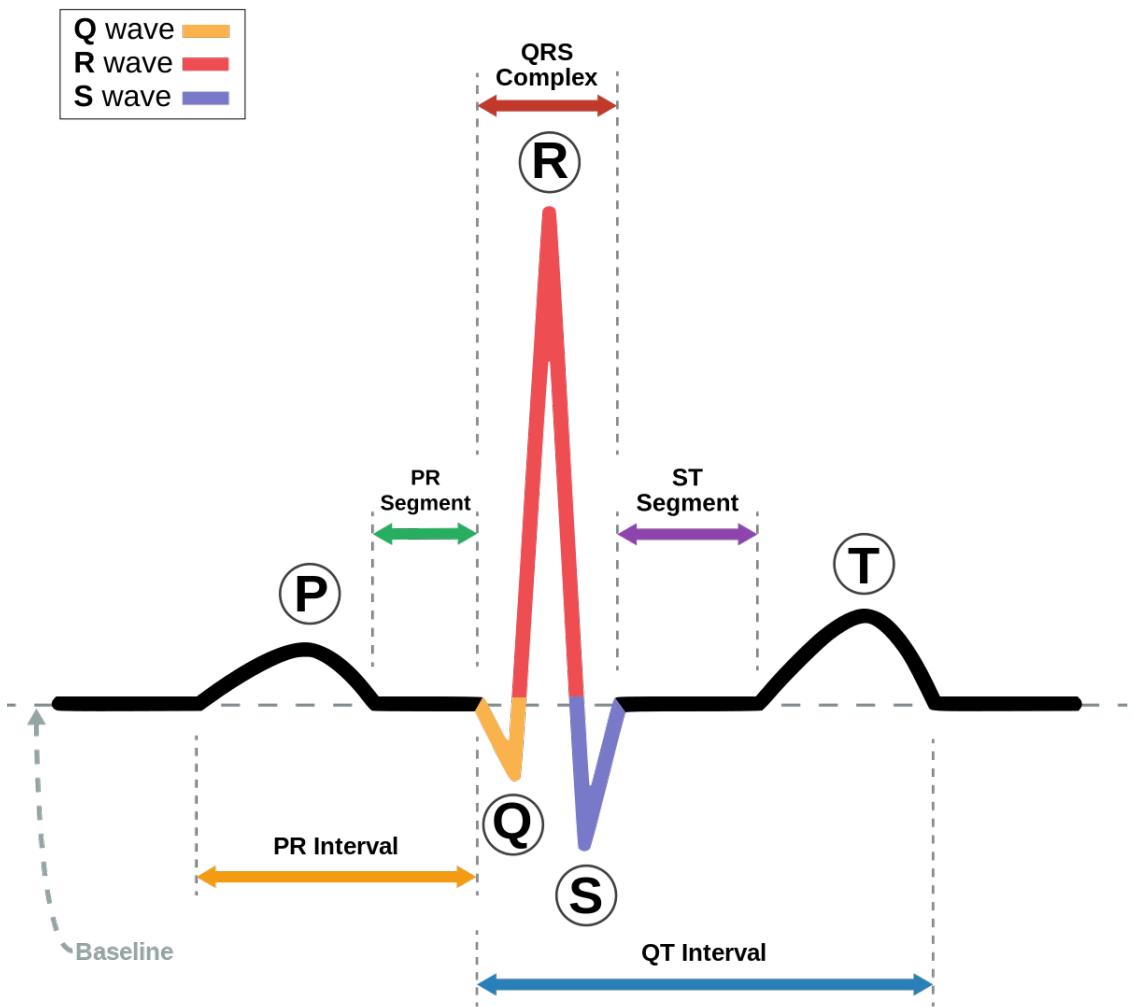


Figura 2.3: Schema di un battito cardiaco

Capitolo 3

In questo capitolo vengono descritte in dettaglio le procedure e scelte che sono state prese nel corso del tirocinio. La prima scelta è ad esempio quella della libreria da usare per l'eleborazione dei segnali degli ECG, dato che è fondamentale poter aver una libreria performante per trattare questo tipo di dati. Pertanto inizialmente è stata usata la libreria suggerita dalla documentazione del dataset, ovvero Biosppy [1], un toolbox per il processo di biosegnali come appunto gli ECG. Tuttavia, come verrà riportato anche più avanti, non si è rilevata la scelta migliore in quanto non riusciva ad esempio a identificare bene i picchi R o ad eliminare il rumore e oscillazione nei segnali che lo avevano (Figura 3.3), e quindi si è passati all'utilizzo escusivo di Neurokit2 [5], molto più preciso in tal senso. Questa è una libreria user-friendly per l'elaborazione dei biosegnali, usata molto spesso anche da ricercatori medici per la sua semplicità di programmazione. Quest'ultima infatti ha diversi metodi, con nomi esplicativi che sono stati usati in seguito: infatti tutti i nomi dei metodi che iniziano con `ecg_` sono proprio di Neurokit2.

3.1 Elaborazione dei dati

Il lavoro di tirocinio è iniziato scegliendo un dataset di ECG. Si è scelto di utilizzare PTB-XL [9] in quanto fornisce una grande quantità e varietà di tracciati. Infatti questo è stato creato per fornire un insieme di dati più ampio e completo rispetto ad altri dataset ECG pubblicamente disponibili, essendo composto da 21799 ECG di 18869 pazienti, divisi in 52% maschi e 48% donne e coprendo la fascia di età tra 0 e 95 anni. All'interno della directory vengono forniti tra gli altri:

- Il file `ptbxl_database.csv`: è un file .csv (**Comma Separated Values**) nel quale sono riportati tutti i dati riguardo il singolo ECG, tra i quali un identificatore univoco del tracciato, del paziente, dell'infermiere coinvolto, del sito di registrazione, età, sesso, altezza, peso del paziente, un rapporto del medico ed altre informazioni. Tra queste ci sono anche il nome del file, in formato WFDB (**WaveForm DataBase**), del tracciato a 500 Hz e a 100 Hz, sebbene i segnali originali siano a 400 Hz. Si è scelto di utilizzare i file a 100 Hz poiché garantiscono un buona velocità di elaborazione, mantenendo comunque una approssimazione affidabile;
- Due cartelle `records_100` e `records_500`: queste contengono i segnali a 100 Hz e a 500 Hz. Nello specifico, al loro interno c'è una suddivisione in altre 22 sottodirectory, ognuna delle quali contiene circa 1000 coppie di ogni segnale. Nello specifico un file ha l'estensione .dat, e contiene il segnale vero e proprio, e un altro .hea, con info sul numero di misurazioni, frequenza e unità di misura.

Ogni segnale a 100 Hz ha una durata pari a 10 secondi ed è 12 derivazioni (I, II, III, aVR, aVL, aVF, V1, V2, V3, V4, V5, V6), le quali permettono di avere una misurazione del potenziale elettrico da diversi punti del cuore. Questi tracciati nello specifico permettono di avere 1000 misurazioni all'interno dei 10 secondi di ognuna delle 12 derivazioni. Nelle successive immagini sono mostrati graficamente alcune informazioni sotto forma di grafici.

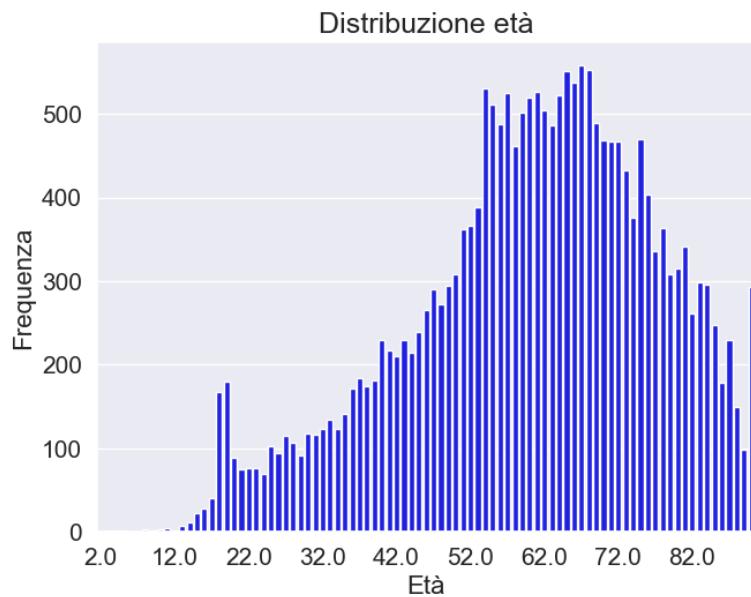


Figura 3.1: Distribuzione età dei pazienti

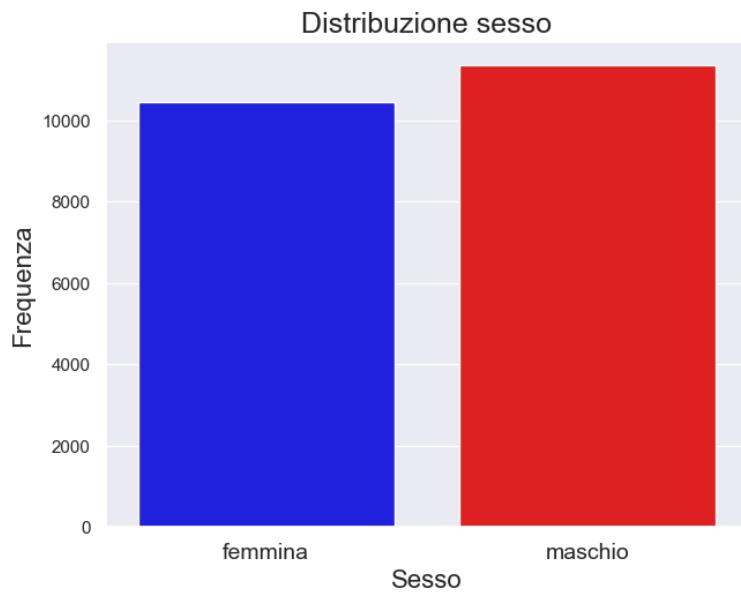


Figura 3.2: Distribuzione sesso dei pazienti

3.2 Pulizia del segnale e scelta delle derivazioni

Analizzando i segnali del dataset, memorizzati come WFDB (Waveform Database), si è visto che alcuni di loro presentano del rumore, il quale non rende i tracciati di qualità sufficiente per proseguire la loro elaborazione. Questo può essere causato dal movimento del paziente, attrito con gli indumenti, problemi agli elettrodi o altro. Pertanto si è vista la necessità di pulire questi dati, sfruttando delle librerie ad hoc, come Neurokit2 [5]. Infatti il metodo `ecg_clean` prende in input il segnale e la frequenza di campionamento (100 Hz in questo caso) e, se il segnale ha troppo rumore, allora ritorna un numpy array con il segnale pulito. Da notare che è stata valutata anche l'utilizzo di Biosppy [1] per questo scopo, ma questa non puliva troppo i segnali e manteneva comunque troppo rumore e quindi si è virato su Neurokit2. Oltre a rimuovere il rumore viene così eliminata anche l'oscillazione del segnale, cioè il `baseline drift`, come visibile in Figura 3.3.

