pyVHR: a Python framework for remote photoplethysmography

Giuseppe Boccignone, Donatello Conte, Vittorio Cuculo, Alessandro Amelio, Giuliano Grossi, Raffaella Lanzarotti, Edoardo Mortara

PeerJ Computer Science

pyVHR: a Python framework for remote photoplethysmography

Giuseppe Boccignone[1], Donatello Conte[2], Vittorio Cuculo[1], Alessandro D'Amelio[1], Giuliano Grossi[1], Raffaella Lanzarotti[1] and Edoardo Mortara[1]

[1] PHuSe Lab - Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy
[2] Laboratoire d'Informatique Fondamentale et Appliquée de Tours, Université de Tours, Tours, France

ABSTRACT

Remote photoplethysmography (rPPG) aspires to automatically estimate heart rate (HR) variability from videos in realistic environments. A number of effective methods relying on data-driven, model-based and statistical approaches have emerged in the past two decades. They exhibit increasing ability to estimate the blood volume pulse (BVP) signal upon which BPMs (Beats per Minute) can be estimated. Furthermore, learning-based rPPG methods have been recently proposed. The present pyVHR framework represents a multi-stage pipeline covering the whole process for extracting and analyzing HR fluctuations. It is designed for both theoretical studies and practical applications in contexts where wearable sensors are inconvenient to use. Namely, pyVHR supports either the development, assessment and statistical analysis of novel rPPG methods, either traditional or learning-based, or simply the sound comparison of well-established methods on multiple datasets. It is built up on accelerated Python libraries for video and signal processing as well as equipped with parallel/accelerated ad-hoc procedures paving the way to online processing on a GPU. The whole accelerated process can be safely run in real-time for 30 fps HD videos with an average speedup of around 5. This paper is shaped in the form of a gentle tutorial presentation of the framework.

Submitted 26 October 2021
Accepted 3 March 2022
Published 15 April 2022

Corresponding author
Alessandro D'Amelio,
alessandro.damelio@unimi.it

Academic editor
Carlos Fernandez-Lozano

Additional Information and Declarations can be found on page 32

DOI **10.7717/peerj-cs.929**

Subjects Human-Computer Interaction, Computer Vision
Keywords Remote photoplethysmography, Contactless monitoring, Deepfake Detection, Heart Rate Estimation, Deep rPPG

INTRODUCTION

Heart rate variability can be monitored *via* photoplethysmography (PPG), an optoelectronic measurement technology first introduced in *Hertzman (1937)*, and then largely adopted due to its reliability and non-invasiveness (*Blazek & Schultz-Ehrenburg, 1996*). Principally, this technique captures the amount of reflected light skin variations due to the blood volume changes.

Successively, remote-PPG (rPPG) has been introduced. This is a contactless technique able to measure reflected light skin variations by using an RGB-video camera as a virtual sensor (*Wieringa, Mastik & Steen, 2005*; *Humphreys, Ward & Markham, 2007*). Essentially, rPPG techniques leverage on the RGB color traces acquired over time and processed to approximate the PPG signal. As a matter of fact, rPPG has sparked great interest by

fostering the opportunity for measuring PPG at distance (*e.g.*, remote health assistance) or in all those cases where contact has to be prevented (*e.g.*, surveillance, fitness, health, emotion analysis) (*Aarts et al., 2013*; *McDuff, Gontarek & Picard, 2014*; *Ramírez et al., 2014*; *Boccignone et al., 2020b*; *Rouast et al., 2017*). Indeed, the rPPG research field has witnessed a growing number of techniques proposed for making this approach more and more robust and thus viable in contexts facing challenging problems such as subject motion, ambient light changes, low-cost cameras (*Lewandowska et al., 2011*; *Verkruysse, Svaasand & Nelson, 2008*; *Tarassenko et al., 2014*; *Benezeth et al., 2018*; *Wang et al., 2016*; *Wang, Stuijk & De Haan, 2015*; *Pilz et al., 2018*; *De Haan & Van Leest, 2014*). More recently, alongside the traditional methods listed above, rPPG approaches based on deep learning (DL) have burst into this research field (*Chen & McDuff, 2018*; *Niu et al., 2019*; *Yu et al., 2020*; *Liu et al., 2020*; *Liu et al., 2021*; *Gideon & Stent, 2021*; *Yu et al., 2021*).

The blossoming of the field and the variety of the proposed solutions raise the issue, for both researchers and practitioners, of a fair comparison among proposed techniques while engaging in the rapid prototyping and the systematic testing of novel methods. Under such circumstances, several reviews and surveys concerning rPPG (*McDuff et al., 2015*; *Rouast et al., 2018*; *Heusch, Anjos & Marcel, 2017a*; *Unakafov, 2018*; *Wang et al., 2016*; *McDuff & Blackford, 2019*; *Cheng et al., 2021*; *McDuff, 2021*; *Ni, Azarang & Kehtarnavaz, 2021*) have conducted empirical comparisons, albeit suffering under several aspects, as discussed in 'Related Works'.

To promote the development of new methods and their experimental analysis, in *Boccignone et al. (2020a)* we proposed `pyVHR`, a preliminary version of a framework supporting the main steps of the traditional rPPG pulse rate recovery, together with a sound statistical assessment of methods' performance. Yet, that proposal exhibited some limits, both in terms of code organization, usability, and scalability, and since it was suitable for traditional approaches only. Here we present a new version of `pyVHR`,[1] with a totally re-engineered code, which introduces several novelties.

[1] Freely available on GitHub: https://github.com/phuselab/pyVHR.

First of all, we provide a dichotomous view of remote heart rate monitoring, leading to two distinct classes of approaches: traditional methods (section 'Pipeline for Traditional Methods') and DL-based methods (section 'Pipeline for Deep-Learning Methods'). Moreover, concerning the former, a further distinction is setup, concerning the Region Of Interest (ROI) taken into account, thus providing both holistic and patch-based methods. The former takes into account the whole skin region, extracted from the face captured in subsequent frames. Undoubtedly, it is the simplest approach, giving satisfying results when applied on video acquired in controlled contexts. However, in more complex settings the illumination conditions are frequently unstable, giving rise to either high variability of skin tone or shading effects. In these cases the holistic approach is prone to biases altering subsequent analyses. Differently, the patch-based approach employs and tracks an ensemble of patches sampling the whole face. The rationale behind this choice is twofold. On the one hand, the face regions affected by either shadows or bad lighting conditions can be discarded, thus avoiding uncorrelated measurements with the HR ground-truth. On the other hand, the amount of observations available allows for making

the final HR estimate more robust, even through simple statistics (*e.g.*, medians), while controlling the confidence levels.

Second, the framework is agile, covers each stage of the pipeline that instantiates it, and it is easily extensible. Indeed, one can freely embed new methods, datasets or tools for the intermediate steps (see section 'Extending the Framework') such as for instance: face detection and extraction, pre- and post-filtering of RGBs traces or BVPs signals, spectral analysis techniques, statistical methods.

`pyVHR` can be easily employed for many diverse applications such as anti-spoofing, aliveness detection, affective computing, biometrics. For instance, in section 'Case Study: DeepFake detection with pyVHR' a case study on the adoption of rPPG technology for a Deepfake detection task is presented.

Finally, computations can be achieved in real-time thanks to the NVIDIA GPU (Graphics Processing Units) accelerated code and the use of optimized Python primitives.

Related works

In the last decade the rPPG domain has witnessed a flourish of investigations (*McDuff et al., 2015*; *Rouast et al., 2018*; *Heusch, Anjos & Marcel, 2017a*; *Unakafov, 2018*; *Wang et al., 2016*; *McDuff & Blackford, 2019*; *Cheng et al., 2021*; *McDuff, 2021*; *Ni, Azarang & Kehtarnavaz, 2021*). Yet, the problem of a fair and reproducible evaluation has been in general overlooked. It is undeniable that theoretical evaluations are almost infeasible, given the complex operations or transformations each algorithm performs. Nevertheless, empirical comparisons could be very informative if conducted in the light of some methodological criteria (*Boccignone et al., 2020a*). In brief: pre/post processing standardization; reproducible evaluation; multiple dataset testing; rigorous statistical assessment.

To the best of our knowledge, a framework respecting all these criteria was missing until the introduction of the early version of `pyVHR` *Boccignone et al. (2020a)*.

In *Heusch, Anjos & Marcel (2017a)* a Python collection of rPPG algorithms is presented, without claiming to be complete in the method assessment.

