



**UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE**

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

**PYVHR: UN FRAMEWORK DI ANALISI
VIDEO PER LA STIMA REMOTA DEL
BATTITO CARDIACO**

RELATORE: Prof. Giuliano Grossi

CORRELATORE: Prof. Donatello Conte

TESI DI LAUREA DI: Edoardo Mortara

MATRICOLA: 954198

ANNO ACCADEMICO 2021-22

Indice

1 rPPG mediante Face Sampling	7
1.1 Definizioni preliminari e assunzioni	7
1.1.1 Analisi spettrale a livello di patch	13
1.2 Stima basate su statistiche e clustering	14
1.2.1 Mediana delle PSD	15
1.2.2 Clustering delle PSD	17
2 Il Framework	22
2.1 Introduzione	22
2.1.1 Installazione	23
2.2 La Pipeline	23
2.2.1 Skin extraction	26
2.2.2 RGB signal processing	29
2.2.3 BVP extraction	30
2.2.4 Pre e Post-filtering	30
2.2.5 BPM estimation	33
2.2.6 Tecniche di stima del BPM basate su multi-stimatori	35
2.3 Pipeline per metodi deep-learning	35
2.3.1 Utilizzo della 'DeepPipeline'	36
2.4 Parallelizzazione del codice	37
2.4.1 CPU	39
2.4.2 GPU - CUDA	40
2.5 GUI per l'elaborazione real time	41
2.6 Estensione del framework	43
2.6.1 Aggiungere un nuovo Metodo rPPG	43
2.6.2 Aggiungere un nuovo Dataset	43
3 Analisi	45
3.1 Valutazione dei metodi rPPG	45
3.2 Metriche di valutazione	47
3.3 Datasets di benchmark	49

3.4	Analisi di un Dataset attraverso la Pipeline	51
3.5	Significance Testing	54
4	Risultati sperimentali	60
4.1	Risultati multi-dataset	60
4.2	Heat maps delle Patches	69
4.3	Caso di studio: rilevare DeepFake con pyVHR	70
4.4	Conclusioni e sviluppi futuri	74

Introduzione

Remote Photoplethysmography (rPPG)

Il metodo tradizionale per il monitoraggio della frequenza cardiaca è la misurazione dei picchi di elettricità generata nel cuore a ogni pulsazione che va sotto il nome di elettrocardiogramma (ECG). Normalmente un dispositivo ECG portatile è collocato su una fascia toracica che lo tiene centrato in corrispondenza del cuore. Questo può trasmettere i valori acquisiti tramite comunicazione wireless a un computer o a uno smartphone. Tuttavia questi tipi di dispositivi sono poco pratici per le diverse attività cui si può sottoporre un soggetto, si pensi ad esempio all'esercizio fisico, e sono spesso usate solo a livello ambulatoriale.

Un'alternativa già ampiamente utilizzata in vari ambiti sia medicali sia sportivi per la misurazione della frequenza cardiaca è la pulsossimetria. A questo riguardo, un metodo di misurazione ottica, noto come fotopletismografia (PPG), misura la variazione del volume del sangue attraverso la distensione di arterie ed arteriole nel tessuto sottocutaneo. Di norma questa tecnologia è implementata in una clip da apporre sulla falange del dito come un anello. Il dispositivo emette un fascio di luce attraverso la pelle (da un Led posto su una delle due facce) e misura le variazioni nella trasmissione della luce all'interno del dito (tramite un fotodiodo posto sull'altra faccia del dispositivo). La quantità di luce rilevata in prossimità del fotosensore si abbassa bruscamente e brevemente a ogni impulso, dato che l'aumento volumetrico del sangue assorbe una maggiore quantità di luce. Il segnale proveniente dal fotodiodo è un'onda simile a un dente di sega, la cui frequenza fondamentale corrisponde alla frequenza cardiaca. Specificatamente, l'assorbimento della luce nella cute dipende dall'ossigenazione del sangue nei microvasi dell'epidermide; l'emoglobina ossigenata assorbe lunghezze d'onda differenti rispetto a quella deossigenata. Il pulsossimetro utilizza due led che emettono luce a specifiche lunghezze d'onda per calcolare la quantità di emoglobina ossigenata e deossigenata presente nel vaso sanguigno (di solito 660 nm per riconoscere la presenza di sangue ossigenato, e 940 nm per quello deossigenato). Dal rapporto di queste due misure può essere ricavata la saturazione dell'ossigeno nel sangue (SpO_2) utilizzando la legge di Beer–Lambert (1).

Attraverso complesse trasformazioni di questi dati il pulsossimetro permette di ot-

tenere un segnale fisiologico chiamato *Blood Volume Pulse* (BVP). La Figura 1 mostra nel dettaglio un segnale BVP; possiamo osservare che questo segnale descrive molti eventi fisiologici: durata della vasocostrizione, intervallo IBI (*Interbeat interval*, ovvero il periodo tra un battito cardiaco e quello successivo), contropulsazione aortica (usato anche per analizzare eventuali problemi cardiaci). Il più importante di questi è

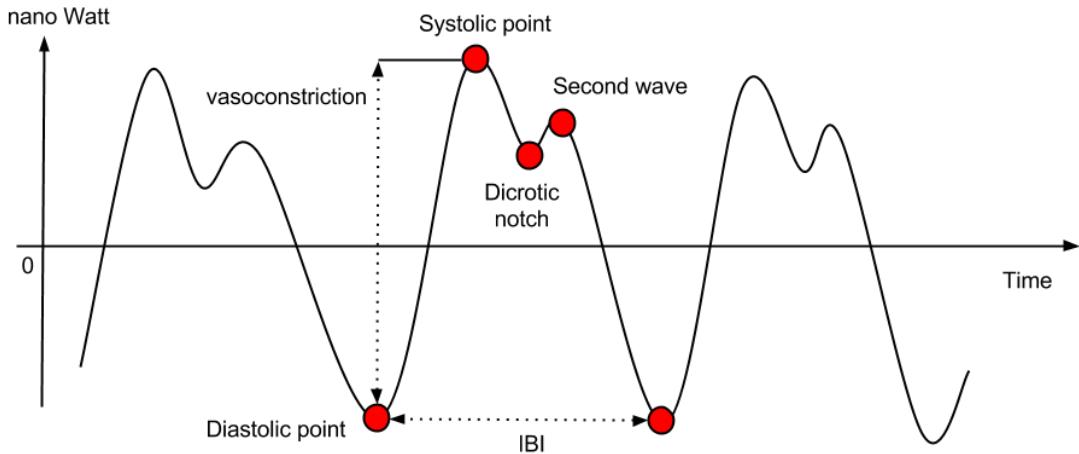


Figura 1: Analisi delle caratteristiche del segnale BVP. Partendo da in alto a sinistra: vasocostrizione (*vasoconstriction*), diastole (*diastolic point*), sistole (*systolic point*), contropulsazione aortica (*dicrotic notch*), dicrotic wave (*second wave*), interbeat interval (*IBI*).

l’intervallo IBI del segnale BVP, perché permette di calcolare la frequenza cardiaca del soggetto (heart rate - HR).

L’utilizzo dei pulsossimetri è estremamente diffuso nella nostra società da vari decenni e permette, in modo poco invasivo ed economico, di monitorare le condizioni vitali di un soggetto/paziente. Mentre la photoplethysmography richiede comune mente una qualche forma di contatto con la pelle umana (ad esempio, orecchio, dito), la *remote photoplethysmography* (rPPG) consente di monitorare i processi fisiologici come il flusso sanguigno senza utilizzare dispositivi a contatto con la pelle. Ciò si ottiene utilizzando il video che inquadra la pelle (per esempio quella del viso) di un soggetto per analizzare sottili cambiamenti momentanei nel colore della stessa, altrimenti non rilevabili dall’occhio umano (Verkruyse et al. 2008 (2), Chihiro Takano and Yuji Ohta 2007 (3)).

Essendo una tecnica non invasiva, può essere utilizzata per monitorare la frequenza cardiaca dei neonati, oppure per analisi di tipo emotivo legate al mondo del cosiddetto affective computing e della salute in persone fragili (health care). La pandemia di COVID-19 ha travolto le infrastrutture sanitarie esistenti in molte parti del mondo. Gli operatori sanitari non sono solo sovraccarichi, ma anche ad alto rischio di trasmissione nosocomiale da pazienti COVID-19. Una tecnica come quella offerta dall’rPPG

permette appunto di monitorare importanti segnali fisiologici in remoto e in totale sicurezza di un gran numero di individui suscettibili o infetti.

Tuttavia, anche se la tecnica risulta molto interessante in virtù dei vantaggi offerti e accennati brevemente sopra, essa non è immune da problemi legati soprattutto alle tipologie di rumore da cui è affetto il segnale video catturato dalla camera che, se non adeguatamente considerato nella modellazione, tende a degradare le prestazioni di stima del battito cardiaco in modo impredicibile. Infatti per meglio comprendere i principi di estrazione della pulsazione che risiedono nei metodi rPPG si fa spesso riferimento a un ben preciso modello rPPG che prende in considerazione le proprietà ottiche e fisiologiche della riflessione della luce da parte della pelle catturata dalla camera. Si veda a tal proposito il lavoro di Wang et al. (4) per comprendere alcuni aspetti di rilievo del modello di riflettanza della pelle come riportato in Figura 2, e il modo in cui è possibile estrarre la componente della pulsazione cardiaca dal segnale luminoso catturato da un sensore video. Nel modello si assume che la sorgente luminosa abbia una composizione spettrale costante ma con intensità variabile e che l'intensità osservata dalla camera dipenda dalla distanza dalla sorgente luminosa, dal tipo di tessuto di cui è composta la pelle e dal tipo di camera. La pelle misurata dalla camera ha un certo colore che varia nel tempo a causa di variazioni di intensità indotte dal moto (del capo) e dalle più impercettibili variazioni cromatiche indotte dalla pulsazione cardiaca. Tutte queste variazioni temporali sono proporzionali al livello di intensità luminosa. Secondo il modello di riflessione dicromatico la radianza spettrale è definita dalla componente speculare e da quella diffusa (Figura 2). Quando i vasi sanguigni contengono sangue ossigenato il derma assorbirà lunghezze d'onda differenti rispetto a quelle assorbite da sangue non ossigenato, quindi la componente diffusa sarà differente. Per questo motivo analizzando nel tempo il segnale luminoso ricevuto da un sensore otteniamo una funzione periodica che descrive il ciclo cardiaco.

Il framework pyVHR

Per promuovere lo sviluppo di nuovi metodi rPPG e la loro analisi sperimentale, nel 2020 è stato proposto pyVHR (5), una versione preliminare di un framework per la stima rPPG tradizionale, insieme a una solida valutazione statistica per la comparazione delle prestazioni dei vari metodi. Tuttavia, quella versione preliminare mostrava alcuni limiti, sia in termini di organizzazione/ingegnerizzazione del codice, sia sotto il profilo dell'usabilità e scalabilità. Inoltre, un limite forte era dato dalla restrizione ai soli approcci cosiddetti tradizionali, cioè basati su tecniche statistiche, data-driven o model-based e non adeguata alla modellazione offerta da tecniche di machine learning, come nel caso di architetture neurali profonde come CNN 2D o 3D.

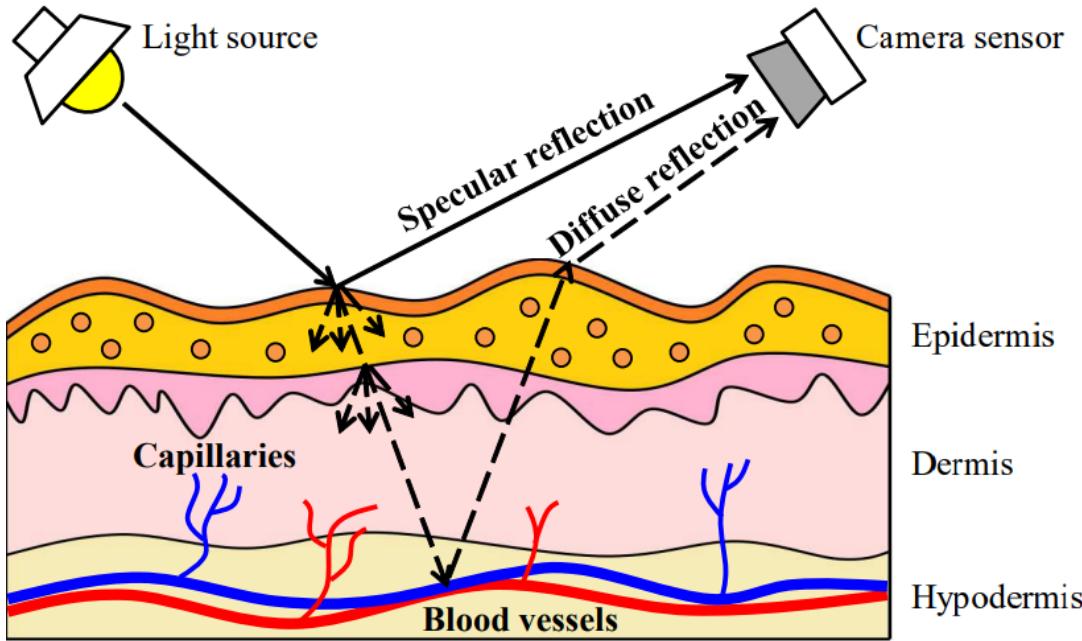


Figura 2: Il modello di riflessione dicromatico della pelle. La radiazione spettrale percepita dal sensore è composta dalla componente speculare e diffusa, dove solo la componente diffusa contiene informazioni pulsatili.

In questo lavoro viene presentata una nuova versione di pyVHR caratterizzata da molti elementi innovativi sotto diversi profili, quali ad esempio aspetti metodologici per rendere più efficienti i metodi implementati, aspetti tecnici o ingegneristici di miglioramento del codice e in generale di usabilità/flessibilità che permette facile estensione a nuovi metodi o nuovi dataset da esaminare, come di seguito riassunto.

- Viene affrontato in modo efficiente il problema della segmentazione della pelle del volto con una tecnica innovativa basata su una rete neurale in grado - non solo di segmentare mediante detection del volto - ma anche di fornire una mesh di 486 landmark fedelmente tracciati nel tempo, un frame dopo l'altro.
- Una tecnica basata su *convex hull* (*inviluppo convesso*) per isolare dai frame del video i pixel associati alla pelle del volto.
- In alternativa alla tecnica illustrata sopra, pyVHR propone una segmentazione semantica del volto basata su rete neurale BiSeNet (6).
- Viene introdotta una visione dicotomica del monitoraggio remoto della frequenza cardiaca, che porta a due distinte classi di approccio: tecniche tradizionale e tecniche di deep-learning.
- Per i metodi deep viene fornita una pipeline distinta dalla pipe per i metodi tradizionali (già presente nel framework originale e qui migliorata dalle tecniche

innovative descritte) capace di integrare un tipico processo end-to-end delle architetture deep (cioè un procedimento che dal video raw porta alla stima del battito) con gli stadi della pipe destinati all’analisi comparativa finale.

- Per le tecniche tradizionali è presente un’ulteriore distinzione riguardo le regioni di interesse (ROI) prese in considerazione: viene distinto il concetto di approccio olistico, che prende in esame l’intero volto del soggetto, da quello basato sul campionamento del volto attraverso molteplici patch (regioni rettangolari centrate in punti specifici del volto detti landmark) distribuite in modo quasi uniforme sul volto e tracciate con coerenza nel tempo.

Stato dell’arte

Nell’ultimo decennio il mondo rPPG ha ricevuto forte interesse da parte di molti ricercatori (4, 7, 8, 9, 10, 11). Tuttavia, è stato in generale trascurato il problema di una valutazione equa e riproducibile dei metodi rPPG; per esempio in (9) viene presentata una collezione di algoritmi rPPG scritti in Python, senza però valutarli. È innegabile che le valutazioni teoriche siano quasi irrealizzabili, date le complesse operazioni o trasformazioni che ogni algoritmo esegue. Tuttavia, i confronti empirici potrebbero essere molto istruttivi se condotti alla luce di alcuni criteri metodologici (5). In sintesi: standardizzazione pre/post elaborazione; valutazione riproducibile; test su Datasets multipli; valutazione statistica rigorosa. La prima versione di pyVHR, infatti, aveva proprio lo scopo di introdurre questi criteri (5).

Altri lavori come (10) sono molto interessanti perché viene presentata una metodologia ben definita; in particolare in questo lavoro viene descritta la stima della frequenza cardiaca come un processo suddiviso in cinque fasi: selezione delle ROI, pre-elaborazione, metodo rPPG, post-elaborazione, stima della frequenza cardiaca. Viene presentato un framework teorico per valutare diverse pipeline al fine di scoprire quale combinazione fornisca la stima PPG più precisa. I risultati sono riportati nel dataset DEAP (12), però non è stato rilasciato codice pubblico.

In (13) viene presentato un toolbox MATLAB, che implementa due nuovi metodi proposti, ovvero *Local Group Invariance* (LGI) (14) e il metodo *Reimannian-PPGI* (SPH) (13); questi sono confrontati con il metodo GREEN (15), e due metodi all’avanguardia, ovvero il *Spatial Subspace Rotation* (SSR) (16) e il *Projection Orthogonal to Skin* (POS) (4).

In (11) gli autori propongono un toolbox MATLAB che implementa diversi metodi: *Green Channel*, *POS*, *CHROM* (17), *ICA* (18) e *BCG* (19). Il toolbox si presenta come una semplice raccolta di implementazioni di metodi tradizionali, senza mirare a creare un framework di confronto rigoroso su uno o più Datasets.

La tabella 1 riassume le principali differenze tra pyVHR e i principali framework esistenti.

	Lang.	Modular	Deep-Ready	Multi-Data	Stat. Assessment
pyVHR	Python	✓	✓	✓	✓
(11)	MATLAB	✗	✗	✗	✗
(9)	Python	✗	✗	✗	✗
(13)	MATLAB	✗	✗	✓	✗

Tabella 1: Un confronto tra i framework rPPG disponibili gratuitamente.

Gli algoritmi rPPG fin'ora proposti si suddividono in due tipologie. La prima è composta dai metodi tradizionali; per esempio CHROM (17) è basato sulla crominanza, POS (16) su piani di proiezione ortogonali al colore della pelle, SSR (16) su *spatial subspace rotation...*

La seconda invece è composta dai metodi basati su reti neurali e apprendimento automatico (*Machine Learning*). Questa tipologia di metodi è molto variegata; abbiamo reti spazio-temporali basate su LSTM e CNN (Yu et al. (20)), oppure *convolutional attention network* (CAN) ibride basate sia su attenzione spaziale che temporale (Liu et al. (21)). In generale essendo modelli ML, queste tecniche sono molto influenzate dal Dataset utilizzato per il *learning* della rete. Per approfondire i pro e contro di questa tipologia di metodi consiglio di leggere il lavoro di Schrumpf et al. (22).

Questo lavoro di tesi è così organizzato. Nel capitolo 1 viene descritto matematicamente il processo rPPG basato su campionamento facciale mediante patch. procedi qui con descrizione dei capitoli

Nel capitolo analizzeremo nel dettaglio la procedura di analisi e valutazione dei metodi rPPG, e i relativi risultati sperimentali.

Capitolo 1

rPPG mediante Face Sampling

1.1 Definizioni preliminari e assunzioni

Di seguito diamo le principali nozioni usate e i principi algoritmici alla base della stima della frequenza della pulsazione cardiaca utilizzando i metodi tradizionali rPPG e tecniche di stima basate sulla combinazione di molteplici stimatori.

Assumiamo di avere in input una sequenza di T frame RGB (finestra). Il t -esimo frame (con $t = 1, \dots, T$) è una collezione di pixel $\{\mathbf{c}_{i,j} : \text{per ogni pixel in posizione } (i,j)\}$ ognuno definito dal vettore

$$\mathbf{c}_{i,j}(t) = (r_{i,j}(t), g_{i,j}(t), b_{i,j}(t))^{\top}$$

dove $r_{i,j}(t), g_{i,j}(t), b_{i,j}(t)$ rappresentano i canali rosso, verde e blu per il pixel in posizione (i,j) e \top denota la trasposizione del vettore.

Se non diversamente specificato, nel resto della tesi si assume implicitamente che un dato video $v \in \mathbb{R}^{h \times w \times 3 \times N}$ di N frame RGB di dimensione (h, w) sia suddiviso in $K = \lfloor N/(T-S) \rfloor$ finestre sovrapposte, dove T è il numero di frame e S lo stride. Pertanto, ogni stima della pulsazione condotta secondo questo modellazione è limitata a una finestra di esattamente T frame e ottenuta indipendentemente l'una dall'altra. Sulla base di questo schema, l'intero processo di valutazione prende come input un video e produce K stime di BPM, una per ogni finestra considerata.

Nell'approccio basato su *patch*, ovvero regioni rettangolari, consideriamo una *mesh* di *patch* sparse centrata su $\mathcal{L} = [1..L]$ punti di riferimento (*landmarks*) da cui estrarre delle tracce RGB come descritto nella prossima sezione. Un esempio di *landmarks* e relative *patch* su volto è fornito in Figura 1.1. La prima immagine (a) mette in evidenza un gruppo di *landmarks* posizionato in modo uniforme su tutta la faccia. Le immagini successive (b-d) mostrano le *patches* ottenute scegliendo casualmente un numero crescente di *landmarks* (da 25 a 100). Ogni *patch* presenta anche il suo numero identifi-

cativo a fianco.

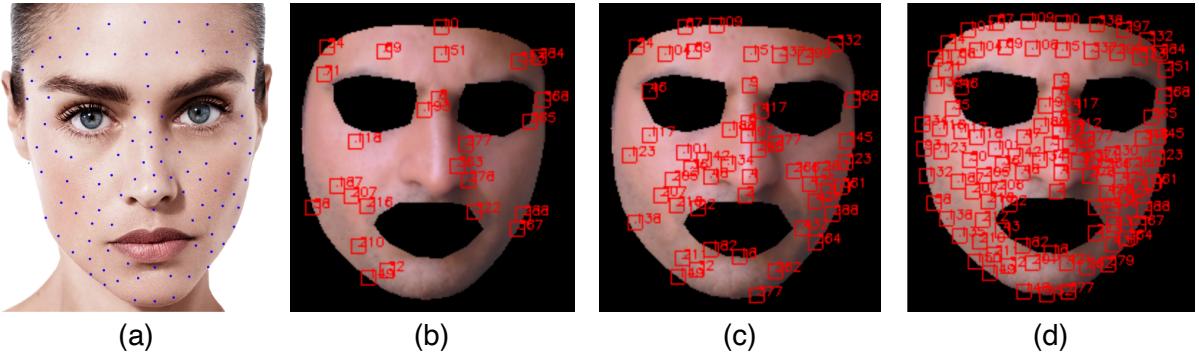


Figura 1.1: Modello con 100 *landmarks* posizionati su un viso in modo quasi uniforme (a). *Patches* centrate rispettivamente su 25 (b), 50 (c) e 100 (d) punti di riferimento (*landmarks*).

Come si può vedere dalla Figura 1.1, i volti sono stati sottoposti a una rimozione dei pixel non cutanei (ad esempio occhi-sopracciglia, bocca e sfondo), perché le informazioni più rilevanti per le tecniche rPPG sono contenute unicamente nelle regioni di pelle del viso. Sulla base di questa osservazione, è fondamentale introdurre un metodo per selezionare solo le regioni di pixel corrispondenti alla pelle del soggetto. È ben nota la poca efficacia della maggior parte dei metodi di segmentazione della pelle che utilizzano soglie predefinite del colore della pelle per stabilire un confine binario tra primo piano e sfondo o gruppi di pelle/non pelle. È anche evidente il dilemma che ne deriva nella scelta di soglie idonee a delimitare tali regioni di interesse.

Come discusso nel prossimo capitolo, il *framework* pyVHR si appoggia pesantemente su uno strumento molto efficace nell'isolare le regioni facciali da quelle irrilevanti e allo stesso tempo di tracciare accuratamente i *landmarks* lungo la sequenza di frame racchiusa nella finestra temporale di dimensione arbitraria T .

La seconda fase del *face sampling* è comune a tutti i metodi rPPG basati su modelli ed è correlata alla quantizzazione spaziale. Data una sequenza video in input contenente tessuto cutaneo vivente, calcoliamo una media spaziale dei valori RGB dei pixel cutanei all'interno delle *patches* in ciascun fotogramma, quindi concateniamo temporaneamente questi valori ottenuti da fotogrammi consecutivi in vari segnali di colore RGB (*RGB colour signals*).

Dato un insieme di *patches* sparse centrata su punti di riferimento \mathcal{L} , per ogni punto di riferimento $l \in \mathcal{L}$ avremo una patch P_l di dimensioni arbitrarie che descrive un insieme di pixel contenenti informazioni relative alla PPG, e per ogni fotogramma

$t \in [1..T]$ calcoliamo le intensità di colore medie su $P_l(t)$ (*RGB colour signals*):

$$\bar{\mathbf{c}}_l(t) = (\bar{r}_l(t), \bar{g}_l(t), \bar{b}_l(t))^\top = \frac{1}{|P_l(t)|} \sum_{(i,j) \in P_l(t)} \mathbf{c}_{i,j}(t) \quad (1.1)$$

dove $|P_l(t)|$ è il numero di pixels in $P_l(t)$.

La Figura 1.2 mostra la procedura di suddivisione e tracciamento basata sulle *patches* e il segnale RGB (1.1) raccolto per ogni fotogramma t sulla patch $P_l(t)$.

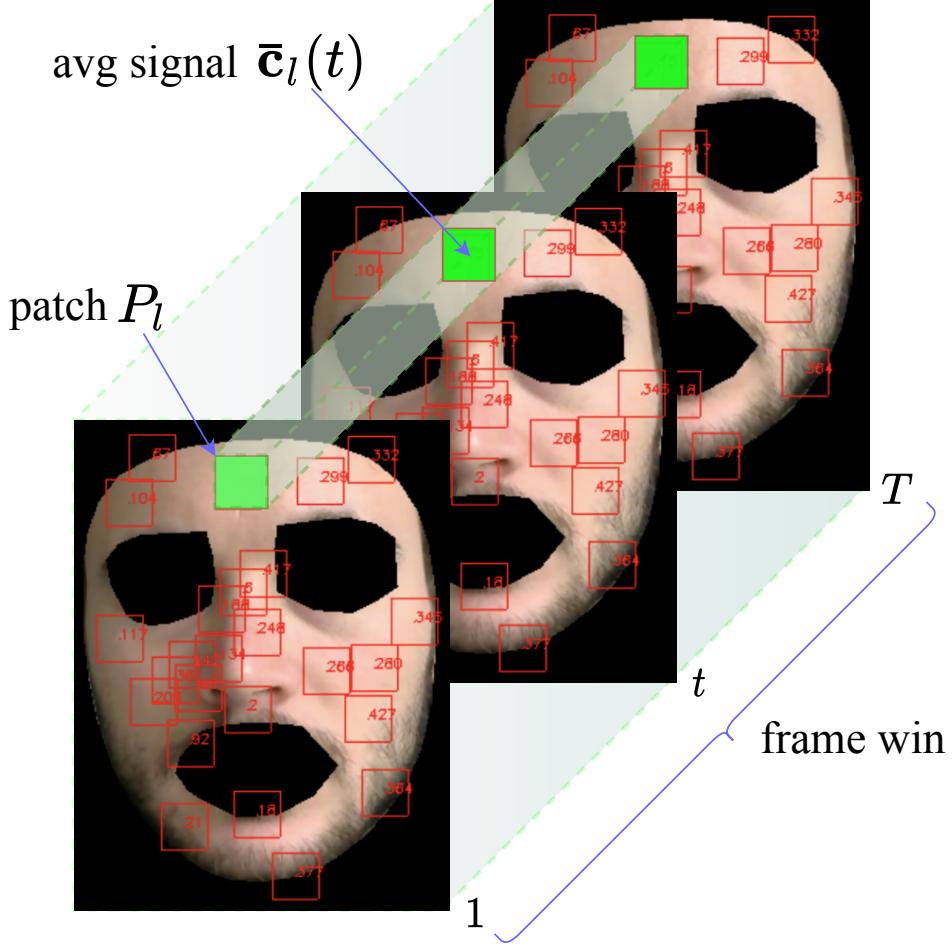


Figura 1.2: Tracciamento della patch all'interno di una finestra temporale di frames e il corrispondente calcolo del segnale colore RGB per ogni frame t .

Supponendo che un intervallo video abbia T frame, le tracce RGB $\bar{\mathbf{c}}_l$ forniscono un vettore nello spazio $\mathbb{R}^{3 \times T}$. La prima dimensione rappresenta i canali di colore e la seconda un totale di tre segnali colore ordinati in ordine R-G-B. Con una videocamera che registra τ fotogrammi al secondo (fps), l'intervallo di tempo dei dati in (1.1) copre T/τ secondi.

È buona norma eliminare la *DC component* nei segnali colore normalizzando tem-

poraneamente ogni riga di (1.1):

$$\tilde{\mathbf{c}}_l = \left[\frac{\bar{\mathbf{r}}_l}{\mu(\bar{\mathbf{r}}_l)}, \frac{\bar{\mathbf{g}}_l}{\mu(\bar{\mathbf{g}}_l)}, \frac{\bar{\mathbf{b}}_l}{\mu(\bar{\mathbf{b}}_l)} \right]^\top \in \mathbb{R}^{3 \times T}$$

dove l'operatore di media temporale $\mu(\cdot)$ viene applicato a ciascun canale RGB (17). Sia la normalizzazione spaziale che quella temporale svolgono un ruolo importante nel dominare le deviazioni rPPG: la media spaziale dei pixel diminuisce l'errore di quantizzazione della fotocamera (4), mentre la normalizzazione temporale mira ad eliminare la dipendenza del segnale \mathbf{c}_l dal colore riflesso medio della pelle, considerato come la componente più stabile in un intervallo di tempo.