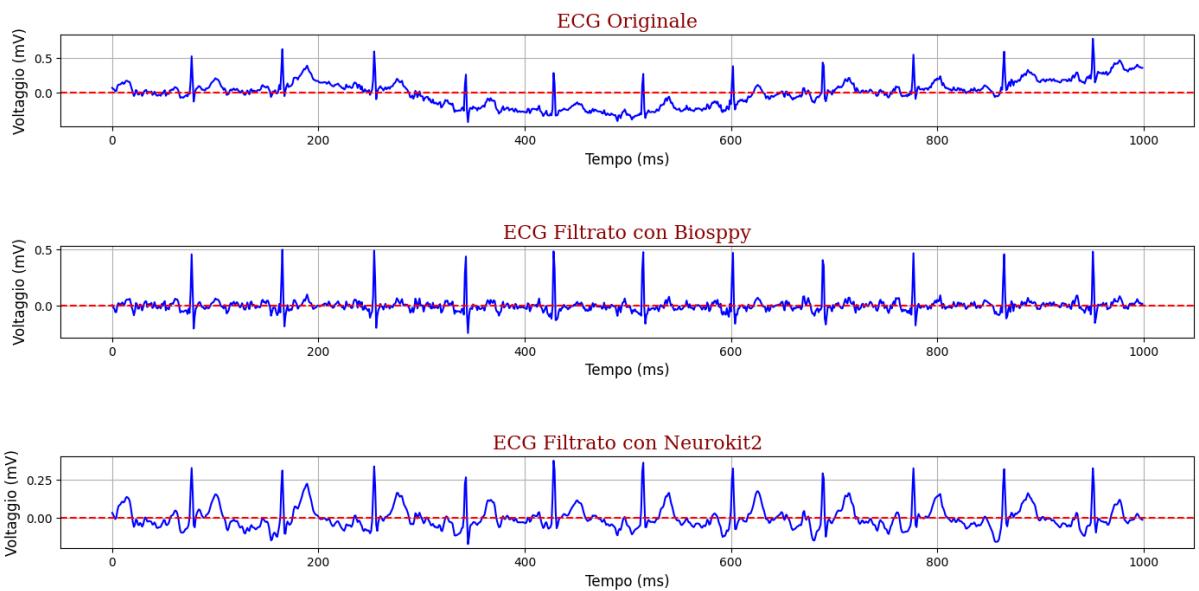


Figura 3.3: Segnale ECG filtrato dalle librerie Biosppy e Neurokit2

Per mostrare il segnale è stata utilizzata la libreria Matplotlib [4], che permette di avere una rappresentazione grafica dei valori del segnale. Una volta puliti i segnali, vengono scelte le 3 derivazioni più rilevanti per trovare caratteristiche affinchè si possa distinguere un segnale normale da uno anomalo. In questa fase sono state fatte delle ricerche al fine di individuare tutte le problematiche che si possono riscontrare in un ECG. Nello specifico si è visto che:

- le anomalie sul ritmo cardiaco, ad esempio bradicardia, tachicardia, fibrillazione ecc, si manifestano principalmente sulle derivazioni periferiche, principalmente 1,2 e 3;
- gli infarti miocardici e altre patologie cardiache sono spesso meglio evidenziati tramite le derivazioni precordiali (V1-V6)

E' inoltre importante che le varie derivazioni avessero picchi R abbastanza alti così da poter più facilmente distinguere i vari battiti presenti. Si è quindi passati ad un analisi del dataset e si è arrivati alla conclusione che le derivazioni più adatte sono I,II e V6, pertanto d'ora in avanti si utilizzeranno proprio queste tre. Come si può vedere in Figura 3.4 tra le derivazioni precordiali si ha un progressivo aumento del picco R fino a V6 che, in condizioni normali, dispone di un'ampiezza notevole.

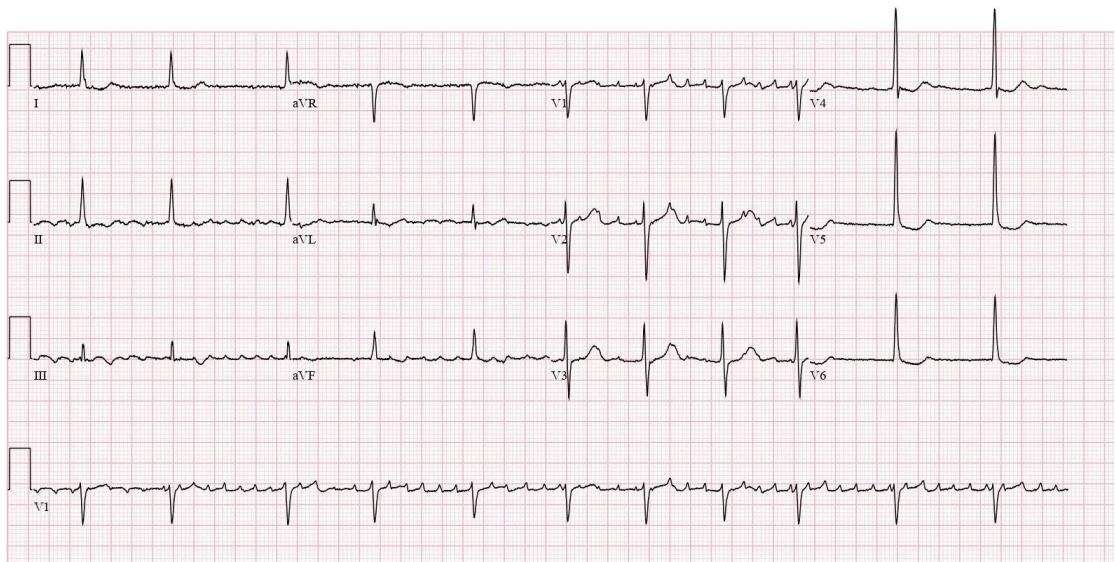


Figura 3.4: Ecg a 12 derivazioni dove si può osservare l'aumento del picco R nelle derivazioni precordiali

3.3 Individuazione dei picchi R

Il passo successivo consiste nel dividere i vari ECG in segmenti, ognuno dei quali corrisponde ad un battito. Per far questo però si devono prima individuare i picchi R, infatti ogni battito è riconoscibile proprio da questo picco. Si sono provati più metodi al fine di avere un'uniformità tra le 3 derivazioni dello stesso ecg. Un primo tentativo è stato quello di considerare le 3 derivazioni dello stesso ecg come se fossero 3 segnali distinti ed utilizzare il metodo `ecg_peaks` di Neurokit2, il quale prende il segnale e la frequenza di campionamento e ritorna un dataframe della stessa lunghezza del segnale in input con 1 nelle posizioni dei picchi e 0 nelle altre e un dizionario con all'interno un numpy array con le posizioni da 0 a 999 dei picchi identificati. Tuttavia i tre array spesso non combaciavano, sia nelle posizioni che nel numero di picchi trovati. Questo non è corretto poichè il numero di battiti (e quindi anche di picchi) tra le 12 derivazioni di un ECG è sempre lo stesso. Analizzando le discrepanze ottenute, si è passato allora ad un altro approccio: ovvero individuare i picchi R nella derivazione V6, e usare qui picchi per tutte e 3 le derivazioni di ogni singolo ecg. Questo approccio si è dimostrato essere valido in quanto la derivazione in questione ha dei segnali dove è molto semplice individuarli, dovuto al fatto che presenta picchi evidenti. In più si è utilizzato anche il metodo `ecg_invert` per verificare se il segnale risultasse invertito o meno. Anche questo metodo prende il segnale in input e la frequenza di campionamento e, se riconosce il segnale come invertito allora ritorna un numpy array con il segnale invertito altrimenti lascia il segnale originario così com'è. E' stato usato questo metodo perchè, se il segnale avesse dei picchi rilevanti non come deflessioni positive ma negative, allora risulta più semplice riconoscerli e quindi riconoscere i vari battiti presenti. Infine, qualora il numero di picchi fosse minore di 4 si vanno cercare nella derivazione II. Questo perchè in tre casi si ha che la derivazione V6 è un segnale quasi piatto mentre nella derivazione II risulta chiari ed evidenti i picchi. In Figura 3.5 è riportato un esempio di un ECG con i relativi picchi R trovati.

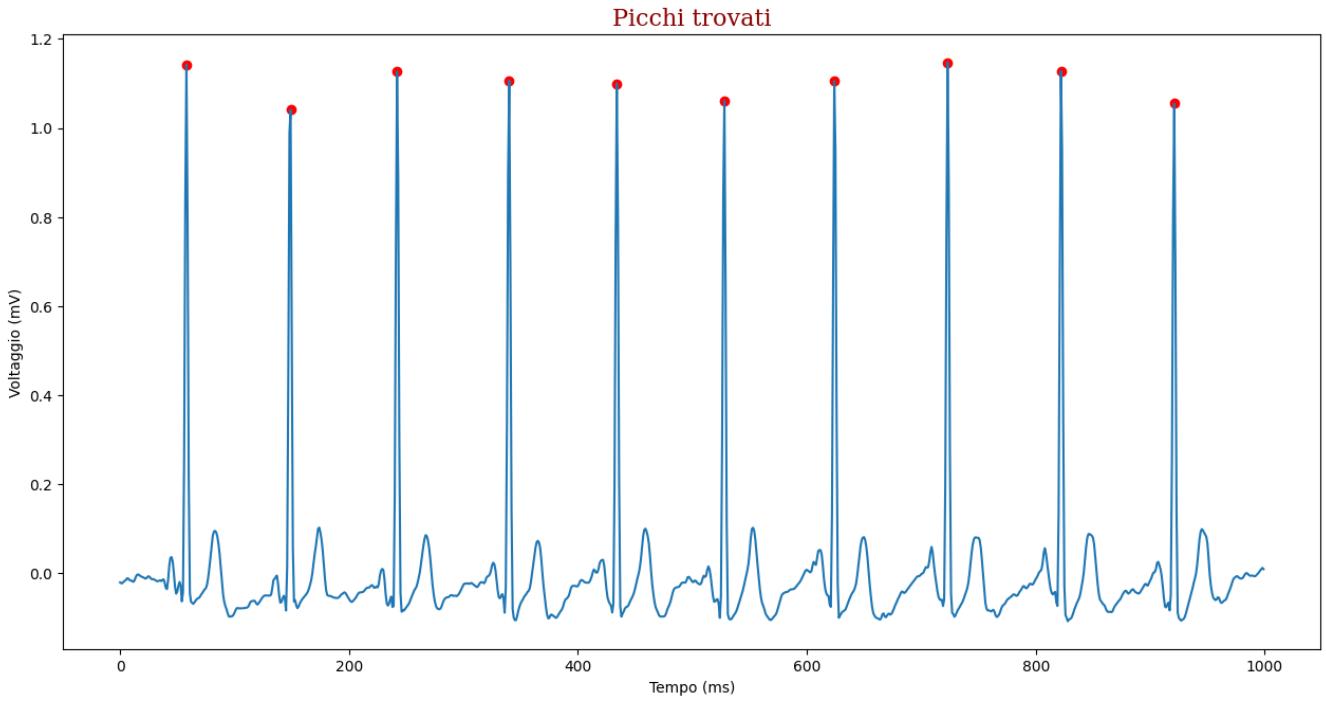


Figura 3.5: ECG con i picchi R in rosso rilevati

3.4 Segmentazione in battiti

Avendo adesso le posizioni dei picchi R si può segmentare il segnale in battiti. In questo passo si è valutato prima l'utilizzo del metodo `ecg_segment`, che prende il segnale e la frequenza di campionamento in input e ritorna un dizionario con i battiti. Tuttavia, ho riscontrato che questo metodo presentava alcune limitazioni in termini di velocità e precisione. Pertanto, ho deciso di non utilizzarlo e di optare per un algoritmo alternativo, usato anche dal lavoro svolto da un collega precedentemente e riportato alla sua pagina 18 [3], che è risultato sia più preciso che più performante. Nello specifico si utilizzano le distanze picco-picco per andare a capire l'inizio e la fine di ogni singolo battito. Nello specifico:

- dal secondo al penultimo battito si calcola la distanza tra il picco in questione e il picco precedente e successivo. A questo punto il battito inizia dalla posizione del picco precedente più la distanza da esso diviso 2 e invece finisce alla posizione del picco successivo meno la distanza da esso diviso 2;
- per il primo battito invece si calcola la distanza con il secondo picco e il battito inizia dalla sua posizione meno questa distanza dimezzata, con l'accortezza che se è negativa allora si considera come margine sinistro la posizione 0. Finisce invece dalla posizione del picco più la distanza dal secondo picco dimezzata. Analogamente, con l'ultimo

battito si procede allo stesso modo ma stavolta la differenza si fa con il penultimo battito e se la somma tra la posizione dell'ultimo picco e questa differenza dimezzata è maggiore di 999, cioè l'ultima posizione del tracciato, allora si prende come limite destro proprio 999 altrimenti il valore calcolato.

Essendo gli stessi picchi R per tutte e tre le derivazioni, queste hanno tutte lo stesso numero di battiti. In Figura 3.6 e dalla Figura 3.7 alla Figura 3.16 viene riportata una rappresentazione visiva di quanto detto.