Interestingly, in *Unakafov (2018)*, the authors highlight the dependency of the pulse rate estimation on five main steps: ROI-selection, pre-processing, rPPG method, post-processing, pulse rate estimation. They present a theoretical framework to assess different pipelines in order to find out which combination provides the most precise PPG estimation; results are reported on the DEAP dataset (*Koelstra et al., 2011*). Unfortunately, no code has been made available.

In *Pilz (2019)* a MATLAB toolbox is presented, implementing two newly proposed methods, namely Local Group Invariance (LGI) (*Pilz et al., 2018*) and Riemannian-PPGI (SPH) (*Pilz, 2019*), and comparing them to the GREEN channel expectation (*Verkruysse, Svaasand & Nelson, 2008*) baseline, and two state-of-the-art methods, *i.e.,* Spatial Subspace Rotation (SSR) (*Wang, Stuijk & De Haan, 2015*), and Projection Orthogonal to Skin (POS) (*Wang et al., 2016*).

In *McDuff & Blackford (2019)* the authors propose `iPhys`, a MATLAB toolbox implementing several methods, such as Green Channel, POS, CHROM (*De Haan & Jeanne, 2013*), ICA (*Poh, McDuff & Picard, 2010*), and BCG (*Balakrishnan, Durand &*

Table 1 A comparison of the freely available rPPG frameworks. Check signs mark conditions fulfilled; crosses, those neglected.

	Lang.	Modular	Deep-Ready	Multi-Data	Stat. Assessment
pyVHR	Python	✓	✓	✓	✓
McDuff & Blackford (2019)	MATLAB	×	×	×	×
Heusch, Anjos & Marcel (2017a)	Python	×	×	×	×
Pilz (2019)	MATLAB	×	×	✓	×

Guttag, 2013). The toolbox is presented as a bare collection of method implementations, without aiming at setting up a rigorous comparison framework on one or more datasets. It is worth noticing that all these frameworks are suitable for traditional methods only. Table 1 summarizes at a glance the main differences between pyVHR and the already proposed frameworks.

INSTALLATION

The quickest way to get started with pyVHR is to install the miniconda distribution, a lightweight minimal installation of Anaconda Python.

Once installed, create a new conda environment, automatically fetching all the dependencies based on the adopted architecture—with or without GPU—, by one of the following commands:

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

for CPU-only architecture, or

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

for a CPU architecture with GPU support. The latest stable release build of pyVHR can be installed, inside the newly created conda environment, with:

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

for CPU-only, or

```
pip install pyvhr
```

for CPU with GPU support.

The source code for pyVHR can be found on GitHub at https://github.com/phuselab/pyVHR and it is distributed under the GPL-3.0 License. On GitHub, the community can report issues, questions as well as contribute with code to the project. The documentation of the pyVHR framework is available at https://phuselab.github.io/pyVHR/.

PYVHR PIPELINE FOR TRADITIONAL METHODS

In this section, we introduce the pyVHR modules to be referred by traditional rPPG methods. They are built on top of both APIs developed for the purpose, and open-source libraries. This pipeline follows a software design strategy that assemble sequential modules or stages, with the output of a stage serving as input to one or more subsequent stages. This responds to the need for the framework to be flexible and extensible in order to be more maintainable and improvable over time with innovative or alternative techniques.

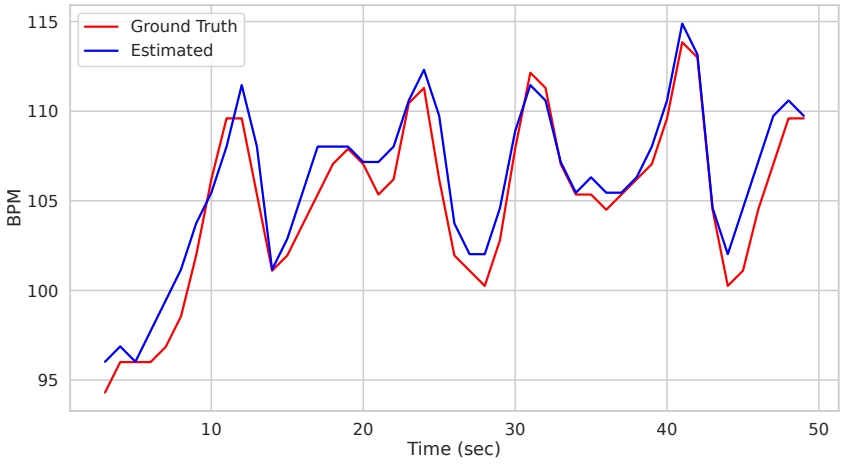

Figure 1 **Prediction example.** Predictions on the `Subject1` of the `UBFC` Dataset.
Full-size DOI: 10.7717/peerjcs.929/fig-1

The pipeline Stages

The `Pipeline()` class implements the sequence of stages or steps that are usually required by the vast majority of rPPG methods proposed in the literature, in order to estimate the BPM of a subject, given a video displaying his/her face. Eventually, going through all these steps in `pyVHR` is as simple as writing a couple of lines of Python code:

```
from pyVHR.analysis.pipeline import Pipeline

pipe = Pipeline()
time, BPM, uncertainty = pipe.run_on_video('/path/to/vid.avi')
```

Calling the `run_on_video()` method of the `Pipeline()` class starts the analysis of the video provided as argument and produces as output the `time` step of the estimated `BPM` and related `uncertainty` estimate. Figure 1 depicts the predicted BPM on `Subject1` of the UBFC[2] dataset (*Bobbia et al., 2019*) (blue trajectory). For comparison, the ground truth BPM trajectory (as recorded from a PPG sensor) is reported in red.

On the one hand the above-mentioned example witnesses the ease of use of the package by hiding the whole pipeline behind a single function call. On the other hand it may be considered too constraining as hinders the user from exploiting its full flexibility. Indeed, the `run_on_video()` method can be thought of as a black box delivering the desired result with the least amount of effort, relying on default parameter setting.

Nevertheless, some users may be interested in playing along with all the different modules composing the `pyVHR` pipeline and the related parameters. The following sections aim at describing in detail each of such elements. These are shown in Fig. 2B and can be recapped as follows:

1. *Skin extraction:* The goal of this first step is to perform a face skin segmentation in order to extract PPG-related areas; the latter are subsequently collected in either a single patch (holistic approach) or a bunch of "sparse" patches covering the whole face (patch-wise approach).

[2] Available at: https://sites.google.com/view/ybenezeth/ubfcrppg.

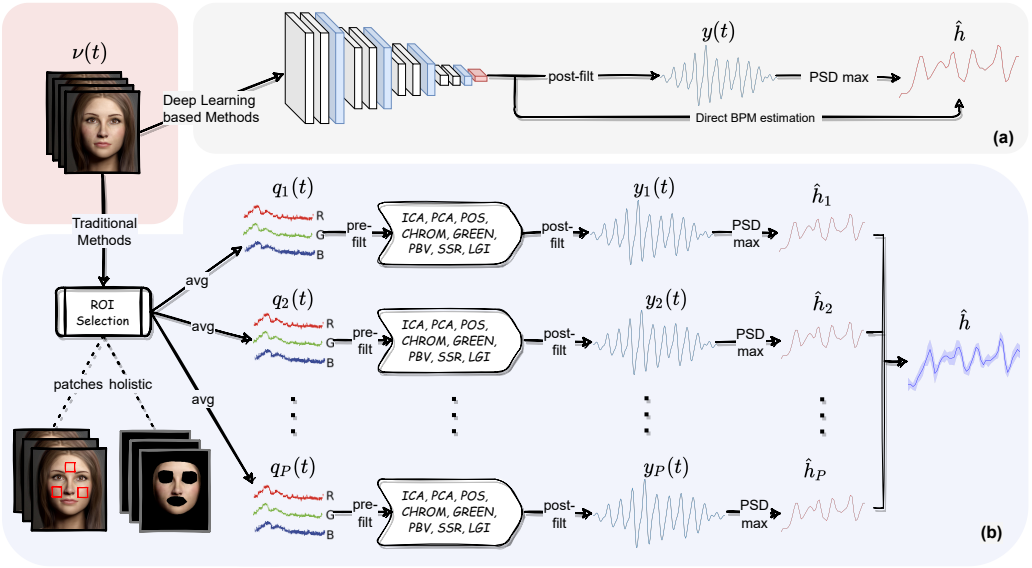

Figure 2 **The pyVHR pipeline at a glance.** (A) The multi-stage pipeline of the pyVHR framework for BPM estimate through PSD analysis exploiting end-to-end DL-based methods. (B) The multi-stage pipeline for traditional approaches that goes through: windowing and patch collection, RGB trace computation, pre-filtering, the application of an rPPG algorithm estimating a BVP signal, post-filtering and BPM estimate through PSD analysis.