Nella fase di *pre-elaborazione*, è anche comune applicare filtri semplici, ma potenti, per aumentare la robustezza degli algoritmi rPPG. Il motivo per concentrarsi su questo punto è dovuto al suo grande impatto sulla stima della frequenza del polso. La caratteristica comune dei filtri è che funzionano nel dominio della frequenza, permettendo di gestire diverse distorsioni di frequenza, mantenendo le traiettorie e le frequenze di colore che sono di interesse per la larghezza di banda della frequenza cardiaca umana (40-240 BPM che corrispondono a 0,65-4 Hz).

Una tipica categoria di filtri di riduzione del rumore molto utilizzata è la famiglia dei filtri temporali che contiene, tra gli altri, il filtro *detrend*, *moving average* e passa-banda. Il *detrend* aiuta a isolare la componente del segnale rPPG che rappresenta la pulsazione, riducendo drasticamente le basse frequenze del segnale grezzo che determinano un andamento non stazionario dei segnali (23, 24). Il filtro *moving average* attenua il segnale sopprimendo il rumore casuale ad alta frequenza causato da improvvisi cambiamenti di colore dovuti alla luce o ai movimenti utilizzando la media temporale di fotogrammi consecutivi. I filtri classici come il filtro passa-banda vengono utilizzati anche per rimuovere le frequenze irrilevanti al di fuori della larghezza di banda della frequenza cardiaca. I tipi più utilizzati sono i filtri passa-banda *Butterworth* o i filtri passa-banda *FIR* ampiamente documentati in letteratura.

E' importante evidenziare che tutti i filtri citati sono spesso usati in combinazione (23); quindi il segnale finale si ottiene applicando k filtri in cascata al segnale normalizzato spazialmente e temporalmente $\tilde{\mathbf{c}}_l$ relativo alla patch P_l :

$$\hat{\mathbf{c}}_l = \text{FILT}_1(\dots \text{FILT}_k(\tilde{\mathbf{c}}_l) \dots).$$

Nella Figura 1.3 sono riportati alcuni esempi (scegliendo un solo canale tra $\hat{\mathbf{r}}_l$, $\hat{\mathbf{g}}_l$ e $\hat{\mathbf{b}}_l$) di segnali colore RGB grigi (immagine in alto) ottenuti da *patches* e le loro versioni filtrate (immagine in basso) dopo l'applicazione di *detrend* e *passa-banda*. In questo

esempio la sequenza temporale analizzata è di 6 secondi e il frame rate è 30.



Figura 1.3: Campioni di segnali RGB prelevati da alcune patches. In alto i segnali grezzi, mentre in basso quelli filtrati con l'applicazione di *detrend* e *passa-banda*. I colori delle linee sono puramente casuali e privi di significato.

La fase successiva prevede di trasformare il segnale RGB in un segnale BVP (*Blood Volume Pulse*). Per far ciò si utilizzano i metodi/algoritmi rPPG; nel capitolo successivo analizzeremo i metodi più utilizzati, ovvero ICA, PCA, GREEN, CHROM, POS, SSR, LGI, PBV.

In generale le differenze essenziali tra questi metodi rPPG sono nel modo di combinare i segnali RGB in un segnale BVP. Ad esempio PCA e ICA sono ispirati alle tecniche di *Blind Source Separation* (BSS) che non impongono alcuna assunzione sui colori associati ai segnali sorgente. PBV e CHROM sono metodi basati su modelli che controllano il *demixing* del segnale sfruttando i vettori di colore delle diverse componenti.

Come affermato, lo spirito di questo lavoro è quello di indagare i principi algoritmici del rPPG in un contesto *patch-based* e soprattutto capire il perché e il come utilizzare più stime ottenute da P_l patches può contribuire ad aumentare la precisione del calcolo della frequenza cardiaca.

Per prima cosa dobbiamo definire un metodo rPPG come la funzione $\mathcal{M}_{\text{rPPG}}$ che mappa un segnale RGB $\hat{\mathbf{c}}_l \in \mathbb{R}^{3 \times T}$ estratto da una patch l e in un segnale BVP $\mathbf{b}_l \in \mathbb{R}^{1 \times T}$:

$$\mathbf{b}_l = \mathcal{M}_{\text{rPPG}}(\hat{\mathbf{c}}_l), \quad \forall l \in \mathcal{L}. \quad (1.2)$$

La Figura 1.4 mostra tre esempi di mappatura (1.2); dall'alto verso il basso i metodi applicati sono: CHROM, POS e LGI.

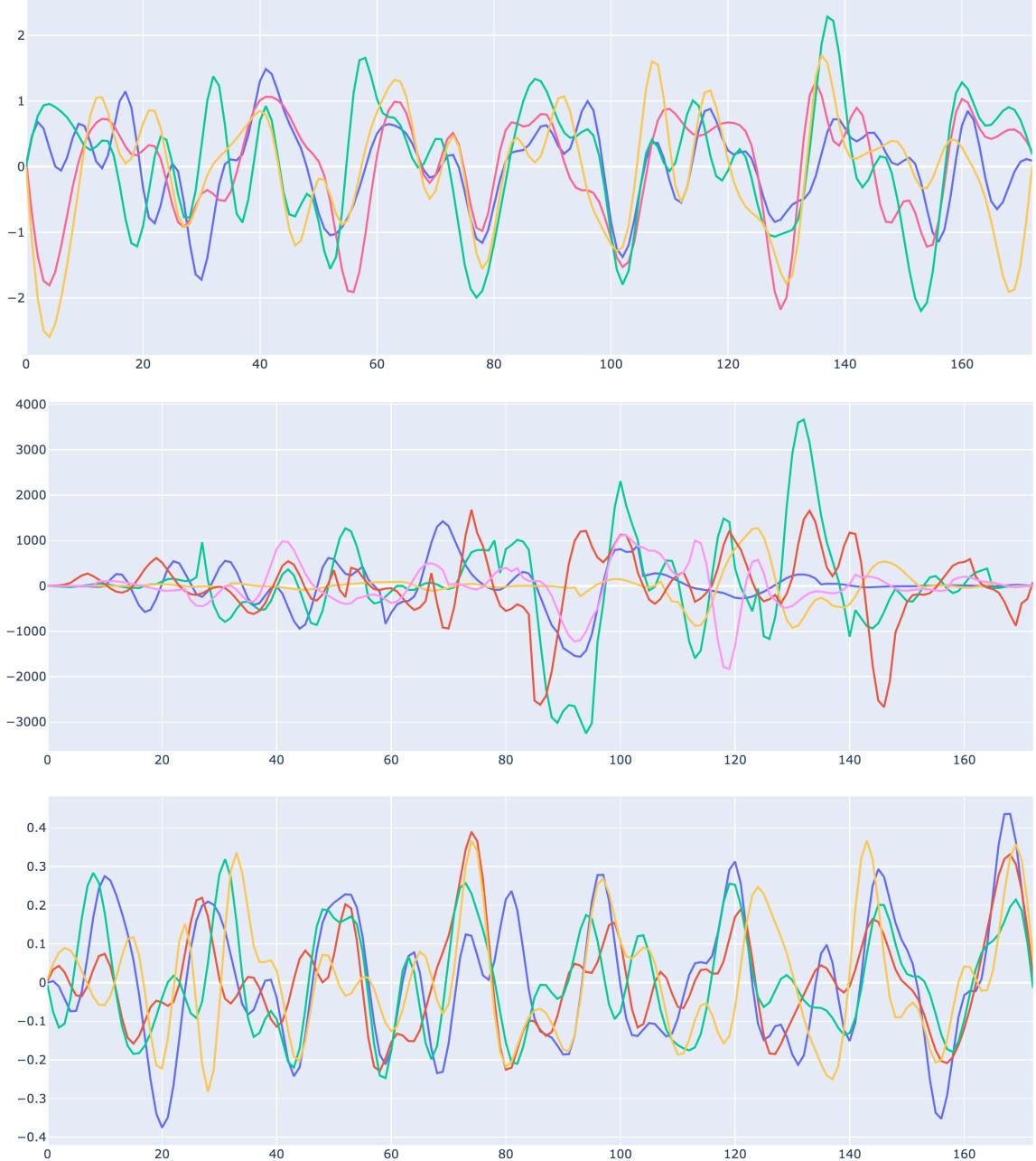


Figura 1.4: Segnali BVP ottenuti tramite la mappatura (1.2) applicando i metodi CHROM, POS e LGI (dall'alto verso il basso) ai segnali di colore RGB estratti da più patch.

1.1.1 Analisi spettrale a livello di patch

Per un segnale caratterizzato da pattern periodici come quello rPPG lo spettro di potenza è concentrato in bande di frequenza estremamente strette che indicano l'esistenza di strutture specifiche e il carattere predicibile del segnale. Le tecniche di stima spettrale basata su FFT, come la densità spettrale di potenza (PSD), possono offrire in questi casi una buona risoluzione in frequenza con ridotta varianza della stima.

Per affrontare lo scenario stocastico tipico delle misurazioni rPPG si dovrebbe tenere conto di due elementi rilevanti: una è la periodicità di base del fenomeno che si analizza, l'altro sono gli effetti stocastici introdotti dal rumore. Per queste ragioni si ritiene più affidabile l'analisi dei picchi che si ha nel dominio delle frequenze rispetto a quella più instabile nel dominio temporale. Una ulteriore ragione a supporto del dominio frequenziale è che il rumore ha quasi sempre una linea spettrale diversa da quella delle fluttuazioni cardiache, fatto che non ostacola la discriminazione dei picchi in frequenza più informativi.

Per quanto detto, l'analisi spettrale viene effettuata mediane l'uso di PSD che provvede informazioni su come si distribuisce la potenza in base alle frequenze. Qui si assume che il segnale su piccoli intervalli di analisi ($1 \div 10$ sec) sia almeno debolmente stazionario così da preservare i picchi significativi nella banda di frequenza [40, 240] BPM. Nel framework, la PSD è calcolata mediante la discrete time Fourier transform (DFT) usando poi il metodo di Welch per effettuare medie e smoothing degli spettri finestrati.

Formalmente, dato un segnale BVP (stimato da rPPG) \mathbf{b}_l di lunghezza T per ogni patch $l \in \mathcal{L}$, la sequenza $\mathbf{b}_l = (x(1), \dots, x(T))$ viene suddivisa in S segmenti (o finestre) di lunghezza M , con uno shift di K campioni tra segmenti adiacenti (che produce una sovrapposizione di $M - K$ punti). Denotando con

$$x_j(t) = x((j-1)K + t), \quad t = 1, \dots, M, \quad j = 1, \dots, S$$

dove $(j-1)K$ è l'elemento di partenza del j -mo segmento, come proposto dal metodo di Welch il j -mo segmento ha un corrispondente periodogramma data da

$$\mathcal{S}_k^l(\omega) = \frac{1}{M\Phi} \left| \sum_{t=1}^M w(t)x_j(t)e^{-j\omega t} \right|^2.$$

dove $w(t)$ è una finestra temporale di smoothing (tipicamente di Hamming o Hanning) e $\Phi = \sum_t w(t)^2$ denota la potenza della finestra. Se si assume una sovrapposizione al 50% allora $K = M/2$, da cui $S \approx 2T/M$

La stima di Welch della PSD è determinata dalla media dei periodogrammi delle

singole finestre sovrapposte:

$$\mathcal{S}^l(\omega) = \frac{1}{S} \sum_{j=1}^S \mathcal{S}_k^l(\omega). \quad (1.3)$$

Il metodo di Welch può essere efficientemente computato via FFT, ed è uno dei metodi più frequentemente usati per la stima dello spettro di potenza.

Per trovare la stima del valore BPM dato dal segnale rPPG \mathbf{b}_l della patch l , si procede trovando il massimo picco dello spettro, cioè della sua PSD i cui pattern tipici mostrano in genere un lobo univoco centrato sulla frequenza correlata con la pulsazione.

Denotando con $\tau = 1/\text{fps}$ il tempo intercorso tra due frame e assumendo che una patch sia analizzata per T frame a partire da un tempo t_0 , i tempi reali di campionamento corrispondono alla sequenza $t_0 + n\tau$, con $n = 0, 1, \dots, T - 1$, per una finestra temporale pari a $T\tau$ secondi. La frequenza $f = \omega/2\pi$ espressa in Hz ha un range che va da $(-1/2\tau + 1/Q\tau)$ a $1/2\tau$ Hz, con una risoluzione di $\nu = 1/Q\tau$ Hz se si calcola la FFT nella (1.3) con Q campioni. Il picco, essendo in genere univoco, è quindi facilmente ottenuto nell'insieme di frequenze $\Omega = \{-1/2\tau + k\nu : k = 1, \dots, Q\}$ come

$$f^l = \arg \max_{f_k \in \Omega} \mathcal{S}(f_k). \quad (1.4)$$

Si noti che la frequenza finale deve essere espressa come $f_{\text{BPM}}^l = 60 \times f^l$, cioè con la risoluzione dei BPM. Il parametro Q è sicuramente dirimente nella stima della PSD e di conseguenza nel determinare una buona stima del suo massimo, e poiché in genere il tasso dei frame è piuttosto basso (normalmente $\text{fps} = 25$ oppure $\text{fps} = 30$ per video standard) è utile definire una risoluzione frequenziale più elevata fissando per esempio $Q = 2048$, che è un ragionevole compromesso per segmentazioni video di meno di 10 secondi.

1.2 Stima basate su statistiche e clustering

Sfruttando al ridondanza spaziale di una camera a colori, molti algoritmi fondamentali per rPPG combinano i segnali RGB catturati dalla camera in una pulsazione mediando sull'intensità dei colori della pelle. Questo approccio è chiamato olistico perché prende in considerazione i pixel dell'intera faccia o, in alternativa, un piccolo gruppo di regioni di interesse (ROI) ampie e predeterminate normalmente localizzate aree frontali o centrali attorno a guance e naso. Come noto questo approccio pone seri problemi in termini di robustezza delle procedure rPPG dovuto a repentini

cambi di intensità luminosa dei pixel causate principalmente da espressioni facciali, illuminazione irregolare del volto, nonché variazioni indotte dal movimento facciale.

Lo scopo di questo lavoro di tesi è quello di porre - anche solo parzialmente - un rimedio a questi problemi di stima attraverso una nuova strategia di elaborazione video che sfrutta una caratteristica chiave dei frame, cioè l'evidente similarità spazio-temporale che esiste nelle sequenze video. Questa proprietà dice che le immagini naturali hanno "componenti" locali che tendono a ripetersi più volte all'interno dell'immagine stessa, esibendo una sostanziale ridondanza interna ai dati (per esempio la ricorrenza di piccole patch), dando così luogo a potenti statistiche interne ottenute dall'immagine in esame (25). Si assume ad esempio che un'immagine sia suddivisa in patch, eventualmente sovrapposte, di piccola taglia (per es. 10×10). La ricorrenza suggerisce che è molto probabile che molte di esse abbiano diversi gemelli simili sparso qui e là in distinte locazioni dell'immagine. Per questa ragione la proprietà è stata sfruttata intensivamente da algoritmi classici di elaborazione delle immagini nell'affrontare i più diversi problemi, che vanno dal denoising, alla super-risoluzione fino alla sintesi testuale (25). Un'idea comune di queste procedure è suddividere l'immagine in piccole patch sovrapposte per poi ricombinarne i risultati in modi opportuni in accordo alla loro similarità per avere miglior comprensione del fenomeno analizzato e in generale incrementare le capacità predittive.

1.2.1 Mediana delle PSD

In questa sezione si mostra come la mediana possa essere impiegata per la stima del BPM relativo a una finestra temporale a partire dalle singole stime operate sulle PSD $\mathcal{S}^l(\omega)$ computate dai segnali BVP \mathbf{b}_l per ognuna delle patch \mathcal{P}_l , con $l \in \mathcal{L}$.

Denotiamo con f_{BPM}^l le frequenze ottenute dai picchi massimi delle PSD. Un esempio con tre esemplari di PSD è riportato in Figura 1.5. Come si vede in figura e come spesso capita, i valori stimati non sempre coincidono e in alcuni casi gli outlier sono molti, per questo motivo la mediana è più affidabile della media.

Se denotiamo con $H = (f_{\text{BPM}}^1, \dots, f_{\text{BPM}}^L)$ il vettore delle L stime BPM di ogni patch e con \hat{H} il corrispondente vettore ordinato avremo:

$$\hat{h} = \text{median}(\hat{H}) = \begin{cases} \hat{H}\left[\frac{L-1}{2}\right] & \text{Se } L \text{ è dispari} \\ \frac{\hat{H}\left[\frac{L}{2}-1\right] + \hat{H}\left[\frac{L}{2}\right]}{2} & \text{Se } L \text{ è pari.} \end{cases} \quad (1.5)$$

Notiamo che se il numero di patch è $L = 1$ (cioè è stato selezionato un singolo patch o è stato scelto l'approccio olistico), allora:

$$\hat{h} = \hat{H}[0]. \quad (1.6)$$

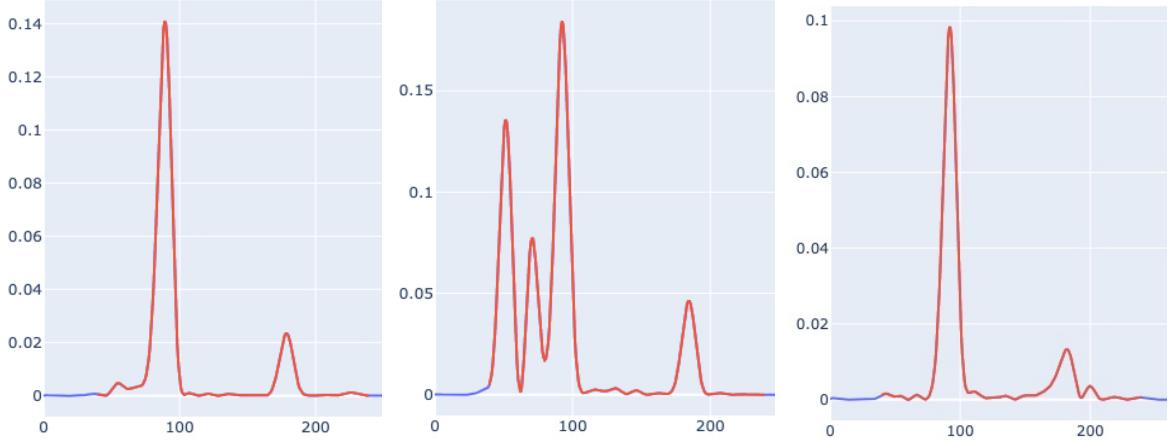


Figura 1.5: PSDs.

Inoltre, quando vengono selezionate più patch, è possibile calcolare una misura di variabilità delle previsioni per quantificare l'incertezza della stima. In particolare, py-VHR calcola la *Median absolute deviation (MAD)* come misura robusta della dispersione statistica. Il *MAD* è definito come:

$$MAD = \text{median}(|\hat{H} - \hat{h}|). \quad (1.7)$$

Chiaramente, il *MAD* tende a 0 quando $L = 1$.

Nella Figura 1.6 sono riportate rispettivamente le mediane e il *ground truth* di un esperimento. In particolare si possono notare anche i valori delle stime dei BPM per ogni istante di tempo rappresentate dalla collezione di punti colorati distribuiti verticalmente per ogni finestra temporale.

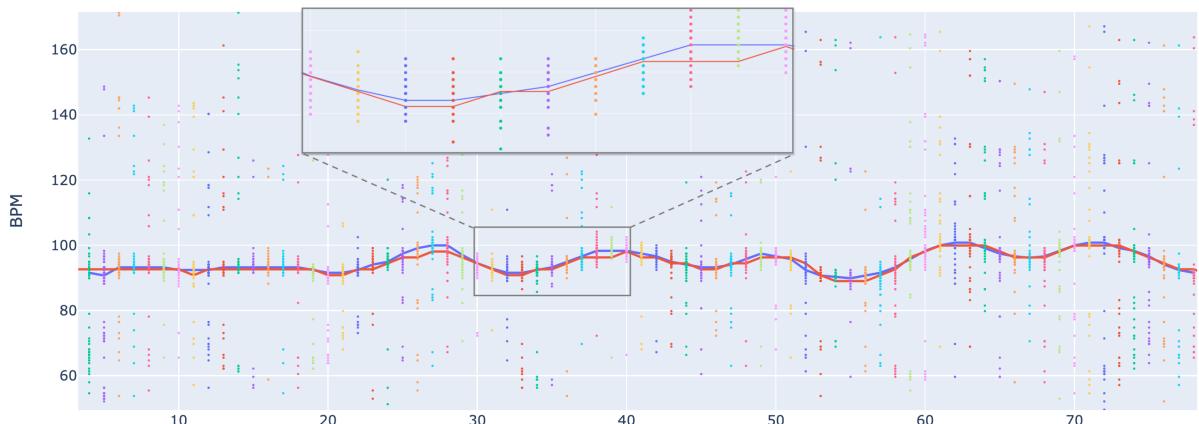


Figura 1.6: Mediane computate su molteplici stime BPM da PSD computate su patch.

1.2.2 Clustering delle PSD

In questa sezione si introduce un nuovo algoritmo di clustering per le PSD $\mathcal{S}^l(\omega)$ computate dai segnali BVP \mathbf{b}_l per ognuna delle patch \mathcal{P}_l , con $l \in \mathcal{L}$. L'idea è quella di separare le PSD più omogenee tra loro da quelle dissimili tra loro in due cluster distinti e poi effettuare la stima finale dei BPM dal cluster che maggiormente concentra attorno a una stessa frequenza i picchi delle PSD che vi appartengono. A questo fine viene definito un nuovo algoritmo di clustering che a differenza di altri, come k -means, riesce ad ottenere i due cluster con la proprietà dichiarata prima, di rispettare cioè l'“asimmetria” nella scelta della bipartizione: il primo cluster A raggruppa le PSD omogenee tra loro (cioè aventi picco concentrato attorno a una stessa frequenza) e un cluster B contenente le PSD non solo disomogenee tra loro ma anche dissimili con quelle di A . Ad esempio, nella Figura 1.5 si può notare che le due PSD esterne sono molto più simili tra loro di quanto ciascuna delle due non lo sia con quella centrale.

Formalmente, assumendo che ogni PSD sia un vettore $\Phi_l = \mathcal{S}^l(\omega) \in \mathbb{R}^Q$, si definisce una matrice delle distanze $W = (w_{ij})$, dove $w_{ij} = \text{similarity}(\Phi_i, \Phi_j)$ fornisce un grado di similitudine (o dissimilitudine) tra i due vettori. Se in particolare si considera la distanza coseno (il complemento del coseno dell'angolo compreso), che negli esperimenti si è dimostrata la più efficace, allora il grado di (dis)similitudine è compreso tra 0 e 2.

Per ottenere i cluster \mathcal{A} e \mathcal{B} con le proprietà dette sopra e quindi una bipartizione dell'insieme $\Theta_{\mathcal{L}} = \{\Phi_l \in \mathbb{R}^Q : l \in \mathcal{L}\}$, per ragioni di ordine pratico che diverranno chiare di seguito, associamo a ogni vettore Φ_l un vettore sul cerchio unitario $\phi_l \in \mathbf{S}_2$. Il principale vantaggio di questa posizione è rendere i funzionali da ottimizzare più semplici rispetto a quelli dipendenti da vettori Φ_l che giacciono in uno spazio di dimensioni elevate. Un secondo non trascurabile vantaggio deriva dal fatto che, poiché il prodotto interno tra due vettori $\phi_i = (\cos \theta_i, \sin \theta_i)$ e $\phi_j = (\cos \theta_j, \sin \theta_j)$ verifica $\phi_i \cdot \phi_j = \cos(\theta_i - \theta_j)$, la similarità relativa a vettori in \mathbf{S}_2 dipende dalle variabili scalari θ_i e θ_j e può essere scritto sotto forma di funzione:

$$\zeta_{i,j}(\phi_i, \phi_j) = \frac{1 - \phi_i \cdot \phi_j}{2} = \zeta_{i,j}(\theta_i, \theta_j) = \frac{1 - \cos(\theta_i - \theta_j)}{2} \quad (1.8)$$

La (1.8) assume valori nell'intervallo reale unitario, dove 0 indica massima similarità e 1 massima dissimilarità, i valori intermedi forniscono un grado di similarità proporzionale alla misura dell'angolo compreso tra i due vettori ϕ_i e ϕ_j .

Combinando i pesi w_{ij} tra i vettori originali in $\Theta_{\mathcal{L}}$ e le funzioni (1.8) per ogni coppia di indici distinti $\{i, j\}$, il problema di trovare una bipartizione di $\Theta_{\mathcal{L}}$, può essere formulato come un problema di ottimizzazione sull'insieme $[0, 2\pi]^L$ con funzione costo

da massimizzare pari a

$$\begin{aligned}\rho(\theta_1, \dots, \theta_L) &= \sum_{i < j} w_{ij} \zeta_{i,j}(\theta_i, \theta_j) \\ &= \sum_{i < j} w_{ij} \frac{1 - \cos(\theta_i - \theta_j)}{2} \\ &= cost - \frac{1}{2} \sum_{i < j} w_{ij} \cos(\theta_i - \theta_j).\end{aligned}$$

Nella precedente equazione, costante a parte, il termine da massimizzare appare con il segno negativo, questo equivale a minimizzare la funzione (che senza perdita di generalità indichiamo con lo stesso funzionale) con segno cambiato:

$$\rho(\theta_1, \dots, \theta_L) = \frac{1}{2} \sum_{i < j} w_{ij} \cos(\theta_i - \theta_j). \quad (1.9)$$

Poiché risulta in generale difficile ottimizzare la funzione (1.9) perché non convessa, un possibile approccio efficiente è dato dalla tecnica di ricerca locale. In particolare è possibile mostrare che a partire da una qualunque condizione iniziale (per esempio valori scelti a caso) l'algoritmo di ricerca locale, chiamato Circle Clustering, il cui pseudocodice è riportato di seguito, è in grado di trovare un insieme di angoli $\{\theta_l^* \in [0, 2\pi] : l \in \mathcal{L}\}$ o equivalentemente di vettori $\{\phi_l^* \in \mathbf{S}_2 : l \in \mathcal{L}\}$ che minimizzano localmente la (1.9). Più precisamente, si può dimostrare che per ogni reale $\varepsilon > 0$ l'algoritmo di ricerca locale Circle Clustering migliorando localmente la funzione ρ ad ogni passaggio del ciclo while al suo interno (Step 1.), ottiene in tempo polinomiale una soluzione tale che:

$$|\rho(\theta_1, \dots, \theta_k, \dots, \theta_n) - \rho(\theta_1, \dots, \alpha_k, \dots, \theta_n)| < \varepsilon.$$

Per avere la partizione finale è sufficiente trovare l'iperpiano separatore che minimizza la funzione riportata allo Step 2 dell'algoritmo. In sostanza è sufficiente valutare come separatore il piano ortogonale $(-\sin \theta_l, \cos \theta_l)$ di ogni vettore $\phi_l = (\cos \theta_l, \sin \theta_l)$ e tenere quello che dà l'ottimo.

Le Figure 1.7 e 1.8 mostrano gli effetti dell'applicazione dell'algoritmo di clustering sopra descritto. I plot nei riquadri a sinistra mostrano la PSD media dei due cluster (partizione \mathcal{A} in alto e partizione \mathcal{B} in basso), rispettivamente in blu e in rosso. Come si nota dai pattern di entrambe le figure, le PSD che contribuiscono a formare la curva media blu del cluster \mathcal{A} sono tra loro molto disomogenee e infatti mostrano uno spettro di potenza medio con un elevato "spread" che non focalizza su nessuna frequenza in particolare. Viceversa, nel caso del cluster \mathcal{B} le PSD sono tra loro molto simili e

Algorithm 1: Circle Clustering

Data: La matrice dei pesi W , un reale $\varepsilon > 0$
Result: Bipartizione \mathcal{A} e \mathcal{B} di $\Theta_{\mathcal{L}}$

```

for  $k := 1$  to  $L$  do
|    $\theta_k \leftarrow$  un numero casuale in  $[0, 2\pi]$ ;
end
for  $k := 1$  to  $L$  do
|    $A_k \leftarrow \sum_j w_{kj} \cos \theta_j$ ;
|    $B_k \leftarrow \sum_j w_{kj} \sin \theta_j$ ;
end
/* STEP 1.
 $CONT \leftarrow True$ ;
while  $CONT$  do
|    $CONT \leftarrow False$ ;
|   for  $k := 1$  to  $L$  do
|   |    $\alpha_k \leftarrow \theta_k$ ;
|   |    $\theta_k \leftarrow \arg \min_{\gamma \in [0, 2\pi]} \{A_k \cos \gamma + B_k \sin \gamma\}$ ;
|   |   if  $|\alpha_k - \theta_k| > \varepsilon$  then
|   |   |    $CONT \leftarrow True$ ;
|   |   |   for  $j := 1$  to  $L$  do
|   |   |   |    $A_j \leftarrow A_j + w_{kj} (\cos \theta_k - \cos \alpha_k)$ ;
|   |   |   |    $B_j \leftarrow B_j + w_{kj} (\sin \theta_k - \sin \alpha_k)$ ;
|   |   |   end
|   |   else
|   |   |    $\theta_k \leftarrow \alpha_k$ ;
|   |   end
|   end
/* STEP 2.
 $\kappa \leftarrow \arg \min_{k \in \mathcal{L}} \left\{ \sum_{i < j} w_{ij} \text{signum}(\sin(\theta_j - \theta_k) \sin(\theta_i - \theta_k)) \right\}$ ;
 $\mathcal{A} \leftarrow \{k \mid \text{signum}(\sin(\theta_k - \theta_\kappa)) = 1\}$ ;
 $\mathcal{B} \leftarrow \{k \mid \text{signum}(\sin(\theta_k - \theta_\kappa)) = 0\}$ ;
end

```

infatti la loro media consente di individuare con molta chiarezza il picco (unico) che corrella fortemente con il ground truth (linea nera verticale nelle figure di destra).