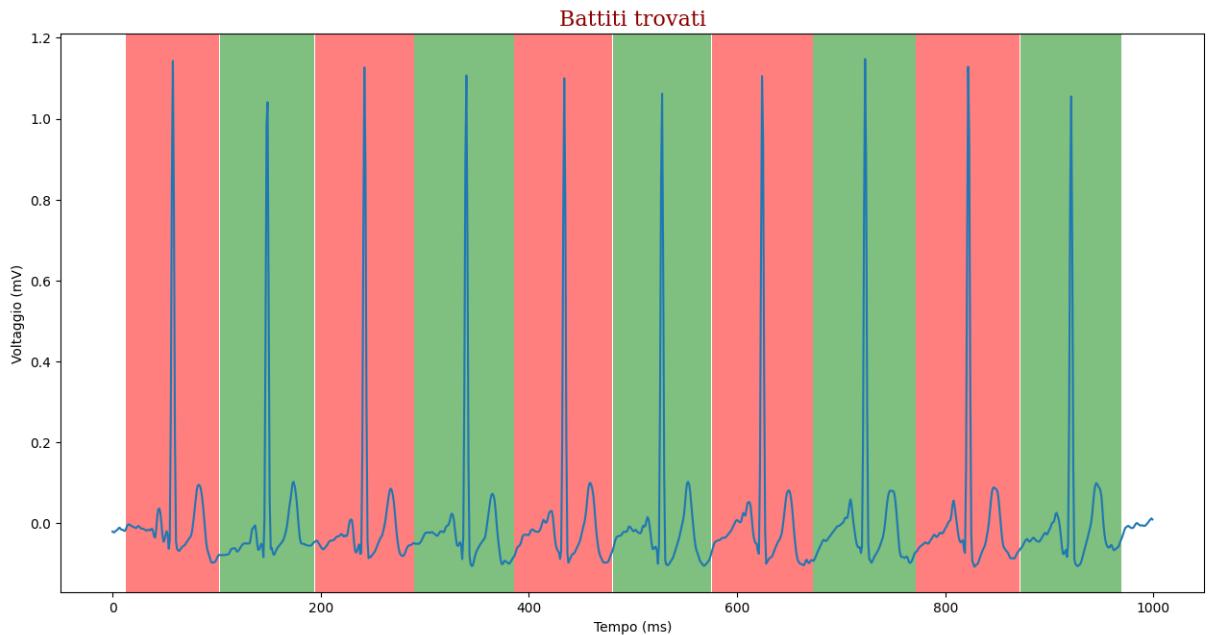


Figura 3.6: Rappresentazione dei margini di ogni battito rilevato

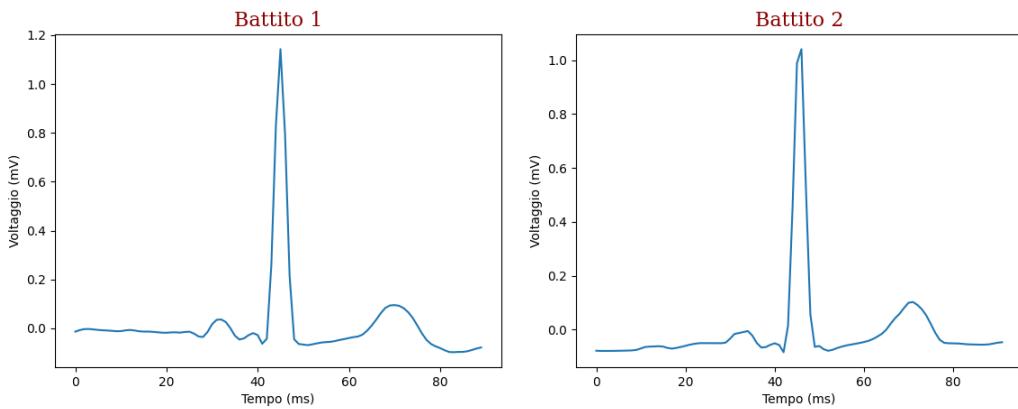


Figura 3.7: Battito 1 segmentato

Figura 3.8: Battito 2 segmentato

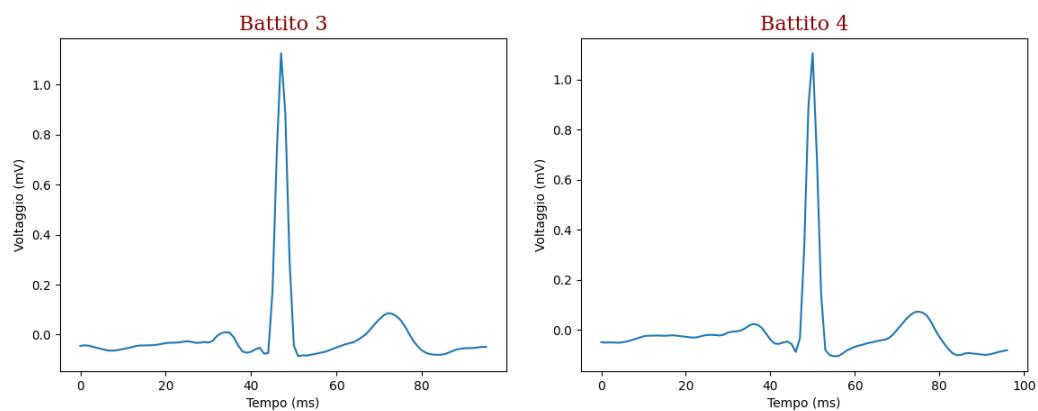


Figura 3.9: Battito 3 segmentato

Figura 3.10: Battito 4 segmentato

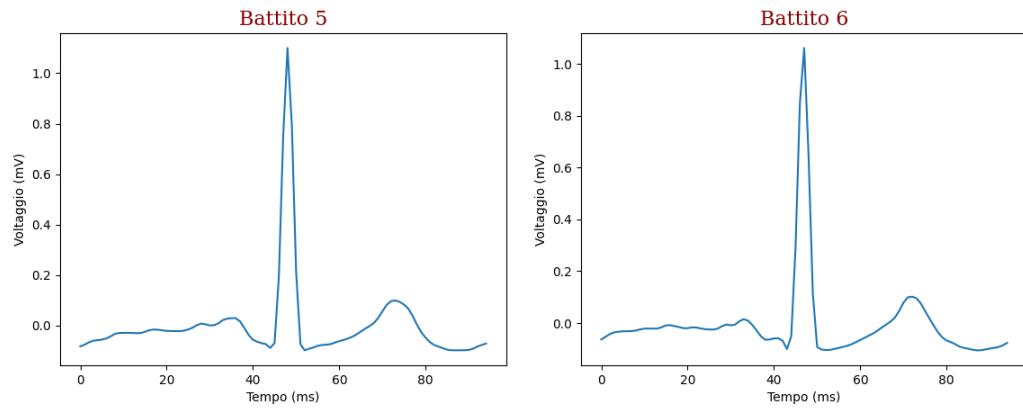


Figura 3.11: Battito 5 segmentato

Figura 3.12: Battito 6 segmentato

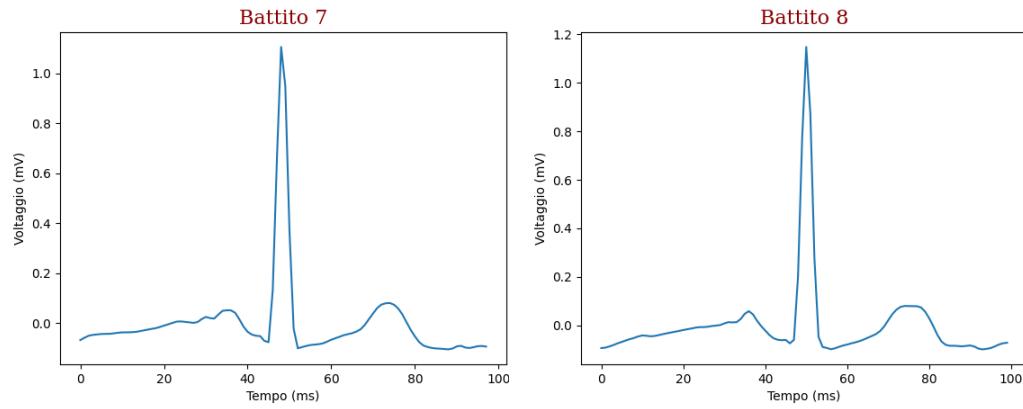


Figura 3.13: Battito 7 segmentato

Figura 3.14: Battito 8 segmentato

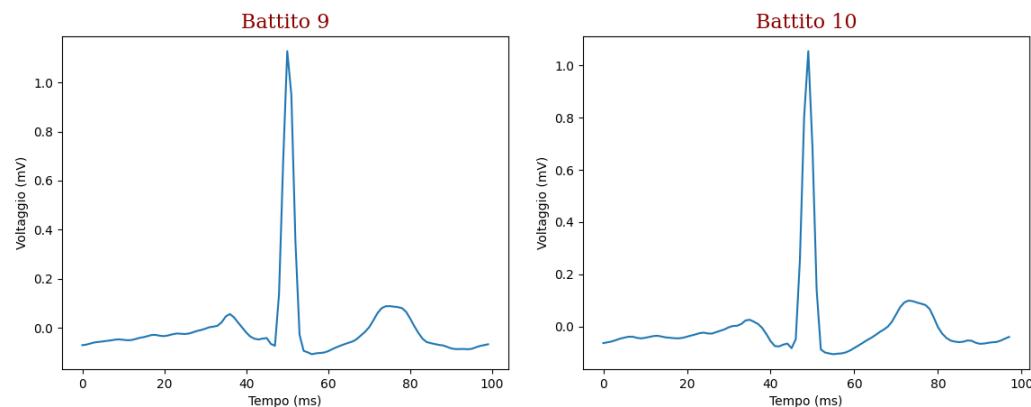


Figura 3.15: Battito 9 segmentato

Figura 3.16: Battito 10 segmentato

Come si può vedere dalle figure, i battiti segmentati sono ancora in forma di segnale. Pertanto i valori rappresenteranno la differenza di potenziale del cuore. Nel passo successivo invece si convertiranno questi valori in colore, andando a popolare l'array bidimensionale dell'immagine.

3.5 Creazione delle immagini caratteristiche

Una volta che si hanno i singoli segmenti per ogni ecg si sono create le immagini caratteristiche. Queste hanno l'altezza che è uguale al numero di battiti rilevati e la larghezza che invece è uguale al numero di sample del battito più grande dell'ecg considerato. Per ogni riga (e quindi per ogni battito) sono stati assegnati i colori rosso, verde e blu (RGB) sfruttando le tre derivazioni considerate, nello specifico:

- il rosso è stato estratto dalla prima derivazione considerata;
- il verde è stato invece estratto dalla seconda derivazione considerata;
- infine il blu è stato estratto dalla terza derivazione considerata.

Tuttavia è necessario che questi valori siano normalizzati. Infatti ogni valore dei tre colori deve variare tra 0, che identifica la non presenza di quel colore in questione, a 255, che invece ne indica la massima presenza. Una volta fatto questo si sono creati gli array numpy a 2 dimensioni e si sono colorate tutte le righe per un numero di colonne pari alla dimensione del battito in questione. Qualora non si arrivasse fino alla fine della riga allora le restanti celle rimangono nere. Questo significa che, nella maggior parte dei casi, si avrà solo una riga completamente colorata. Esempi di immagini sono visibili nelle Figure 3.17, 3.18 e 3.19 a seguire.