Full-size DOI: 10.7717/peerjcs.929/fig-2

2. *RGB signal processing:* The patches, either one or more, are coherently tracked and are used to compute the average colour intensities along overlapping windows, thus providing multiple time-varying RGB signals for each temporal window.
3. *Pre-filtering:* Optionally, the raw RGB traces are pre-processed *via* canonical filtering, normalization or de-trending; the outcome signals provide the inputs to any subsequent rPPG method.
4. *BVP extraction:* The rPPG method(s) at hand is applied to the time-windowed signals, thus producing a collection of heart rate pulse signals (BVP estimates), one for each patch.
5. *Post-filtering:* The pulse signals are optionally passed through a narrow-band filter in order to remove unwanted out-of-band frequency components.
6. *BPM estimation:* A BPM estimate is eventually obtained through simple statistics relying on the apical points of the BVP power spectral densities.

Skin extraction

The skin extraction step implemented in pyVHR consists in the segmentation of the face region of the subject. Typically, the regions corresponding to the eyes and mouth are discarded from the analysis. This can be accomplished by pyVHR in two different ways, denoted as:

1. the *Convex-hull* extractor,
2. the *Face parsing* extractor.

(A) Convex Hull (B) BiSeNet

Figure 3 Comparison of the two implemented skin extraction methods. Output of the Convex-hull approach (A) and face parsing by BiSeNet (B) on a subject of the LGI-PPGI dataset (*Pilz et al., 2018*).
Full-size DOI: 10.7717/peerjcs.929/fig-3

The *Convex-hull* extractor considers the skin region as the convex-hull of a set of the 468 facial fiducial points delivered by the MediaPipe face mesh (*Lugaresi et al., 2019*). The latter provides reliable face/landmark detection and tracking in real-time. From the convex-hulls including the whole face, pyVHR subtracts those computed from the landmarks associated to the eyes and mouth. The resulting mask is employed to isolate the pixels that are generally associated to the skin. An example is shown in the left image of Fig. 3 on a subject of the LG-PPGI dataset (*Pilz et al., 2018*).[3]

[3] Available for download at https://github.com/partofthestars/LGI-PPGI-DB.

Alternatively, the *Face parsing* extractor computes a semantic segmentation of the subject's face. It produces pixel-wise label maps for different semantic components (*e.g.*, hair, mouth, eyes, nose, c...), thus allowing to retain only those related to the skin regions. Face semantic segmentation is carried over with BiSeNet (*Yu et al., 2018*), which supports real-time inference speed. One example is shown in the right image of Fig. 3.

Both extraction methods are handled in pyVHR by the `SignalProcessing()` class. The following lines of code set-up the extractor with the desired skin extraction procedure:

```
from pyVHR.extraction.sig_processing import SignalProcessing

sig_processing = SignalProcessing()
if skin_method == 'convexhull':
  sig_processing.set_skin_extractor(SkinExtractionConvexHull(target_device))
elif skin_method == 'faceparsing':
  sig_processing.set_skin_extractor(SkinExtractionFaceParsing(target_device))
```

Holistic approach

The skin extraction method paves the way to the RGB trace computation which is accomplished in a channel-wise fashion by averaging the facial skin colour intensities. This is referred to as the holistic approach, and within the pyVHR framework it can be instantiated as follows:

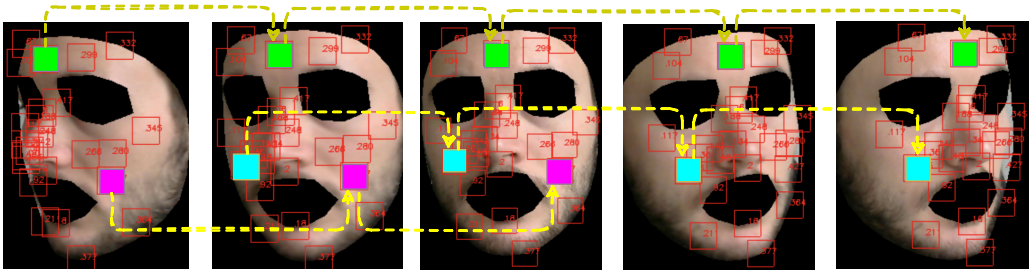

Figure 4 **Landmarks automatically tracked by MediaPipe and correspondent patch tracking on a subject of the LGI-PPGI dataset (*Pilz et al., 2018*).**
Full-size DOI: 10.7717/peerjcs.929/fig-4

```
sig = sig_processing.extract_holistic(videoFileName)
```

Patch-based approach

In contrast to the holistic approach, the patch-based one takes into account a bunch of localized regions of interest, thus extracting as many RGB traces as patches. Clearly, imaging photoplethysmography in unconstrained settings is sensitive to subjects changing pose, moving their head or talking. This calls for a mechanism for robust detection and tracking of such regions.

To such end, pyVHR again relies on the MediaPipe Face Mesh, which establishes a metric 3D space to infer the face landmark screen positions by a lightweight method to drive a robust and performant tracking. The analysis runs on CPU and has a minimal speed or memory footprint on top of the inference model.

The user can easily select up to 468 patches centered on a subset of landmarks and define them as the set of informative regions on which the subsequent steps of the pipeline are evaluated. An example of landmark extraction and tracking is shown in Fig. 4. Note that eventually, a patch may disappear due to subject's movements, hence delivering only partial or none contribution.

It is worth noting how the user is allowed to arbitrarily compose its own set of patches by exploiting pyVHR utility functions. In the example below, three patches have been selected corresponding to the forehead, left and right cheek areas. Usually, several patches are chosen in order to better control the high variability in the results and to achieve high level of confidence, while making smaller the margin of error.

As for the holistic approach, video loading and patch extraction are handled by few APIs available in the `SignalProcessing()` class, as shown in the following script.

```
from pyVHR.extraction.utils import MagicLandmarks

ldmks_list = [MagicLandmarks.cheek_left_top[16], MagicLandmarks.cheek_right_top
    [14], MagicLandmarks.forehead_center[1]]
sig_processing.set_landmarks(ldmks_list)
# set squares patches side dimension
sig_processing.set_square_patches_side(28.0)
#Extract square patches and compute the RGB trajectories as the channel-wise mean
sig = sig_processing.extract_patches(videoFileName, 'squares', 'mean')
```

RGB signal computation

In this step, the skin regions detected and tracked on the subject's face are split in successive overlapping time windows. Next, the RGB traces associated to each region are computed by averaging their colour intensities. More formally, let us consider an RGB video $v \in \mathbb{R}^{w \times h \times 3 \times T}$ of T frames containing a face, split on P (possibly overlapped) patches. Once the ith patch has been selected, an RGB signal $q_i(t)$ is computed. Denote $\{p_i^j(t)\}_{j=1}^{N_i}$ the set of N_i pixels belonging to the ith patch at time t, where $p_i^j(t) \in [0,255]^3$. Then, $q_i(t)$ is recovered by averaging on pixel colour intensities, *i.e.*,

$$q_i(t) = \frac{1}{N_i} \sum_{j=1}^{N_i} p_i^j(t), \qquad i = 1, \ldots, P.$$

In the time-splitting process, fixed an integer $\tau > 0$, $q_i(t)$ is sliced into K overlapping windows of $M = W_s F_s$ frames, thus obtaining

$$q_i^k(t) = q_i(t) w(t - k\tau F_s), \qquad k = 0, \ldots, K-1.$$

where F_s represents the video frame rate, W_s the window length in seconds, while w is the rectangular window defined as:

$$w(t) = \begin{cases} 1, & 0 \leq t < M \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

In order for the signal segments to actually overlap, the overlap inequality $\tau < W_s$ must be verified.

Figure 5 shows how the above described patch-based split and tracking procedure is put in place.

In pyVHR , the extraction of the windowed RGB signals is computed by the following code snippet.

```
from pyVHR.extraction.utils import sig_windowing, get_fps

Ws = 6 #window lenght in seconds
overlap = 1 #window overlap in seconds
fps = get_fps(videoFileName)

windowed_sig, timesES = sig_windowing(sig, Ws, overlap, fps)
```

Notably, beside being able to switch between convex-hull and face parsing, the user can easily change the main process parameters such as the window length and the amount of frame overlapping.