Per effettuare una scelta finale tra i due cluster e stabilire definitivamente chi darà la stima del BPM finale per una data finestra temporale caratterizzata dalle patch dell'insieme \mathcal{L} , si procede con un fitting Gaussiano come mostrato nei plot dei riquadri a destra delle due figure. Le misure di bontà del fitting sono riportate al di sotto delle figure dove si riporta alcune misure della stima del fitting: sigma, chi square e criteri aic e bic. Quindi, in definitiva, sulla base di un meccanismo di voting si decide chi dei due fitting delle PSD medie è il migliore e il BPM finale è quello dato dal massimo picco della PSD prescelta.

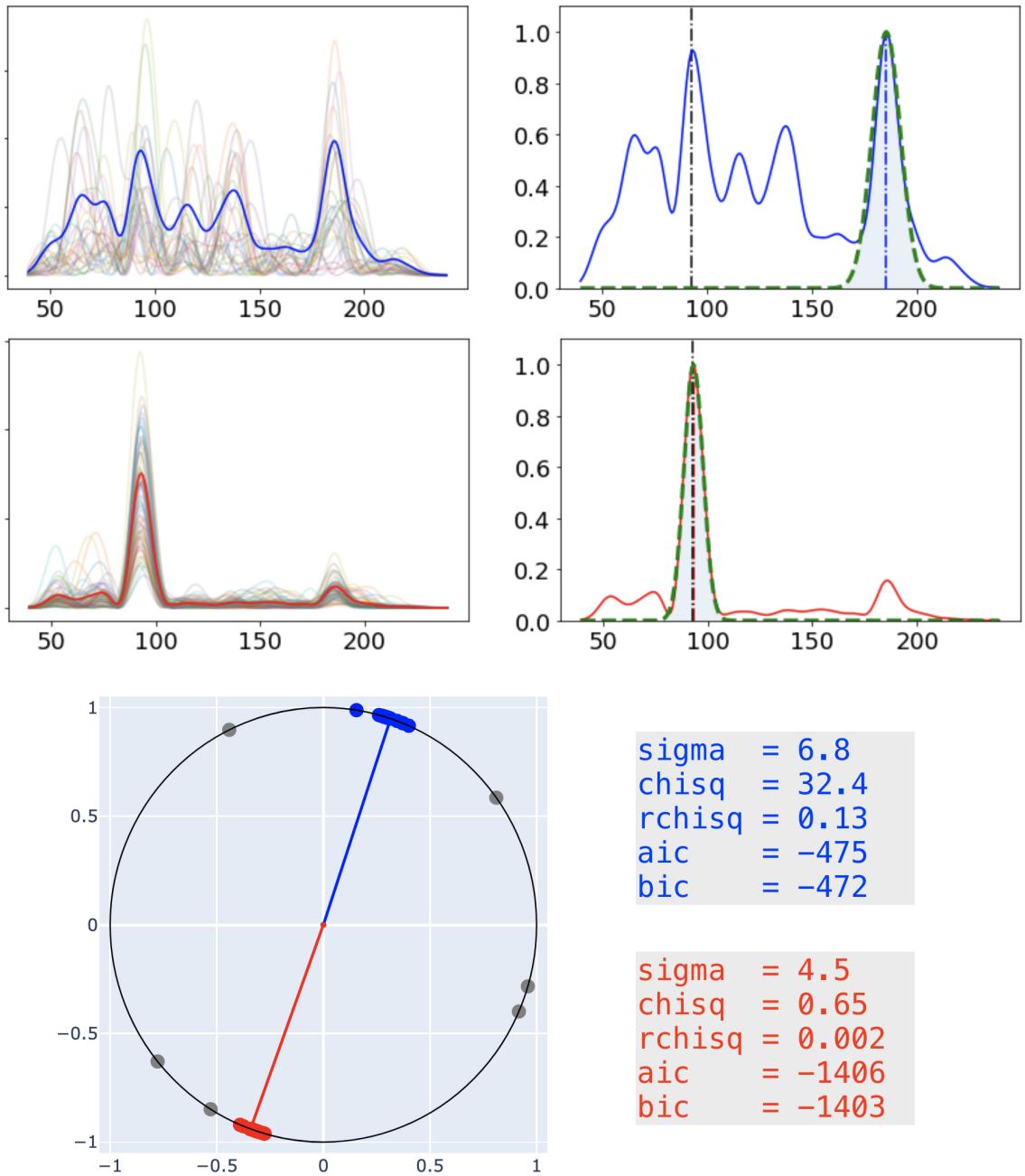


Figura 1.7: Circle Clustering. In alto l'analisi delle PSD; in basso il clustering su \mathbb{R}^2 , e le caratteristiche delle Gaussiane trovate in corrispondenza della frequenza con potenza spettrale massima.

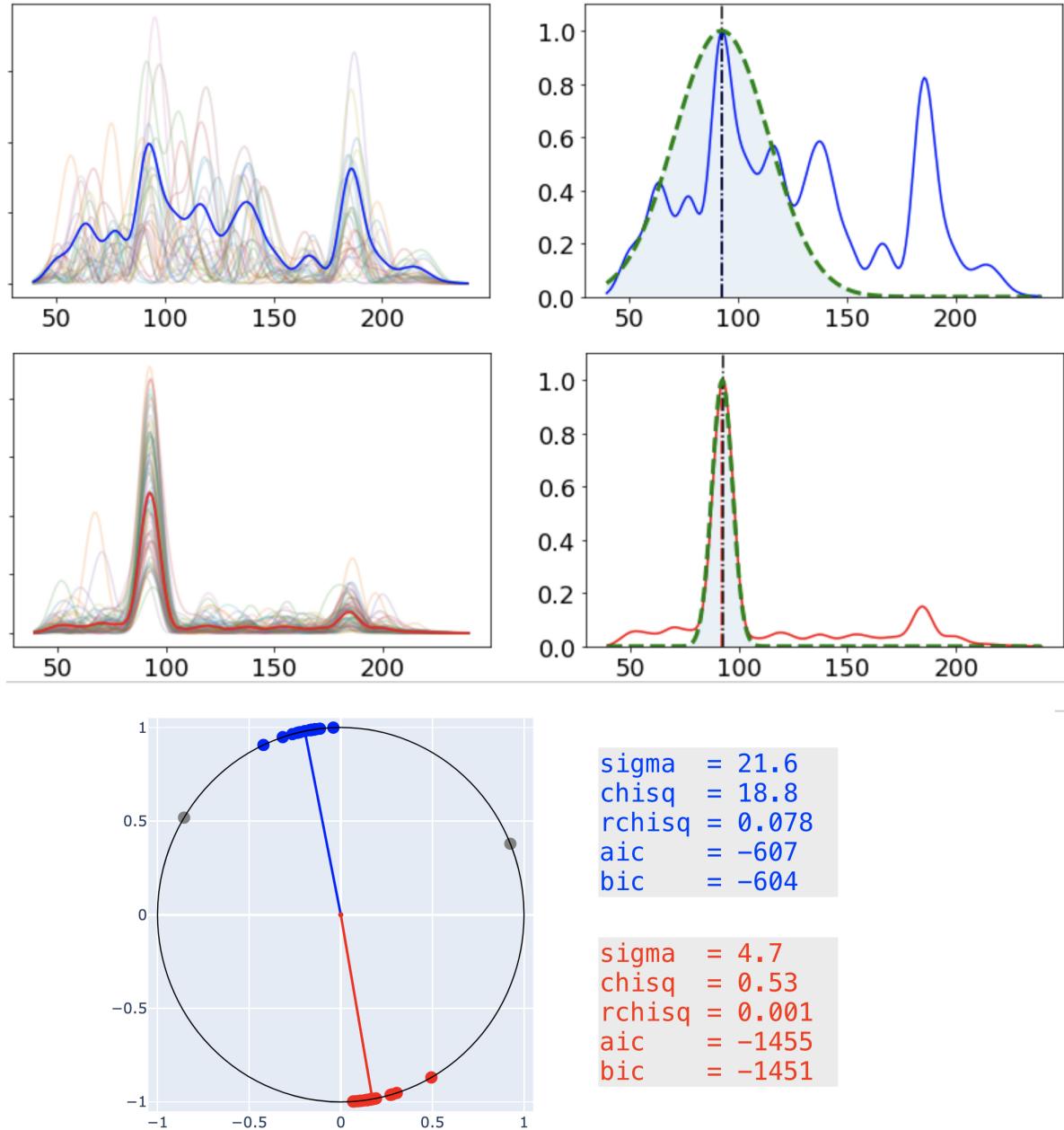


Figura 1.8: Circle Clustering. In alto l'analisi delle PSD; in basso il clustering su \mathbb{R}^2 , e le caratteristiche delle Gaussiane trovate in corrispondenza della frequenza con potenza spettrale massima.

Capitolo 2

Il Framework

2.1 Introduzione

pyVHR (Python framework for Virtual Heart Rate) è un framework per studiare, sviluppare e confrontare i metodi di stima del battito cardiaco basati su remote Photoplethysmography (rPPG).

La logica metodologica alla base del framework è che al fine di studiare, sviluppare e confrontare nuovi metodi rPPG in modo riproducibile, dovrebbero essere soddisfatte le seguenti condizioni: i) una pipeline strutturata per monitorare l'input, l'output e il controllo dei parametri degli algoritmi rPPG; ii) la disponibilità e l'utilizzo di più dataset; iii) una solida valutazione statistica delle prestazioni dei metodi. Le sue caratteristiche principali sono le seguenti:

- **Orientato all'analisi.** *pyVHR* permette di effettuare esperimenti su un numero arbitrario di metodi rPPG attraverso una pipeline end-to-end sistemica, che consente di impostare facilmente parametri e meta-parametri.
- **Openess.** *pyVHR* permette all'utente di utilizzare metodi rPPG custom e di aggiungere facilmente nuovi dataset.
- **Valutazione robusta.** I risultati sono organizzati in formati strutturati per analisi statistiche approfondite. Il confronto delle prestazioni viene effettuato sulla base di robusti test statistici non parametrici.

Il pacchetto *pyVHR* comprende vari notebooks Python (.ipynb) utili per imparare ad utilizzare il framework, e molto efficaci per effettuare analisi statistiche e sviluppare nuovi metodi rPPG.

2.1.1 Installazione

Il modo più semplice per installare pyVHR si basa sull’ambiente Miniconda, ovvero una versione minimale di Anaconda Python. Una volta installato, dobbiamo creare un ambiente per poter utilizzare pyVHR; nel GitHub del framework sono presenti degli ambienti già pronti: il primo è per CPU+GPU ed utilizza python 3.8 e CudaToolkit 10.2:

```
1 $ conda env create --file https://github.com/phuselab/pyVHR/blob/  
    pyVHR_CPU/pyVHR_CPU_env.yml
```

il secondo è per CPU ed utilizza python 3.8:

```
1 $ conda env create --file https://github.com/phuselab/pyVHR/blob/main/  
    pyVHR_env.yml
```

Una volta completata l’installazione dell’ambiente sarà necessario attivarlo e installare pyVHR:

```
1 pip install pyvhr
```

oppure nel caso CPU-only:

```
1 $ pip install pyvhr -cpu
```

Il codice sorgente di pyVHR è contenuto su GitHub (<https://github.com/phuselab/pyVHR>) ed è distribuito secondo la licenza GPL-3.0. Nella homepage GitHub del framework è presente anche la documentazione ufficiale realizzata con Sphinx.

2.2 La Pipeline

La classe `Pipeline()` implementa la sequenza di passaggi normalmente richiesti dalla stragrande maggioranza dei metodi rPPG proposti in letteratura per stimare il BPM di un soggetto dato un video che mostra il suo volto. Per utilizzare la classe `Pipeline()` sarà necessario scrivere un paio di righe di codice Python:

```
1 from pyVHR.analysis.pipeline import Pipeline  
2  
3 pipe = Pipeline()  
4 time, BPM, uncertainty = pipe.run_on_video('/path/to/vid.avi')
```

La chiamata al metodo `run_on_video()` della classe `Pipeline()` avvia l’analisi del video fornito come argomento e produce come output il campionamento temporale `time` della stima BPM e relativa stima di `uncertainty`. La figura 2.1 mostra il BPM previsto su Subject1 dell’UBFC¹ dataset (26) (traiettoria blu). La traiettoria del BPM ground truth (registrata da un sensore PPG) è riportata in rosso.

¹Disponibile su: <https://sites.google.com/view/ybenezeth/ubfcrppg>.

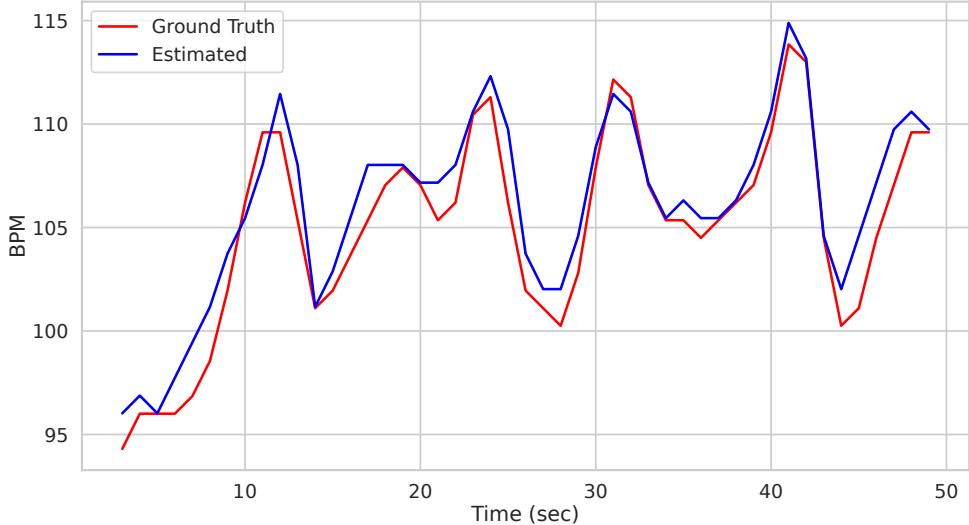


Figura 2.1: Esempio di stima BPM Stima BPM sul Subject1 del dataset UBFC.

Questo esempio dimostra la facilità di utilizzo del pacchetto nascondendo l'intera pipeline dietro una singola chiamata di funzione.

Questo approccio *black-box* può essere molto comodo, ma allo stesso tempo molto limitante. Alcuni utenti potrebbero essere interessati a utilizzare tutti i diversi moduli che compongono la pipeline pyVHR. Questi sono mostrati in Figura 2.2 (regione azzurra "Traditional Methods") e possono essere riassunti come segue:

- *Skin extraction*: L'obiettivo di questo primo passo è eseguire una segmentazione della pelle del viso; le aree individuate vengono successivamente raccolte in una singola *patch* (approccio olistico) o in un gruppo di *patches* "sparse" che coprono l'intera faccia (approccio *patches*).
- *RGB signal processing*: le *patches*, una o più, sono tracciate in modo coerente e sono utilizzate per calcolare le intensità di colore medie lungo finestre temporali sovrapposte, fornendo così più segnali RGB variabili nel tempo per ciascuna finestra temporale.
- *Pre-filtering*: facoltativamente, le tracce RGB grezze vengono pre-filtrate tramite filtraggio canonico, normalizzazione o de-trend; i segnali ottenuti saranno l'input di qualsiasi successivo metodo rPPG.
- *BVP extraction*: il metodo, o i metodi rPPG a disposizione vengono applicati ai segnali RGB, producendo così una raccolta di segnali *Blood Volume Pulse* (stime BVP), uno per ogni patch.
- *Post-filtering*: i segnali BVP possono essere filtrati utilizzando gli stessi filtri della fase *Pre-filtering* e, in più, il filtro *band-pass* per rimuovere componenti di frequenza fuori dalla banda di interesse.

- *BPM estimation*: Una stima del BPM è infine ottenuta attraverso semplici statistiche basate sui valori massimi delle densità spettrali di potenza del BVP. Esistono anche tecniche più avanzate che tratteremo in seguito.

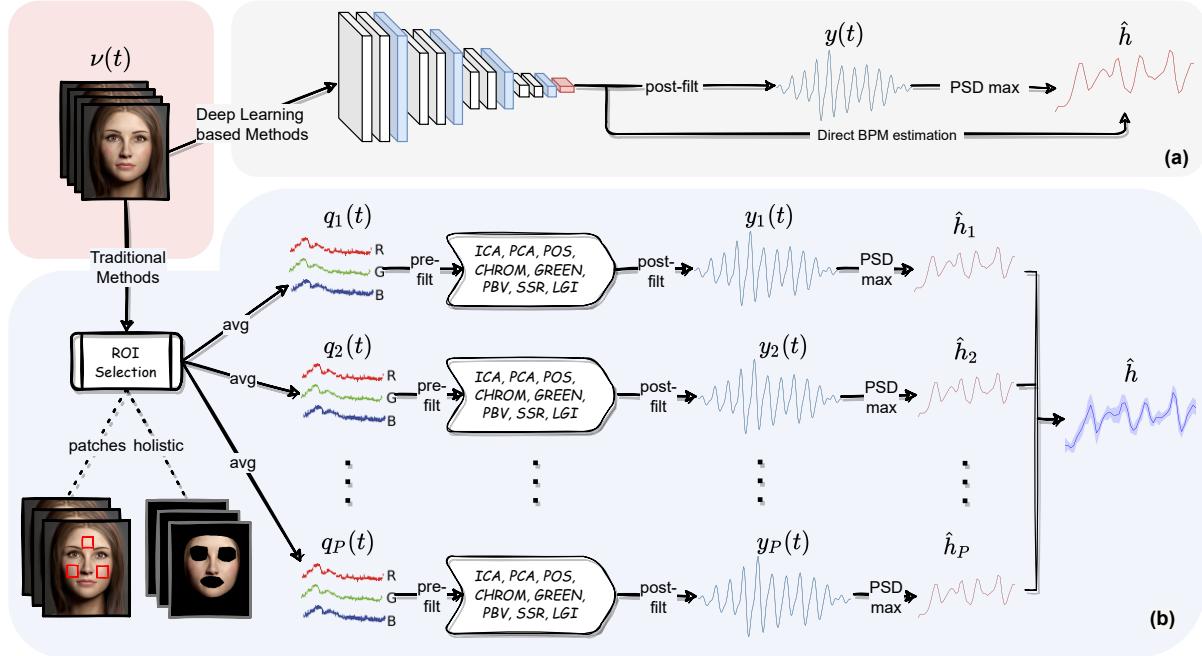


Figura 2.2: **La pipeline pyVHR Pipeline.** La pipeline *multi-stage* del framework pyVHR. Partendo da un video contenente un volto, lo step di selezione della ROI produce una raccolta di patches P_l ($l \in \mathcal{L} = [1 \dots L]$), che può essere anche solo una (*holistic*). I segnali RGB vengono calcolati tramite la media delle intensità di colore dei pixel di ciascuna *patch*. Il segnale RGB multicanale viene eventualmente pre-filtrato e viene passato in input ad un metodo rPPG che restituisce un segnale BVP. Successivamente, questo segnale viene filtrato e finestrato nel tempo; infine, il k -esimo BPM istantaneo è calcolato come il valore massimo della PSD della k -esima finestra (o con altri metodi più elaborati). Alla fine i L segnali BPM vengono combinati in uno solo utilizzando la mediana. L'incertezza della stima in ogni fase temporale viene calcolata con la *Median Absolute Deviation* (MAD) delle L stime. In alternativa, partendo dallo stesso video si può eseguire la pipeline per metodi Deep. Il segnale BVP restituito dai metodi Deep può essere filtrato e in seguito trasformato in un segnale BVP utilizzando gli stessi *step* della pipeline *tradizionale*.

La pipeline è stata progettata in modo da avere moduli indipendenti che descrivono un processo sequenziale dove ogni output di uno *step* può essere utilizzato come input di una o più fasi successive. Ciò risponde alla necessità di avere un framework flessibile, estensibile, mantenibile e migliorabile nel tempo con tecniche innovative o alternative.

2.2.1 Skin extraction

La fase di estrazione della pelle implementata in pyVHR consiste nella segmentazione della regione di pelle del viso del soggetto. Tipicamente, le regioni corrispondenti agli occhi e alla bocca vengono scartate dall'analisi. Esistono due strumenti per effettuare la *skin extraction*:

1. l'estrattore *Convex-Hull*,
2. l'estrattore *Face parsing*.

L'estrattore *Convex-hull* è basato sulla *solution 'Face Mesh'* del framework MediaPipe (27), che permette di rilevare e tracciare con alta affidabilità 468 punti facciali (vedi Figura 2.4) in tempo reale utilizzando solo la CPU. L'estrattore calcola l'inviluppo convesso dei punti individuati (in situazioni di occlusione del volto anche meno di 468) ottenendo una regione bidimensionale che li contiene. Vengono calcolati anche gli inviluppi convessi dei punti che descrivono gli occhi e la bocca; questi sono sottratti alla regione che descrive l'intero volto, in modo da ottenere una maschera per isolare i pixel associati alla pelle. L'immagine a sinistra della Figura 2.3 mostra un esempio di estrazione *Convex-hull* su un soggetto del dataset LGI-PPGI² (14).

L'estrattore *Face parsing*, invece, effettua una segmentazione semantica del volto del soggetto utilizzando la rete neurale BiSeNet (6). Ogni pixel analizzato è identificato semanticamente da una label (es. capelli, bocca, occhi, naso, ecc...), come mostrato nella Figura 2.5; l'estrattore conserverà solo i pixel corrispondenti a regioni cutanee. Questa rete permette di analizzare video in real-time anche se eseguita su CPU (non a risoluzioni maggiore dell'HD). Un esempio di estrazione è mostrato nell'immagine a destra della Figura 2.3.



Figura 2.3: Esempi di skin extraction. Output dell'approccio Convex-hull (sinistra) e Face Parsing di BiSeNet (destra) su un soggetto del dataset LGI-PPGI (14).

²Disponibile per il download su <https://github.com/partofthestars/LGI-PPGI-DB>

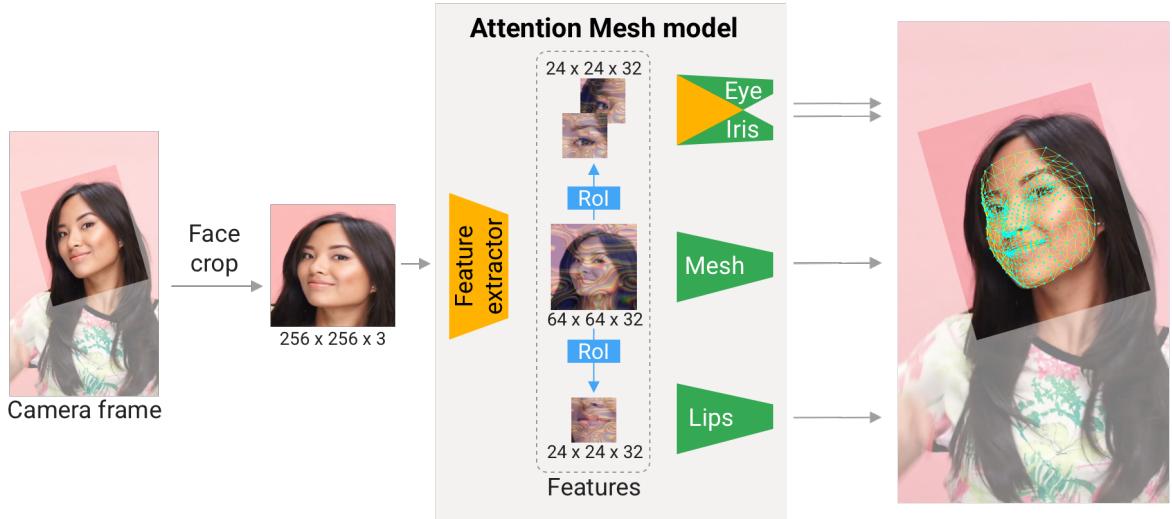


Figura 2.4: Architettura del Modello Face Mesh di MediaPipe.

La fase di skin extraction è gestita dalla classe `SignalProcessing` di `pyVHR`. Le seguenti righe di codice mostrano come assegnare uno skin extractor a una istanza di `SignalProcessing`:

```

1 from pyVHR.extraction.sig_processing import SignalProcessing
2
3 sig_processing = SignalProcessing()
4 if skin_method == 'convexhull':
5     sig_processing.set_skin_extractor(SkinExtractionConvexHull(
6         target_device))
7 elif skin_method == 'faceparsing':
8     sig_processing.set_skin_extractor(SkinExtractionFaceParsing(
9         target_device))

```

ROI selection

Una volta estratta la pelle del soggetto l’utente può scegliere di considerare regioni specifiche di interesse (ROI). Il metodo più semplice prevede di utilizzare una singola regione equivalente a tutta la pelle estratta (*olistico*), mentre quello più avanzato considera varie regioni indipendenti (*patches*). Analizziamo con maggiore dettaglio l’approccio *Patches*.

Durante la fase di *skin extraction*, a prescindere dall’estrattore scelto, `pyVHR` tiene traccia del volto del soggetto utilizzando Face Mesh di MediaPipe; questo significa che sono noti ad ogni frame 468 punti facciali (o meno in caso di occlusione). Definiamo una *patch* come una regione 2D di forma quadrata o rettangolare centrata su



Figura 2.5: Segmentazione semantica di Face Parsing BiSeNet.

uno dei 468 punti facciali individuati (detti anche *landmarks*). L'utente può scegliere le dimensioni dei lati e uno dei 468 punti su cui centrare la *patch*.

Ogni *patch* è indipendente, quindi le fasi successive della pipeline lavoreranno in parallelo su più *patches*. L'approccio olistico può essere visto come un approccio *Patches* dove esiste una sola *patch* che racchiude tutti i pixel del volto del soggetto.

Un esempio di estrazione e tracciamento dei punti di riferimento facciali è mostrato in Figura 2.6. E' importante evidenziare che una *patch* potrebbe scomparire se il soggetto non è perfettamente dentro la inquadratura o per occlusione, fornendo quindi solo un contributo parziale o nullo alle fasi successive della pipeline.

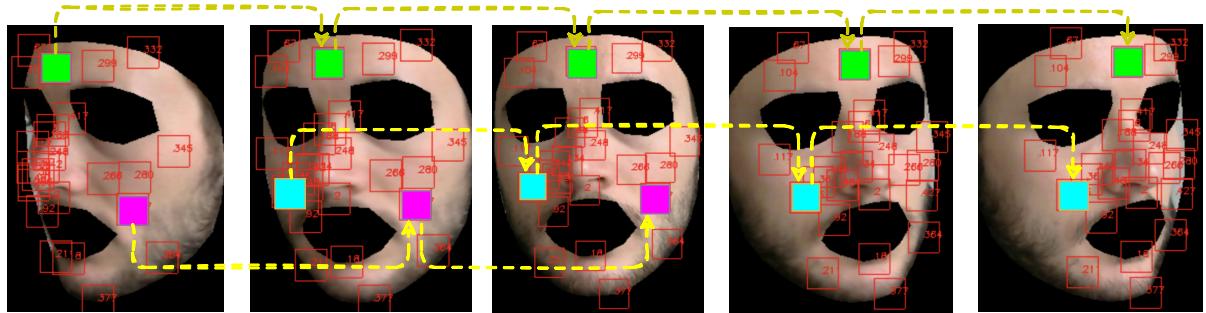


Figura 2.6: Tracciamento automatico di alcuni landmarks individuati da MediaPipe e corrispondente patch quadrata su un soggetto del dataset LGI-PPGI (14).

Vale la pena notare come l'utente possa comporre arbitrariamente il proprio set di *patch*. Nell'esempio seguente, sono stati selezionati tre *patches* corrispondenti alle aree della fronte, della guancia sinistra e destra.

```

1 from pyVHR.extraction.utils import MagicLandmarks
2
3 ldmks_list = [MagicLandmarks.cheek_left_top[16], MagicLandmarks.
    cheek_right_top[14], MagicLandmarks.forehead_center[1]]
4 # set square patches side dimension
5 sig_processing.set_square_patches_side(28.0)
6 # set patches
7 sig_processing.set_landmarks(ldmks_list)

```

L'approccio olistico non necessita di essere configurato come quello *Patches*.

Solitamente si scelgono più *patches* per controllare meglio l'elevata variabilità dei risultati e per ottenere un alto livello di confidenza, riducendo al contempo il margine di errore.

2.2.2 RGB signal processing

Per estrarre il segnale RGB $\hat{\mathbf{c}}_l(t)$ con pyVHR utilizziamo la classe *SignalProcessing*:

```

1 from pyVHR.extraction.sig_processing import SignalProcessing
2
3 sig_processing = SignalProcessing()
4 # ...
5 # steps della pipeline precedenti
6 # ...
7 # For Patches approach
8 sig = sig_processing.extract_patches(videoFileName, 'squares', 'mean')
9 # For Holistic approach
10 sig = sig_processing.extract_holistic(videoFileName)

```

E' importante sottolineare che questa fase è totalmente indipendente dalla scelta dell'estrattore facciale *Convex Hull* o *Face Parsing*.

Una volta ottenuto il segnale RGB possiamo dividerlo in finestre sovrapposte:

```

1 from pyVHR.extraction.utils import sig_windowing, get_fps
2
3 Ws = 6 # Lunghezza della finestra in secondi
4 overlap = 1 # Sovrapposizione delle finestre in secondi
5 fps = get_fps(videoFileName) # Frame rate
6
7 windowed_sig, timesES = sig_windowing(sig, Ws, overlap, fps)
8 # timesES e' la lista dei centri di ogni finestra temporale in secondi.