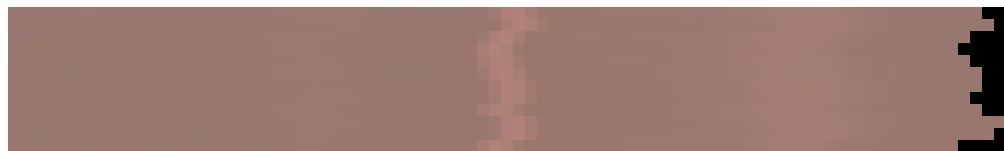


Figura 3.17: Immagine caratteristica ecg_id 00008

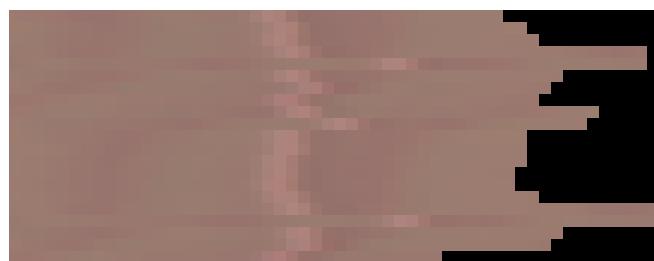


Figura 3.18: Immagine caratteristica ecg_id 00018



Figura 3.19: Immagine caratteristica `ecg_id 00020`

Le immagini qui riportare hanno una dimensione molto piccola, infatti Figura 3.17 ha dimensioni 83 x 12, Figura 3.18 ha dimensioni 54 x 21 e Figura 3.19 43 x 22. Pertanto sono state ingrandite su questo documento per renderle più visibili.

Le immagini così prodotte sono state salvate usando il modulo `Image` della libreria `Pillow` [2] assegnandole come nome il codice univoco `ecg_id`. Nello specifico è stata organizzata una directory che contiene a sua volta altre 22 sottodirectory, in modo da avere la stessa organizzazione del dataset originario.

3.6 Organizzazione del dataset

A questo punto si hanno le immagini, ognuna delle quali rappresenta in modo alternativo un’istantanea dell’attività elettrica del cuore in un determinato intervallo di tempo, che sono state organizzate nel nuovo dataset. Bisogna ora capire quale immagine rappresenta un ecg classificato come normale e quale invece risulta anomalo. A tal proposito è stato sfruttato il campo `scp_codes` del file `ptbxl_database.csv`. Infatti questo campo rappresenta il responso del medico sotto forma di dizionario, dove la chiave è una classe e il valore indica la certezza del cardiologo di quella classe.

E’ stata quindi creata una nuova colonna del file csv, chiamata ’normale/anomalo’, dove appunto vengono etichettati come normali tutti i record con la classe ’NORM’ contenuta nel dizionario come chiave e anormali gli altri. Come mostrato in Figura 3.20, si hanno così 9514 record etichettati come normali e i restanti 12285 come anomali. In questo modo se l’etichetta è normale allora significa che l’ECG è normale, altrimenti è anomalo. Si ricorda infatti che questa assegnazione non avviene ad uno o più battiti di un ECG ma all’intero ECG (infatti si sta usando l’immagine caratteristica per riferisti all’intero ECG). Più precisamente l’assegnazione si riferisce all’immagine caratteristica dell’ECG con lo stesso `ecg_id` presente nel dataset iniziale, ed ha un numero di righe pari al numero di battiti di quel ECG e un numero di colonne pari alla loro lunghezza.

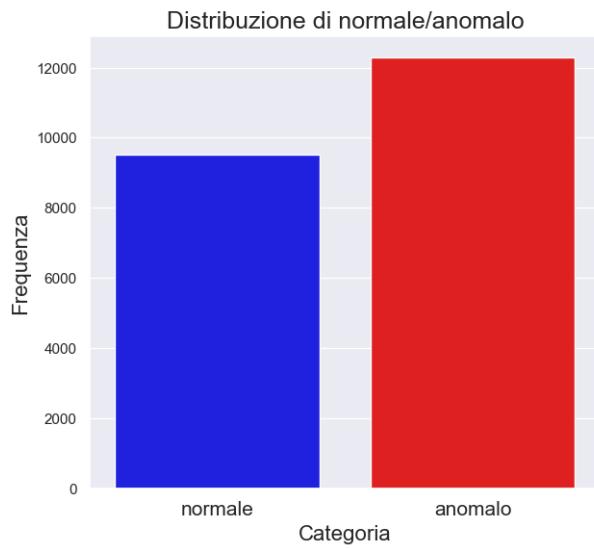


Figura 3.20: Distribuzione degli ecg normali e anomali

Viene quindi organizzato un nuovo dataset, il quale contiene le immagini (che hanno come nome il relativo `ecg_id` univoco) degli ecg etichettati come normale e anormali in due directory separate.

3.7 Train, Validation e Test set

A questo punto si passa all'ultimo passo del tirocinio, ovvero andare a classificare le immagini in Anomalo e Normale. Per far questo è stata costruita, come si vedrà più avanti, una piccola rete neurale convoluzionale, la quale prende in input queste immagini e riesce a capire con una buona probabilità se si tratta di un ECG etichettato come Normale o Anomalo. Tuttavia prima di far ciò si è dovuto procedere con un ulteriore divisione del dataset. Infatti, dato che si andrà a creare ed utilizzare un modello di machine learning, si è dovuto creare un set di train, un set di validazione ed un set di test. Il set di train servirà al modello per andare ad imparare dai dati in input a distinguere la differenza tra le due tipologie di immagini, cercando di minimizzare l'errore tra le previsioni del modello e le reali etichette. Il set di validazione serve invece a validare i risultati ottenuti durante il training, andando ad ottimizzare i parametri interni del modello. Fa quindi parte ancora della fase di apprendimento e l'obbiettivo di questo set è quello di evitare l'*overfitting*, fenomeno che si verifica quando il modello impara troppo bene dai dati del train set tanto da non riuscire a generalizzare su nuovi dati. In tal caso il grafico dell'accuratezza e dell'errore avrà un aspetto simile a quello della Figura 3.21. Il test set invece è l'insieme di dati che contiene le immagini etichettate ma che verranno date in pasto al modello senza queste etichette. L'obbiettivo è quello di vedere se il modello riesce a dare delle predizioni quanto più corrette possibili. Infatti, sebbene le etichette reali non vengano usate dal modello durante il test, verranno poi chiamate in causa quando si confronteranno i valori predetti dalla rete. Di solito la divisione in questi 3 insiemi prevede un 60/80% train, 10/20% validazione e 10/20% test, ma queste proporzioni possono più o meno variare. L'insieme di train è il più grande perché il modello deve ovviamente imparare a generalizzare da un insieme possibilmente eterogeneo nel contesto di una classe. In questo caso si è proceduto con una suddivisione dell'80% train, 10% validazione e il restante 10% per il test. Per la precisione l'insieme di train e quello di validazione si trovano nella stessa directory e la loro suddivisione avviene solo durante il caricamento delle immagini. Gli insiemi sono disgiunti. Le immagini sono state organizzate usando la libreria `os` integrata in Python. A questo punto ci sono quindi 2 directory:

- La directory `Train validazione`, con all'interno altre due sottodirectory `Anomalo` e `Normale`
- La directory `Test`, con all'interno le due sottodirectory `Anomalo` e `Normale`, diverse da quelle in `Train validazione`.

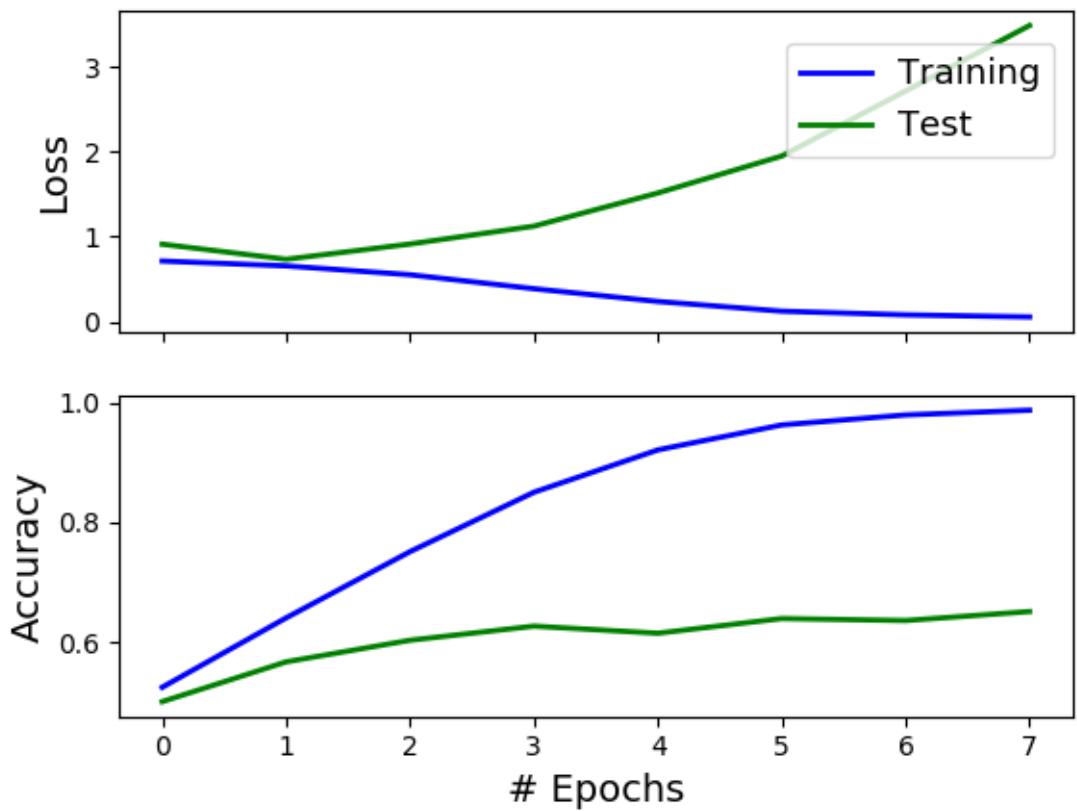


Figura 3.21: L’overfit è un problema che può essere visualizzato su grafico sia sul validation set che sul test set. In questo esempio sul test set si può vedere che l’accuratezza del train è alta ma sul test set man mano diminuisce mentre il grafico dell’errore tra la predizione e la classe corretta è esattamente al contrario.

3.8 Caricamento delle immagini

Dopo aver suddiviso il dataset si è passato a caricare questi set. Da questo punto in avanti è stata usata una libreria comunemente usata nel machine learning e deep learning, TensorFlow [6], sviluppata da parte di Google. Quindi per caricare i dati di train e validazione si usa, in TensorFlow, la classe `ImageDataGenerator`: questa prende in input un parametro per la suddivisione tra train e validazione e un altro parametro per scalare i valori dei pixel. In pratica il modello usato, come detto, è una rete neurale convoluzionale e quello che si fa in questi casi è forzare i valori dei pixel, che vanno da 0 a 255 per ognuno dei tre canali RGB, in un intervallo comune e possibilmente piccolo per ridurre il tempo di addestramento e migliorare le prestazioni del modello. Nel mio caso questo intervallo va tra 0 e 1, quindi le immagini avranno tutti i valori dei loro array numpy in questo range.

Sono poi stati caricate le immagini di train e validazione. Per questo si è usato il metodo `flow_from_directory` applicato sul datagen tornato dalla classe `ImageDataGenerator`, specificando che si tratta del sottoinsieme di training, il path fino alla directory `Train validazione` e che si tratta di una classificazione binaria (Anomalo e Normale). La stessa cosa viene fatta per la validazione, specificando ovviamente che si sta caricando il sottoinsieme di validazione. Durante questo caricamento dei dati TensorFlow assegna le etichette, o meglio dire le classi, ad ogni immagine. Lo fa andando a vedere la directory di appartenenza di ogni immagine, nello specifico:

- Se un'immagine viene presa dalla directory Anomalo allora appartiene a quella classe
- Se invece viene presa dalla directory Normale allora appartiene alla classe Normale.

Questa assegnazione avviene però sotto forma di intero. Quindi, dato che si va a vedere l'ordine lessicografico dei nomi delle directory, la directory Anomalo sarà rappresentata da qui in avanti come uno 0 e la directory Normale come un 1. Inoltre, dato che le immagini sono tutte di dimensioni diverse tra loro, occorre standardizzarle in modo che la rete possa ricevere immagini tutte della stessa forma. Ciò viene fatto in questo passo durante il caricamento, ed è stata proposta una dimensione comune di 110 x 110. Sono state provate anche dimensioni maggiori, come 224 x 224 ma la rete era assai più lenta e le performance non hanno portato a scegliere questa dimensione. Un esempio delle dimensioni delle immagini è visibile in Figura 3.22 e in Figura 3.23. Per caricare le immagini di test invece è stato usato un'altra istanza di `ImageDataGenerator`, dove questa volta si scalano solo i valori tra 0 e 1 (perchè invece in quello di train e validazione si è riservata una parte, pari al 11,2% del dataset di train, pari a sua volta al 10% del dataset totale per la validazione). Il caricamento avviene anche qui con `flow_from_directory`, specificando però la directory Test e, anche qui, che si tratta di classificazione binaria, la `target_size` delle immagini ed un parametro `shuffle` settato a False: questo parametro permette di evitare

di mescolare le immagini e le rispettive etichette e quindi, quando poi si andrà più avanti a valutare le performance del modello, di mantenere per ogni immagine la sua etichetta vera. Questo parametro viene settato di default a True, infatti durante il test è stato oggetto di performance scarse inizialmente.