Methods for BVP estimation

Given that the framework can rely on holistic-wise and patch-wise processing, pyVHR estimates the BVP signal either from a single trace or leveraging on multiple traces. In both cases it employs a wide range of state of the art rPPG methods.

In particular, the windowed RGB traces $q_i^k(t)$ ($i = 1, \ldots, P$, with $P = 1$ in the holistic case) of length K are given in input to the rPPG method at hand, which outputs the signals $y_i^k(t)$ representing the estimated BVP associated to the ith patch in the k-th time window.

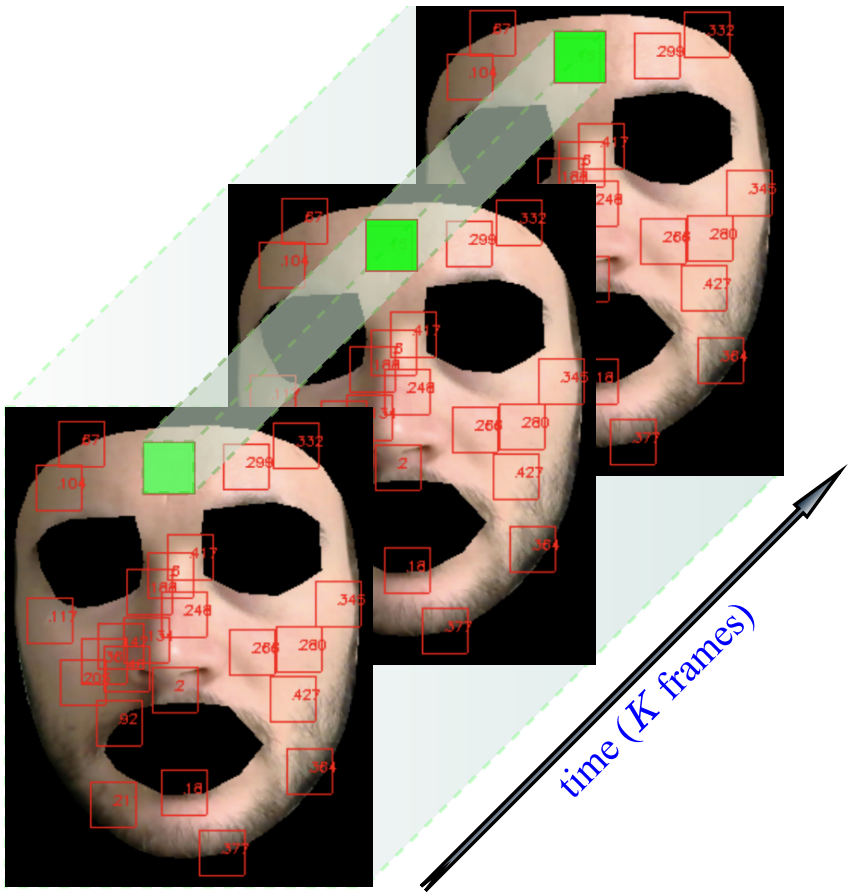

Figure 5 **Patch tracking within a frame temporal window on a subject of the LGI-PPGI dataset (*Pilz et al., 2018*).**

Full-size DOI: 10.7717/peerjcs.929/fig-5

The many methods that have been proposed in the recent literature mostly differ in the way of combining such RGB signals into a pulse-signal. The pool of methods provided by `pyVHR` , together with a description of the main concepts grounding them, is provided in Table 2. A review of the principles/assumptions behind each of the implemented algorithms is out of the scope of the present work. The interested reader might refer to *Wang et al. (2016)*, *McDuff et al. (2015)* and *Rouast et al. (2018)*.

Currently, the package implements the following methods for the estimation of the pulse signal from the RGB traces: GREEN (*Verkruysse, Svaasand & Nelson, 2008*), CHROM (*De Haan & Jeanne, 2013*), ICA (*Poh, McDuff & Picard, 2010*), LGI (*Pilz et al., 2018*), PBV (*De Haan & Van Leest, 2014*), PCA (*Lewandowska et al., 2011*), POS (*Wang et al., 2016*), SSR (*Wang, Stuijk & De Haan, 2015*). However, the user may define any custom method for estimating BVP by extending the `pyVHR.BVP.methods` module.

The BVP signal can be estimated in `pyVHR` as follows:

```
bvp = RGB_sig_to_BVP(windowed_sig, fps, method=cpu_POS)
```

Table 2 Traditional rPPG algorithms implemented in pyVHR.

Method	Description
GREEN (*Verkruysse, Svaasand & Nelson, 2008*)	The Green (G) temporal trace is directly considered as an estimate of the BVP signal. Usually adopted as a *baseline* method.
ICA (*Poh, McDuff & Picard, 2010*)	Independent Component Analysis (ICA) is employed to extract the pulse signal via Blind Source Separation of temporal RGB mixtures.
PCA (*Lewandowska et al., 2011*)	Principal Component Analysis (PCA) of temporal RGB traces is employed to estimate the BVP signal.
CHROM (*De Haan & Jeanne, 2013*)	A Chrominance-based method for the BVP signal estimation.
PBV (*De Haan & Van Leest, 2014*)	Computes the signature of blood volume pulse changes to distinguish the pulse-induced color changes from motion noise in RGB temporal traces.
SSR (S2R) (*Wang, Stuijk & De Haan, 2015*)	Spatial Subspace Rotation (SSR); estimates a spatial subspace of skin-pixels and measures its temporal rotation for extracting pulse signal.
POS (*Wang et al., 2016*)	Plane Orthogonal to the Skin (POS). Pulse signal extraction is performed via a projection plane orthogonal to the skin tone.
LGI (*Pilz et al., 2018*)	Local Group Invariance (LGI). Computes a feature representation which is invariant to action and motion based on differentiable local transformations.

Figure 6 depicts the BVP signals estimated by four different rPPG methods implemented in pyVHR (POS, GREEN, CHROM, PCA), on the same time window using the holistic patch.

Pre and post-filtering

pyVHR offers simple APIs to apply filters on either the RGB traces $q_i(t)$ (pre-filtering) or the estimated pulse signal $y_i(t)$ (post-filtering). A set of ready to use filters are implemented, namely:

- *Band Pass (BP) filter*: filters the input signal using a bandpass N-th order Butterworth filter with a given passband frequency range.
- *Detrending*: subtracts offsets or linear trends from time-domain input data.
- *Zero-Mean*: Removes the DC component from a given signal.

However, the user can adopt any custom filter complying with the function signature defined in `pyVHR.BVP.filters`. The following provides an example of how to detrend an RGB trace $q_i(t)$:

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

Additionally, a Band-Pass filter can be applied on the estimated BVP signals $y_i(t)$ in order to rule out the frequencies that leave outside the feasible range of typical heart rates (which is usually between 40 Hz and 200 Hz):

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

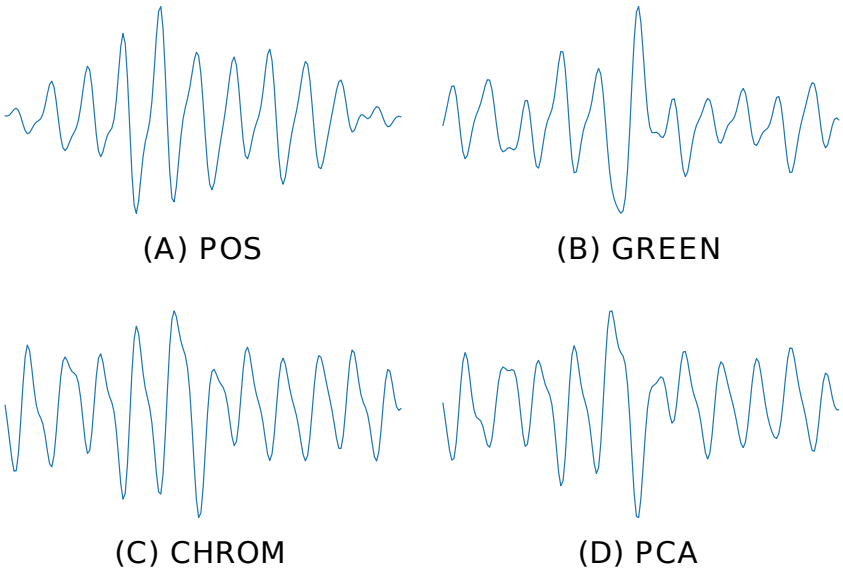

Figure 6 Predicted BVP signals. An example of estimated BVP signals on the same time window by four different methods. (A) POS. (B) GREEN. (C) CHROM. (D) PCA.
Full-size DOI: 10.7717/peerjcs.929/fig-6

From BVP to BPM

Given the estimated BVP signal, the beats per minute (BPM) associated to a given time window can be easily recovered *via* analysis of its frequency domain representation. In particular, pyVHR estimates the Power Spectral Density (PSD) of the windowed pulse signal $y_i^k(t)$ *via* discrete time Fourier transform (DFT) using the Welch's method. The latter employs both averaging and smoothing to analyze the underlying random process.