```

2.2.3 BVP extraction

Dato che il framework può fare affidamento su un'elaborazione olistica e *patch*, pyVHR stima il segnale BVP da una singola traccia o sfruttando più tracce. In entrambi i casi impiega un'ampia gamma di metodi rPPG allo stato dell'arte.

In particolare, le tracce RGB finestrate $\hat{\mathbf{c}}_l(t)$ ($l = 1, \dots, L$, con $l = 1$ nel caso olistico) sono date in input al metodo rPPG in uso, che restituisce i segnali BVP $\mathbf{b}_l(t)$ associati alla l -esima *patch* nella finestra temporale k -esima.

I molti metodi che sono stati proposti nella letteratura recente differiscono principalmente nel modo di combinare tali segnali RGB in un segnale a impulsi. Una revisione dei principi/presupposti alla base di ciascuno degli algoritmi implementati non è parte dello scopo del presente lavoro. Il lettore interessato potrebbe fare riferimento a (4, 7, 8). Di seguito la lista dei metodi implementati in pyVHR: GREEN (15), CHROM (17), ICA (18), LGI (14), PBV (28), PCA (29), POS (4), SSR (16). La Tabella 2.1 fornisce una descrizione dei concetti principali che caratterizzano questi metodi.

L'utente può definire metodi rPPG *custom* per stimare il BVP (fare riferimento alla *function signature* definita nel modulo `pyVHR.BVP.methods`).

Il segnale BVP può essere stimato in pyVHR come segue:

```
1 # Per Metodi basati su Numpy (CPU)
2 bvp = RGB_sig_to_BVP(windowed_sig, fps, device_type='cpu', method=
cpu_CHROM)
3 # Per Metodi basati su Cupy (CUDA-GPU)
4 bvp = RGB_sig_to_BVP(windowed_sig, fps, device_type='cuda', method=
cupy_CHROM)
5 # Per Metodi basati su pyTorch
6 bvp = RGB_sig_to_BVP(windowed_sig, fps, device_type='torch', method=
=torch_CHROM)
```

La Figura 2.7 raffigura i segnali BVP stimati da quattro metodi rPPG differenti implementati in pyVHR (POS, GREEN, CHROM, PCA) utilizzando la stessa finestra temporale di un segnale RGB *olistico*.

2.2.4 Pre e Post-filtering

pyVHR offre semplici API per applicare filtri sulle tracce RGB $\hat{\mathbf{c}}_l(t)$ (pre-filtraggio) o sulla stima del segnale BVP $\mathbf{b}_l(t)$ (post-filtraggio). I filtri implementati sono:

- *Band Pass (BP) filter*: filtra il segnale di ingresso utilizzando un filtro bandpass Butterworth dell' N -esimo ordine con un certo intervallo di frequenza di banda passante.
- *Detrending*: sottrae offset o trend lineari dai dati di input nel dominio del tempo.

Tabella 2.1: Algoritmi rPPG implementati in pyVHR.

Metodo	Descrizione
ICA	Decomposizione basata sulla <i>blind source separation</i> (BSS) per ottenere i componenti indipendenti dalla mistura temporale RGB.
PCA	Tecnica Statistica per estrarre un sottoinsieme dei componenti non correlati della traccia temporale RGB.
GREEN	Estrazione del canale RGB verde, in quanto contiene meno artefatti di quello rosso e blu.
CHROM	Metodo basato sulla <i>Chrominance</i> ; viene eseguita la normalizzazione dei canali colore per attenuare le distorsioni del segnale.
POS	Utilizza un piano ortogonale alla tonalità della pelle nello spazio RGB temporalmente normalizzato.
SSR	Basato sul <i>spatial subspace</i> dei pixels della pelle e sui <i>temporal rotation measurements</i> per l'estrazione del segnale BVP.
LGI	Fornisce <i>features</i> invarianti rispetto al movimento basate su trasformazioni locali differenziabili.
PBV	Utilizza le caratteristiche dei <i>blood volume changes</i> in diverse lunghezze d'onda per distinguere esplicitamente i cambiamenti di colore indotti dal battito cardiaco, rispetto a quelli generati da rumore di movimento nelle misurazioni RGB..

- *Z-score*: rimuove la componente DC da un dato segnale.

Anche in questo caso l'utente può utilizzare dei filtri *custom* (fare riferimento alla *function signature* definita nel modulo `pyVHR.BVP.filters`.

Di seguito un esempio di utilizzo del filtro di *Detrend* sul segnale RGB $\hat{\mathbf{c}}_l(t)$:

```
1 filtered_sig = apply_filter(sig, detrend)
```

Ora proviamo ad applicare un filtro *Band Pass* sul segnale BVP $\mathbf{b}_l(t)$ in modo da rimuovere le frequenze che sono al di fuori della gamma di frequenze cardiache tipiche (dai 40 ai 200 BPM):

```
1 filtered_bvp = apply_filter(bvp, BPfilter, params={'order':6, 'minHz':  
:0.65, 'maxHz':4.0, 'fps':fps})
```

Infine, pyVHR permette di effettuare due tipologie di *color thresholding*. Il primo viene utilizzato nel calcolo del segnale RGB per escludere tutti i pixel il cui colore

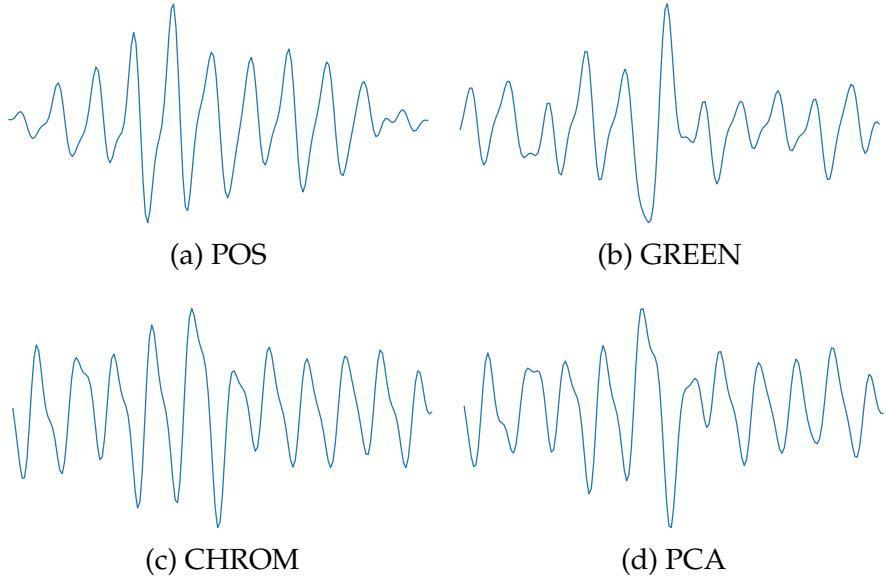


Figura 2.7: **Segnali BVP stimati.** Un esempio di segnali BVP stimati utilizzando quattro metodi differenti sulla stessa finestra temporale di un segnale RGB olistico.

non rientra in un determinato range $[X_r X_g X_b] : [Y_r Y_g Y_b]$ (per esempio $[75, 75, 75] : [230, 230, 230]$); immaginando il cubo RGB, questo significa definire un *sotto-spazio cubico* di colori. Utilizzare questo filtro è utile per escludere facilmente colori troppo scuri (ombre, barba) o troppo chiari (zone sovraesposte).

Di seguito il codice per impostare le soglie X e Y utilizzando i valori della classe *SignalProcessingParams*:

```

1 # set sig-processing color thresholding
2 SignalProcessingParams.RGB_LOW_TH = 75
3 SignalProcessingParams.RGB_HIGH_TH = 230

```

Possiamo utilizzare lo stesso tipo di *thresholding* anche per escludere pixels nella fase di *skin extraction*:

```

1 # set skin-processing color thresholding
2 SkinProcessingParams.RGB_LOW_TH = 75
3 SkinProcessingParams.RGB_HIGH_TH = 230

```

in questo caso però la classe è *SkinProcessingParams*.

La seconda tipologia di *color thresholding* permette di filtrare il segnale RGB calcolato. Ipotizziamo di avere un totale di \mathcal{L} tracce RGB, dove $\mathcal{L} = [1 \dots L]$ è il numero di *patches*; vorremmo ottenere un sottoinsieme di segnali RGB caratterizzato dalla seguente proprietà: ogni valore RGB assunto dal segnale deve essere contenuto nel range $[X_r X_g X_b] : [Y_r Y_g Y_b]$ (per esempio $[60, 60, 60] : [240, 240, 240]$). In questo modo possiamo escludere facilmente tracce RGB che assumono valori troppo scuri o chiari, seguendo la stessa logica del *color thresholding* descritto prima.

Questo tipo di filtraggio è pericoloso perché potrebbe rimuovere in una finestra temporale tutte le nostre tracce RGB, ovvero tutti i nostri L stimatori. Nel caso di finestre temporali a *zero estimatori*, pyVHR restituisce come stima del BPM il valore 0. Allo stesso tempo, utilizzare questo filtro migliora molto la precisione di stima se si utilizza un numero elevato di *patches*.

Di seguito il codice per utilizzare questo *color thresholding* su un segnale finestrato *windowed_sig* utilizzando come valori dell'intervallo $X = 60$, $Y = 240$:

```

1 filtered_windowed_sig = apply_filter(
2
3     windowed_sig ,
4     rgb_filter_th ,
5     params={ 'RGB_LOW_TH': 60 ,
6             'RGB_HIGH_TH':240})

```

2.2.5 BPM estimation

Dato il segnale BVP stimato, possiamo ottenere i battiti al minuto (BPM) associati a una data finestra temporale. Facendo riferimento a quanto spiegato nella sezione "*Analisi spettrale a livello di patch*", la Figura 2.8 mostra il PSD ottenuto attraverso il metodo di Welch a partire dai segnali BVP della figura 2.7. Il picco nello spettro sarà associato alla frequenza cardiaca istantanea f^l .

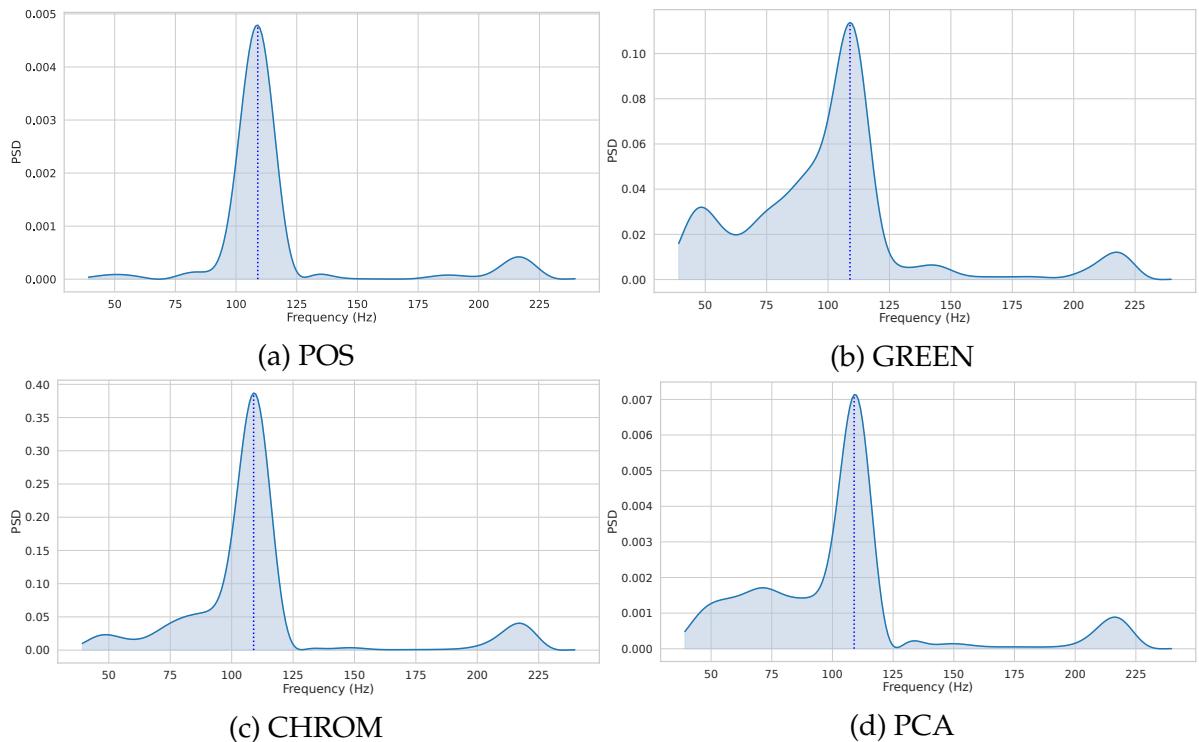


Figura 2.8: PSD stimato. Densità spettrale di potenza stimate (PSD) per i segnali BVP riportati in Figura 2.7. La stima del BPM, data dai massimi della PSD, è rappresentata dalla linea tratteggiata blu.

Quando sono state selezionate più *patches* ($L > 1$), otteniamo un unico BPM per la k -esima finestra temporale come la Mediana delle L stime BPM ottenute da ciascuna *patch* P_l .

La Figura 2.9 mostra la somma degli spettri di potenza delle PSD ottenute da $L = 100$ *patches*. Notiamo come la mediana è in grado di fornire previsioni precise, mentre il *MAD* rappresenta una misura robusta dell'incertezza. La Figura 2.10 mostra le stime BPM per un video del Dataset *UBFC*; anche in questo caso questa tecnica risulta molto affidabile.

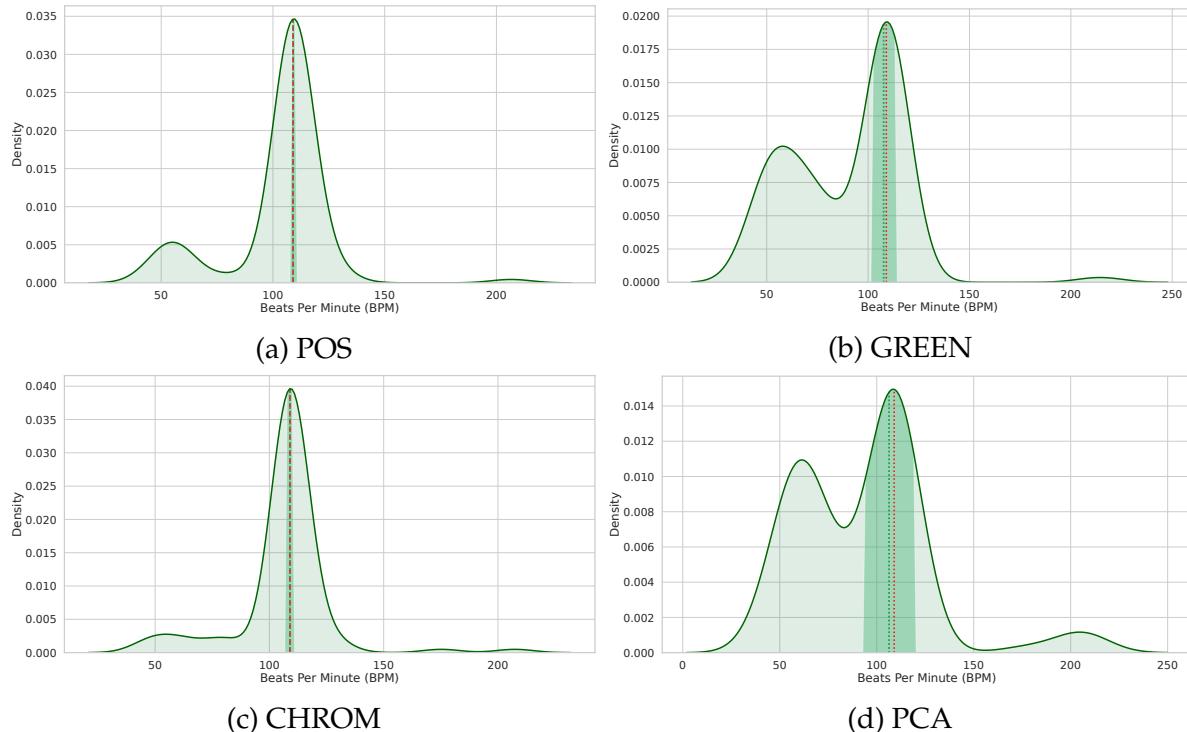


Figura 2.9: **PSD di quattro metodi rPPG su L patches.** La previsione del BPM finale è data dalla mediana (linea tratteggiata verde). La stima dell'incertezza fornita dalla *Median absolute deviation (MAD)* è mostrata in verde più scuro. la linea tratteggiata rossa rappresenta il BPM effettivo.

Il calcolo del BPM dai segnali BVP può essere facilmente ottenuto in pyVHR come segue:

```

1 from pyVHR.BPM.BPM import BVP_to_BPM, multi_est_BPM_median
2
3 bpmES = BVP_to_BPM(bvp, fps)
4 # Mediana dei BPM
5 bpm, uncertainty_MAD = multi_est_BPM_median(bpmES)

```

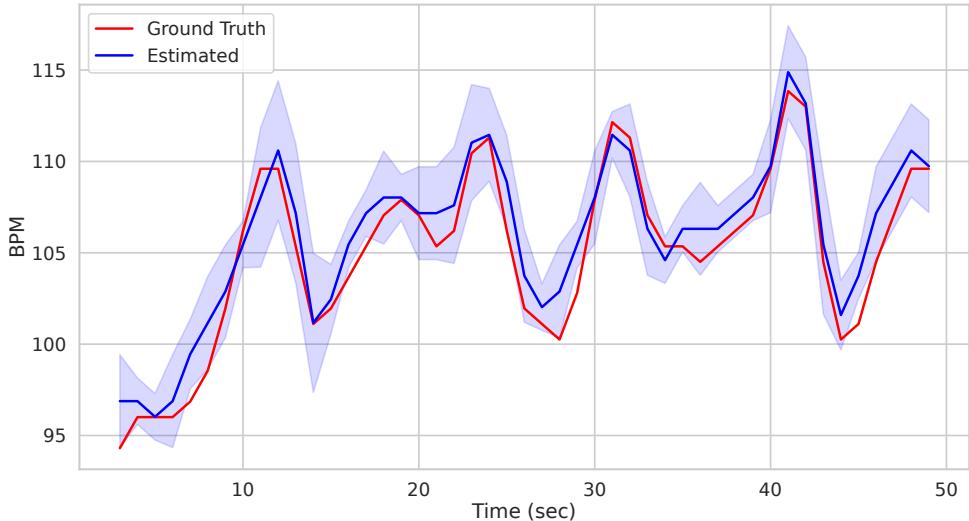


Figura 2.10: **Confronto dei BPM stimati rispetto al ground truth utilizzando l’approccio Patches.** Il BPM stimato (blu) per il Subject1 del Dataset UBFC. L’incertezza è tracciata in blu sfumato, mentre il ground truth è rappresentato dalla linea rossa.

2.2.6 Tecniche di stima del BPM basate su multi-stimatori

In alternativa al semplice approccio descritto prima per il calcolo del BPM dai segnali BVP, è possibile utilizzare la tecnica *Circle Clustering* (descritta nella sezione *Clustering delle PSD*). Di seguito è mostrato come utilizzarla in pyVHR:

```

1 from pyVHR.BPM.BPM import BVP_to_BPM_PSD_clustering,
2     BVP_to_BPM_PSD_clustering_cuda
3
4 # CPU
5 bpm = BVP_to_BPM_PSD_clustering(bvp, fps)
6
7 # GPU
8 bpm = BVP_to_BPM_PSD_clustering_cuda(bvp, fps)
```

In questo caso non servirà utilizzare il metodo *multi_est_BPM_median*, perché la tecnica *Circle Clustering* restituisce già per ogni finestra temporale un singolo valore BPM.

2.3 Pipeline per metodi deep-learning

La letteratura sulla visione artificiale ha recentemente dato ampio risalto alle reti neurali deep end-to-end e alla loro capacità di superare i metodi tradizionali che richiedono la progettazione manuale di algoritmi efficaci. In questo contesto sono nati molti progetti per il recupero dei segnali fisiologici basati su tecniche *deep-learning* (DL) (30, 31, 32, 33, 34, 35, 36). Tra questi, DeepPhys (30) è stato il primo lavoro a dimostra-

re l'efficacia della rete neurale *deep* superando tutti gli approcci tradizionali basati su modelli o tecniche statistiche.

La natura end-to-end degli approcci basati su DL si riflette in una pipeline molto più semplice; infatti, questi metodi richiedono tipicamente come input i fotogrammi di un video lossless, che vengono elaborati dall'architettura DL in uso e producono direttamente un segnale BVP o la frequenza cardiaca stimata. La figura 2.2 (regione grigia "*Deep Learning based Methods*") mostra le fasi coinvolte nella stima della frequenza cardiaca utilizzando approcci basati su DL.

Chiaramente, la semplicità dei metodi DL comporta degli aspetti negativi: il modello deve essere addestrato su enormi quantità di dati, e possono presentarsi varie problematiche relative alla capacità di generalizzazione del modello.

Nonostante l'efficacia dichiarata e le prestazioni superiori, pochi modelli DL sono stati resi pubblicamente disponibili (sia in termini di codice che di *weights* del modello). Questo è un problema per la corretta riproducibilità dei risultati e la valutazione del metodo.

In (21) è stata proposta una recente ed efficiente architettura neurale chiamata *MTTS-CAN*, che rappresenta un prezioso contributo poiché il modello pre-addestrato e il codice sono stati rilasciati. Questa rete sfrutta essenzialmente un modulo di *tensor-shift* e operazioni convoluzionali 2D per eseguire un'efficiente modellazione spazio temporale al fine di consentire misurazioni cardiovascolari e respiratorie in tempo reale. *MTTS-CAN* può essere inquadrato come un modello end-to-end poiché non necessita di alcun passaggio di pre-elaborazione prima che i dati vengano inseriti nella rete, ad eccezione dell'esecuzione di banali normalizzazioni dell'immagine. *MTTS-CAN* è incluso nel framework pyVHR e di seguito viene mostrato quanto sia pratico utilizzarlo.

2.3.1 Utilizzo della 'DeepPipeline'

La classe *DeepPipeline()* permette di effettuare la stima rPPG utilizzando un metodo DL end-to-end implementato nel pacchetto pyVHR. Nello specifico, la pipeline comprende la gestione dei video in input, la trasformazione del segnale BVP in segnale BPM, e eventualmente, le fasi di pre/post-filtraggio.

Il seguente frammento di codice permette di eseguire la pipeline Deep su un video:

```
1 from pyVHR.analysis.pipeline import DeepPipeline  
2  
3 pipe = DeepPipeline()  
4 time, BPM = pipe.run_on_video('/path/to/vid.avi', method='MTTS_CAN')
```

La figura 2.2 (regione grigia "Deep Learning based Methods") riassume i passaggi coinvolti in una chiamata a `run_on_video()`. Come nella pipeline con metodi tradizionali, `run_on_video()` restituisce i BPM stimati e i relativi timestamps (time).

Un utente può replicare gli step della `DeepPipeline()` attraverso poche righe di codice:

```

1 from pyVHR.extraction.sig_processing import SignalProcessing
2 from pyVHR.extraction.utils import get_fps
3 from pyVHR.BPM import BVPsignal
4
5 sp = SignalProcessing()
6 frames = sp.extract_raw('/path/to/videoFileName')
7 fps = get_fps('/path/to/videoFileName')
8
9 bvp_pred = MTTS_CAN_deep(frames, fps)
10 bvp = BVPsignal(bvp_pred, fps) # BVP object
11
12 winsize = 6 #6 seconds long time window
13 bpm, time = bvp.getBPM(winsize)

```

Per utilizzare un metodo rPPG DL differente in questo *snippet* di codice, basta sostituire il metodo *MTTS_CAN_deep* con un altro che abbia la stessa *function signature*.

2.4 Parallelizzazione del codice

La maggior parte dei passaggi della pipeline sono adatti per essere implementati attraverso *parallel computation*. Ad esempio, le operazioni di algebra lineare coinvolte nel recupero del segnale di impulso BVP dal segnale RGB o, più in generale, le fasi di elaborazione del segnale (es. filtraggio, stima spettrale, ecc.), per non parlare delle procedure di segmentazione della pelle da video ad alta risoluzione.

A tal fine, pyVHR sfrutta il massiccio parallelismo delle unità di elaborazione grafica (GPU). Vale la pena ricordare che le GPU non sono strettamente necessarie per eseguire il codice pyVHR; tuttavia, in alcuni casi, il codice accelerato dalla GPU consente di eseguire la pipeline in tempo reale.

La figura 2.11 mostra il tempo medio richiesto per frame per superare l'intera pipeline quando si utilizza il metodo POS. Vale la pena notare che, quando si utilizza l'approccio Holistic (o equivalentemente una singola patch), un frame video può essere elaborato in meno di 0.025 secondi, indipendentemente dall'architettura adottata (CPU o GPU). Ciò significa che l'intera pipeline può essere eseguita in sicurezza in tempo reale per i video a 30 fotogrammi al secondo (il limite di tempo di 30 fps è rappresentato dalla linea verde tratteggiata).

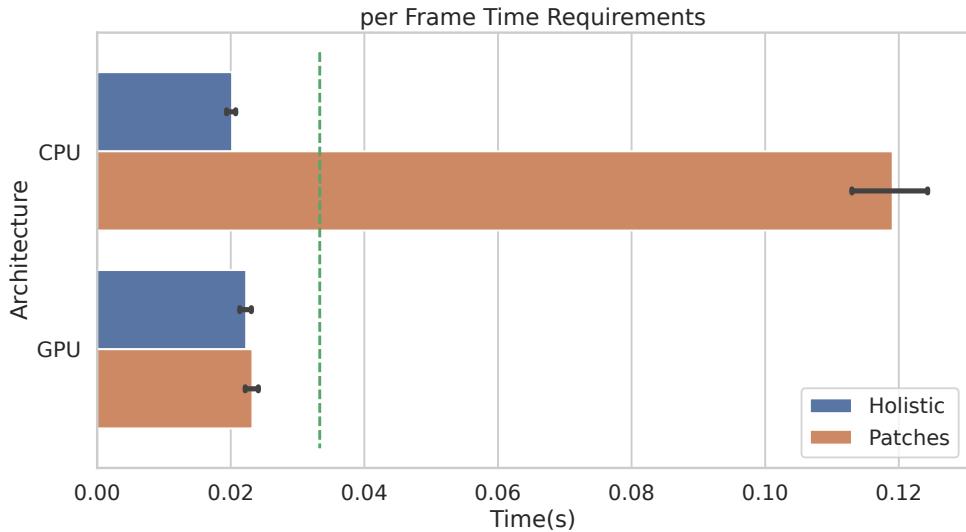


Figura 2.11: **Requisiti di tempo per frame.** Tempo medio richiesto per elaborare un frame con gli approcci Holistic e Patches quando si utilizzano implementazioni con accelerazione CPU vs. GPU. La linea tratteggiata verde rappresenta il limite *real-time* di 30 fotogrammi al secondo (fps).

Ovviamente, quando vengono impiegate più patch (nell'esempio di Figura 2.11 vengono utilizzate $L = 100$ patches), il tempo medio richiesto dalle CPU per elaborare un singolo frame sale fino a circa 0,12 secondi; invece, se si utilizza il codice accelerato con GPU potremo eseguire l'intera pipeline in tempo reale. Sorprendentemente, si osserva un guadagno simile nelle prestazioni se si adotta qualsiasi altro metodo rPPG.

Il risultato mostrato nella Figura 2.11 si riferisce alla seguente configurazione hardware: Intel Xeon Silver 4214R 2,40 GHz (CPU), NVIDIA Tesla V100S PCIe 32 GB (GPU). Risultati simili sono stati ottenuti basandosi su una configurazione non server: Intel Core i7-8700K 4,70 GHz (CPU), NVIDIA GeForce GTX 960 2GB (GPU). L'utilizzo massimo della RAM per l'analisi di video HD di 1 minuto è di 2,5 GB (in media 2 GB); l'utilizzo massimo della memoria della GPU per l'analisi video HD di 1 minuto è di 1,8 GB (la media è di 1,4 GB).

Di seguito sono elencati i passaggi della Pipeline accelerati con GPU-CUDA:

- Skin extraction: Convex Hull e Face Parsing. L'utente può scegliere facilmente di eseguire questo passaggio con CPU o GPU:

```

1 target_device = 'GPU' # or 'CPU'
2 sig_processing = SignalProcessing()
3 sig_processing.set_skin_extractor(
4     SkinExtractionConvexHull(target_device))
5 sig_processing.set_skin_extractor(
6     SkinExtractionFaceParsing(target_device))
7

```

- rPPG Methods: il pacchetto contiene versioni diverse dello stesso metodo. Ad esempio il metodo CHROM è implementato sia per CPU che per GPU:

```

1 # GPU
2 bvp = RGB_sig_to_BVP(sig, fps, device_type='cuda', method=
3   cupy_CHROM)
4 # CPU
5 bvp = RGB_sig_to_BVP(sig, fps, device_type='cpu', method=
6   cpu_CHROM)
7

```

- BPM estimation:

```

1 bpmES = BVP_to_BPM_cuda(bvp, fps)      #GPU
2 bpmES = BVP_to_BPM(bvp, fps)            #CPU
3
4 # Circle Clustering
5 bpmES = BVP_to_BPM_PSD_clustering_cuda(bvp, fps)    #GPU
6 bpmES = BVP_to_BPM_PSD_clustering(bvp, fps)          #CPU
7

```

2.4.1 CPU

Per fornire prestazioni real-time (con alcune configurazioni della Pipeline) al nuovo pyVHR senza l'utilizzo della GPU, è stato necessario parallelizzare il codice eseguito sulla CPU utilizzando tecniche *Multithreading*. Molte librerie utilizzate in pyVHR eseguono questo tipo di ottimizzazione in modo automatico; per citarne alcune: Numpy (<https://numpy.org>), PyTorch (<https://pytorch.org>), Scipy (<https://scipy.org>). In tutti gli altri casi pyVHR utilizza *Numba* (<https://numba.pydata.org>), ovvero una libreria per la compilazione *just-in-time* (JIT) e la parallelizzazione automatica del codice.