Figura 3.22: Immagine dell'ECG 00008 ridimensionata alle dimensioni usate nella rete, 110 x 110. Questa immagine ha le dimensioni reali. L'immagine nella rete ha i valori scalati tra 0 e 1 (questa ha ancora i valori tra 0 e 255 per i 3 canali)



Figura 3.23: Immagine dell'ECG 00017 ridimensionata alle dimensioni usate nella rete, 110 x 110. Questa immagine ha le dimensioni reali. L'immagine nella rete ha i valori scalati tra 0 e 1 (questa ha ancora i valori tra 0 e 255 per i 3 canali)

3.9 Creazione della rete

Ora è stata creata la rete neurale convolutiva, anche detta CNN (Convolutional Neural Networks).

3.9.1 Le reti neurali

Queste sono un tipo di reti neurali artificiali progettate principalmente per elaborare dati a griglia, come in questo caso immagini oppure video. Sono particolarmente efficaci per compiti di visione artificiale, riconoscimento di modelli e analisi di dati spaziali. Queste reti sono si differenziano dalle reti neurali classiche in quanto presentano alcune caratteristiche specializzate per l'elaborazione di dati di tipo griglia:

- Strati convoluzionali: Nelle CNN, i layer convoluzionali svolgono un ruolo chiave. Questi strati utilizzano filtri (kernel) per eseguire operazioni di convoluzione sui dati in ingresso. Questo permette alla rete di rilevare pattern e caratteristiche locali nell'immagine, come bordi, texture o forme;
- Pooling: Dopo ciascun strato convoluzionale, di solito si applica uno strato di pooling (ad esempio, MaxPooling o AveragePooling). Questo riduce progressivamente la dimensione dei dati e conserva le caratteristiche più importanti. Il pooling riduce il carico computazionale e aiuta a rendere la rete più robusta ai piccoli cambiamenti nella posizione delle caratteristiche nell'immagine;
- Località: Le CNN sfruttano l'assunzione di località, il che significa che caratteristiche importanti sono tipicamente localizzate in piccole regioni dell'immagine. I filtri convoluzionali catturano queste caratteristiche locali, riducendo la complessità computazionale.
- Strati completamente connessi: Anche se le CNN iniziano con strati convoluzionali, spesso terminano con strati completamente connessi, tipici delle reti neurali tradizionali. Questi strati finali combinano le caratteristiche estratte dalle parti precedenti della rete per produrre l'output.

In Figura 3.24 viene mostrata un esempio di rete neurale tradizionale. Si può vedere come ogni neurone di ogni livello sia collegato con tutti i neuroni dello strato successivo. Questo implica che ci sarà un certo peso, detto **parametro** per ognuno di questo collegamento e quindi la rete, dato che apprende modificando man mano questi parametri, dovrà adattare questo valore molto frequentemente, rendendola abbastanza pesante durante l'esecuzione. In Figura 3.25 invece si può vedere un esempio di rete neurale convoluzionale. Come si può vedere anche qui ci sono dei livelli completamente connessi, ma questi sono solo in fondo alla rete e di solito sono in numero limitato. Nella prima parte ogni neurone, o per meglio dire ogni

filtro, va ad analizzare l'intera immagine con dei parametri fissi, detto **kernel**. Questa è una matrice, di solito 3x3, che va ad effettuare il prodotto tra il kernel stesso e la porzione di immagine considerata, in questo caso tutte le porzioni 3x3 dell'immagine in input. Viene poi usata una funzione di attivazione, come la **ReLU** che va a mappare i valori negativi a zero e mantiene i positivi così come sono, per fare emergere le caratteristiche rilevanti e va a produrre una **mappa** dell'immagine per ogni kernel. Si hanno quindi più layer, ognuno per mappa. Successivamente c'è uno strato di **pooling** che va a diminuire la dimensione delle mappe prodotte. Queste operazioni vengono ripetute più volte in sequenza all'interno della rete e più si va in fondo e più lo stack delle mappe diminuisce in larghezza e altezza ma aumenta in profondità. Si arriva poi alla fine della rete dove viene appiattito questo stack a dimensione 1xN e dato in pasto alla sezione completamente connessa per procedere con una classificazione in classi. In questo esempio verranno date delle probabilità che l'immagine in input sia una macchina, camion, van ecc.

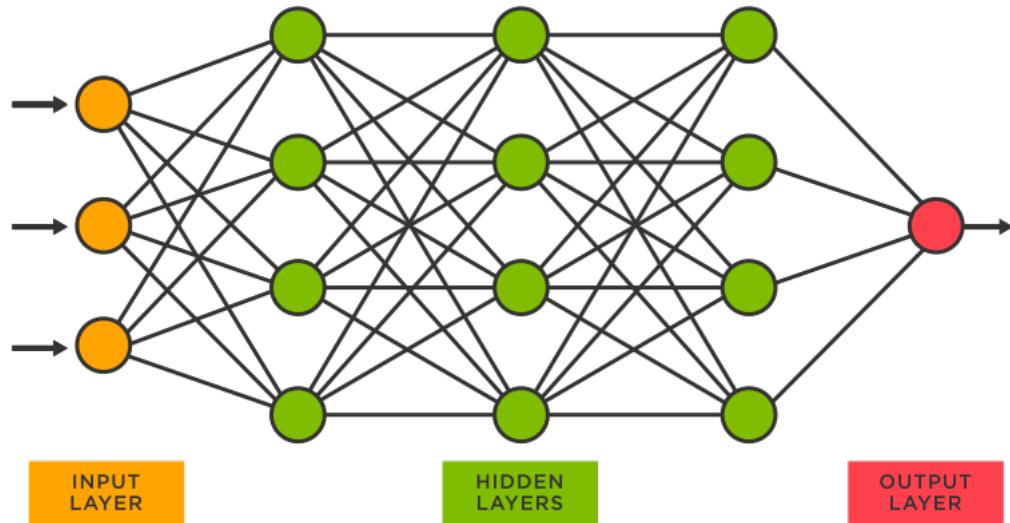


Figura 3.24: Esempio di rete neurale classica. Ogni cerchio rappresenta un neurone

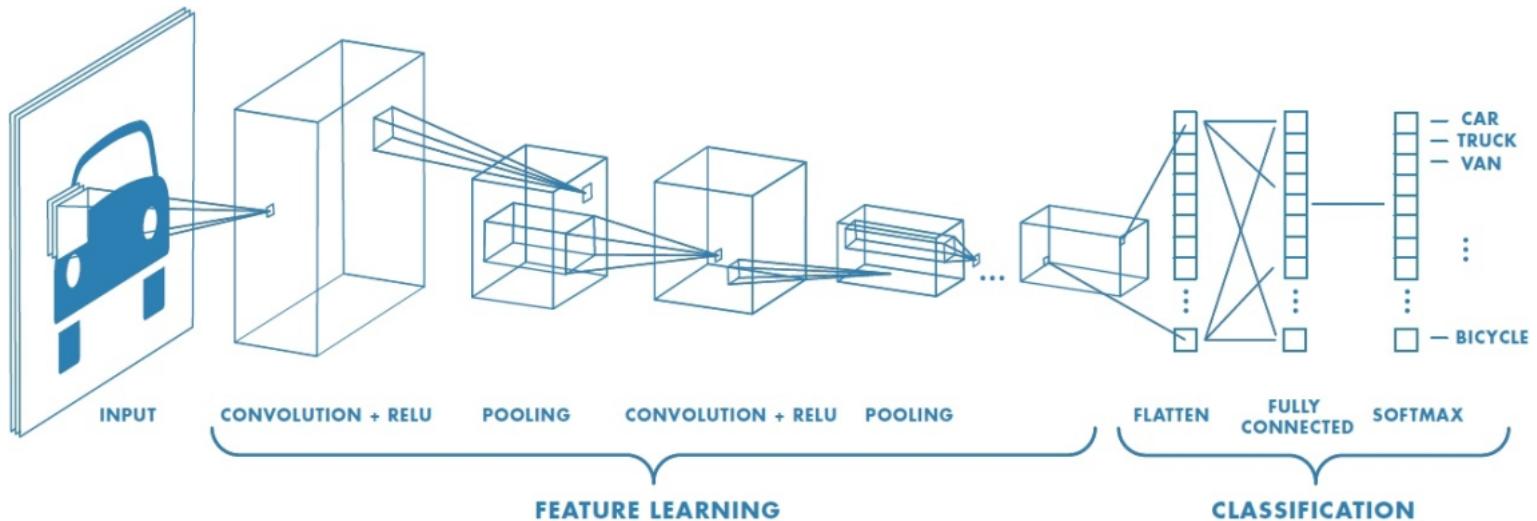


Figura 3.25: Esempio di rete neurale CNN per la classificazione di immagini in input

3.9.2 La rete

La rete utilizzata per il tirocinio è stata progettata da zero, senza usare alcune delle reti già esistenti, usando la giusta quantità di neuroni senza eccedere. Per farlo è stato usata la classe `Sequential` di TensorFlow e sono stati inseriti i seguenti livelli:

- Un primo strato convolutivo: Questo è stato fatto con la classe `Conv2D`. Ha 32 filtri 3×3 , attivazione `ReLU` e `padding = Same` per calcolare la feature map su tutta l'immagine, senza escludere i margini;
- Un maxpool: Creato usando la classe `MaxPool2D`. È 2×2 . In Figura 3.26 un esempio di come funziona;
- Un secondo strato convolutivo: Anche questo è stato fatto con la classe `Conv2D`, con però 64 filtri 3×3 , attivazione `ReLU` e `padding = Same` per non escludere i margini;
- Un maxpool;
- Un ultimo strato convolutivo: Creato sempre usando la classe `Conv2D` ma con 128 filtri 3×3 , attivazione `ReLU` e `padding = Same`;
- Un maxpool;
- Un flatten: Creato con la classe `Flatten` di TensorFlow. Usato per appiattire lo stack delle mappe e darlo in input allo strato denso successivo;
- Uno strato denso: Questo si trova alla fine della rete, nella parte completamente connessa. Infatti ognuno dei 128 neuroni che lo compongono riceverà tutti i valori contenuti del flatten precedente. È stato creato con `Dense`, attivazione `ReLU`;

- dropout: Il dropout è una tecnica di regolarizzazione utilizzata per prevenire l’overfitting durante l’addestramento della rete neurale e a rendere la rete più robusta. Durante l’addestramento, il dropout imposta casualmente un sottoinsieme dei nodi del livello a zero con una certa probabilità, nel mio caso del 50%. Crato usando Dropout;
- L’ultimo strato denso: Questo è l’ultimo strato della rete usata e prevede solo un neurone, che riceve dai 128 neuroni dello strato precedente, dei quali la metà di loro sono settati a 0 dal dropout. In questo caso però la funzione di attivazione usata non è una ReLU. Infatti non si deve semplicemente prendere il valore ricevuto e portarlo a zero se negativo altrimenti lo si mantiene così com’è, ma si vuole avere una classificazione dell’immagine ricevuta in 0 (Anomalo) o in 1(cioè Normale). Pertanto si utilizza la sigmoide, visibile in Figura 3.28, che forza il valore calcolato dal singolo neurone tra 0 e 1. Se è più vicino a 0 allora l’immagine viene classificata come 0, se più vicino a 1 invece viene classificata come 1.