Given a sequence $y_i^k(t)$, call $S_i^k(\nu)$ its power spectra (periodogram) estimated *via* the Welch's method. The BPM is recovered by selecting the normalized frequency associated to the peak of the periodogram:

$$\hat{\nu}_i^k = \underset{\nu \in \Omega}{\arg\max} \left\{ S_i^k(\nu) \right\},$$

corresponding to the PSD maxima as computed by Welch's method on the range $\Omega = [39, 240]$ of feasible BPMs.

The instantaneous BPM associated to the k-th time window ($k \in 1, \ldots, K$) for the ith patch ($i \in 1, \ldots, P$), is recovered by converting the normalized peak frequency $\hat{\nu}_i^k$ into an actual frequency,

$$\hat{h}_i^k = \hat{\nu}_i^k \frac{F_s}{L},$$

where F_s is the video frame rate and L is the DFT size. Figure 7 shows the Welch's estimates for the BVP signals of Fig. 6. The peak in the spectrum represents the instantaneous Heart Rate ($\hat{h}_i^k$).

When multiple patches have been selected ($P > 1$), the predicted BPM for the k-th time window can be obtained resorting to simple statistical measures. Specifically, pyVHR computes the median BPM value of the predictions coming from the P patches.

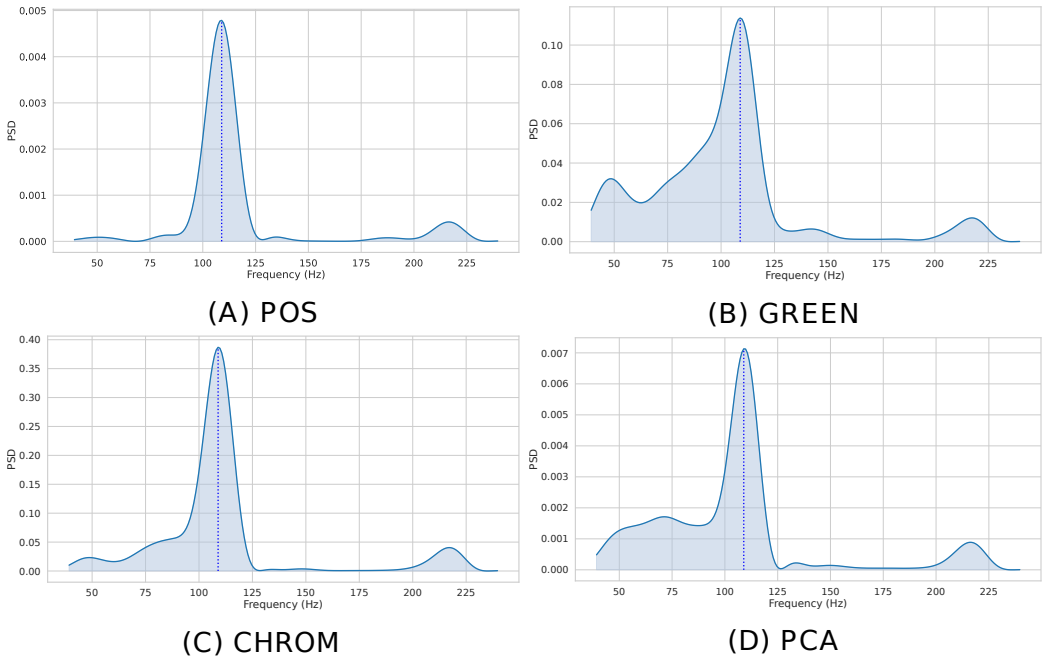

Figure 7 **Estimated PSD.** Estimated Power Spectral Densities (PSD) for the BVP signals plotted in Fig. 6. The BPM estimate, given by the maxima of the PSD, is represented by the blue dashed line. (A) POS. (B) GREEN. (C) CHROM. (D) PCA.

Full-size DOI: 10.7717/peerjcs.929/fig-7

Formally, call H^k the ordered list of P BPM predictions coming from each patch in the k-th time window; then:

$$\hat{h}^k = \text{median}(H^k) = \begin{cases} H^k\left[\dfrac{P-1}{2}\right] & \text{if P is odd} \\ \dfrac{\left(H^k\left[\frac{P}{2}-1\right]+H^k\left[\frac{P}{2}\right]\right)}{2} & \text{if P is even.} \end{cases} \tag{2}$$

Note that if the number of patches $P = 1$ (*i.e.*, a single patch has been selected or the holistic approach has been chosen), then:

$$\hat{h}^k = H^k[0]. \tag{3}$$

Moreover, when multiple patches have been selected, a measure of variability of the predictions can be computed in order to quantify the uncertainty of the estimation. In particular, pyVHR computes the Median Absolute Deviation (*MAD*) as a robust measure of statistical dispersion. The *MAD* is defined as:

$$MAD^k = \text{median}(|H^k - \hat{h}^k|). \tag{4}$$

Clearly, the *MAD* drops to 0 when $P = 1$. Figure 8 depicts the distribution of predicted BPM in a given time window, when $P = 100$ patches are employed. The results from different methods are shown for comparison. Note how the median is able to deliver precise predictions, while the MAD represents a robust measure of uncertainty.

Computing the BPM from the BVP signal(s) can be easily accomplished in pyVHR as follows:

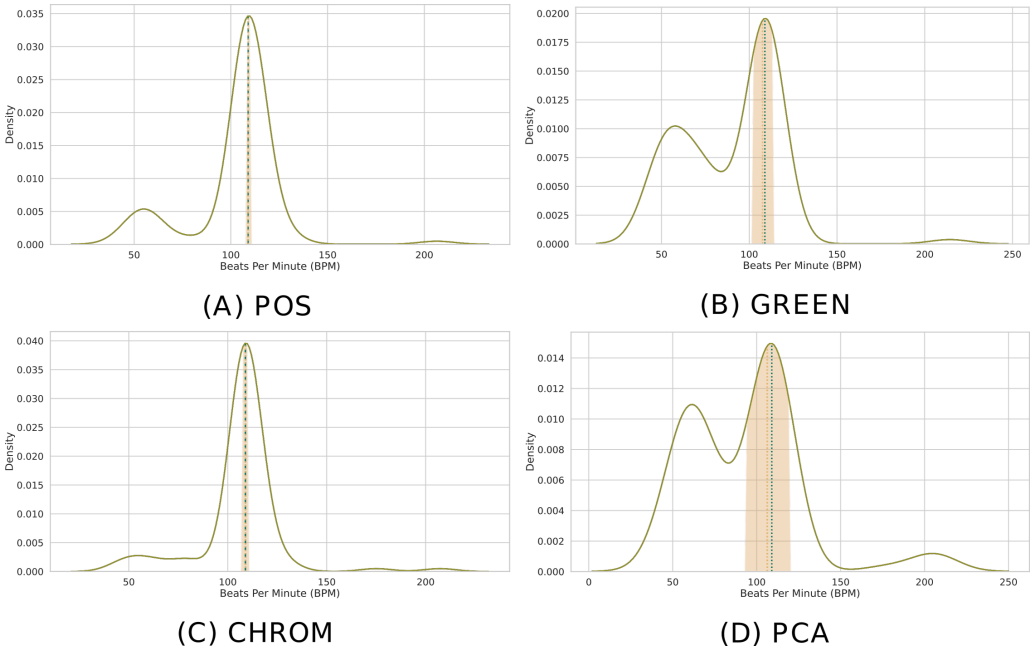

Figure 8 Distribution of BPM predictions by four methods on P patches. (A) POS. (B) GREEN. (C) CHROM. (D) PCA. Kernel Density Estimates (KDEs) of the predicted BPMs in a time window from $P = 100$ patches. The ultimate BPM prediction is given by the median (gold dashed line). The uncertainty estimate delivered by the Median Absolute Deviation (MAD) is shown by the golden band around the median. The blue dashed line represents the actual BPM.