Il limite principale di Python è il fatto di essere un linguaggio interpretato. Questo non permette di avere le stesse performance di linguaggi compilati come C o C++. Malgrado ciò, con *Numba* possiamo trasformare il codice Python in codice macchina ottimizzato sfruttando il compilatore LLVM (<https://llvm.org>).

Utilizzare Numba è molto semplice e immediato; ipotizziamo di definire una funzione per calcolare la somma di tutti gli elementi di un array Numpy. In Python questo si può scrivere come:

```

1 def prange_test(A):
2     s = 0
3     for i in range(A.shape[0]):
4         s += A[i]

```

```
5     return s
```

Questo codice non avrebbe performance real-time se per esempio A avesse 10^6 elementi (o più).

Con Numba possiamo trasformare questa funzione in un kernel compilato, ottimizzato e parallelizzato:

```
1 from numba import jit, prange
2
3 @jit(nopython=True, parallel=True)
4 def prange_test(A):
5     s = 0
6     for i in prange(A.shape[0]):
7         s += A[i]
8     return s
```

Il parametro $nopython = True$ è fondamentale per ottimizzare il codice, in quanto assicura a Numba che la funzione definita non accede alle API C di Python. Invece la funzione $prange$ serve per parallelizzare automaticamente i cicli for .

Esistono moltissimi altri modi in Numba per ottimizzare al meglio il nostro codice, però già utilizzando il solo decoratore jit non parametrizzato si ottengono miglioramenti sorprendenti delle performance. Questa pagina di GitHub <https://gist.github.com/jfpuget/6f8c82b729677b0173d0> confronta i tempi di esecuzione di funzioni scritte in C, Python e Python ottimizzato con Numba; in alcuni casi Numba riesce ad ottenere tempi quasi identici a C.

2.4.2 GPU - CUDA

pyVHR permette di eseguire la sua Pipeline in real-time con qualsiasi configurazione utilizzando una scheda grafica NVIDIA con $cudatoolkit >= 10.2$. Questo è possibile perché alcune fasi della Pipeline possono essere eseguite con kernel GPU.

Esistono vari approcci per creare codice GPU in Python. Il più immediato e complicato è utilizzare codice CUDA e richiamarlo in Python; l'approccio più moderno e semplice è utilizzare librerie per trasformare automaticamente codice Python in kernel GPU NVIDIA. In qualsiasi caso, prima di scrivere un kernel da zero consiglio di controllare se questo è già implementato in PyTorch o CuSignal.

Per scrivere kernel in Python, pyVHR utilizza principalmente Cupy (<https://docs.cupy.dev>). Questa libreria è molto semplice da utilizzare e nasconde all'utente il processo di creazione e ottimizzazione del kernel GPU.

Utilizzare Cupy è immediato; prendiamo un qualsiasi codice Numpy:

```
1 import numpy as np
2
```

```

3 x_cpu = np.array([1, 2, 3])
4 l2_cpu = np.linalg.norm(x_cpu)

```

ora sostituiamo con Cupy:

```

1 import cupy as cp
2
3 x_gpu = cp.array([1, 2, 3])
4 l2_gpu = cp.linalg.norm(x_gpu)

```

Abbiamo finito! Cupy fornisce quasi tutte le funzionalità Numpy su GPU e si occupa automaticamente di allocare e liberare la memoria GPU necessaria.

Quando però sono necessari kernel *ad-hoc* che non possono essere costruiti con Cupy, pyVHR utilizza Numba. Questa incredibile libreria non si limita a creare codice ottimizzato per CPU, ma permette anche di creare kernel GPU molto facilmente.

Per creare un kernel CUDA con Numba utilizziamo il decoratore *cuda.jit* e definiamo il tipo delle variabili in ingresso al kernel.

```

1 @cuda.jit(void(uint8[:, :, :]))
2 def increment_by_one(an_array):
3     x, y = cuda.grid(2)
4     if x < an_array.shape[0] and y < an_array.shape[1]:
5         an_array[x, y] += 1

```

Essendo un vero e proprio kernel CUDA abbiamo accesso alla posizione del *thread* rispetto alle *grid* e i *block*. In questo esempio chiamando il metodo *cuda.grid(2)* otteniamo la posizione 2D del *thread*.

Con Numba dobbiamo allocare e liberare manualmente le risorse GPU, e per invocare il kernel è necessario definire il numero di Blocchi e di Thread:

```

1 threadsperblock = (16, 16)
2 blockspergrid_x = math.ceil(an_array.shape[0] / threadsperblock[0])
3 blockspergrid_y = math.ceil(an_array.shape[1] / threadsperblock[1])
4 blockspergrid = (blockspergrid_x, blockspergrid_y)
5 increment_by_one[blockspergrid, threadsperblock](an_array)

```

Sia Cupy che Numba permettono di ottenere performance estremamente migliori rispetto al semplice codice Python eseguito su CPU. La Figura 2.11 dimostra l'impatto che ha l'esecuzione su GPU in pyVHR.

2.5 GUI per l'elaborazione real time

Molti steps della Pipeline sono implementati con metodi real-time. In questo senso le varie componenti del framework pyVHR possono essere usate come *building blocks*

per applicazioni offline o online.

Dentro alla libreria è presente una Graphical User Interface (GUI) online basata interamente su pyVHR. Questa fornisce l'accesso alla maggior parte delle funzionalità disponibili in pyVHR, e permette di monitorare il processo di stima dei BPM in tempo reale; è possibile impostare facilmente i parametri della pipeline e la sorgente video di input, scegliendo una webcam o un file video.

Per avviare la GUI, si può eseguire il comando:

```
1 $ python pyVHR realtime/GUI.py
```

La figura 2.12 mostra uno screenshot della GUI durante l'analisi online di un video. In alto a destra vengono presentati il nome del file video, gli FPS del video, la risoluzione e un elenco di pulsanti di opzione per selezionare il tipo di fotogramma visualizzato. La faccia originale può essere visualizzata selezionando l'opzione *Original Video*, la faccia segmentata selezionando *Skin*, mentre il pulsante *Patches* abilita la visualizzazione delle *patches* (in rosso).

Il pulsante *Stop* termina l'analisi, e i risultati possono essere salvati su disco premendo il pulsante *Save BPMs*.

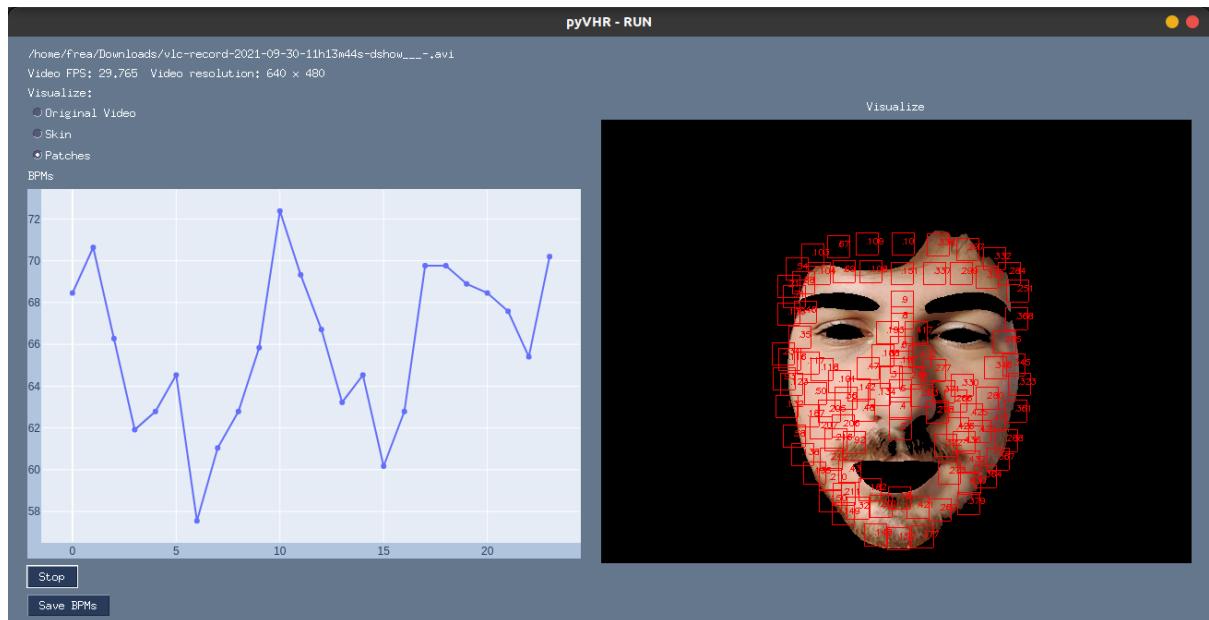


Figura 2.12: **Graphical User Interface.** Uno screenshot della *Graphical User Interface* (GUI) per l'analisi video online. Il grafico a sinistra mostra i BPM previsti, mentre a destra vengono mostrati i fotogrammi video (catturati con una webcam) della pelle segmentata e delle patch tracciate.

2.6 Estensione del framework

Oltre a valutare i vari metodi rPPG *built-in* su Dataset pubblici inclusi nel framework, la piattaforma è concepita per consentire l'aggiunta di nuovi metodi o Dataset. In questo modo è possibile valutare una nuova tecnica, confrontarla con metodi *built-in* e testarla su dataset già inclusi o su nuovi, sfruttando tutti i moduli di pre e post-elaborazione messi a disposizione in pyVHR. L'estensione del framework può essere ottenuta seguendo semplici passaggi come descritto nelle sottosezioni successive.

2.6.1 Aggiungere un nuovo Metodo rPPG

Supponiamo di voler aggiungere al framework pyVHR un nuovo metodo, chiamato MY_NEW_METHOD. Per sfruttare i moduli *built-in* pyVHR, questa nuova funzione dovrebbe essere concepita per ricevere in input un `signal` nella forma prodotta dai moduli *built-in* pre-processing, e facoltativamente insieme ad altri parametri. Nello specifico, ciò si traduce nella seguente *function signature*:

```
1 MY_NEW_METHOD(signal, **kargs)
```

dove `signal` è un array (Numpy nel caso di elaborazione CPU, Cupy nel caso di elaborazione CUDA-GPU, Torch nel caso di elaborazione con pyTorch) di `shape` (`L, 3, K`); `L` è il numero di *Patches* (può essere 1 nel caso olistico), `3` è il numero di canali RGB e `K` è il numero di fotogrammi. `**kargs` si riferisce a un dizionario che contiene tutti i possibili parametri richiesti dal metodo.

Una funzione appropriata che definisce un metodo rPPG deve restituire un segnale BVP come array di `shape` (`L, K`).

2.6.2 Aggiungere un nuovo Dataset

Attualmente pyVHR fornisce le API per la gestione di cinque Dataset comunemente adottati per la valutazione dei metodi rPPG, ovvero LGI-PPGI (14), UBFC (26), PURE (37), MAHNOB (38) e COHFACE (9). Inoltre, la piattaforma consente di aggiungere nuovi Dataset favorendo la valutazione dei metodi rPPG noti su nuovi set di dati. Il framework concepisce i Dataset come classi (vedi Fig.2.13) che ereditano della classe `Dataset` alcuni metodi per caricare video e dati PPG di ground truth.

I metodi da implementare sono i seguenti:

- `loadFilenames()` definisce due variabili di classe, vale a dire `videoFilenames` e `BVPfilenames`. Queste sono entrambe liste Python che contengono rispettivamente, i percorsi dei file video e dei file BVP ground-truth del Dataset);
- `readSigfile(filename)` restituisce il segnale BVP di ground truth di un certo `filename` (ovvero il percorso del file video).

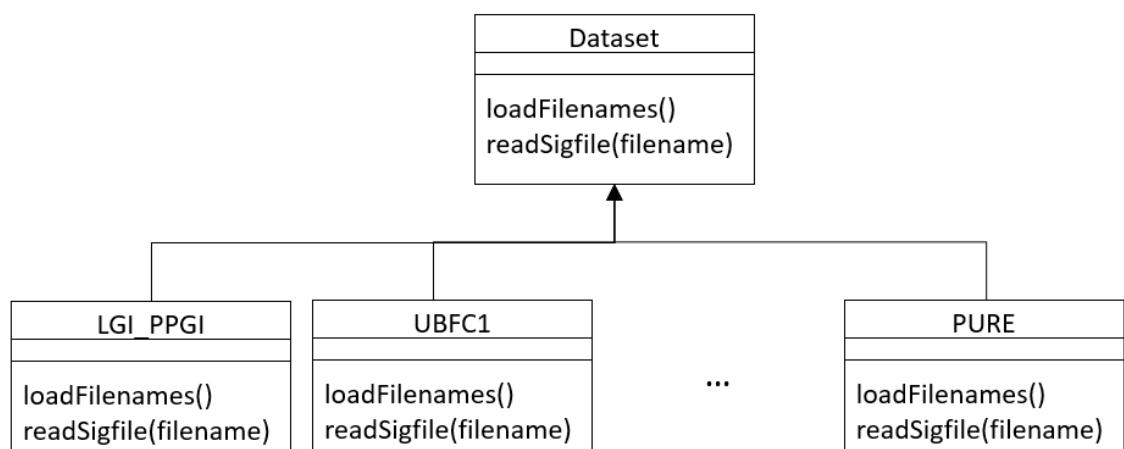


Figura 2.13: Diagramma della gerarchia della classe **Dataset**.

Capitolo 3

Analisi

3.1 Valutazione dei metodi rPPG

Dato un nuovo algoritmo rPPG, questo è migliore di quelli esistenti? Fino a che punto? La differenza di prestazioni è significativamente grande? Un particolare algoritmo di post-filtraggio causa un aumento/calo delle prestazioni?

Rispondere a tutte queste domande richiede una rigorosa valutazione statistica dei metodi rPPG. In effetti, sebbene il settore abbia recentemente registrato un notevole aumento di interesse da parte della comunità scientifica, manca ancora una metodologia di valutazione solida e riproducibile che consenta di acquisire intuizioni significative e fornire *best practices*.

In generale, nuovi algoritmi proposti in letteratura sono confrontati su set di dati non disponibili pubblicamente, ostacolando così la corretta riproducibilità dei risultati. Inoltre, in molti casi, i risultati riportati sono ottenuti con pipeline differenti; ciò rende difficile identificare con precisione il contributo del metodo proposto sulla misurazione finale della performance.

Oltre a ciò, la valutazione delle prestazioni si basa principalmente su tecniche di base e di buon senso, come i nuovi metodi di classificazione approssimativa rispetto allo stato dell'arte. Queste metodologie rozze spesso rendono la valutazione ingiusta e statisticamente infodata. Al contrario, una buona pratica di ricerca non dovrebbe limitarsi a indicare le prestazioni, ma piuttosto mirare ad analisi di principio e attentamente progettate. Ciò è in accordo con la crescente ricerca di procedure statistiche per la valutazione corretta delle prestazioni, come nel caso dell'apprendimento automatico e della visione artificiale (39, 40, 41, 42, 43).

Sulla scia del suo predecessore (5), pyVHR affronta tutti questi problemi per mezzo del suo modulo di valutazione statistica. I principi di *design* di questo modulo possono essere riassunti come segue:

- *Standardized pipeline*: Quando si imposta un esperimento per valutare un nuovo

algoritmo rPPG, l'intera pipeline (tranne l'algoritmo) dovrebbe essere sempre la stessa.

- *Valutazione riproducibile*: Il protocollo di valutazione dovrebbe essere riproducibile. Ciò comporta l'adozione di *Datasets* e codice disponibili pubblicamente.
- *Confronto su più Datasets*: Al fine di evitare *Dataset bias*, l'analisi dovrebbe essere condotta sul maggior numero possibile di *Datasets* diversi.
- *Valutazione statistica rigorosa*: I risultati ottenuti dovrebbero essere calcolati attraverso adeguate procedure statistiche, che ne valutano la significatività statistica.

Il *workflow* del Modulo di valutazione statistica è illustrato nella Figura 3.1.

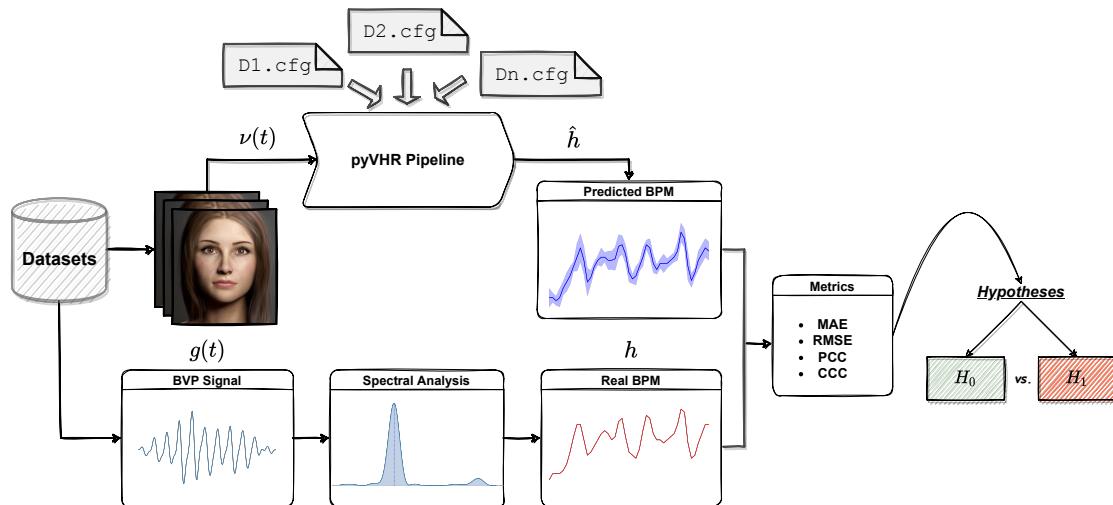


Figura 3.1: **Il modulo di valutazione statistica.** Vengono caricati uno o più set di dati; i video vengono elaborati dalla pipeline pyVHR mentre vengono recuperati i segnali BPM di ground-truth. I BPM stimati e reali vengono confrontati con metriche standard e i risultati vengono analizzati rigorosamente tramite procedure di verifica delle ipotesi.

Ogni video che compone un particolare Dataset rPPG viene elaborato dalla pipeline pyVHR. In contemporanea i segnali BVP reali registrati per esempio da un *Pulsiossimetro* vengono trasformati in segnali BPM attraverso vari metodi presenti nel modulo *pyVHR.BVP*. In particolare, il segnale BVP originale $g(t)$ è suddiviso in finestre temporali sovrapposte; per ogni finestra viene recuperato il Ground Truth BPM h^k (il BPM associato alla k -esima finestra temporale, con $k = 1, \dots, K$) tramite la massimizzazione della Power Spectral Density (PSD) fornita dal metodo Welch.

A questo punto, i segnali BPM stimati (\hat{h}^k) e *ground truth* (h^k) vengono confrontati tra loro sfruttando metriche standard (vedi sezione *Metriche di valutazione*). Alla fine, possono essere eseguiti vari confronti statisticamente rigorosi (vedi sezione *Significance Testing*).

I numerosi parametri che compongono ogni fase della pipeline (dal metodo di selezione delle ROI alle operazioni di pre/post filtraggio, passando per la stima BVP di uno o più algoritmi rPPG) possono essere facilmente specificati in un file di configurazione (`.cfg`). L'utilizzo di un file `.cfg` consente di svolgere la procedura sperimentale secondo i principi di *design* del modulo precedentemente descritti. Una breve descrizione delle metriche di confronto implementate e le specifiche del file `.cfg` sono fornite nella sezione seguente.

3.2 Metriche di valutazione

pyVHR fornisce metriche comuni per valutare le prestazioni di uno o più metodi rPPG nella stima della frequenza cardiaca (BPM) nel tempo.

Per misurare l'accuratezza della stima del BPM \hat{h} , questo viene confrontato con il BPM di riferimento h recuperato dal BVP di ground-truth. A tal fine, il segnale BVP ground-truth $g(t)$ viene suddiviso in finestre sovrapposte, analogamente alla procedura descritta nella Sezione *BVP extraction*, producendo così K segnali finestrati g^k ($k \in 1, \dots, K$). Il BPM di riferimento si trova tramite l'analisi spettrale di ciascuna finestra, come descritto nella Sezione *BPM estimation*. Ciò produce K BPM di riferimento h^k da confrontare con gli \hat{h}^k stimati adottando una delle seguenti metriche:

Mean Absolute Error (MAE). Il *Mean Absolute Error* misura la differenza assoluta media tra il \hat{h} stimato e il BPM h di riferimento. Si calcola come:

$$\text{MAE} = \frac{1}{K} \sum_k |\hat{h}^k - h^k|.$$

Valori bassi di MAE suggeriscono previsioni migliori. Il MAE è una misura facilmente interpretabile in quanto è la distanza media, in termini di BPM, delle previsioni rispetto al ground-truth.

Root Mean Squared Error (RMSE). Il *Root-Mean-Square Error* rappresenta la deviazione standard della differenza assoluta tra riferimento e stima, quindi un RMSE più piccolo suggerisce una stima più accurata. Contrariamente al MAE, anche poche grandi differenze aumenteranno l'RMSE in misura maggiore.

$$\text{RMSE} = \sqrt{\frac{1}{K} \sum_k (\hat{h}^k - h^k)^2}.$$

Pearson Correlation Coefficient (PCC). Il *Pearson Correlation Coefficient* misura la cor-

relazione lineare tra la stima \hat{h} e la verità fondamentale h . È definito come:

$$PCC = \frac{\sum_k (\hat{h}^k - \hat{\mu})(h^k - \mu)}{\sigma_1 \sigma_2},$$

dove $\hat{\mu}$ e μ indicano le medie dei rispettivi segnali, mentre σ_1 e σ_2 sono le loro deviazioni standard. Le previsioni il cui andamento segue perfettamente quello del BPM di riferimento, daranno un $PCC = 1$. Al contrario, $PCC = 0$ non suggerisce alcuna correlazione tra le quantità. Ovviamente valori negativi di PCC suggeriscono la presenza di trend opposti.

Concordance Correlation Coefficient (CCC). Il *Concordance Correlation Coefficient* (44) è una misura dell'*accordo* tra due segnali. Come la correlazione di Pearson, CCC varia da -1 a 1, e per segnali identici avremo $CCC = 1$. È definito come:

$$CCC = \frac{2\sigma_{12}}{(\hat{\mu} - \mu)^2 + \sigma_1^2 + \sigma_2^2}$$

dove $\hat{\mu}$ e μ indicano rispettivamente le medie delle tracce BPM stimate e di riferimento. Allo stesso modo, σ_1 e σ_2 sono le loro deviazioni standard, mentre σ_{12} è la loro covarianza.

La CCC supera uno svantaggio della PCC. Infatti, la correlazione di Pearson non tiene conto della presenza di bias tra le previsioni e i segnali di riferimento. Per i nostri scopi, questo significa che se la traccia BPM stimata segue perfettamente quella di riferimento ma, per esempio, 10 BPM sotto, allora la PCC non sarà in grado di notare la mancanza di *accordo* tra le due misure, producendo così valori di PCC vicini a 1. La presenza di bias sarebbe eventualmente rivelata da altre misure, come MAE o RMSE, che, d'altra parte, non fornirebbero alcuna informazione sulla covarianza lineare dei segnali. Al contrario, il CCC è in grado di tenere conto sia della corrispondenza (bias) che della correlazione lineare.

Signal to Noise Ratio (SNR). L'SNR (17) misura il rapporto tra la potenza attorno alla frequenza cardiaca di riferimento più la prima armonica del segnale di impulso stimato e la potenza residua contenuta nello spettro del BVP stimato. Formalmente è definito come:

$$SNR = \frac{1}{K} \sum_K 10 \log_{10} \left(\frac{\sum_v (U^k(v) S^k(v))^2}{\sum_v (1 - U^k(v)) S^k(v))^2} \right)$$

dove $S^k(v)$ è la densità spettrale di potenza del BVP stimato nella k -esima finestra temporale, e $U^k(v)$ è una maschera binaria che seleziona solo la potenza

contenuta in ± 12 BPM intorno al battito cardiaco di riferimento e la sua prima armonica. Un valore alto di SNR è indice di un segnale poco rumoroso.

3.3 Datasets di benchmark

Come già anticipato, il framework è pensato anche per effettuare analisi multi-dataset. In particolare, pyVHR fornisce il supporto per i seguenti 5 Datasets:

PURE (37). Il dataset PURE consiste di 60 sequenze video di 1 minuto ciascuna. In totale 10 soggetti (8 maschi, 2 femmine) che sono stati registrati in 6 diverse configurazioni, vale a dire Steady (S); Talking (T); Slow translation (ST); Fast translation (FT); Small rotation (SR); Medium rotation (MR). L'illuminazione è fornita principalmente dalla luce frontale del sole, con presenza di nuvole che cambiano leggermente le condizioni di illuminazione nel tempo. I soggetti erano seduti davanti a una telecamera con una distanza di circa 1,1 metri; i video sono *non compressi* con una risoluzione di 640×480 a 30 fps. La frequenza del polso ground-truth è stata acquisita utilizzando un *Pulsiossimetro* da dito con una frequenza di campionamento di 60 Hz.

UBFC (26). Il dataset UBFC fornisce 50 video con una durata di circa 2 minuti. I video sono registrati a 30 fps con una risoluzione di 640×480 in formato RGB a 8 bit non compresso. Gli autori hanno diviso questo dataset in due sottoinsiemi: il primo, UBFC1 è composto da 8 video, in cui ai partecipanti è stato chiesto di stare fermi; il secondo, UBFC2 è composto da 42 video, in cui ai partecipanti è stato chiesto di giocare a un gioco matematico che mirava ad aumentare la frequenza cardiaca emulando contemporaneamente un normale scenario di interazione uomo-computer. I partecipanti erano seduti davanti a una telecamera posta a una distanza di circa 1 metro.

LGI (14). Il Dataset LGI è progettato per la stima della frequenza cardiaca da video di volti non compressi acquisiti in natura. Viene registrato in quattro diverse sessioni: 1) uno scenario di riposo senza movimento della testa né cambi di illuminazione, 2) sono consentiti i movimenti della testa (con illuminazione statica), 3) le persone vengono registrate mentre eseguono esercizi su una bicicletta in palestra; 4) vengono registrate le conversazioni urbane, inclusi i movimenti della testa e del viso, nonché le condizioni di illuminazione naturale variabili. I video sono stati catturati a 25 fps mentre la frequenza di campionamento del segnale a impulsi era di 60Hz. Vale la pena notare che sebbene il Dataset originale fornisca nominalmente 25 soggetti, al momento della scrittura solo 6 sono ufficialmente rilasciati e quindi utilizzati nell'analisi.

MAHNOB-HCI (38). Sebbene questo database sia stato concepito principalmente per l’analisi delle emozioni, è stato adottato per i test degli algoritmi rPPG, (45), (46). Il Dataset contiene video di 17 femmine e 13 maschi, di età compresa tra 19 e 40 anni. Sono state utilizzate 6 videocamere disposte in diverse angolazioni, un microfono indossato sulla testa, un tracker dello sguardo, e sensori fisiologici che misurano l’ECG, elettroencefalogramma, respirazione e temperatura cutanea. Il problema principale di questo Dataset è che contiene solo video a cui è stata applicata una forte compressione; di seguito una descrizione completa delle proprietà video: compressione H.264 o MPEG-4 AVC, bit rate $\approx 4200 \text{ kb/s}$, 61 fps, 780×580 pixel, $\approx 1,5 \times 10^{-5}$ bit per pixel,

COHFACE (47). Questo Dataset contiene 160 video RGB della durata di un minuto sincronizzati con i battiti cardiaci e la frequenza respiratoria di 40 soggetti (12 femmine e 28 maschi). Ad ogni partecipante è stato chiesto di stare fermo davanti a una webcam per consentire di catturare l’intero viso. Sono stati considerati due tipi di condizioni di illuminazione: studio, utilizzando un faretto, e luce naturale. I video sono compressi in MPEG-4 Visual, ovvero MPEG-4 Part 2, con bit rate $\approx 250 \text{ kb/s}$, risoluzione 640×480 pixel, 20 fotogrammi al secondo, e $\approx 5 \times 10^{-5}$ bit per pixel. In altre parole, i video sono stati fortemente compressi.

Un elenco completo dei *datasets* che sono tipicamente impiegati per la stima e la valutazione dei metodi rPPG è riportato nella Tabella 3.1. Nel Capitolo successivo riguardante i risultati sperimentali ottenuti sono stati esclusi sia MAHNOB-HCI che COHFACE, perché non esistono ancora metodi rPPG efficaci per stimare la frequenza cardiaca da video compressi.

Dataset	Currently Implemented in pyVHR
UBFC-rPPG (26)	✓
PURE (37)	✓
LGI-PPGI (14)	✓
MAHNOB-HCI (38)	✓
COHFACE (47)	✓
UBFC-Phys (48)	✗
AFRL (49)	✗
MMSE-HR (50)	✗
OBF (51)	✗
VIPL-HR (52)	✗
VIPL-HR-V2 (53)	✗
ECG-Fitness (54)	✗

Tabella 3.1: Un elenco di Datasets rPPG comunemente usati.

3.4 Analisi di un Dataset attraverso la Pipeline

Il file .cfg permette di impostare facilmente gli esperimenti per la valutazione dei metodi rPPG. È strutturato in sei blocchi principali descritti di seguito:

Dataset. Questo blocco contiene le informazioni relative a un particolare dataset rPPG; il primo campo è il nome del modulo che permette di interagire con il Dataset, mentre il secondo è il percorso dove è situato questo modulo.

```
1 [DATASET]
2 dataset      = MyDataset
3 path = path/to/module/
4 videodataDIR= /path/to/vids/
5 BVPdataDIR   = /path/to/gt/
6 ...
```

E' importante sottolineare che il modulo *MyDataset* deve contenere una classe chiamata *MyDataset* (non importano maiuscole e minuscole) che estende la classe *pyVHR.datasets.dataset.Dataset*.