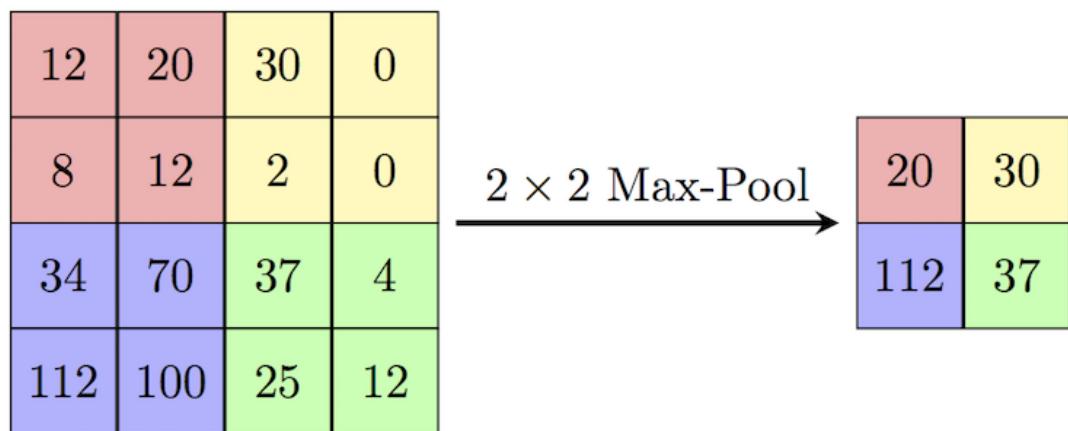


Figura 3.26: Esempio di funzionamento del MaxPooling2D

Come si può notare man mano che si va in fondo alla rete questa presenta sempre più filtri. Questo perchè l’immagine in input diventa sempre più astratta e quindi la rete ha a che fare con feature map sempre più elaborate.

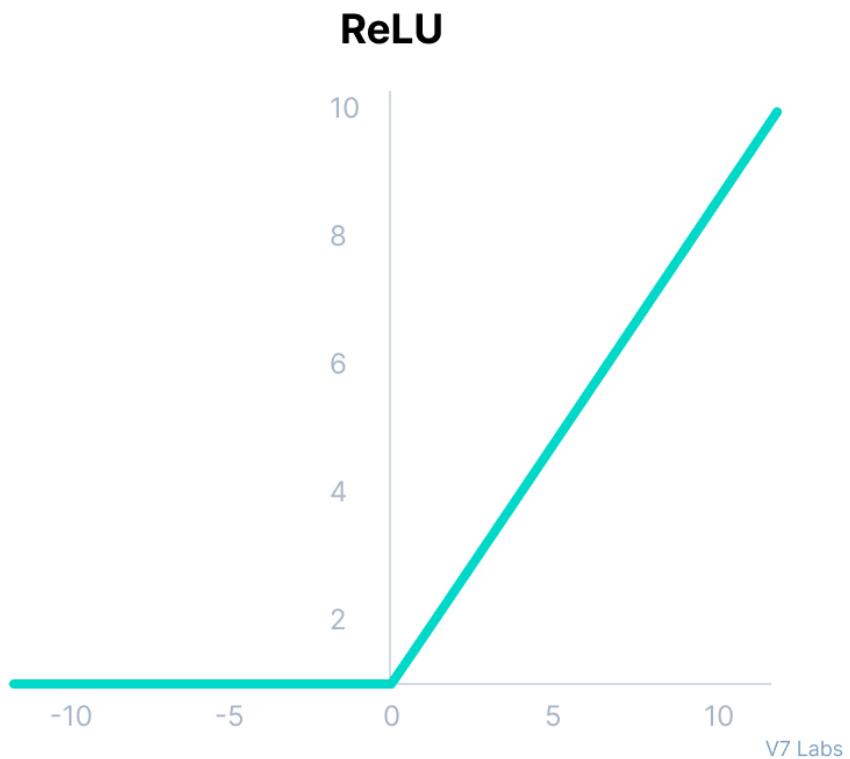


Figura 3.27: Funzione ReLU usata nella rete

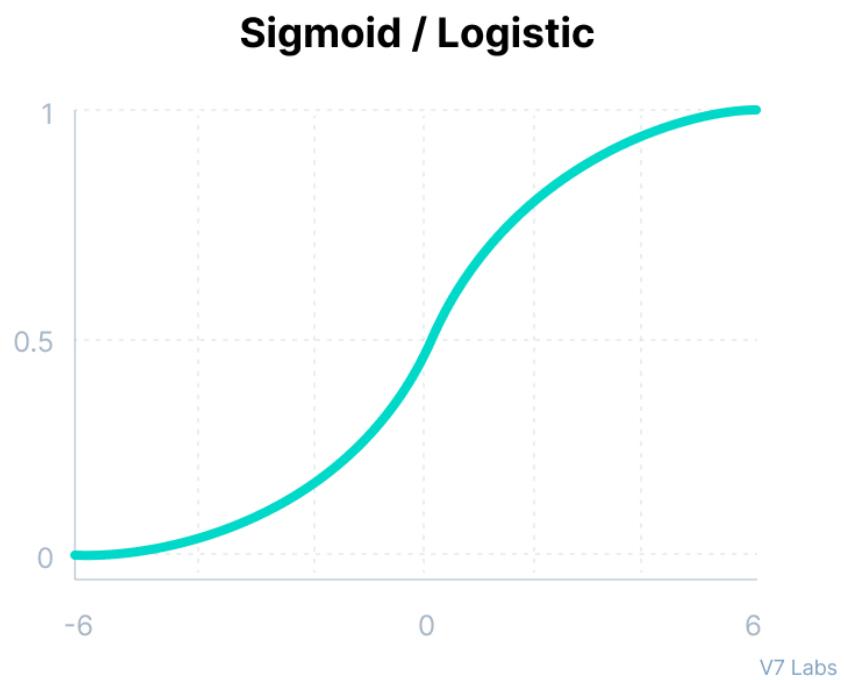


Figura 3.28: Funzione sigmoide usata nell'ultimo strato della rete per la classificazione

Una panoramica della rete è visibile in Figura 3.29, dove vengono mostrati anche gli output shape da ogni livello ed il numero di parametri. Come si può vedere man mano che si va all'interno della rete lo stack delle feature map si riduce in larghezza ed altezza (da 110 x 110 del primo livello a 13 x 13 dell'ultimo maxpool prima dello strato completamente connesso) e contestualmente aumenta in profondità (da 32 a 128).

| Model: "sequential" | | |
|---|----------------------|---------|
| Layer (type) | Output Shape | Param # |
| conv2d (Conv2D) | (None, 110, 110, 32) | 896 |
| max_pooling2d (MaxPooling2D) | (None, 55, 55, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 55, 55, 64) | 18496 |
| max_pooling2d_1 (MaxPooling2D) | (None, 27, 27, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 27, 27, 128) | 73856 |
| max_pooling2d_2 (MaxPooling2D) | (None, 13, 13, 128) | 0 |
| flatten (Flatten) | (None, 21632) | 0 |
| dense (Dense) | (None, 128) | 2769024 |
| dropout (Dropout) | (None, 128) | 0 |
| dense_1 (Dense) | (None, 1) | 129 |
| Total params: 2862401 (10.92 MB) | | |
| Trainable params: 2862401 (10.92 MB) | | |
| Non-trainable params: 0 (0.00 Byte) | | |

Figura 3.29: Architettura della rete, con output shape e numero di parametri per ogni strato

Questa rete è stata creata in questo modo perchè si è rivelato essere il migliore in termini di performance e tempistiche di allenamento. Infatti si è provato anche ad usare più livelli convolutivi, con filtri 5 x 5 invece di 3 x 3 e cambiando anche il numero di filtri per ogni livello ma i risultati sono stati peggiori.

Successivamente è stato eseguito il training del modello, usando il metodo `fit`. A questo sono stati passati in input il train-generator conte-

nente come detto il train set, l’insieme di validazione contenuto in validation_generator ed il numero di epoche, impostato su 20. Questo numero è un iperparametro della rete che specifica il numero di volte che il training (e validazione) deve essere ripetuto sull’intero dataset di train e validazione. E’ stato inoltre settato un EarlyStopping di TensorFlow sulla loss del set di validazione con `patience = 3`: questo significa che il modello interromperà il training qualora il valore della loss calcolata sul set di validazione non migliori per 3 epoche di fila in modo significativo. Anche questa è stata una opzione usata al fine di evitare overfitting del modello. Durante l’addestramento il modello divide il tutto in batch da 32 e per ognuno calcola l’accuratezza e la loss, cioè l’errore che il modello commette durante questa fase.

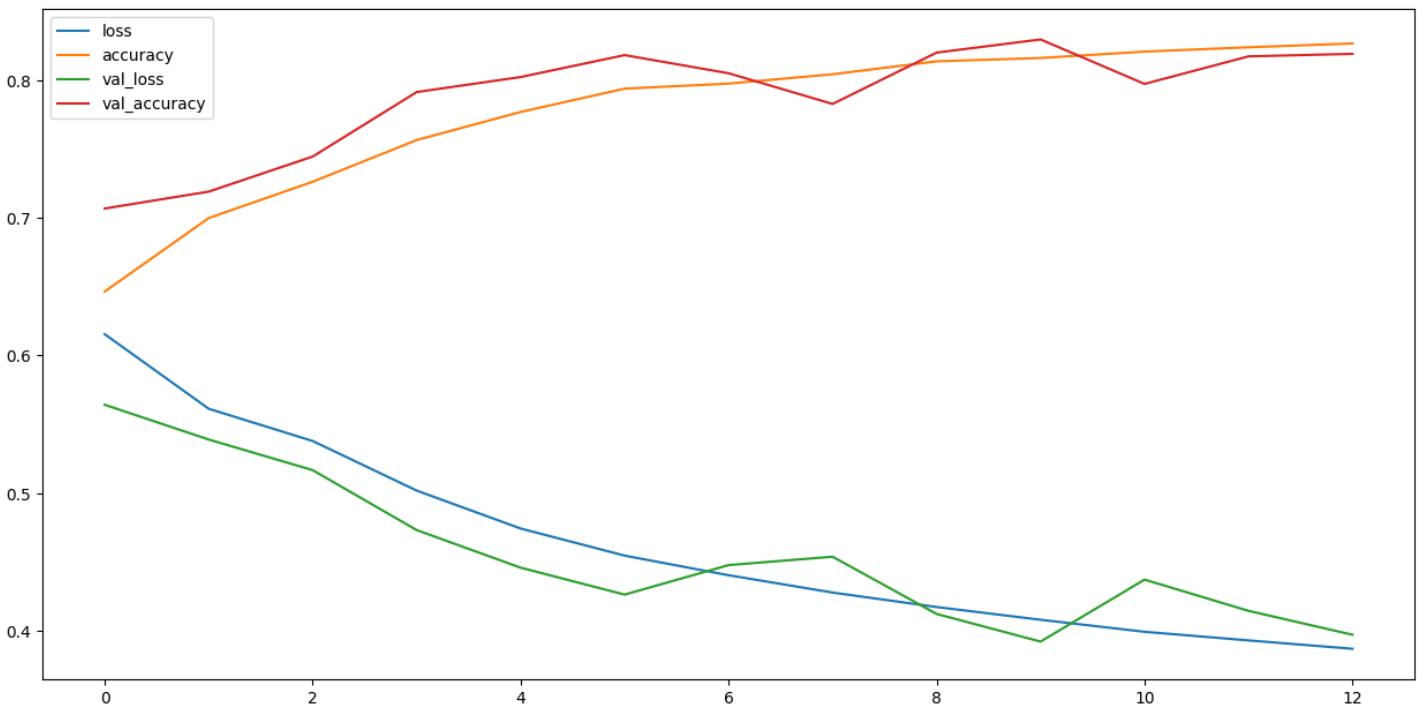


Figura 3.30: Plot dell’andamento dell’accuratezza e della loss durante il training

Come si può vedere in Figura 3.30, l’accuratezza è dell’80% e la loss è sotto al 0.4.