Full-size DOI: 10.7717/peerjcs.929/fig-8

```
from pyVHR.BPM.BPM import BVP_to_BPM, multi_est_BPM_median

bpmES = BVP_to_BPM(bvp, fps)
# median BPM from multiple estimators BPM
bpm, uncertainty = multi_est_BPM_median(bpmES)
```

The result along with the ground-truth are shown in Fig. 9.

Efficient computation and GPU acceleration

Most of the steps composing the pipeline described above are well suited for parallel computation. For instance, the linear algebra operations involved in the pulse signal recovery from the RGB signal or, more generally, the signal processing steps (*e.g.*, filtering, spectral estimation, *etc.*), not to mention the skin segmentation procedures from high resolution videos.

To such end, pyVHR exploits the massive parallelism of Graphical Processing Units (GPUs). It is worth mentioning that GPUs are not strictly required to run pyVHR code; nevertheless, in some cases, GPU accelerated code allows to run the pipeline in real-time.

Figure 10 shows the average per-frame time requirement for getting through the whole pipeline when using the POS method. It is worth noticing that, when using the Holistic approach (or equivalently one single patch), a video frame can be processed in less than 0.025 seconds, regardless of the adopted architecture (either CPU or GPU). This means

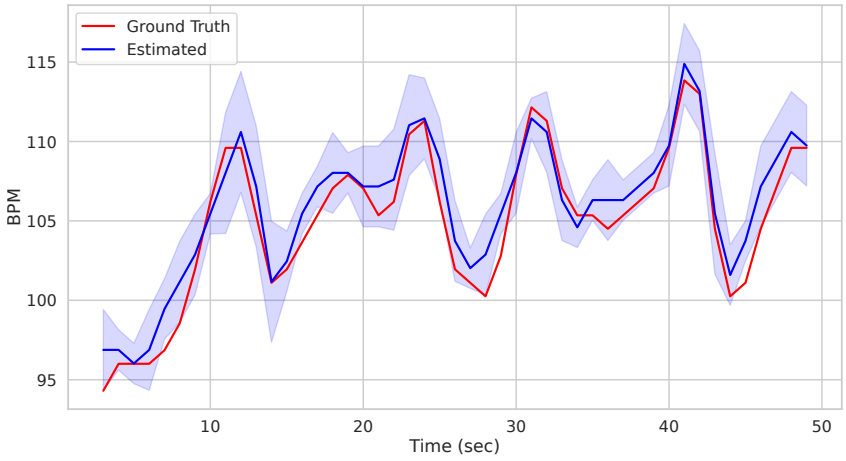

Figure 9 Comparison of predicted *vs* ground truth BPMs using the patch-wise approach. Predicted BPM (blue) for the `Subject1` of the UBFC Dataset. The uncertainty is plotted in shaded blue, while the ground truth is represented by the red line.

Full-size DOI: 10.7717/peerjcs.929/fig-9

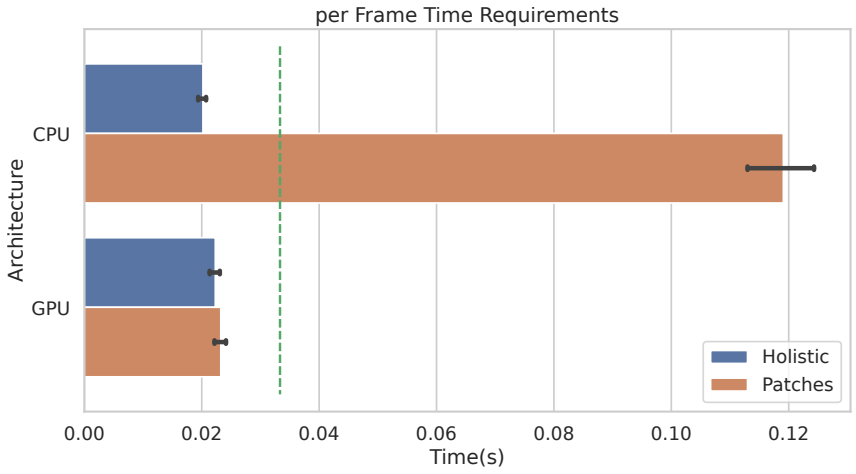

Figure 10 Per-frame time requirements. Average time requirements to process one frame by the Holistic and Patches approaches when using CPU *vs.* GPU accelerated implementations. The green dashed line represents the real-time limit at 30 frames per second (fps).

Full-size DOI: 10.7717/peerjcs.929/fig-10

that the whole pipeline can be safely run in real-time for videos at 30 frames per second (the 30 fps time limit is represented by the dashed green line).

Obviously, when multiple patches are employed (in the example of Fig. 10, $P = 100$ patches are used), the average time required by CPUs to process a single frame rises up to about 0.12 seconds. Notably, the adoption of GPU accelerated code allows to run the whole pipeline in real-time, even when using a huge number of patches. Indeed, the ratio to CPU time and GPU time, *i.e.*, the speedup defined as $time_{\text{seq}}/time_{\text{parall}}$, is about 5. Remarkably, similar gain in performances are observed if adopting any other rPPG method.

The result shown in Fig. 10 refers to the following hardware configuration: Intel Xeon Silver 4214R 2.40 GHz (CPU), NVIDIA Tesla V100S PCIe 32GB (GPU). Similar results were obtained relying on a non-server configuration: Intel Core i7-8700K 4.70 GHz (CPU), NVIDIA GeForce GTX 960 2GB (GPU). The maximum RAM usage for 1 min HD video analysis is 2.5 GB (average is 2 GB); the maximum GPU memory usage for 1 min HD video analysis is 1.8 GB (average is 1.4 GB).

In the following it is shown how to enable CUDA GPU acceleration on different steps in the Pipeline:

- Skin extraction: Convex Hull and Face Parsing. The user can easily choose to run this step with CPU or GPU:

```
target_device = 'GPU' # or 'CPU'
sig_processing = SignalProcessing()
sig_processing.set_skin_extractor(
          SkinExtractionConvexHull(target_device))
sig_processing.set_skin_extractor(
          SkinExtractionFaceParsing(target_device))
```

- rPPG Methods: the package contains different version of the same method. For example the CHROM method is implemented for both CPU and GPU.

```
bvp = RGB_sig_to_BVP(sig, fps, device_type='cuda', method=cupy_CHROM)
bvp = RGB_sig_to_BVP(sig, fps, device_type='cpu', method=cpu_LGI)
```

- BPM Estimation:

```
  bpmES = BVP_to_BPM_cuda(bvp, fps)   #GPU
  bpmES = BVP_to_BPM(bvp, fps)        #CPU
```

GUI for online processing

Besides being used as a Python library, pyVHR makes available a Graphical User Interface (GUI). It provides access to most of the available functionalities, while showing the BPMs estimation process in real-time. It is straightforward to use and it allows for setting up the pipeline parameters and the operating mode, by choosing either a webcam or a video file.

To start the GUI, one can run the command:

```
$ Python pyVHR/realtime/GUI.py
```

Figure 11 shows a screenshot of the GUI during the online analysis of a video. On the top right are presented the video file name, the video FPS, resolution, and a radio button list to select the type of frame displayed. The original or segmented face can be visualized either selecting the *Original Video* or the *Skin* option, while the *Patches* radio button enables the visualization of the patches (in red). The *Stop* button ends the analysis, and results can be saved on disk by pushing the *Save BPMs* button.