Filters. Definisce vari filtri da utilizzare eventualmente in fase di pre/post filtraggio. Nell'esempio seguente viene definito un filtro passa-banda Butterworth di sesto ordine, con una banda passante compresa tra 42Hz e 240Hz.

```
1 ### FILTERS ###
2 [BPFILTER]
3 path = None
4 name = BPfilter
5 params = {'minHz':0.7, 'maxHz':4.0, 'fps':'adaptive', 'order':6}
```

Questo filtro è identificato dall'etichetta "BPFILTER"; un'utente può anche creare N filtri che utilizzando lo stesso metodo, purché abbiano etichette diverse.

Segnale RGB. Definisce tutti i parametri per l'estrazione del segnale RGB (es. metodo di selezione della ROI, dimensione della finestra temporale, numero e tipo di patch da utilizzare, ecc.).

```
1 [SIG]
2 ...
3 winSize = 6
4 skin_extractor = convexhull
5 approach = patches
6 ...
```

Questa sezione non è necessaria nel .cfg della Pipeline per metodi DL.

Methods. Permette di configurare vari metodi rPPG (con eventuali parametri e filtri pre/post). In questo esempio definiamo due metodi CPU, il primo, POS non

utilizzerà alcun filtro pre/post, mentre il secondo, GREEN, utilizzeremo il filtro passa banda sopra definito sia in *pre* che *post* filtraggio.

```

1 ##### METHODS #####
2 [POS]
3 ...
4 name = cpu_POS
5 device_type = cpu
6 pre_filtering = []
7 post_filtering = []
8
9 [GREEN]
10 ...
11 name = cpu_GREEN
12 device_type = cpu
13 pre_filtering = ['BPFILTER']
14 post_filtering = ['BPFILTER']
```

Ogni metodo è identificato da una etichetta; un'utente può anche creare N metodi che utilizzando lo stesso algoritmo rPPG, purché abbiamo etichette diverse. Questo permette di testare lo stesso metodo con diverse configurazioni di parametri e filtri.

BVP. In questo blocco possiamo selezionare i metodi rPPG da utilizzare per la stima del segnale BVP. In questo esempio verranno analizzati i due metodi definiti precedentemente, ovvero POS e GREEN.

```

1 [BVP]
2 methods = ['POS', 'GREEN']
3 ...
```

Nel caso della *DeepPipeline* questa sezione è struttura come segue:

```

1 [BVP]
2 winSize = 10
3 methods = ['MTTS_CAN']
4 post_filtering = ["DETREND", "BPFILTER"]
```

dove *methods* è una lista di metodi rPPG DL implementati in pyVHR e *post_filtering* è una lista di filtri da applicare al BVP restituito dalla rete deep. Ovviamente il .cfg della *DeepPipeline* non possiede la sezione *METHODS*.

BPM. In questo blocco possiamo selezionare il modo in cui il BVP è trasformato in BPM. In questo esempio utilizziamo la massimizzazione della Power Spectral Density (PSD) fornita dal metodo Welch, e applichiamo un filtro passa-banda con una banda passante compresa tra 42Hz e 240Hz.

```

1 [BPM]
2 type = welch
3 minHz = 0.7
4 maxHz = 4.0

```

L'esperimento sul dataset definito nel file .cfg può essere semplicemente lanciato con le seguenti righe di codice:

```

1 from pyVHR.analysis.pipeline import Pipeline
2
3 pipe = Pipeline()
4 results = pipe.run_on_dataset('/path/to/config.cfg')
5 results.saveResults("/path/to/results.h5")

```

Nel codice precedente, il metodo `run_on_dataset` della classe `Pipeline`, analizza il file .cfg e avvia una pipeline per ogni metodo rPPG definito in esso. La Pipeline viene utilizzata per elaborare ogni video del Dataset. Allo stesso tempo, vengono caricati i BPM ground-truth, che saranno confrontati con i BPM stimati attraverso le metriche di confronto. I risultati vengono salvati in un file *HDF5*, che sarà utilizzato durante il *Significance Testing*.

Per utilizzare la Pipeline per metodi DL basterà sostituire nello *snippet* di codice `Pipeline` con `DeepPipeline` (e correggere l'*import* di riga 1).

E' importante ricordare che il file .cfg può contenere anche metodi e dataset *Custom* definiti dall'utente. Di seguito un esempio in cui l'utente definisce lo stesso metodo due volte, cambiando il valore dei suoi parametri:

```

1 [MyNewMethod_HighValue]
2 path = 'path/to/module.py'
3 name = 'MY_NEW_METHOD'
4 params = {'myparam': 'high'}
5 ...
6
7 [MyNewMethod_LowValue]
8 path = 'path/to/module.py'
9 name = 'MY_NEW_METHOD'
10 params = {'myparam': 'low'}
11 ...

```

Mentre per un dataset custom:

```

1 [DATASET]
2 dataset      = myCustomDataset
3 path = /path/to/myCustomDataset/
4 videodataDIR= /path/to/videos/
5 BVPdataDIR  = /path/to/gtfiles/
6 ...

```

Il modulo `myCustomDataset` deve definire al suo interno una classe chiamata `myCustomDataset` (non importano maiuscole e minuscole) che estende la classe `pyVHR.datasets.dataset.Dataset`.

3.5 Significance Testing

Una volta calcolate le metriche di confronto per tutti i metodi considerati, è possibile valutare la significatività delle differenze tra le loro prestazioni. In altre parole, vogliamo garantire che tale differenza non sia tracciata a caso, ma rappresenti un effettivo miglioramento di un metodo rispetto a un altro. A tal fine, pyVHR ricorre a procedure standard di test di ipotesi statistica. Nel caso di due popolazioni, tipicamente, si impiega il paired t-test; in alternativa sono disponibili alcune versioni non parametriche del t-test, cioè il Sign Test o il Wilcoxon signed ranks Test. In generale il secondo è preferito al primo per la sua maggiore potenza. Per lo stesso motivo si raccomanda di adottare il paired t-test parametrico invece del test non parametrico di Wilcoxon. Tuttavia, l'uso del t-test è soggetto al vincolo della normalità delle popolazioni. Se tale condizione non è soddisfatta, si dovrebbe scegliere un test non parametrico.

Allo stesso modo, con più di due *Pipelines*, l'ANOVA a misure ripetute è il test parametrico che viene solitamente adottato. Il ricorso all'ANOVA richiede che siano soddisfatte le condizioni di normalità ed *Heteroskedasticity* (uguaglianza delle varianze). In alternativa, quando queste non possono essere garantite, si sceglie il test di Friedman.

In pyVHR le condizioni di Normalità ed *Heteroskedasticity* sono controllate automaticamente attraverso il test di Normalità di Shapiro-Wilk e, a seconda della Normalità con il test di Levene o il test di Bartlett per l'omogeneità dei dati.

Nel caso di confronti multipli (ANOVA/Friedman), è necessaria un'adeguata analisi post-hoc per stabilire le differenze a coppie tra le *Pipelines*. In particolare, il test post-hoc Tukey viene adottato a valle del rifiuto dell'ipotesi nulla dell'ANOVA (le medie delle popolazioni sono uguali), mentre il test post-hoc Nemenyi viene utilizzato dopo il rifiuto dell'ipotesi nulla di Friedman di uguaglianza delle mediane dei campioni.

Oltre alla significatività delle differenze, è conveniente riportare anche la loro grandezza. La *effect size* può essere calcolata tramite la *d* di Cohen nel caso di popolazioni normali, altrimenti si usa la *gamma* di Akinshin.

pyVHR gestisce automaticamente tutto questo con la classe `StatAnalysis()`, facendo affidamento sul pacchetto *Autorank* di Python (55); `StatAnalysis()` prende in input i risultati prodotti dalla *Pipeline* ed esegue il test statistico appropriato su una metrica di confronto scelta:

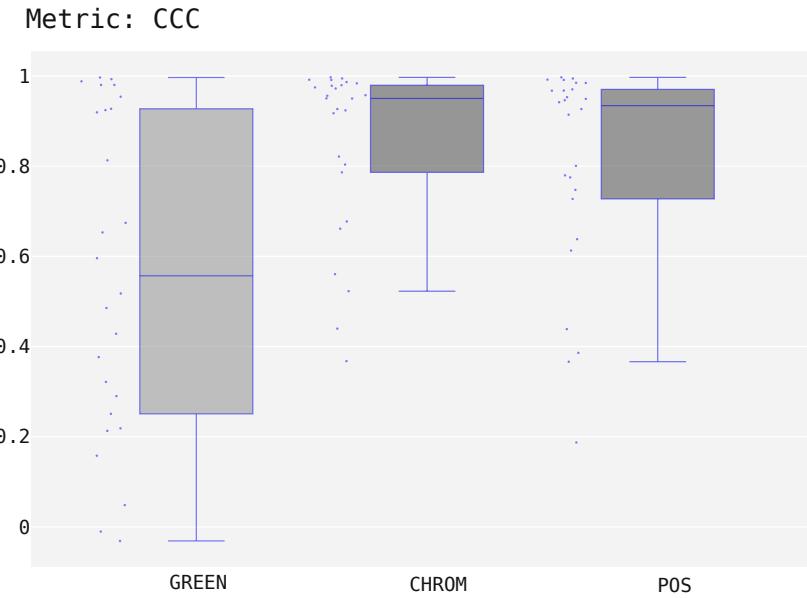


Figura 3.2: Box plot di esempio che mostrano la distribuzione dei valori CCC per i metodi POS, CHROM e GREEN sul dataset UBFC.

```

1 from pyVHR.analysis.stats import StatAnalysis
2
3 st = StatAnalysis("/path/to/results.h5")
4 # -- box plot statistics (medians)
5 st.displayBoxPlot(metric='CCC')
6
7 #testing
8 st.run_stats(metric='CCC')

```

Un esempio di output della procedura di test statistico è riportata nelle Figure 3.2 e 3.3; possiamo interpretare questi risultati nel modo seguente:

Abbiamo rifiutato l'ipotesi nulla che la popolazione sia normale per le popolazioni POS ($p = 0,000$) e GREEN ($p = 0,000$). Pertanto, assumiamo che non tutte le popolazioni siano normali. Poiché abbiamo solo due popolazioni ed entrambe non sono normali, usiamo il Wilcoxon signed ranks test per determinare le differenze nelle misure di tendenza centrale e riportiamo la mediana (MD) e la Median absolute deviation (MAD) per ogni popolazione. Rifiutiamo l'ipotesi nulla ($p = 0,000$) del Wilcoxon signed ranks test, ovvero che la popolazione POS ($MD = 1,344 \pm 1,256$, $MAD = 0,688$) non sia maggiore della popolazione GREEN ($MD = 2,297 \pm 3,217$, $MAD = 1,429$). Pertanto, assumiamo che la mediana di POS sia significativamente più grande del valore mediano di GREEN, e che la *effect size* sia molto grande ($\gamma = -0.850$).

Come si può osservare, è stato selezionato correttamente il test statistico appropriato per due popolazioni non normali. Il coefficiente di correlazione di concordanza (CCC) per il metodo POS è risultato significativamente più grande del CCC del metodo GREEN. Oltre ad essere significativa, tale differenza è sostanziale, come testimoniato dalla dimensione dell'*effect size*.

Supponiamo ora di strutturare il suddetto .cfg in modo da eseguire tre metodi invece di due. Modifichiamo il file come segue:

```

1  #####BVP#####
2  [BVP]
3  methods = ['POS', 'GREEN', 'CHROM']
4  ...
5  #####METHODS#####
6  [POS]
7  ...
8
9  [GREEN]
10 ...
11
12 [CHROM]
13 ...
14 name = CHROM
15 pre_filtering = ['BPFILTER']
16 post_filtering = ['BPFILTER']
```

Rieseguire l'analisi statistica produrrebbe il seguente output:

Abbiamo rifiutato l'ipotesi nulla che la popolazione sia normale per le popolazioni CHROM ($p = 0.000$), POS ($p = 0.000$) e CPU_GREEN ($p=0.000$). Pertanto, assumiamo che non tutte le popolazioni siano normali. Poiché abbiamo più di due popolazioni e alcune di esse non sono normali, utilizziamo il test di Friedman non parametrico come *omnibus test* per determinare se ci sono differenze significative tra i valori mediani delle popolazioni. Usiamo il test Nemenyi post-hoc per dedurre quali differenze sono significative. Riportiamo la mediana (MD), la Median Absolute Deviation (MAD) e il rango medio (MR) di tutte le popolazioni sui vari campioni. Rifiutiamo l'ipotesi nulla ($p=0.000$) del test di Friedman che non vi sia differenza nella misure di tendenza centrale delle popolazioni CHROM ($MD = 1.263 \pm 1.688$, $MAD = 0.515$, $MR = 1.385$), POS ($MD = 1,344 \pm 1,513$, $MAD = 0,688$, $MR = 1,769$) e GREEN ($MD = 2,297 \pm 4,569$, $MAD = 1,429$, $MR = 2,846$). Pertanto, assumiamo che vi sia una differenza statisticamente significativa tra i valori mediani delle popolazioni. Sulla base del test Nemenyi post-hoc, assumiamo che non

vi siano differenze significative all'interno dei seguenti gruppi: CHROM e POS. Tutte le altre differenze sono significative.

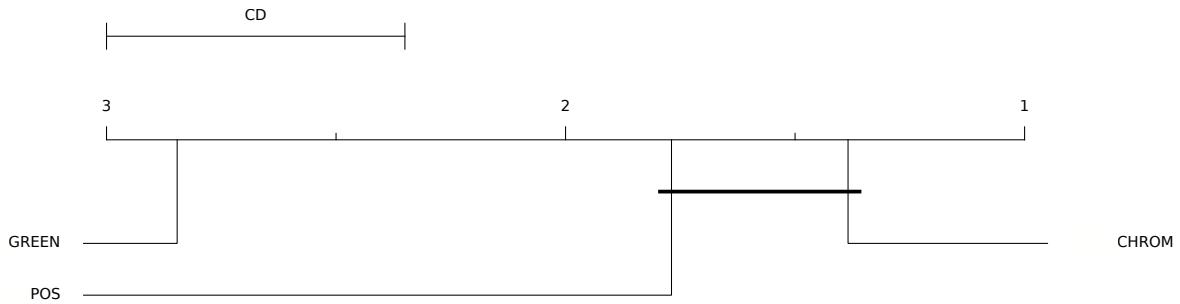


Figura 3.3: Risultati di esempio della procedura di valutazione statistica. Diagramma CD che mostra i risultati del test post-hoc Nemenyi sui valori CCC delle tre popolazioni POS, CHROM e GREEN sul dataset UBFC.

In particolare, la presenza di più di due popolazioni non normali porta alla scelta del Test di Friedman non parametrico come omnibus test per determinare se vi siano differenze significative tra i valori mediani delle popolazioni. La Figura 3.2 mostra i boxplot delle distribuzioni dei valori CCC per questi tre metodi sul dataset UBFC, mentre l'output del test Nemenyi post-hoc può essere visualizzato attraverso il diagramma di Differenze Critiche (CD) (39) mostrato in Figura 3.3; I diagrammi CD mostrano il grado medio di ciascun metodo (gradi più alti significa punteggi medi più alti); i modelli la cui differenza di rango non supera CD_{α} ($\alpha = 0.05$) sono uniti da linee spesse e non possono essere considerati significativamente differenti.

Confronto tra Pipeline Deep e Tradizionale

In che modo un determinato metodo rPPG basato su DL si confronta con gli approcci tradizionali sopra menzionati? Il seguente frammento di codice consente di eseguire sia la pipeline tradizionale che quella *Deep*.

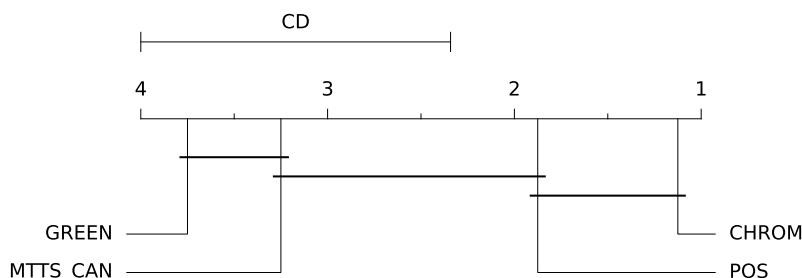


Figura 3.4: Risultati della procedura di valutazione statistica. Diagramma CD che mostra i risultati del test post-hoc di Nemenyi sui valori SNIR delle quattro popolazioni (POS, CHROM, MTTS-CAN e GREEN) sul dataset UBFC1.

Definiamo un’istanza della classe StatAnalysis passando in input il path della cartella dove sono stati salvati i risultati delle due analisi; il flag join_data=True permette di *unire* i risultati prodotti dalle due pipeline (perché strutturati differentemente), consentendo così il confronto statistico tra i metodi scelti.

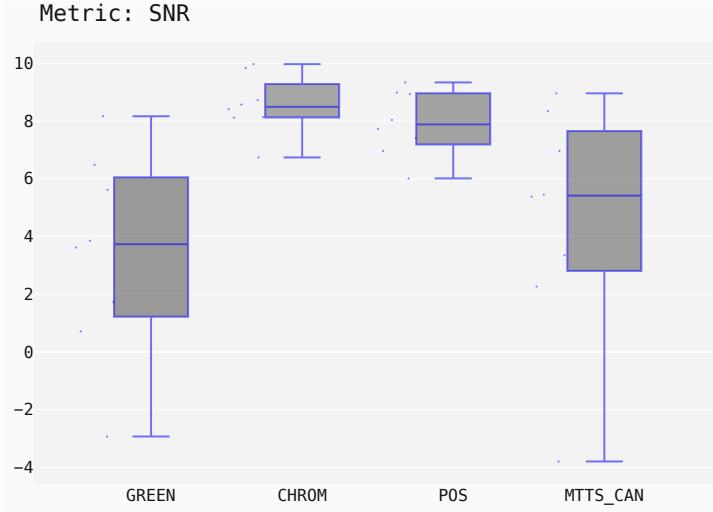


Figura 3.5: Box plot che mostrano la distribuzione dei valori SNR per i metodi POS, CHROM, MTTS-CAN e GREEN sul dataset UBFC1.

```

1 from pyVHR.analysis.stats import StatAnalysis
2 from pyVHR.analysis.pipeline import Pipeline, DeepPipeline
3
4 #Pipeline for Traditional Methods
5 traditional_pipe = Pipeline()
6 traditional_results = traditional_pipe.run_on_dataset('/path/to/config.
    cfg')
7 traditional_results.saveResults("/path/to/results_folder/
    traditional_results.h5")
8
9 #Pipeline for Deep Methods
10 deep_pipe = DeepPipeline()
11 deep_results = deep_pipe.run_on_dataset('/path/to/deep_config.cfg')
12 deep_results.saveResults("/path/to/results_folder/deep_results.h5")
13
14 #Statistical Analysis
15 st = StatAnalysis("/path/to/results_folder/", join_data=True)
16 # -- box plot statistics
17 st.displayBoxPlot(metric='SNR')
18 #Significance testing on the SNR metric
19 st.run_stats(metric='SNR')
```

In questo caso, come metrica di confronto è stato scelto il rapporto segnale/rumore (SNR); La figura 3.5 mostra qualitativamente i risultati del confronto dei metodi tradi-

zionali sopra menzionati con l'approccio basato su DL MTTS-CAN (21). L'esito della valutazione statistica è mostrato nel diagramma CD della Figura 3.4.

Capitolo 4

Risultati sperimentali

4.1 Risultati multi-dataset

Per prima cosa forniamo un riepilogo, tramite *box plot*, delle prestazioni raggiunte dai sette algoritmi rPPG sui Dataset di riferimento (il set di dati UBFC è diviso in due parti, ovvero UBFC1 e UBFC2). Misurando la tendenza centrale tramite la mediana siamo in grado di ottenere informazioni sulla distribuzione sottostante e di identificare possibili valori anomali. Le Figure 4.1 e 4.2 mostrano i *box plot* standard calcolati dal framework pyVHR e associati alla metrica MAE. I dati sono tracciati in scala logaritmica per enfatizzare i valori migliori per ciascuna metrica.

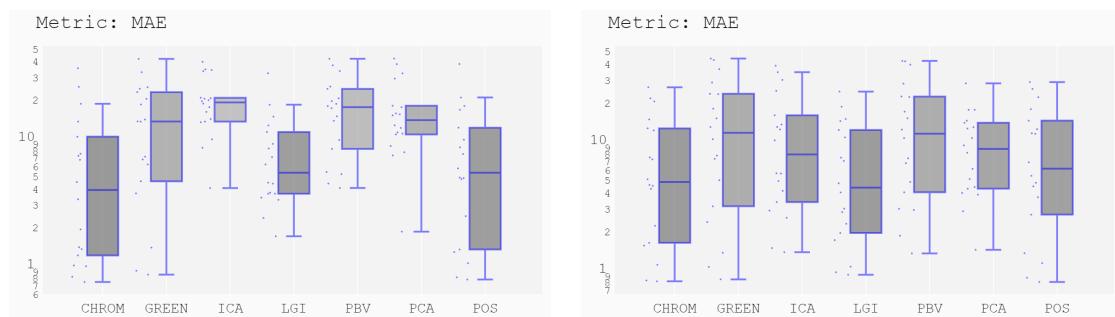
I parametri di configurazione scelti sono i seguenti: dimensione delle *patches* di 40×40 pixels, 100 *landmarks* equi spaziati sul volto, *skin color thresholding* stretto con valori di $X = 65$ e $Y = 235$, ampio con $X = 1, Y = 255$, *RGB signal color thresholding* (detto anche semplicemente *color thresholding*) stretto con $X = 65$ e $Y = 235$, ampio con $X = 1, Y = 255$ ($X = 1$ permette di non avere finestre senza stimatori), *window size* di 8 secondi; per aggregare più BPM stimati (numero di *patches* > 1) in un unico BPM è stata utilizzata in un caso la tecnica basata su Mediana (punti *b, c* delle Figure 4.1 e 4.2), nell'altro quella basata su *Circle Clustering* (punto *d*).

I filtri utilizzati dipendono dal metodo, ma in generale è consigliato utilizzare in *pre* e *post* filtraggio un filtro passa banda con frequenza di taglio inferiore di 0.75Hz e superiore di 3.8Hz .

La considerazione generale che si può trarre a colpo d'occhio (oltre alla scala logaritmica) è che gli estremi superiori e inferiori non si allontano troppo rispetto al primo e al terzo quartile. È anche evidente che gli estremi superiori sono sempre più estesi rispetto a quelli inferiori.

Notiamo una elevata variabilità dei risultati, che non permette di stabilire un vincitore o un perdente assoluto tra i metodi rPPG rispetto a tutti i Datasets. Questi risultati si basano tutti su Datasets non compressi in quanto non esistono ancora tecniche efficaci

LGI-PPGI



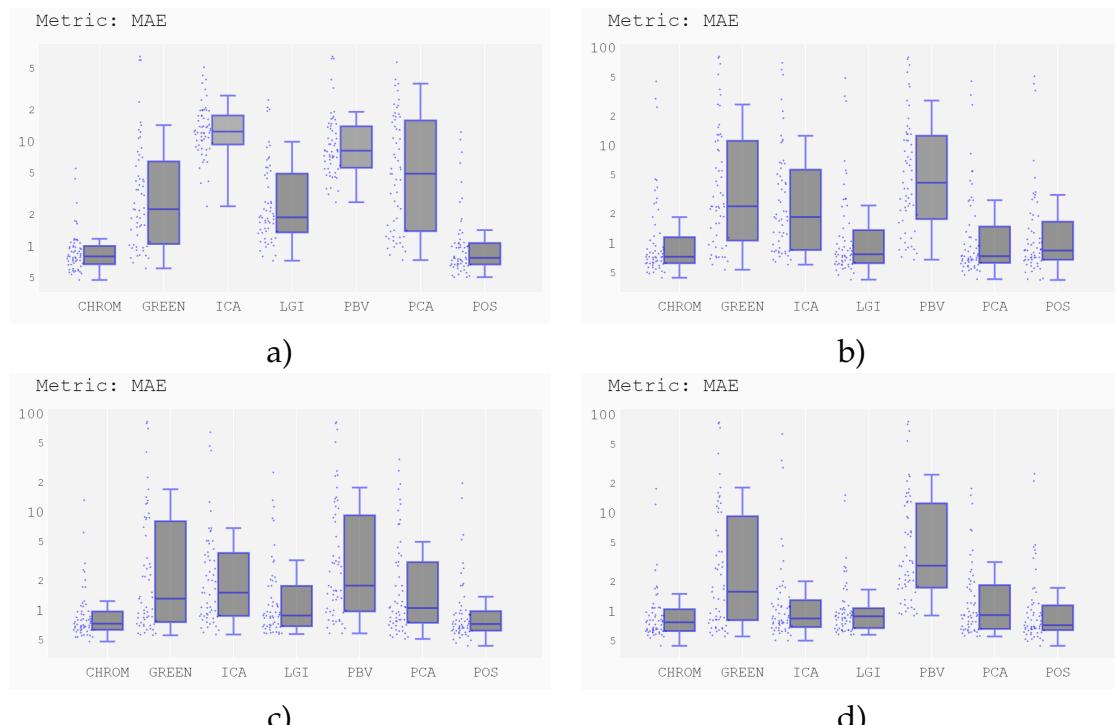
a)

b)

c)

d)

PURE



a)

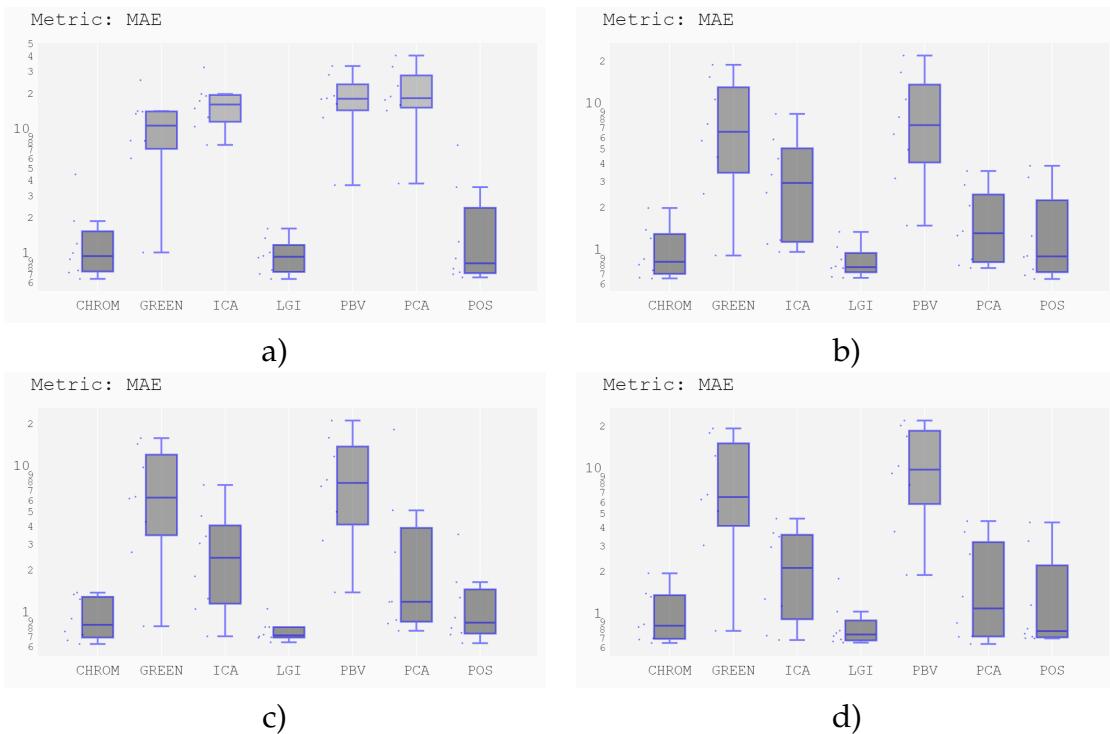
b)

c)

d)

Figura 4.1: Risultati dei Datasets LGI-PPGI e PURE. Per ogni metodo rPPG viene riportato: a) risultati dell'approccio olistico, b) risultati dell'approccio *patches* con intervallo ampio di *RGB signal color thresholding*, c) risultati dell'approccio *patches* con intervallo stretto di *RGB signal color thresholding*, d) risultati dell'approccio *patches* con intervallo stretto di *RGB signal color thresholding* e utilizzo della tecnica *Circle Clustering* per la stima del BPM. In tutti i casi è stato utlizzato *skin color thresholding* stretto.