3.10 Valutazione delle performance

Dopo aver addestrato il modello si sono potute valutare le performance sul set di test. Per far ciò sono state fatte le predizioni, tramite il metodo `predict`, al quale è stato passato il `test_generator`. Come detto le predizioni consistono in un intero tra 0 e 1 e si è scelto di decidere, come soglia di divisione tra le due classi, il valore di 0.5. A questo punto è stata creata la `matrice di confusione`. Questa è una matrice 2 x 2 dove vengono confrontati le classi reali di ogni immagine di test con quelle predette dal modello. Ovviamente si vuole avere quanta più corrispondenza possibile con le classi reali e minimizzare invece il numero di predizioni errate. Come si può vedere in Figura 3.31 si ha un gran numero di **True Negative** (cioè il numero di immagini di classe reale Anomalo 0 che effettivamente sono state previste dal modello come classe 0) e di **True Positive** (cioè di classi reali 1 Normali predette come 1) ed un basso numero dei **False Positive**, cioè delle classi reali 0 predette come 1, e dei **False Negative**, cioè delle classi reali 1 predette come 0.

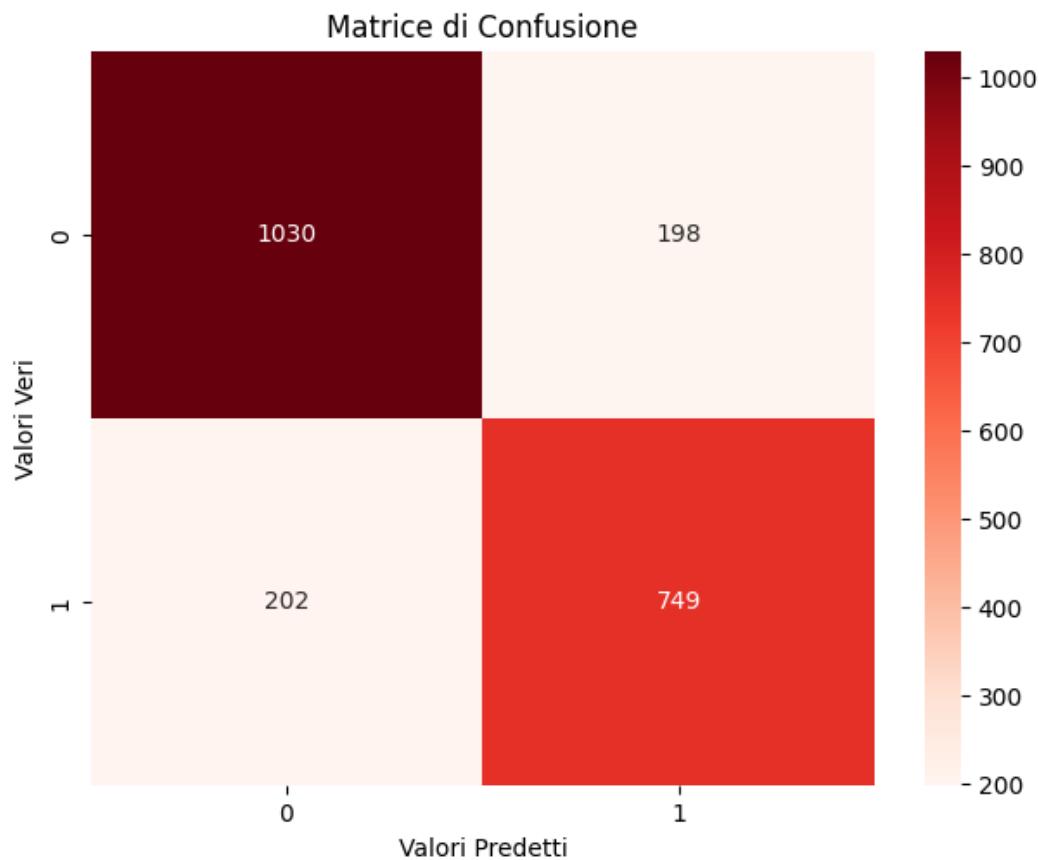


Figura 3.31: Matrice di confusione prodotta dal test della rete

Da questa matrice quello che si fa è calcolare altre metriche per vedere

la bontà del modello. Queste sono:

- **Precision** : Questa metrica indica il numero di TP ripetuto a tutte le positive predette. Più è vicino a 1, più il classificatore è performante. Formalmente : $Precision = TP/(TP + FP)$
- **Recall** : Questa metrica indica il numero di TP ripetuto a tutte le positive effettive. Più è vicino a 1, più il classificatore è performante. Formalmente : $Precision = TP/(TP + FN)$
- **F1-Score**: Questa metrica utilizza entrambe le precedenti per dare un valore complessivo. Più è vicino a 1, più il classificatore è performante. Formalmente : $F1Score = 2 \cdot (Precision \cdot Recall) / (Precision + Recall)$
- **Accuratezza** : Questa metrica è, come durante il training, il numero di classi corrette rispetto a tutte. Più è vicino a 1, più il classificatore è performante. Formalmente : $Accuracy = TP + TN / (TP + TN + FN + FP)$

I valori di tali metriche vengono calcolati con il metodo `classification_report` della libreria sklearn [8] e vengono riportati in Tabella 3.1.

| | Precision | Recall | F1-Score | Support |
|----------|-----------|--------|----------|---------|
| 0 | 0.84 | 0.84 | 0.84 | 1228 |
| 1 | 0.79 | 0.79 | 0.79 | 951 |
| accuracy | | | 0.82 | 2179 |

Tabella 3.1: Metriche prodotte durante il test del modello

Oltre a queste viene usata anche la metrica AUC (Area Under Curve). Nello specifico si tratta dell'area calcolata sotto la curva ROC (Receiver Operating Characteristic). Quest'ultima è una curva che mette in relazione il tasso di veri positivi, detto TPR o come detto precedentemente recall, e il tasso di falsi positivi, detto FPR, calcolato come $FP/(FP + TN)$. La curva ROC viene mostrata su un piano cartesiano tra (0, 0) e (1, 1) e la AUC indica proprio l'area al di sotto di tale curva. Più è grande quest'area, migliore sarà il modello a classificare le classi. In questo caso il valore di AUC è di 0.81. Per avere un buon valore di AUC è importante avere una curva ROC quanto più in alto a sinistra nel grafico, in modo da massimizzare il numero di veri positivi e minimizzando i falsi positivi e di conseguenza aumentare l'area interessata. La curva ROC prodotta è visibile in Figura 3.32.

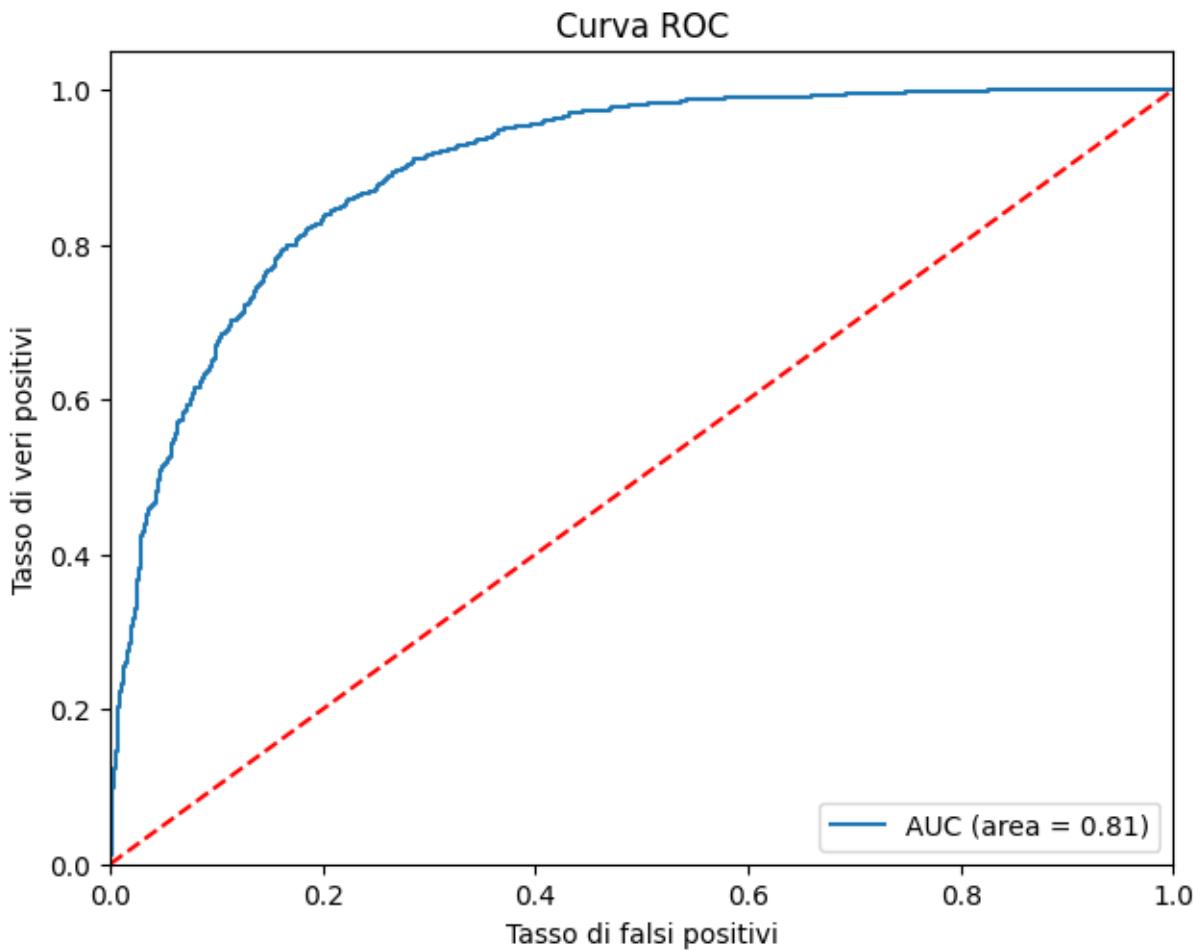


Figura 3.32: Curva ROC prodotta (in blu). La retta rossa invece mostra la ROC di un classificatore casuale

Come si può vedere il modello prodotto riesce a classificare nel modo corretto le immagini con una probabilità di oltre l'80%, confermando i valori ottenuti durante training e test.

Esistono diversi studi fatti analizzando, piuttosto che le immagini caratteristiche, i segnali così come sono. Non è questo il caso ovviamente, ma è utile per capire le differenze prestazionali del modello ottenuto durante questo lavoro e rapportarlo con modelli invece che lavorano in modo diretto sui segnali, per di più analizzando tutte e 12 le derivazioni messe a disposizione. Ad esempio in questo articolo [7] gli autori vanno a classificare i segnali ECG usando tutte e 12 le derivazioni tramite FSL (Few-Shot Learning). Questo è un tipo di metodo di apprendimento automatico in cui il dataset di addestramento contiene informazioni limitate. L'obiettivo di FSL è di costruire modelli di apprendimento automatico accurati con meno dati di addestramento. In questo paper gli autori hanno confrontato la classificazione degli ECG in 2, 5 e 20 classi diverse usando appunto FSL e

la classica softmax, sfruttando tra l'altro lo stesso dataset PTB-XL usato un questo lavoro. Ci si focalizza sulla classificazione a 2 classi, in quanto è la stessa tipologia di classificazione usata durante il tirocinio. Vanno poi ad addestrare una rete neurale convolutiva per il softmax e un'altra per FSL, usando in entrambi i complessi QRS di ognuna delle 12 derivazioni di ogni ECG. Anche i complessi QRS vengono ricavati tramite rete neurale. Dividono quindi il dataset in 70% train, 15% validazione e 15% test e procedono con un addestramento **k-fold**, cioè usando ripetendo il train e validazione k volte e usando in totale k test set diversi. Al fine di confrontare questo lavoro con quello del paper si riportano solo i risultati della classificazione a 2 classi.