PYVHR PIPELINE FOR DEEP-LEARNING METHODS

Recent literature in computer vision has given wide prominence to end-to-end deep neural models and their ability to outperform traditional methods requiring hand-crafted

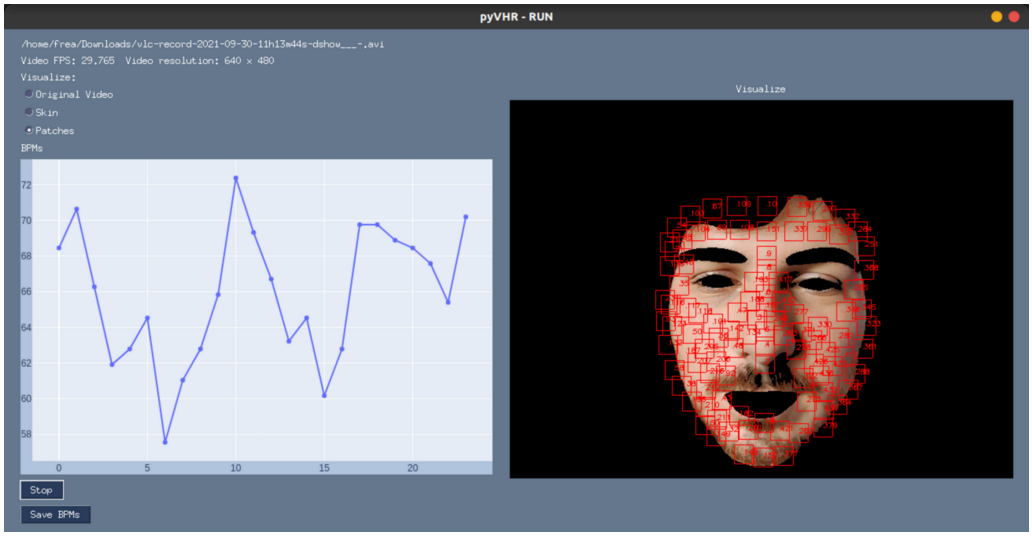

Figure 11 **The graphical user interface.** A screenshot of the graphical user interface (GUI) for online video analysis. The plot on the left shows the predicted BPMs, while on the right it is shown the processed video frames (captured with a webcam) with an example of the segmented skin and the tracked patches.
Full-size DOI: 10.7717/peerjcs.929/fig-11

feature design. In this context, learning frameworks for recovering physiological signals were also born (*Chen & McDuff, 2018*; *Niu et al., 2019*; *Yu et al., 2020*; *Yu et al., 2021*; *Gideon & Stent, 2021*; *Liu et al., 2021*; *Nowara, McDuff & Veeraraghavan, 2020*). The end-to-end nature of the DL based approaches is reflected by a much simpler pipeline; indeed, these methods typically require as input raw video frames that are processed by the DL architecture at hand and produce either a BVP signal or the estimated heart rate, directly. Figure 2A depicts at a glance the flow of stages involved in the estimation of heart rate using DL based approaches. Clearly, this gain in simplicity comes at the cost of having to train the model on huge amounts of data, not to mention the issues related to the assessment of the model's generalization abilities.

In the last few years the literature has witnessed a flourish of DL-based approaches(for two recent reviews see *Cheng et al. (2021)* and *Ni, Azarang & Kehtarnavaz (2021)*. Nonetheless, despite the claimed effectiveness and superior performances, few solutions have been made publicly available (both in terms of code and learned model weights). This raises issues related to proper reproducibility of the results and the method assessment. For instance, a recent efficient neural architecture called MTTS-CAN has been proposed in *Liu et al. (2020)* being a valuable contribution since the pre-trained model and code are released. It essentially leverages a tensor-shift module and 2D-convolutional operations to perform efficient spatial temporal modeling in order to enable real-time cardiovascular and respiratory measurements. MTTS-CAN can be framed as an end-to-end model since it does not need any pre-processing step before data is fed into the network, except performing trivial image normalizations. MTTS-CAN is included in the pyVHR framework, and below it is shown how practical is to extend the framework with similar DL-based approaches provided that the pre-trained model is available.

As for the pipeline for traditional methods shown in previous section, pyVHR also defines a sequence of stages that allows to recover the time varying heart rate from a sequence of images displaying a face. These are detailed in the following.

The stages for end-to-end methods

Given a video displaying a subject face, the `DeepPipeline()` class performs the necessary steps for the rPPG estimate using a chosen end-to-end DL method. Specifically, the pipeline includes the handling of input videos, the estimation from the sequence of raw frames and, eventually, the pre/post-processing steps. The following code snippet carries out the above procedure with few statements:

```
from pyVHR.analysis.pipeline import DeepPipeline

pipe = DeepPipeline()
time, BPM = pipe.run_on_video('/path/to/vid.avi', method='MTTS_CAN')
```

Figure 2A summarizes the steps involved in a `run_on_video()` call on a given input video. As in the pipeline using traditional methods (see section 'Pipeline for Traditional Methods'), after a predetermined chain of analysis steps it produces as output the estimated `BPM` and related timestamps (`time`).

For instance, consider the `MTTS-CAN` model currently embedded into the `DeepPipeline()` class; it estimates the rPPG pulse signal from which the BPM computation can be carried out by following the very same procedure outlined in section 'From BVP to BPM', namely time windowing and spectral estimation. Eventually, some optional pre/post filtering operations (section 'Pre and Post-Filtering') can be performed.

The following few lines of Python code allow to carry out the above steps explicitly:

```
from pyVHR.extraction.sig_processing import SignalProcessing
from pyVHR.extraction.utils import get_fps
from pyVHR.BPM import BVP_to_BPM
from pyVHR.utils.errors import BVP_windowing

sp = SignalProcessing()
frames = sp.extract_raw('/path/to/videoFileName')
fps = get_fps('/path/to/videoFileName')

bvps_pred = MTTS_CAN_deep(frames, fps)

winsize = 6 #6\ s long time window
bvp_win, timesES = BVP_windowing(bvp_pred, winsize, fps, stride=1)
bpm = BVP_to_BPM(bvp_win, fps)
```

In order to embed a new DL-method, the code above should be simply modified substituting the function `MTTS_CAN_deep` with a new one implementing the method at hand, while respecting the same signature (cfr. 'Extending the Framework').

ASSESSMENT OF RPPG METHODS

Does a given rPPG algorithm outperforms the existing ones? To what extent? Is the difference in performance significantly large? Does a particular post-filtering algorithm cause an increase/drop of performance?

Answering all such questions, calls for a rigorous statistical assessment of rPPG methods. As a matter of fact, although the field has recently experienced a substantial

gain in interest from the scientific community, it is still missing a sound and reproducible assessment methodology allowing to gain meaningful insights and delivering best practices.

By and large, novel algorithms proposed in the literature are benchmarked on non-publicly available datasets, thus hindering proper reproducibility of results. Moreover, in many cases, the reported results are obtained with different pipelines; this makes it difficult to precisely identify the actual effect of the proposed method on the final performance measurement.

Besides that, the performance assessment mostly relies on basic and common-sense techniques, such as roughly rank new methods with respect to the state-of-the-art. These crude methodologies often make the assessment unfair and statistically unsound. Conversely, a good research practice should not limit to barely report performance numbers, but rather aiming at principled and carefully designed analyses. This is in accordance with the growing quest for statistical procedures in performance assessment in many different fields, including machine learning and computer vision (*Demšar, 2006*; *Benavoli et al., 2017*; *Torralba & Efros, 2011*; *Graczyk et al., 2010*; *Eisinga et al., 2017*).

In the vein of its forerunner (*Boccignone et al., 2020a*), `pyVHR` deals with all such problems by means of its statistical assessment module. The design principles can be recapped as follows:

- *Standardized pipeline:* When setting up an experiment to evaluate a new rPPG algorithm, the whole pipeline (except the algorithm) should be held fixed.
- *Reproducible evaluation:* The evaluation protocol should be reproducible. This entails adopting publicly available datasets and code.
- *Comparison over multiple datasets:* In order to avoid dataset bias, the analysis should be conducted on as many diverse datasets as possible.
- *Rigorous statistical assessment:* The reported results should be the outcome of proper statistical procedures, assessing their statistical significance.

The workflow of the Statistical Assessment Module is depicted in Fig. 12.

In a nutshell, each video composing a particular rPPG dataset is processed by the `pyVHR` pipeline as described above. Moreover, the package provides primitives for loading and processing real BVP signals as recorded from pulse-oximeters. Such signals undergo a treatment similar to the estimated BVP. In particular, the original BVP signal $g(t)$ is sliced into overlapping time windows; for each window the ground truth BPM h^k (the BPM associated to the k-th time window, with $k = 1, \ldots, K$) is recovered *via* maximization of the Power Spectral Density (PSD) estimate provided by the Welch's method.

Finally, the estimated ($\hat{h}^k$) and ground truth (h^k) BPM signals are compared with one another exploiting standard metrics (c.f.r 'Metrics'). Eventually, statistically rigorous comparisons can be effortlessly performed (c.f.r 'Significance Testing').