UBCF1



UBCF2

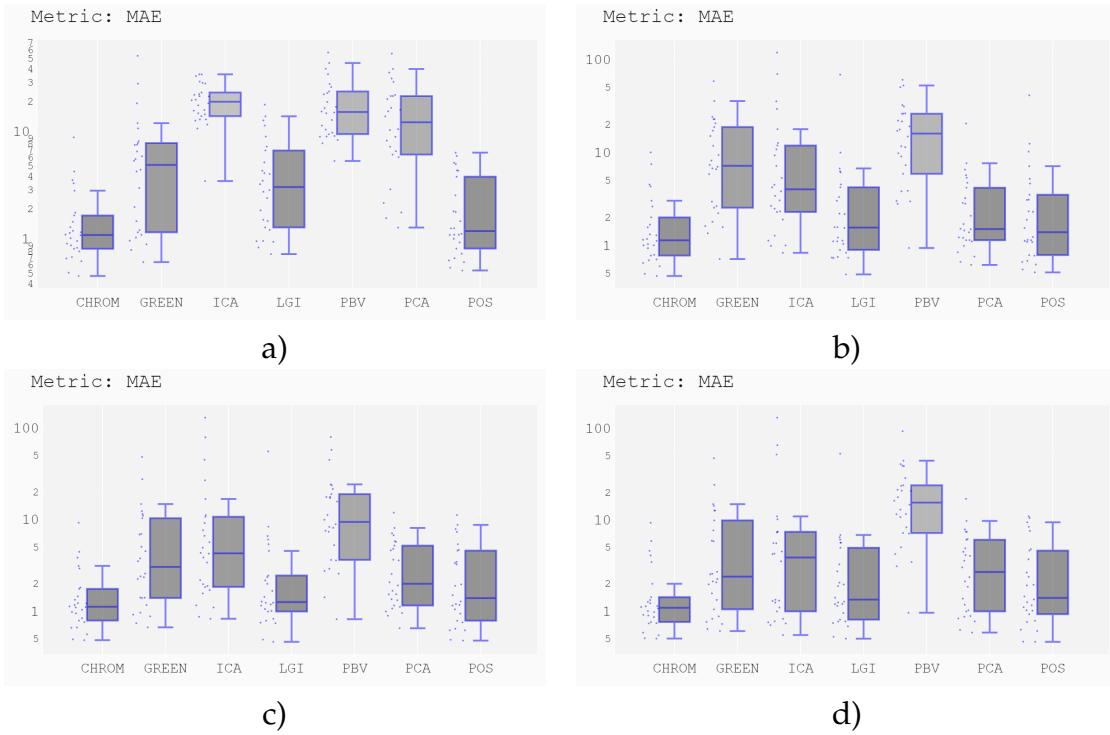


Figura 4.2: Risultati dei Datasets UBCF1 e UBCF2. Per ogni metodo rPPG viene riportato: a) risultati dell'approccio olistico, b) risultati dell'approccio *patches* con intervallo ampio di *RGB signal color thresholding*, c) risultati dell'approccio *patches* con intervallo stretto di *RGB signal color thresholding*, d) risultati dell'approccio *patches* con intervallo stretto di *RGB signal color thresholding* e utilizzo della tecnica *Circle Clustering* per la stima del BPM. In tutti i casi è stato utilizzato *skin color thresholding* stretto.

per gestire video non compressi. Future versioni di pyVHR si concentreranno anche su questo aspetto.

Oltre alla mediana, anche l'intervallo interquartile (IQR) (definito come la differenza tra il terzo e il primo quartile e che rappresenta la dimensione del *box*), che include il 50% dei dati centrali, fornisce spunti utili per la procedura di valutazione. Ispezionando gli IQR rappresentati nelle figure, vale la pena notare che in generale i metodi POS e CHROM forniscono una migliore mediana MAE, e una minore diffusione rispetto agli altri.

Possiamo inoltre dimostrare che utilizzare *RGB signal color thresholding* stretto o ampio è equivalente, se si utilizza *skin color thresholding* stretto. Le Tabelle 4.3 e 4.4 riportano per ogni Dataset (LGI-PPGI e PURE sono stati suddivisi in base alla tipologia di video) e per ogni metodo rPPG i valori medi della metrica MAE, il valore mediano e lo scarto interquartile; nella prima tabella è stato scelto *RGB signal color thresholding* stretto, mentre nell'altra ampio. Notiamo anche che i migliori metodi sono CHROM, POS. Oltre alla tecnica *patches* vengono riportati anche i risultati della tecnica *holistic*; nel caso di CHROM e LGI *patches* è simile ad *holistic*. Metodi come GREEN e PBV performano meglio utilizzando approccio *holistic*. Infine, LGI e PCA hanno risultati nettamente migliori (in media) con approccio *patches*. Infine le Tabelle 4.5 e 4.6 mostrano i risultati ottenuti utilizzando parametri simili, ma con *skin color thresholding* ampio. E' difficile definire in modo assoluto quale sia la configurazione migliore da utilizzare per *RGB signal color thresholding* e *skin color thresholding*; per quanto riguarda CHROM e POS con approccio *patches* è preferibile l'utilizzo di *RGB signal color thresholding* stretto e *skin color thresholding* stretto.

E' importante ricordare che le Tabelle 4.3, 4.4, 4.5 e 4.6 utilizzano la tecnica basata su Mediana per aggregare più BPM stimati (numero di *patches* > 1) in un unico BPM.

In generale modificare la configurazione dei parametri, utilizzando tecnica *patches*, impatta molto sui risultati ottenuti; per questo motivo, servirebbe definire una configurazione dei parametri *ad-hoc* per ogni Dataset. Nel caso dell'approccio olistico questo non è sempre vero.

Una valutazione più rigorosa e statisticamente valida rispetto alla differenza delle mediane tra i metodi è lasciata alla prossima analisi attraverso i *Significance Tests*.

Come spiegato nella sezione precedente di *Significance Testing*, pyVHR effettua automaticamente il test di *Friedman* (FT) o il test post-hoc *Nemenyi*, se viene rifiutata l'ipotesi nulla di *Friedman* di uguaglianza delle mediane dei campioni.

Per quanto riguarda il FT, i *p*-values = $9 \cdot 10^{-6}$ calcolati tramite la statistica χ^2 , suggerisce fortemente l'esistenza di differenze significative tra gli algoritmi considerati. Per i tests Nemenyi sono forniti i risultati con valori critici al 95%.

Utilizzando i risultati mostrati nelle Figure 4.1 e 4.2 possiamo ottenere l’analisi statistica mostrata in Tabella 4.7 per tutti i 4 Datasets e per ogni metodo rPPG; è importante precisare che questi risultati riguardano solo l’approccio *patches*.

Sorprendentemente, si scopre che le prestazioni raggiunte dai quattro metodi migliori, vale a dire POS, CHROM, PCA e LGI, sono non significativamente diversi dal punto di vista statistico. Utilizzando tre diversi livelli di significato, vale a dire $\alpha \in 0,05, 0,01, 0,001$, le figure di Tabella 4.7 rappresentano nella forma di *heatmaps* le varie ipotesi rifiutate/accettate. In particolare, le *heatmaps* raccolgono una famiglia di 21 ipotesi (tutte le coppie di algoritmi) evidenziando quali algoritmi hanno miglioramenti rispetto agli altri, ad ogni dato livello di significatività. La heatmap rappresenta differenti livelli di significance per la metrica MAE; le *p*-values significative sono colorate in blue, mentre quelle non significative in rosso. In generale osserviamo per gli 8 metodi rPPG differenze sostanziali.

Osservando i risultati del Dataset LGI-PPGI, vediamo che c’è differenza significativa tra i metodi LGI e GREEN, GREEN e CHORM, LGI e PBV ($p < 0,001$). Invece non abbiamo differenza significativa tra CHROM e LGI, CHROM e POS, POS e LGI.

	CHROM	GREEN	ICA	LGI	PBV	PCA	POS
LGI-PPGI gym (hol.)	12.809922	21.737579	28.285880	10.912366	27.704324	11.975057	12.599763
LGI-PPGI gym (patch)	11.986661	33.334972	28.626246	10.422358	35.709941	11.233662	14.169911
LGI-PPGI resting (hol.)	1.777448	1.366271	25.453643	24.244002	9.256011	26.348773	2.587683
LGI-PPGI resting (patch)	1.665666	1.715175	2.217767	1.331963	2.247652	9.759703	1.729802
LGI-PPGI rotation (hol.)	2.498833	5.599749	17.979064	9.761490	10.021431	15.646901	3.394135
LGI-PPGI rotation (patch)	3.989115	7.950017	5.307606	3.825792	8.147088	8.624730	5.558395
LGI-PPGI talk (hol.)	8.108997	15.442641	14.422577	8.003216	15.300019	12.590261	9.962678
LGI-PPGI talk (patch)	9.686265	15.272165	10.849713	7.973409	14.188426	9.806313	11.706517
PURE fast t. (hol.)	0.827562	1.748665	19.397632	19.049602	6.805171	20.201345	0.822364
PURE fast t. (patch)	0.812482	4.324306	5.341223	3.714807	6.828197	5.329400	0.783981
PURE medium r. (hol.)	0.885246	5.699120	12.400026	5.753562	10.730639	9.483734	0.800287
PURE medium r. (patch)	1.021778	11.675221	6.556333	1.711916	12.493463	1.104949	1.515995
PURE slow t. (hol.)	0.688093	0.912934	25.863714	28.593960	7.513829	27.595850	0.712164
PURE slow t. (patch)	0.695018	1.992217	6.396230	2.422921	5.226833	5.670052	0.699210
PURE small r. (hol.)	0.770327	1.914160	13.219702	10.082605	6.632653	13.997143	0.787334
PURE small r. (patch)	0.803416	9.328225	5.870738	1.283932	11.023178	0.989979	0.871118
PURE steady (hol.)	1.065449	3.619029	37.665731	41.302661	9.546569	40.948835	0.964482
PURE steady (patch)	1.149150	5.479150	12.470675	12.831994	10.180551	11.552854	0.955969
PURE talking (hol.)	1.913529	10.464509	13.323529	4.809922	15.504448	9.375506	2.038446
PURE talking (patch)	2.032750	11.228913	7.394940	2.499399	12.989873	2.608175	2.806553
UBFC1 (hol.)	0.969232	4.165661	17.463350	6.597866	11.147240	22.720535	1.216783
UBFC1 (patch)	0.917828	4.453929	1.961075	0.771057	6.100001	5.558355	1.164851
UBFC2 (hol.)	1.531705	3.747272	20.771576	9.800684	12.218104	20.541943	1.944539
UBFC2 (patch)	1.614967	5.238829	9.867236	2.590036	9.805470	3.550622	2.339647
Median	1.340428	5.358990	12.845188	7.285638	10.100991	10.519988	1.622898
IQR	1.278445	7.443285	13.224811	7.977483	5.275144	11.143384	2.094519

Tabella 4.3: Per ogni Dataset e metodo rPPG e tecnica ("patch" indica *patches*, "hol." indica *holistic*): valori medi della metrica MAE, valore mediano (*Median*) e scarto interquartile (*IQR*). I parametri utilizzati sono elencati all'inizio della sezione *Risultati multi-dataset*; in particolare è stato utilizzato *skin color thresholding* stretto, e *RGB signal color thresholding* stretto.

	CHROM	GREEN	ICA	LGI	PBV	PCA	POS
LGI-PPGI gym (hol.)	12.809922	21.737579	28.285880	10.912366	27.704324	11.975057	12.599763
LGI-PPGI gym (patch)	11.986661	33.334972	28.626246	10.422358	35.709941	11.233662	14.169911
LGI-PPGI resting (hol.)	1.777448	1.366271	25.453643	24.244002	9.256011	26.348773	2.587683
LGI-PPGI resting (patch)	1.665666	1.715175	2.217767	1.331963	2.247652	9.759703	1.729802
LGI-PPGI rotation (hol.)	2.498833	5.599749	17.979064	9.761490	10.021431	15.646901	3.394135
LGI-PPGI rotation (patch)	3.989115	7.950017	5.307606	3.825792	8.147088	8.624730	5.558395
LGI-PPGI talk (hol.)	8.108997	15.442641	14.422577	8.003216	15.300019	12.590261	9.962678
LGI-PPGI talk (patch)	9.686265	15.272165	10.849713	7.973409	14.188426	9.806313	11.706517
PURE fast t. (hol.)	0.827562	1.748665	19.397632	19.049602	6.805171	20.201345	0.822364
PURE fast t. (patch)	0.809830	4.324905	5.325997	3.626776	6.828225	5.283965	0.784637
PURE medium r. (hol.)	0.885246	5.699120	12.400026	5.753562	10.730639	9.483734	0.800287
PURE medium r. (patch)	1.015716	11.675979	6.551787	1.715705	12.500282	1.112526	1.509933
PURE slow t. (hol.)	0.688093	0.912934	25.863714	28.593960	7.513829	27.595850	0.712164
PURE slow t. (patch)	0.698895	1.999325	6.402047	2.430030	5.228771	5.662297	0.698564
PURE small r. (hol.)	0.770327	1.914160	13.219702	10.082605	6.632653	13.997143	0.787334
PURE small r. (patch)	0.803416	9.328225	5.870738	1.283932	11.023178	0.989979	0.871118
PURE steady (hol.)	1.065449	3.619029	37.665731	41.302661	9.546569	40.948835	0.964482
PURE steady (patch)	1.149907	5.483619	12.457820	12.735730	10.173719	11.558967	0.956726
PURE talking (hol.)	1.913529	10.464509	13.323529	4.809922	15.504448	9.375506	2.038446
PURE talking (patch)	2.031922	11.233051	7.401561	2.499399	12.985735	2.607389	2.807380
UBFC1 (hol.)	0.969232	4.165661	17.463350	6.597866	11.147240	22.720535	1.216783
UBFC1 (patch)	0.917828	4.453929	1.961075	0.771057	6.100001	5.558355	1.164851
UBFC2 (hol.)	1.531705	3.747272	20.771576	9.800684	12.218104	20.541943	1.944539
UBFC2 (patch)	1.614967	5.238829	9.867236	2.590036	9.805470	3.550622	2.339647
Median	1.340806	5.361224	12.838761	7.285638	10.097575	10.519988	1.619868
IQR	1.277825	7.442542	13.226766	7.977483	5.279217	11.149201	2.095140

Tabella 4.4: Per ogni Dataset e metodo rPPG e tecnica ("patch" indica *patches*, "hol." indica *holistic*): valori medi della metrica MAE, valore mediano (*Median*) e scarto interquartile (*IQR*). I parametri utilizzati sono elencati all'inizio della sezione *Risultati multi-dataset*; in particolare è stato utilizzato *skin color thresholding* stretto, e *RGB signal color thresholding* ampio.

	CHROM	GREEN	ICA	LGI	PBV	PCA	POS
LGI-PPGI gym (hol.)	14.621087	28.904626	29.477524	11.387493	33.882305	11.675843	12.218069
LGI-PPGI gym (patch)	13.399292	31.874553	25.571766	11.230834	34.317137	11.545140	13.392421
LGI-PPGI resting (hol.)	1.764976	3.166889	18.712970	2.003287	13.165558	24.153952	2.492878
LGI-PPGI resting (patch)	1.696792	2.539719	1.981082	1.276145	6.058889	2.381537	1.773525
LGI-PPGI rotation (hol.)	3.146319	12.077515	19.504569	3.855361	20.742667	11.814494	4.264294
LGI-PPGI rotation (patch)	4.406816	8.608769	5.113319	4.153643	10.148918	5.980780	5.560685
LGI-PPGI talk (hol.)	7.626646	20.564306	14.260513	7.483125	18.782693	7.628923	9.575632
LGI-PPGI talk (patch)	13.163387	18.535613	14.764290	12.483122	17.887221	13.220841	15.079204
PURE fast t. (hol.)	0.850781	7.285391	18.698029	1.911707	15.832414	5.289430	0.857216
PURE fast t. (patch)	3.702371	9.865417	7.838512	4.427193	11.545092	3.707508	3.678238
PURE medium r. (hol.)	0.840771	11.387703	13.022707	1.258269	16.294558	2.330364	0.952826
PURE medium r. (patch)	1.304903	12.958737	7.651596	1.962357	13.795387	1.398817	1.224239
PURE slow t. (hol.)	0.716696	7.856740	21.902274	1.457649	15.824392	9.214610	0.731734
PURE slow t. (patch)	0.721532	6.192048	4.559477	0.808944	9.271787	0.713628	0.727396
PURE small r. (hol.)	0.806603	11.309883	18.511786	1.329977	19.039159	3.421984	1.003899
PURE small r. (patch)	0.884263	11.835039	6.144453	1.071109	12.855937	0.878133	0.867834
PURE steady (hol.)	1.064085	11.438588	22.354772	4.755855	20.380452	8.292966	0.959683
PURE steady (patch)	3.142293	12.512906	11.582345	6.881433	15.464766	4.372116	3.021552
PURE talking (hol.)	2.098170	21.720565	16.825935	2.338772	23.883277	2.163719	2.653271
PURE talking (patch)	3.315797	14.114727	9.220439	3.720407	15.678009	3.377794	4.192828
UBFC1 (hol.)	1.118830	13.636998	12.195961	1.052470	22.038820	14.800978	1.214376
UBFC1 (patch)	0.969933	5.814686	1.905758	0.997650	7.620606	1.638191	1.167986
UBFC2 (hol.)	1.716308	22.171307	17.533615	2.084463	26.337644	7.626980	1.909688
UBFC2 (patch)	1.607218	14.928749	6.261933	2.793062	16.924381	2.408825	1.994767
Median	1.706550	11.956277	13.641610	2.211617	16.063486	4.830773	1.952228
IQR	2.463925	7.409703	11.397584	3.192839	7.382853	7.428498	3.217849

Tabella 4.5: Per ogni Dataset e metodo rPPG e tecnica ("patch" per *patches*, "hol." per *holistic*): valori medi della metrica MAE, valore mediano (*Median*) e scarto interquartile (*IQR*). I parametri utilizzati sono elencati all'inizio della sezione *Risultati multi-dataset*; in particolare è stato utilizzato *skin color thresholding* ampio, e *RGB signal color thresholding* stretto.

	CHROM	GREEN	ICA	LGI	PBV	PCA	POS
LGI-PPGI gym (hol.)	14.621087	28.904626	29.477524	11.387493	33.882305	11.675843	12.218069
LGI-PPGI gym (patch)	13.574479	32.558479	25.795601	11.540049	34.352927	11.611071	14.547109
LGI-PPGI resting (hol.)	1.764976	3.166889	18.712970	2.003287	13.165558	24.153952	2.492878
LGI-PPGI resting (patch)	1.666911	2.938686	2.041458	1.307682	6.872028	2.016762	1.766962
LGI-PPGI rotation (hol.)	3.146319	12.077515	19.504569	3.855361	20.742667	11.814494	4.264294
LGI-PPGI rotation (patch)	5.842507	7.748601	5.454982	5.791391	10.064145	7.230676	5.437179
LGI-PPGI talk (hol.)	7.626646	20.564306	14.260513	7.483125	18.782693	7.628923	9.575632
LGI-PPGI talk (patch)	8.985932	14.898862	10.941128	8.521744	14.180422	9.100645	11.625304
PURE fast t. (hol.)	0.850781	7.285391	18.698029	1.911707	15.832414	5.289430	0.857216
PURE fast t. (patch)	0.829141	8.908080	5.993486	1.596510	11.047927	0.856593	0.806013
PURE medium r. (hol.)	0.840771	11.387703	13.022707	1.258269	16.294558	2.330364	0.952826
PURE medium r. (patch)	1.386486	14.296532	8.214278	1.943029	14.579213	1.373544	2.236141
PURE slow t. (hol.)	0.716696	7.856740	21.902274	1.457649	15.824392	9.214610	0.731734
PURE slow t. (patch)	0.709170	8.886440	5.867564	0.929131	12.386381	0.709739	0.716935
PURE small r. (hol.)	0.806603	11.309883	18.511786	1.329977	19.039159	3.421984	1.003899
PURE small r. (patch)	1.095032	13.052317	7.297731	1.473751	13.633859	1.087315	0.948747
PURE steady (hol.)	1.064085	11.438588	22.354772	4.755855	20.380452	8.292966	0.959683
PURE steady (patch)	1.212113	13.535206	10.791854	5.746766	15.716801	1.241356	0.947723
PURE talking (hol.)	2.098170	21.720565	16.825935	2.338772	23.883277	2.163719	2.653271
PURE talking (patch)	2.230870	14.090533	8.536934	2.850772	15.383559	2.266422	3.013698
UBFC1 (hol.)	1.118830	13.636998	12.195961	1.052470	22.038820	14.800978	1.214376
UBFC1 (patch)	0.972691	5.855151	1.718258	0.950507	7.402685	1.403707	1.178404
UBFC2 (hol.)	1.716308	22.171307	17.533615	2.084463	26.337644	7.626980	1.909688
UBFC2 (patch)	1.911564	15.796736	6.268049	3.385721	17.295189	2.455842	2.330209
Median	1.526698	12.564916	12.609334	2.043875	15.828403	4.355707	1.838325
IQR	1.517519	6.494315	11.661454	3.577852	6.954222	7.265638	2.374541

Tabella 4.6: Per ogni Dataset, metodo rPPG e tecnica ("patch" indica *patches*, "hol." indica *holistic*): valori medi della metrica MAE, valore mediano (*Median*) e scarto interquartile (*IQR*). I parametri utilizzati sono elencati all'inizio della sezione *Risultati multi-dataset*; in particolare è stato utilizzato *skin color thresholding* ampio, e *RGB signal color thresholding* ampio.

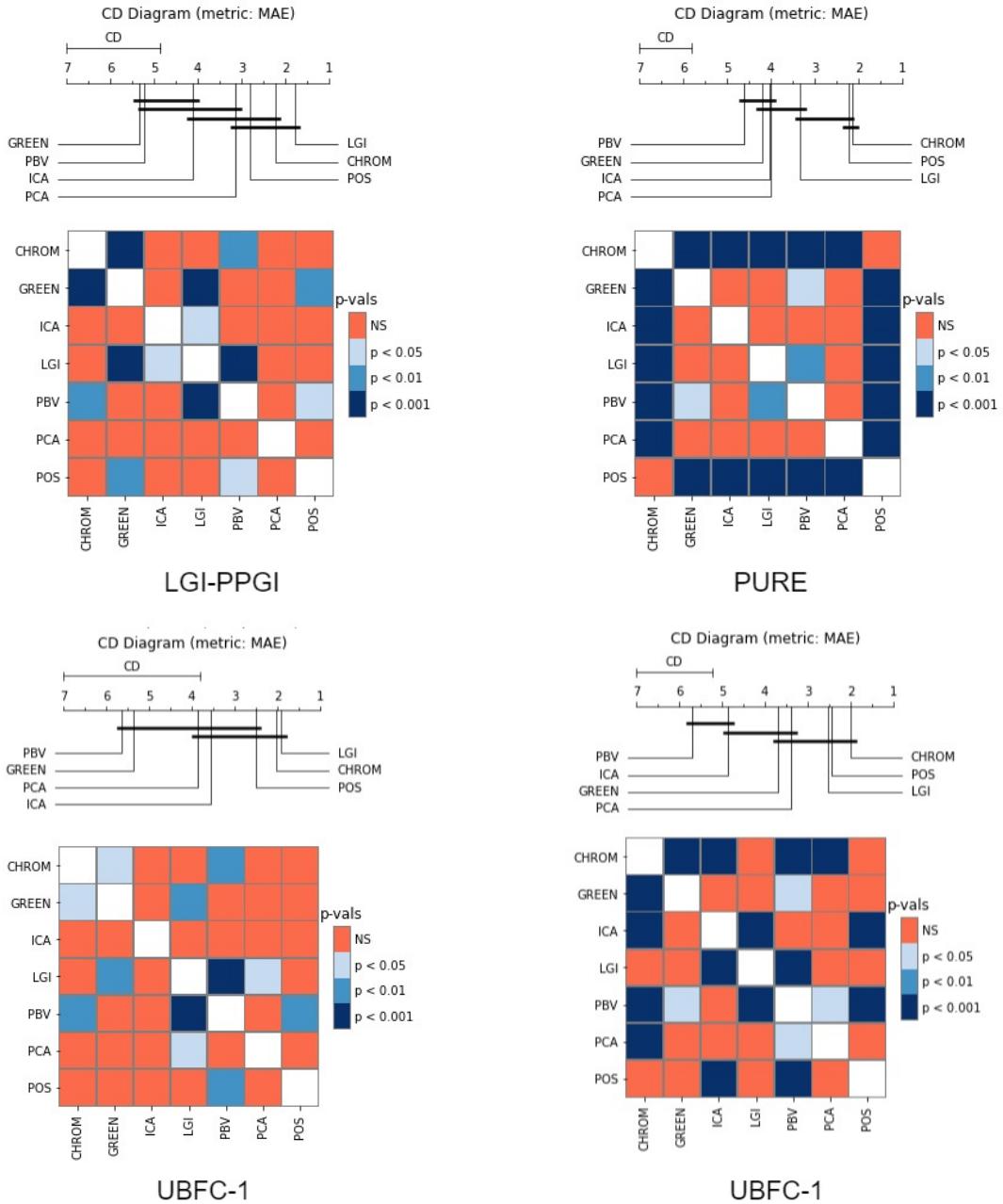


Tabella 4.7: Per ogni Dataset e metodo rPPG: diagramma delle differenze critiche (CD) e *heatmaps*. Per le CD viene utilizzata la metrica MAE e vengono collegati gruppi di metodi che non sono significativamente diversi (a $p = 0,05$). La heatmap rappresenta differenti livelli di *significance* per la metrica MAE; le p -values significative sono colorate in blu, mentre quelle non significative in rosso (NS).

4.2 Heat maps delle Patches

Avendo a disposizione 468 *landmarks* facciali è importante sapere quali di questi è più affidabile per stimare correttamente il BPM. Di seguito è descritto un possibile approccio per rispondere a questo dubbio.

Per prima cosa analizziamo i Datasets UBFC, PURE e LGI-PPGI con approccio *Patches* per ognuno dei 468 *landmarks*. La dimensione della *patch* dipende dalla larghezza del volto analizzato; per questi Datasets sono state utilizzate *patches* quadrate con lato di 30 pixels (dimensioni maggiori fino a 50 pixels hanno dato risultati analoghi). Il metodo rPPG scelto è CHROM senza l'utilizzo di alcun pre post filtering.

A questo punto selezioniamo un sottoinsieme equi spaziato dei 468 *landmarks*. Per ogni metrica a nostra disposizione (MAE, RMSE, MAX) creiamo una mappa di colore in cui associamo a colori caldi (rosso scuro) errori bassi, e a colori freddi (azzurro chiaro) errori alti.

Le Figure 4.2, 4.3, 4.4 rappresentano le Heat Maps delle metriche MAE, RMSE e MAX sui Dataset UBFC, PURE e LGI-PPGI. Come si può notare i punti che forniscono una stima rPPG più precisa sono quelli del centro della fronte e delle guance. Questo risultato è interessante, perché conferma che la maggior parte dell'informazione è contenuta nelle zone di viso più illuminate e piatte. La metrica PCC non è molto utile per realizzare questo tipo di mappe colore, in quanto, da sola non è significativa (è possibile avere PCC alto anche con MAE alto).

Conoscendo i migliori *landmarks* per le metriche MAE, RMSE, MAX e PCC, possiamo calcolarne l'intersezione e ottenere i migliori 35 *landmarks* del volto. Questi sono rappresentati nella Figura 4.1. In questo caso non stiamo considerando un sottoinsieme equi spaziato di *landmarks*, malgrado ciò notiamo che questi sono posizionati nelle guance e nel centro della fronte come nelle Heat Maps.

Utilizzare i *migliori landmarks* di un Dataset per analizzarne un altro non ha portato risultati migliori rispetto ad utilizzare 100 *landmarks* equi-spaziati sul volto. Questo è principalmente dovuto al fatto che i video di un Dataset sono molto eterogenei: i soggetti sono illuminati in parti differenti del volto, effettuano varie rotazioni della testa, e non hanno la stessa acconciatura di capelli e/o peluria facciale. Inoltre le analisi condotte hanno dimostrato che è sempre preferibile utilizzare un grande numero di stimatori (le *patch*) in modo da rendere statisticamente più efficace la Mediana utilizzata in equazione 1.5.

4.3 Caso di studio: rilevare DeepFake con pyVHR

DeepFakes è un insieme di tecniche basate sul deep learning che permettono di creare video falsi scambiando il volto di una persona con quello di un'altra. Questa tecnologia ha molte applicazioni diverse come la ricostruzione di espressioni (56) o la *de-identificazione* del volto (57). Negli ultimi anni la qualità dei *deepfake* ha raggiunto enormi livelli di realismo, ponendo così una serie di problemi legati alla possibilità

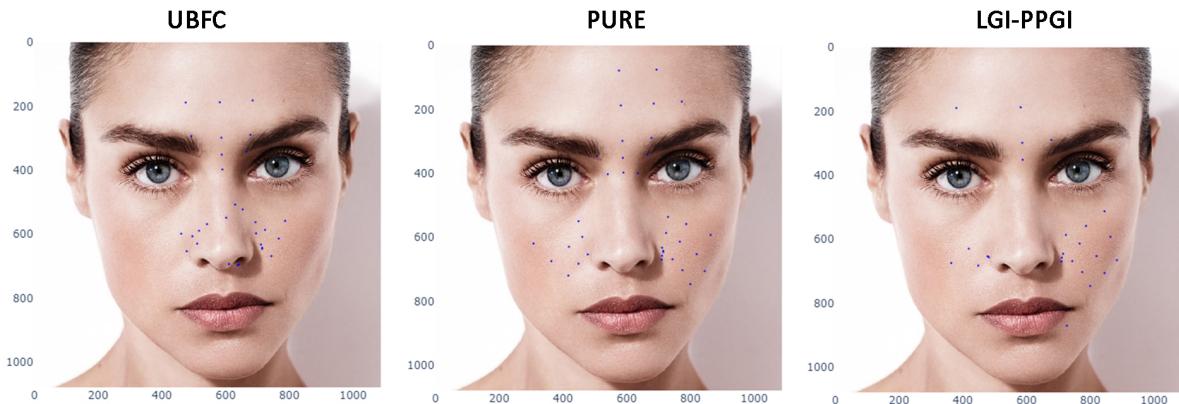


Figura 4.1: Intersezione dei migliori *landmarks*

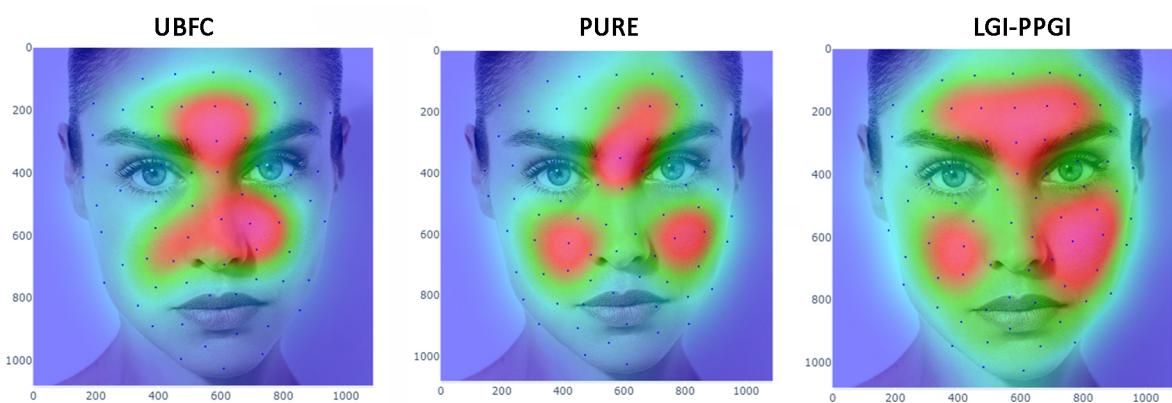


Figura 4.2: Heat Map - metrica MAE. Per ogni Dataset (UBFC, PURE, LGI-PPGI) è presente la mappa colore in cui l'errore della stima rPPG di *patches* equi spaziati è associato a un colore caldo (errore basso) o a un colore freddo (errore alto).

di manipolazione arbitraria dell'identità, come la propaganda politica, il ricatto e la creazione di fake news.