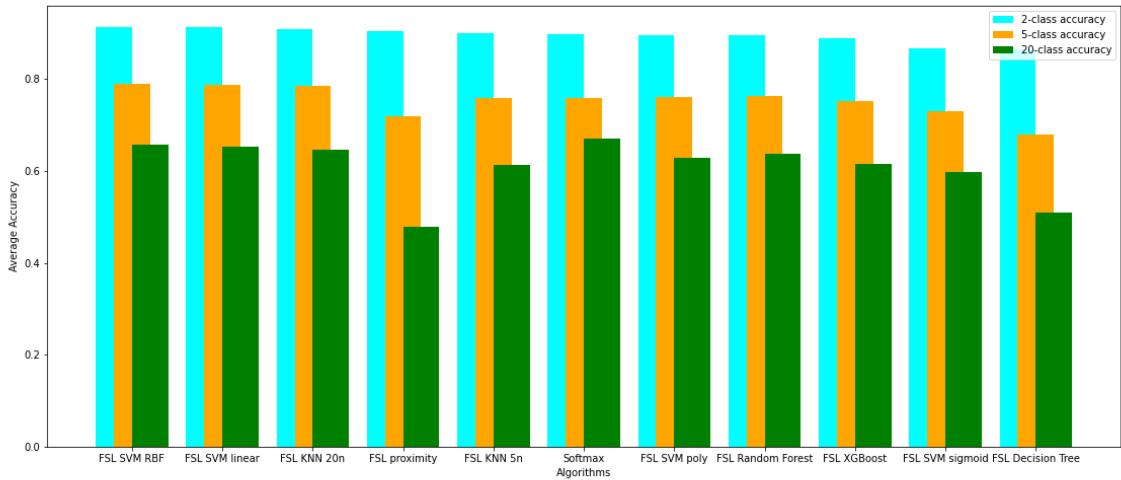


Figura 3.33: Confronto dell'accuratezza media per il rilevamento di 2 (celeste), 5 (arancione) e 20 (verde) classi

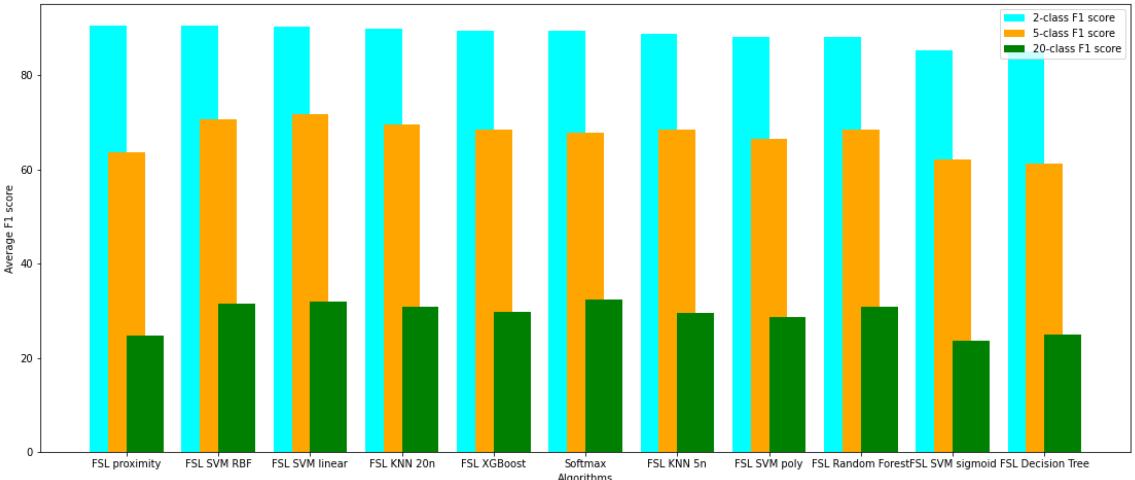


Figura 3.34: Confronto del punteggio F1 medio nel rilevamento di 2 (celeste), 5 (arancione) e 20 (verde) classi

Osservando la Figura 3.33 e la Figura 3.34 si possono vedere, limitandosi alla classificazione a 2 classi (cioè normale e anomalo), un'accuratezza di

circa il 90% ed un F1 degli stessi valori. Una migliore valutazione è visibile nella Figura 3.35.

| Technique | Acc | Acc Avg Std | F1 | F1 Avg Std | AUC | AUC Avg Std |
|----------------------------------|------------|---------------|-----------|--------------|-----------|---------------|
| FSL proximity-based | 89.5–91.1% | 90.4% 0.5% | 89.1–90.8 | 90.6 0.6 | 92.5–94.4 | 93.7 0.8 |
| Softmax-based classification | 89.2–90.5% | 89.7% 0.4% | 89.0–90.2 | 89.4 0.4 | 94.8–95.9 | 95.5 0.4 |
| FSL + XGBoost | 87.9–89.7% | 88.9% 0.7% | 86.5–88.5 | 87.7 0.8 | 95.1–97.2 | 96.1 0.7 |
| FSL + Random Forest | 87.8–91.2% | 89.4% 1.1% | 86.2–90.1 | 88.1 1.3 | 95.5–97.1 | 96.3 0.5 |
| FSL + Decision Tree | 84.9–88.9% | 86.4% 1.4% | 82.8–87.5 | 85.0 1.8 | 82.8–87.5 | 85.0 1.8 |
| FSL + KNN – 5 neighbors | 88.7–92.0% | 89.9% 1.2% | 87.1–91.2 | 88.8 1.4 | 93.9–96.4 | 94.6 0.9 |
| FSL + KNN – 20 neighbors | 88.1–93.3% | 90.9% 1.9% | 86.6–92.6 | 89.8 2.2 | 96.0–97.8 | 96.6 0.7 |
| FSL + SVM with linear kernel | 88.6–93.3% | 91.2% 1.6% | 87.4–92.9 | 90.3 1.8 | 96.1–97.6 | 96.9 0.5 |
| FSL + SVM with polynomial kernel | 87.2–93.0% | 89.6% 1.9% | 85.5–92.3 | 88.2 2.3 | 94.9–97.6 | 96.0 1.0 |
| FSL + SVM with RBF kernel | 89.2–93.3% | 91.3% 1.4% | 88.1–92.8 | 90.5 1.6 | 92.2–95.6 | 93.8 1.3 |
| FSL + SVM with Sigmoid kernel | 68.6–92.9% | 86.6% 9.1% | 66.2–92.2 | 85.3 9.6 | 83.6–95.3 | 88.2 4.0 |

Figura 3.35: Valori di accuratezza, F1 e AUC per la classificazione a 2 classi

Vengono anche riportate le due matrici di confusione, in modo da confrontarle con quella ottenuta da questo lavoro, visibile in Figura 3.31. Figura 3.36 e Figura 3.37 mostrano le rispettive matrici ottenute. Da notare che i valori ottenuti dalle matrici, in termini di accuratezza, sono pressoché identici tra le due modalità sperimentate. La differenza infatti si sente maggiormente sulla classificazione a 5 e a 20 classi, che però non essendo coerente con il lavoro di tirocinio non è stata riportata su questa relazione. Analizzando i risultati del paper si possono vedere valori ottimi, si parla di circa il 90% di accuratezza sul test set e un AUC addirittura del 95%. Si prendono in considerazione specialmente i valori ottenuti dalla classificazione binaria basata su softmax, in quanto è più tradizionale ed è più vicina al tipo di rete usata in questo lavoro. E' possibile fare qualche considerazione partendo da questi risultati: in particolare il paper ha analizzato in primis tutte e 12 le derivazioni, invece che sole 3 del tirocinio. Questo ovviamente porta ad avere una maggior quantità di dati utile per valutare anche altre derivazioni per considerare anche il più piccolo dei dettagli che magari, usando solo alcune derivazioni, si potrebbe perdere. In aggiunta il lavoro del paper con cui è stato effettuato il confronto ha utilizzato il segnale vero e proprio ed una rete neurale convolutiva per ricavare i segmenti QRS, aven-

do così un collegamento più diretto tra input e classificazione. Tuttavia, sebbene nel tirocinio si siano usate immagini caratteristiche di ogni ECG invece del segnale grezzo, si può dire che comunque il divario non è poi così alto. Infatti l'accuratezza è dell'82% sul test set e AUC è 0.81.

| | | Network | | |
|-----------|---------|---|---|---|
| | | NORM | others | Actual |
| Predicted | NORM | 1313 50.79% | 119 4.60% | 1432 91.69% 8.31% |
| | others | 155 6.00% | 998 38.61% | 1153 86.56% 13.44% |
| | sum_col | 1468 89.44% 10.56% | 1117 89.35% 10.65% | 2585 89.40% 10.60% |

Figura 3.36: Matrice di confusione per classificazione binaria con softmax

| | | Network | | |
|-----------|---------|---|--|---|
| | | NORM | others | Actual |
| Predicted | NORM | 706 54.43% | 83 6.40% | 789 89.48% 10.52% |
| | others | 52 4.01% | 456 35.16% | 508 89.76% 10.24% |
| | sum_col | 758 93.14% 6.86% | 539 84.60% 15.40% | 1297 89.59% 10.41% |

Figura 3.37: Matrice di confusione per classificazione binaria con Few-Shot

Un altro lavoro con il quale è possibile confrontare i risultati ottenuti da questo tirocinio si può fare analizzando la relazione di tirocinio di un collega [3]. In quel lavoro il collega ha usato lo stesso dataset PTB-XL, ha selezionato 3 derivazioni ed ha estratto i battiti da ogni ECG. Poi ha

creato le immagini per ogni battito (e quindi non per ogni ECG come invece avviene in questo lavoro), cercando di capire quale battiti fossero normali e quali no. Per farlo ha selezionato alcuni complessi QRS con i picchi evidenti, ha creato il **Gallery**, cioè un set di dati che contiene i pattern sotto forma di immagini di complessi QRS normali e anomali, ed il **Probe**, che contiene invece le immagini dei battiti da verificare per capire se si tratta di un probe normale o anomalo. In sostanza il collega va a confrontare le immagini dei complessi QRS del Probe con i pattern riconosciuti del Gallery. L'immagine del probe viene considerata di un determinato pattern quando è molto simile ad esso. In questo modo il collega riesce ad a riconoscere nel modo corretto circa l'87% dei battiti, il che è simile al risultato di questo lavoro. Il collega però ha selezionato le 3 derivazioni in modo diverso: infatti ha selezionato 2 delle 3 prendendo, per ogni ECG, le 2 derivazioni con i picchi R più alti e non, come in questo lavoro, le stesse per tutti gli ECG. In più, anche in questo caso, il collega utilizza l'immagine della forma del complesso QRS e non l'immagine caratteristica. Questo conferma una volta in più che i risultati di questo lavoro sono più che validi.

3.11 Conclusioni

In conclusione si può dire che il lavoro prodotto da questo tirocinio ha prodotto dei buoni risultati, con un'accuratezza dell'82% e un'AUC di 0.81. Questi valori potrebbero essere migliorati cambiando le linee guida di questo lavoro. Ad esempio si potrebbe cercare di includere nelle immagini tutte e 12 le derivazioni in qualche modo o almeno un'altra in più, sfruttando ad esempio il campo alfa dei canali RGBA piuttosto che il classico RGB. Infatti in questo lavoro ci si è focalizzati sulla selezione di sole 3 derivazioni per avere una corrispondenza diretta con i 3 canali RGB per la creazione delle immagini caratteristiche.

Bibliografia

- [1] Carlos Carreiras et al. *BioSPPy: Biosignal Processing in Python*. [Online; accessed]. 2015-. URL: <https://github.com/PIA-Group/BioSPPy/>.
- [2] Alex Clark. “Pillow (PIL Fork) Documentation”. In: (2015). URL: <https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf>.
- [3] Vlad Ghitun. “Classificazione di battiti da ECG tramite pattern caratteristici”. Tesi di dott. Università di Roma La Sapienza, Anno accademico 2021/2022.
- [4] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.
- [5] Dominique Makowski et al. “NeuroKit2: A Python toolbox for neurophysiological signal processing”. In: *Behavior Research Methods* 53.4 (feb. 2021), pp. 1689–1696. DOI: 10.3758/s13428-020-01516-y. URL: <https://doi.org/10.3758/s13428-020-01516-y>.
- [6] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.
- [7] Krzysztof Pałczyński et al. “Study of the Few-Shot Learning for ECG Classification Based on the PTB-XL Dataset”. In: *Sensors* 22.3 (2022). ISSN: 1424-8220. DOI: 10.3390/s22030904. URL: <https://www.mdpi.com/1424-8220/22/3/904>.
- [8] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [9] Patrick Wagner et al. “PTB-XL, a large publicly available electrocardiography dataset”. In: *Scientific data* 7.1 (2020), p. 154.