Di conseguenza, sono stati dedicati vari sforzi nello studio e sviluppo di metodi che permettono di discriminare tra video reali e falsi (58, 59). È interessante notare che esistono metodi molto efficaci basati sulle informazioni fisiologiche (60, 61, 62), infatti, ci si aspetta che i segnali provenienti da azioni biologiche come il battito cardiaco, il flusso sanguigno o la respirazione siano (in gran parte) disturbati dopo il face-swapping. Pertanto, metodi come il PPG remoto possono essere adattati per valutare la presenza di deep fakes.

Di seguito, viene mostrato come *pyVHR* possa essere efficacemente impiegato per eseguire facilmente un compito di rilevamento DeepFake. A tal fine, ci basiamo sul dataset FaceForensics++¹ (63) costituito da 1000 sequenze video originali (per lo più viso frontale senza occlusioni) che sono state manipolate con quattro metodi automatici di manipolazione del viso.

¹Disponibile su: <https://github.com/ondyari/FaceForensics>.

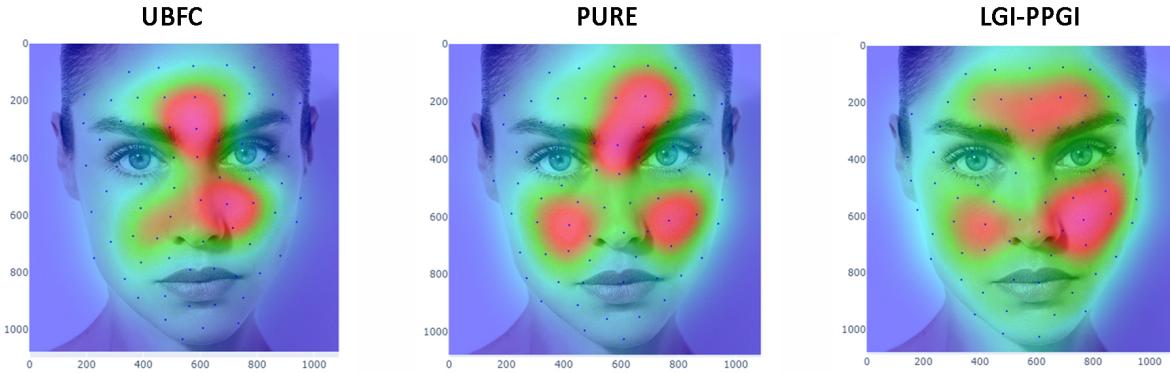


Figura 4.3: **Heat Map - metrica RMSE.** Per ogni Dataset (UBFC, PURE, LGI-PPGI) è presente la mappa colore in cui l'errore della stima rPPG di *patches* equi spaziati è associato a un colore caldo (errore basso) o a un colore freddo (errore alto).

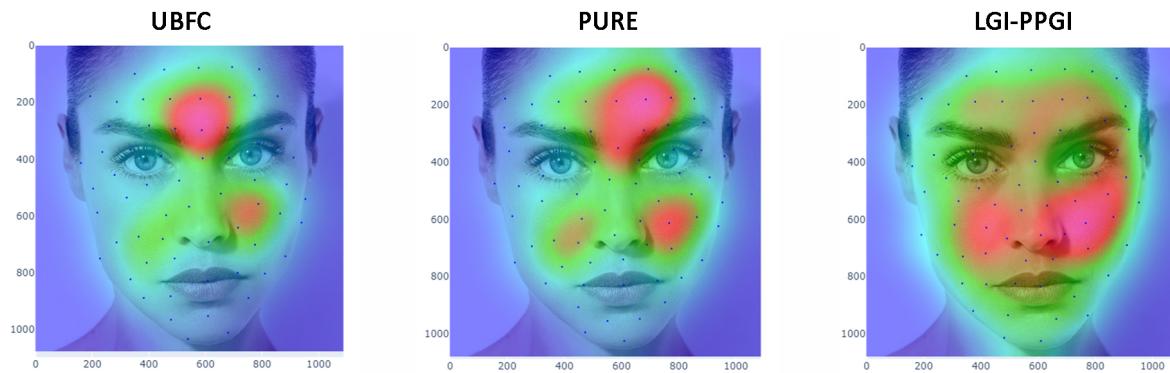


Figura 4.4: **Heat Map - metrica MAX.** Per ogni Dataset (UBFC, PURE, LGI-PPGI) è presente la mappa colore in cui l'errore della stima rPPG di *patches* equi spaziati è associato a un colore caldo (errore basso) o a un colore freddo (errore alto).

Ogni video, sia originale che scambiato, viene fornito come input alla pipeline di *pyVHR*. È ragionevole immaginare che i segnali BVP stimati sui video originali avrebbero una complessità molto più bassa rispetto a quelli scambiati, a causa della maggiore presenza di informazioni relative alla PPG che verrebbero eventualmente escluse durante le procedure di scambio. Di conseguenza, i segnali BVP dei Deep-Fake presenterebbero forse livelli più alti di rumore e quindi un comportamento più complesso.

Esistono molti modi per misurare la complessità di un segnale; qui abbiamo scelto di calcolare la dimensione frattale (FD) dei BVP; in particolare viene impiegato il metodo di Katz (64).

L'FD del BVP stimato dalla i -esima patch sulla k -esima finestra temporale (D_i^k) può essere calcolato come (64):

$$D_i^k = \frac{\log_{10}(L/a)}{\log_{10}(d/a)},$$

dove L è la somma delle distanze tra punti successivi, a è la loro media, e d è la distanza massima tra il primo punto e qualsiasi altro punto del segnale BVP stimato. L'FD associato a un dato video può quindi essere ottenuto tramite la media:

$$\hat{FD}_{vid} = \frac{1}{PK} \sum_{i=0}^P \sum_{k=0}^K D_i^k.$$

Allo stesso modo, si potrebbe considerare l'adozione della *Median absolute deviation* (MAD) delle stime BPM su un video come indice della presenza di DeepFake:

$$\hat{MAD}_{vid} = \frac{1}{K} \sum_{k=0}^K MAD^k.$$

La figura 4.5 mostra come si distribuiscono i video *real* e *fake* (metodo *FaceShifter*) di FaceForensics++ nello spazio 2-dimensionale definito dalla dimensione frattale media (\hat{FD}) dei BVP stimati e dalle MAD medie dei BPM (\hat{MAD}).

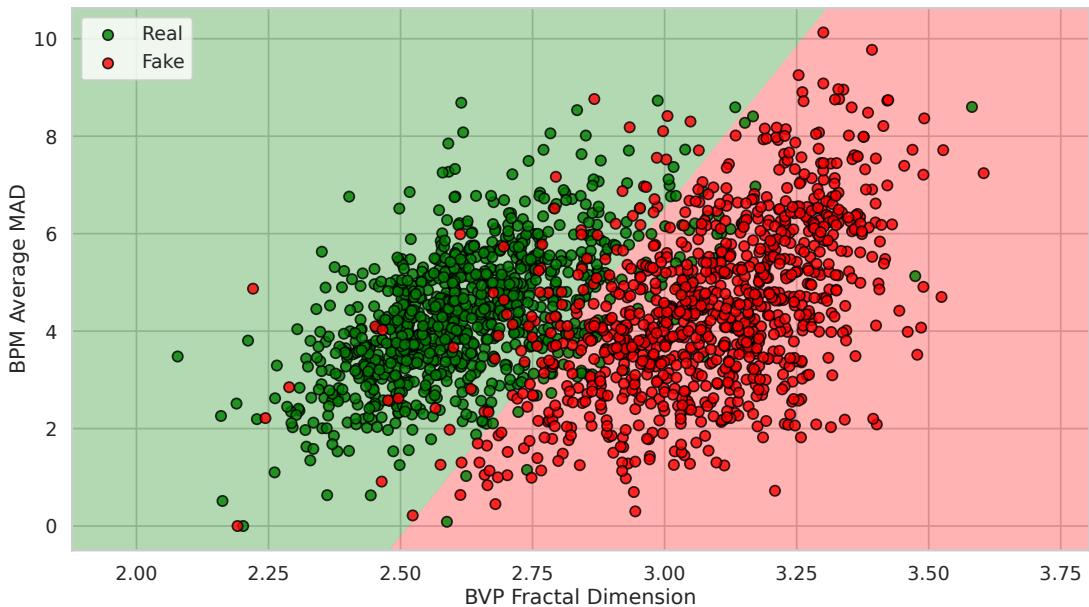


Figura 4.5: Risultati del rilevamento di deepfake. I 1000 video originali di FaceForensics++ (verde) e le loro versioni scambiate (rosso) rappresentate nello spazio 2-D della dimensione frattale dei BVP vs. BPM MAD medi. Gli spazi verde e rosso sono semplicemente appresi tramite una SVM lineare.

È facile vedere come l'adozione di queste semplici statistiche sulle previsioni di *py-VHR* permetta di discriminare i video originali dai DeepFake. In particolare, utilizzando la SVM lineare mostrata in Figura 4.5 si ottiene una *Accuracy* di $91.41\% \pm 2.05$ con tecnica 10-fold Cross-Validation. Questo risultato è paragonabile allo stato dell'arte degli approcci che di solito adottano soluzioni molto più complesse.

4.4 Conclusioni e sviluppi futuri

Nel contesto degli algoritmi della remote Photoplethysmography (rPPG) non sono noti al momento framework open-source completi per creare, analizzare e comparare i vari algoritmi di stima del battito cardiaco basati sul segnale rPPG. Per questo motivo l’obiettivo alla base di questo lavoro di tesi è duplice: il primo avente finalità di ridisegno di un framework software dato in forma preliminare (5), il secondo di studio e sviluppo di tecniche statistiche e di clustering per rendere più efficienti metodi tradizionali di rPPG. Nella prima fase, causa limiti manifesti del prototipo iniziale, si è reso necessario creare una nuova versione del framework pyVHR (An Open Framework for Remote-PPG Methods and Their Assessment) puntando a una reingegnerizzazione completa del codice avendo cura di:

1. rendere più semplice manutenzione ed estensione del framework. In particolare, la modularità che caratterizza il framework e l’implementazione in Python dovrebbero favorire il suo utilizzo, future estensioni, e la possibilità di costruire applicazioni *custom* a partire da pyVHR.
2. Dotare il framework di capacità real-time mediante accelerazione di procedure di signal e video processing particolarmente onerose con implementazione su GPU NVIDIA assieme a parziale parallelizzazione di parti del codice anche su CPU.
3. Sviluppare nuove tecniche di pre- e post-processing (tipicamente filtri standard e ad-hoc) per migliorare la stima del battito cardiaco con approcci rPPG tradizionali.
4. Viene affrontato in modo efficiente il problema della segmentazione della pelle del volto con una tecnica innovativa basata su una rete neurale in grado – non solo di segmentare mediante detection del volto – ma anche di fornire un mesh di 486 landmark fedelmente tracciati nel tempo, un frame dopo l’altro.
5. Sviluppo di una tecnica basata su convex hull (inviluppo convesso) sui landmark per isolare dai frame del video i pixel associati alla pelle del volto.
6. In alternativa alla tecnica convex hull, in pyVHR si propone una segmentazione semantica del volto in regioni ben determinate (per es. fronte, naso, guance, bocca, etc) basata su rete neurale BiSeNet.
7. Viene introdotta una visione dicotomica del monitoraggio remoto della frequenza cardiaca, che porta a due distinte classi di approccio: tecniche tradizionale e tecniche di deep-learning.

8. Per i metodi di deep learning viene fornita una pipeline distinta dalla pipe per i metodi tradizionali (già presente nel framework originale e qui migliorata dalle tecniche innovative descritte) capace di integrare un tipico processo end-to-end delle architetture deep (cioè un procedimento che dal video raw porta alla stima del battito) con gli stadi della pipe destinati all'analisi comparativa finale.
9. Per le tecniche tradizionali è presente un'ulteriore distinzione riguardo le regioni di interesse (ROI) prese in considerazione: viene distinto il concetto di approccio olistico, che prende in esame l'intero volto del soggetto, da quello basato sul campionamento del volto attraverso molteplici patch (regioni rettangolari centrate in punti specifici del volto detti landmark) distribuite in modo quasi uniforme sul volto e tracciate con coerenza nel tempo.
10. Il framework consente facilmente di creare, analizzare e confrontare statisticamente gli algoritmi rPPG tradizionali (e/o le tecniche rPPG basate su deep-learning).

La seconda fase, più marcatamente di ricerca, è stata rivolta allo studio di tecniche di campionamento facciale mediante uso massivo di landmark uniformemente distribuiti sul volto per migliorare le stime del battito cardiaco impiegando semplici statistiche e nuovi metodi di clustering applicati a patch individuate e tracciate da un mesh di landmark computato efficientemente. In particolare, si è inteso:

1. incrementare il livello di confidenza delle stime rPPG prodotte da molteplici stimatori derivanti da patch usando statistiche standard come la mediana sugli spettri di potenza ottenuti dalle singole patch;
2. filtrare le numerose stime spettrali mediante tecniche di clustering disegnate ad hoc per ricavarne una separazione in due cluster: uno contenente spettri tra loro massimamente coerenti (massima similarità), l'altro contenente spettri tra loro incoerenti e dissimili da quelli del primo cluster (massima dissimilarità).

I risultati ottenuti hanno portato alla scrittura di un articolo su rivista (65).

Per quanto riguarda gli sviluppi futuri, è sicuramente una sfida aperta quella di sviluppare tecniche rPPG robuste per video compressi. Questo vale indifferentemente per approcci basati su deep learning come quelli basati su metodi tradizionali. Puntare a comprendere quali siano i parametri di compressione più importanti che assieme a ridurre il bitrate aiutino a preservare le informazioni più importanti per la stima del battito potrebbe essere una tattica di indagine, perché al momento per video compressi con i codec più diffusi ad elevato tasso di compressione non si ottengono prestazioni apprezzabili.

Sono aperti anche molti degli altri punti visti nello sviluppo della pipeline di stima che è stata mostrata nei capitoli precedenti. Sicuramente pre e post processing giocano anch'essi un ruolo importante. Il pre filtraggio, ad esempio, richiede modellazione più specifica e capace di rimuovere il rumore più insidioso che proviene non solo dalla degradazione video ma soprattutto dal movimento del corpo il quale confonde spesso le tecniche di stima. Il post filtraggio applicato al segnale BVP potrebbe operare un detrending per rendere i pattern del segnale più aderenti a segnali realistici, rimuovendone di fatto artefatti dannosi prodotti da molti dei metodi allo stato dell'arte.

Assieme a queste principali sfide si possono aggiungere anche operazioni di miglioramento generale che toccano punti come: l'aggiunta di nuovi metodi deep e non, lo studio di *Datasets* per studiare a fondo i metodi rPPG, accelerare con GPU o CPU altri moduli di pyVHR per ottenere prestazioni ancora più real time.

Bibliografia

- [1] Pierre Bouguer. *Essai d'optique sur la gradation de la lumière*. Claude Jombert, 1729.
- [2] Wim Verkruyse, Lars O Svaasand, and J Stuart Nelson. Remote plethysmographic imaging using ambient light. *Optics express*, 16(26):21434–21445, 2008.
- [3] Chihiro Takano and Yuji Ohta. Heart rate measurement based on a time-lapse image. *Medical Engineering & Physics*, 29(8):853–857, 2007. ISSN 1350-4533. doi: <https://doi.org/10.1016/j.medengphy.2006.09.006>.
- [4] Wenjin Wang, Albertus C den Brinker, Sander Stuijk, and Gerard de Haan. Algorithmic principles of remote ppg. *IEEE Transactions on Biomedical Engineering*, 64(7):1479–1491, 2016.
- [5] Giuseppe Boccignone, Donatello Conte, Vittorio Cuculo, Alessandro D’Amelio, Giuliano Grossi, and Raffaella Lanzarotti. An open framework for remote-ppg methods and their assessment. *IEEE Access*, 8:216083–216103, 2020.
- [6] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 325–341, 2018.
- [7] D. J. McDuff, J. R. Estepp, A. M. Piasecki, and E. B. Blackford. A survey of remote optical photoplethysmographic imaging methods. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 6398–6404, 2015.
- [8] Philipp V. Rouast, Marc T. P. Adam, Raymond Chiong, David Cornforth, and Ewa Lux. Remote heart rate measurement using low-cost rgb face video: a technical literature review. *Frontiers of Computer Science*, 12(5):858–872, 2018. doi: 10.1007/s11704-016-6243-6. URL <https://doi.org/10.1007/s11704-016-6243-6>.

- [9] Guillaume Heusch, André Anjos, and Sébastien Marcel. A reproducible study on remote heart rate measurement. *CoRR*, abs/1709.00962, 2017. URL <http://arxiv.org/abs/1709.00962>.
- [10] Anton M Unakafov. Pulse rate estimation using imaging photoplethysmography: generic framework and comparison of methods on a publicly available dataset. *Biomedical Physics & Engineering Express*, 4(4):045001, 2018.
- [11] Daniel McDuff and Ethan Blackford. iPhys: An open non-contact imaging-based physiological measurement toolbox. In *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 6521–6524. IEEE, 2019.
- [12] Sander Koelstra, Christian Muhl, Mohammad Soleymani, Jong-Seok Lee, Ashkan Yazdani, Touradj Ebrahimi, Thierry Pun, Anton Nijholt, and Ioannis Patras. Deap: A database for emotion analysis; using physiological signals. *IEEE transactions on affective computing*, 3(1):18–31, 2011.
- [13] Christian Pilz. On the vector space in photoplethysmography imaging. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.
- [14] Christian S Pilz, Sebastian Zaunseder, Jarek Krajewski, and Vladimir Blazek. Local group invariance for heart rate estimation from face videos in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1254–1262, 2018.
- [15] Wim Verkruyse, Lars O Svaasand, and J Stuart Nelson. Remote plethysmographic imaging using ambient light. *Opt. Express*, 16(26):21434–21445, 2008. doi: 10.1364/OE.16.021434.
- [16] Wenjin Wang, Sander Stuijk, and Gerard De Haan. A novel algorithm for remote photoplethysmography: Spatial subspace rotation. *IEEE transactions on biomedical engineering*, 63(9):1974–1984, 2015.
- [17] Gerard De Haan and Vincent Jeanne. Robust pulse rate from chrominance-based rppg. *IEEE Transactions on Biomedical Engineering*, 60(10):2878–2886, 2013.
- [18] Ming-Zher Poh, Daniel J McDuff, and Rosalind W Picard. Non-contact, automated cardiac pulse measurements using video imaging and blind source separation. *Optics express*, 18(10):10762–10774, 2010.

- [19] Guha Balakrishnan, Fredo Durand, and John Guttag. Detecting pulse from head motions in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3430–3437, 2013.
- [20] Zitong Yu, Xiaobai Li, and Guoying Zhao. Remote photoplethysmograph signal measurement from facial videos using spatio-temporal networks. *arXiv preprint arXiv:1905.02419*, 2019.
- [21] Xin Liu, Josh Fromm, Shwetak Patel, and Daniel McDuff. Multi-task temporal shift attention networks for on-device contactless vitals measurement. *arXiv preprint arXiv:2006.03790*, 2020.
- [22] Fabian Schrumpf, Patrick Frenzel, Christoph Aust, Georg Osterhoff, and Mirco Fuchs. Assessment of deep learning based blood pressure prediction from ppg and rppg signals. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3820–3830, 2021.
- [23] Xiaobai Li, Jie Chen, Guoying Zhao, and Matti Pietikäinen. Remote heart rate measurement from face videos under realistic situations. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 4264–4271, 2014. doi: 10.1109/CVPR.2014.543.
- [24] M.P. Tarvainen, P.O. Ranta-aho, and P.A. Karjalainen. An advanced detrending method with application to hrv analysis. *IEEE Transactions on Biomedical Engineering*, 49(2):172–175, 2002. doi: 10.1109/10.979357.
- [25] Maria Zontak and Michal Irani. Internal statistics of a single natural image. *CVPR 2011*, pages 977–984, 2011.
- [26] Serge Bobbia, Richard Macwan, Yannick Benezeth, Alamin Mansouri, and Julien Dubois. Unsupervised skin tissue segmentation for remote photoplethysmography. *Pattern Recognition Letters*, 124:82–90, 2019.
- [27] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweha, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.
- [28] G de Haan and A van Leest. Improved motion robustness of remote-PPG by using the blood volume pulse signature. *Physiological Measurement*, 35(9):1913–1926, aug 2014. doi: 10.1088/0967-3334/35/9/1913.
- [29] Magdalena Lewandowska, Jacek Rumiński, Tomasz Kocejko, and Jędrzej Nowak. Measuring pulse rate with a webcam - a non-contact method for evaluating

- cardiac activity. In *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 405–410, 2011.
- [30] Weixuan Chen and Daniel McDuff. Deepphys: Video-based physiological measurement using convolutional attention networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 349–365, 2018.
 - [31] Xuesong Niu, Shiguang Shan, Hu Han, and Xilin Chen. Rhythmnet: End-to-end heart rate estimation from face via spatial-temporal representation. *IEEE Transactions on Image Processing*, 29:2409–2423, 2019.
 - [32] Zitong Yu, Xiaobai Li, Xuesong Niu, Jingang Shi, and Guoying Zhao. Autohr: A strong end-to-end baseline for remote heart rate measurement with neural searching. *IEEE Signal Processing Letters*, 27:1245–1249, 2020.
 - [33] Zitong Yu, Yuming Shen, Jingang Shi, Hengshuang Zhao, Philip Torr, and Guoying Zhao. Physformer: Facial video-based physiological measurement with temporal difference transformer. *arXiv preprint arXiv:2111.12082*, 2021.
 - [34] John Gideon and Simon Stent. The way to my heart is through contrastive learning: Remote photoplethysmography from unlabelled video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3995–4004, 2021.
 - [35] Xin Liu, Ziheng Jiang, Josh Fromm, Xuhai Xu, Shwetak Patel, and Daniel McDuff. Metaphys: few-shot adaptation for non-contact physiological measurement. In *Proceedings of the Conference on Health, Inference, and Learning*, pages 154–163, 2021.
 - [36] Ewa Nowara, Daniel McDuff, and Ashok Veeraraghavan. The benefit of distraction: Denoising remote vitals measurements using inverse attention. *arXiv preprint arXiv:2010.07770*, 2020.
 - [37] Ronny Stricker, Steffen Müller, and Horst-Michael Gross. Non-contact video-based pulse rate measurement on a mobile service robot. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 1056–1062. IEEE, 2014.
 - [38] Mohammad Soleymani, Jeroen Lichtenauer, Thierry Pun, and Maja Pantic. A multimodal database for affect recognition and implicit tagging. *IEEE transactions on affective computing*, 3(1):42–55, 2011.
 - [39] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006. ISSN 1532-4435.

- [40] Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *The Journal of Machine Learning Research*, 18(1):2653–2688, 2017.
- [41] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528. IEEE, 2011.
- [42] Magdalena Graczyk, Tadeusz Lasota, Zbigniew Telec, and Bogdan Trawiński. Nonparametric statistical analysis of machine learning algorithms for regression problems. In Rossitza Setchi, Ivan Jordanov, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 111–120. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-15387-7.
- [43] Rob Eisinga, Tom Heskes, Ben Pelzer, and Manfred Te Grotenhuis. Exact p-values for pairwise comparison of friedman rank sums, with application to comparing classifiers. *BMC Bioinformatics*, 18(1):68, 2017. doi: 10.1186/s12859-017-1486-2.
- [44] I Lawrence and Kuei Lin. A concordance correlation coefficient to evaluate reproducibility. *Biometrics*, pages 255–268, 1989.
- [45] Xiaobai Li, Jie Chen, Guoying Zhao, and Matti Pietikainen. Remote heart rate measurement from face videos under realistic situations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4264–4271, 2014.
- [46] Sergey Tulyakov, Xavier Alameda-Pineda, Elisa Ricci, Lijun Yin, Jeffrey F Cohn, and Nicu Sebe. Self-adaptive matrix completion for heart rate estimation from face videos under realistic conditions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2396–2404, 2016.
- [47] Guillaume Heusch, André Anjos, and Sébastien Marcel. A reproducible study on remote heart rate measurement. *arXiv preprint arXiv:1709.00962*, 2017.
- [48] Rita Meziati Sabour, Yannick Benezeth, Pierre De Oliveira, Julien Chappe, and Fan Yang. Ubfc-phys: A multimodal database for psychophysiological studies of social stress. *IEEE Transactions on Affective Computing*, 2021.
- [49] Justin R Estepp, Ethan B Blackford, and Christopher M Meier. Recovering pulse rate during motion artifact with a multi-imager array for non-contact imaging photoplethysmography. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1462–1469. IEEE, 2014.
- [50] Zheng Zhang, Jeff M Girard, Yue Wu, Xing Zhang, Peng Liu, Umur Ciftci, Shaun Canavan, Michael Reale, Andy Horowitz, Huiyuan Yang, et al. Multimodal spon-

taneous emotion corpus for human behavior analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3438–3446, 2016.

- [51] Xiaobai Li, Iman Alikhani, Jingang Shi, Tapio Seppanen, Juhani Juntila, Kirsi Majamaa-Voltti, Mikko Tulppo, and Guoying Zhao. The obf database: A large face video database for remote physiological signal measurement and atrial fibrillation detection. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pages 242–249. IEEE, 2018.
- [52] Xuesong Niu, Hu Han, Shiguang Shan, and Xilin Chen. Vipl-hr: A multi-modal database for pulse estimation from less-constrained face video. In *Asian Conference on Computer Vision*, pages 562–576. Springer, 2018.
- [53] Xiaobai Li, Hu Han, Hao Lu, Xuesong Niu, Zitong Yu, Antitza Dantcheva, Guoying Zhao, and Shiguang Shan. The 1st challenge on remote physiological signal sensing (repss). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 314–315, 2020.
- [54] Radim Špetlík, Vojtech Franc, and Jirí Matas. Visual heart rate estimation with convolutional neural network. In *Proceedings of the british machine vision conference, Newcastle, UK*, pages 3–6, 2018.
- [55] Steffen Herbold. Autorank: A python package for automated ranking of classifiers. *Journal of Open Source Software*, 5(48):2173, 2020. doi: 10.21105/joss.02173. URL <https://doi.org/10.21105/joss.02173>.
- [56] Aayush Bansal, Shugao Ma, Deva Ramanan, and Yaser Sheikh. Recycle-gan: Unsupervised video retargeting. In *Proceedings of the European conference on computer vision (ECCV)*, pages 119–135, 2018.
- [57] Sathya Bursic, Alessandro D’Amelio, Marco Granato, Giuliano Grossi, and Rafaella Lanzarotti. A quantitative evaluation framework of video de-identification methods. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 6089–6095. IEEE, 2021.
- [58] Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales, and Javier Ortega-Garcia. Deepfakes and beyond: A survey of face manipulation and fake detection. *Information Fusion*, 64:131–148, 2020.
- [59] Yisroel Mirsky and Wenke Lee. The creation and detection of deepfakes: A survey. *ACM Computing Surveys (CSUR)*, 54(1):1–41, 2021.

- [60] Javier Hernandez-Ortega, Ruben Tolosana, Julian Fierrez, and Aythami Morales. Deepfakeson-phys: Deepfakes detection based on heart rate estimation. *arXiv preprint arXiv:2010.00400*, 2020.
- [61] Umur Aybars Ciftci, Ilke Demir, and Lijun Yin. How do the hearts of deep fakes beat? deep fake source detection via interpreting residuals with biological signals. In *2020 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–10. IEEE, 2020.
- [62] Hua Qi, Qing Guo, Felix Juefei-Xu, Xiaofei Xie, Lei Ma, Wei Feng, Yang Liu, and Jianjun Zhao. Deeprhythm: Exposing deepfakes with attentional visual heartbeat rhythms. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 4318–4327, 2020.
- [63] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. FaceForensics++: Learning to detect manipulated facial images. In *International Conference on Computer Vision (ICCV)*, 2019.
- [64] Michael J Katz. Fractals and the analysis of waveforms. *Computers in biology and medicine*, 18(3):145–156, 1988.
- [65] Giuseppe Boccignone, Donatello Conte, Vittorio Cuculo, Alessandro D’Amelio, Giuliano Grossi, Raffaella Lanzarotti, and Edoardo Mortara. Python framework for remote photoplethysmography. *PeerJ*, 2022. Submitted.