Notably, the many parameters that make up each step of the pipeline (from the ROI selection method to the pre/post filtering operations, passing through the BVP estimation by one or multiple rPPG algorithms) can be easily specified in a configuration (`.cfg`) file. Setting up a `.cfg` file allows to design the experimental procedure in accordance with the

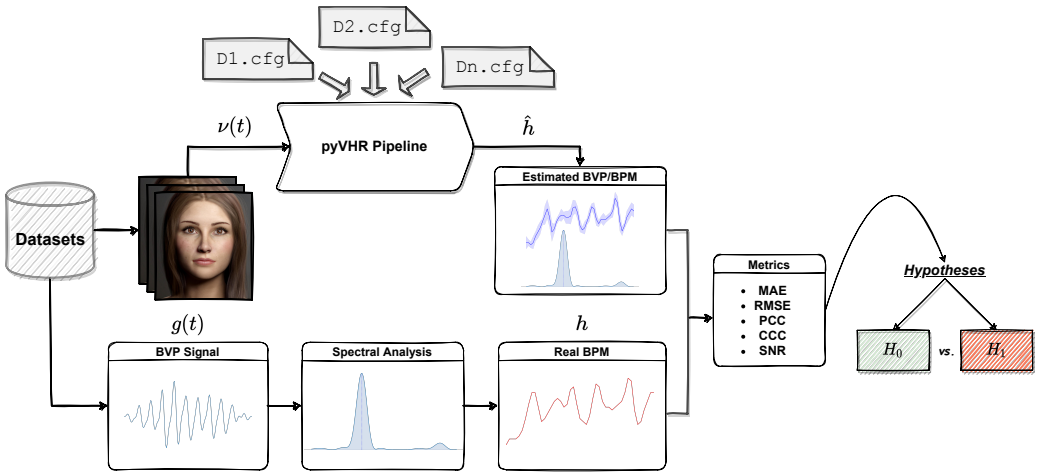

Figure 12 The assessment module at a glance. One or more datasets are loaded; videos are processed by the pyVHR pipeline while ground-truth BPM signals are retrieved. Predicted and real BPM are compared with standard metrics and the results are rigorously analyzed *via* hypothesis testing procedures.
Full-size DOI: 10.7717/peerjcs.929/fig-12

principles summarized above. A brief description of the implemented comparison metrics and the `.cfg` file specifications are provided in the following Sections.

Metrics

pyVHR provides common metrics to evaluate the performance of one or more rPPG methods in estimating the correct heart rate (BPM) over time. These are briefly recalled here.

In order to measure the accuracy of the BPM estimate $\hat{h}$, this is compared to the reference BPM as recovered from contact BVP sensors h. To this end, the reference BVP signal $g(t)$ is splitted into overlapping windows, similarly to the procedure described in section 'Methods for BVP estimation' for the estimated BVP, thus producing K windowed signals g^k $(k \in 1, \ldots, K)$. The reference BPM is found *via* spectral analysis of each window, as described in section 'From BVP to BPM'. This yields the K reference BPM h^k to be compared to the estimated one $\hat{h}^k$ by adopting any of the following metrics:

Mean Absolute Error (MAE). The Mean Absolute Error measures the average absolute difference between the estimated $\hat{h}$ and reference BPM h. It is computed as:

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

Root Mean Squared Error (RMSE). The Root-Mean-Square Error measures the difference between quantities in terms of the square root of the average of squared differences, *i.e.*,

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

Pearson Correlation Coefficient (PCC). Pearson Correlation Coefficient measures the linear correlation between the estimate $\hat{h}$ and the ground truth h. It is defined as:

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

here $\hat{\mu}$ and μdenote the means of the respective signals, while σ_1 and σ_2 are their standard deviations.

Concordance Correlation Coefficient (CCC). The Concordance Correlation Coefficient (*Lawrence & Lin, 1989*) is a measure of the agreement between two quantities. Like Pearson's correlation, CCC ranges from -1 to 1, with perfect agreement at 1. It is defined as:

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

where $\hat{\mu}$ and μdenote the means of the prediceted and reference BPM traces, respectively. Likewise, σ_1 and σ_2 are their standard deviations, while σ_{12} is their covariance.

Signal to Noise Ratio (SNR). The SNR (*De Haan & Jeanne, 2013*) measures the ratio of the power around the reference HR frequency plus the first harmonic of the estimated pulse-signal and the remaining power contained in the spectrum of the estimated BVP. Formally it is defined as:

$$\text{SNR} = \frac{1}{K} \sum_K 10 \log_{10} \frac{\sum_v (U^{k(v)} S^{k(v)})^2}{\sum_v (1 - U^{k(v)}) S^{k(v)^2}} \tag{5}$$

where $S^{k(v)}$ is the power spectral density of the estimated BVP in the k-th time window and $U^k(v)$ is a binary mask that selects the power contained within ± 12 BPM around the reference Heart Rate and its first harmonic.

The configuration (.cfg) file

The `.cfg` file allows to set up the experimental procedure for the evaluation of models. It is structured into 6 main blocks that are briefly described here:

Dataset. This block contains the information relative to a particular rPPG dataset, namely its name, and its path.

```
[DATASET]
dataset     = DatasetName
path = None
videodataDIR= /path/to/vids/
BVPdataDIR  = /path/to/gt/
...

```

Filters. It defines the filtering methods to be eventually used in the pre/post filtering phase. In the following example a band-pass butterworth filter of 6-th order is defined, with a passing band between 40 Hz and 240 Hz.

```
[BPFILTER]
path = None
name = BPfilter
params = {'minHz':0.65, 'maxHz':4.0, 'fps':'adaptive', 'order':6}

```

RGB Signal. Defines all the parameters for the extraction of the RGB signal (*e.g.*, ROI selection method, temporal windowing size, number and type of patches to be used, *etc.*).

```
[SIG]
...
winSize = 6
skin_extractor = convexhull
approach = patches
...
```

BVP. Sets up the rPPG method to be adopted for the estimation of the BVP signal. Multiple methods can be provided in order to compare them. In this example two methods will be analyzed, namely `POS` and `GREEN` (adopting their CPU implementations).

```
[BVP]
methods = ['POS', 'GREEN']
...
```

Methods. It allows to configure each rPPG method to be analyzed (*e.g.*, eventual parametrs and pre/post filters). The two methods chosen above are configured here. In particular, `POS` will not employ any pre/post filtering, while for the `GREEN` method, the above-defined band pass filter will be applied for both *pre* and *post* filtering.

```
[POS]
...
name = cpu_POS
device_type = cpu
pre_filtering = []
post_filtering = []

[GREEN]
...
name = cpu_GREEN
device_type = cpu
pre_filtering = ['BPFILTER']
post_filtering = ['BPFILTER']
```

The experiment on the dataset defined in the `.cfg` file can be simply launched as:

```
from pyVHR.analysis.pipeline import Pipeline

pipe = Pipeline()
results = pipe.run_on_dataset('/path/to/config.cfg')
results.saveResults("/path/to/results.h5")
```

In the above code, the `run_on_dataset` method from the `Pipeline` class, parses the `.cfg` file and initiates a pipeline for each rPPG method defined in it. The pipelines are used to process each video in the dataset. Concurrently, ground truth BPM data is loaded and comparison metrics are computed w.r.t. the predictions (cfr. Figure 12). The results are delivered as a table containing for each method the value of the comparison metrics computed between ground truth and predicted BPM signals, on each video belonging to

the dataset, which are then saved to disk. The same considerations hold for the definition of `.cfg` files associated to DL-based methods. Clearly, in this case the information related to the `RGB Signal` block are unnecessary.

Significance testing

Once the comparison metrics have been computed for all the considered methods, the significance of the differences between their performance can be evaluated. In other words, we want to ensure that such difference is not drawn by chance, but it represents an actual improvement of one method over another.

To this end, `pyVHR` resorts to standard statistical hypothesis testing procedures. Clearly, the results eventually obtained represent a typical repeated measure design, in which two or more pipelines are compared on paired samples (videos). A number of statistical tests are available in order to deal with such state of affairs.

In the two populations case, typically, the paired t-test is employed; alternatively some non-parametric versions of the paired t-test are at hand, namely the Sign Test or the Wilcoxon signed ranks Test; in general the latter is preferred over the former due to its higher power. For the same reason it is recommended to adopt the parametric paired t-test instead of the non-parametric Wilcoxon test. However, the use of the paired t-test is subject to the constraint of normality of the populations. If such condition is not met, a non-parametric test should be chosen.

Similarly, with more than two pipelines, repeated measure ANOVA is the parametric test that is usually adopted. Resorting to ANOVA, requires Normality and Heteroskedasticity (equality of variances) conditions to be met. Alternatively, when these cannot be ensured, the Friedman Test is chosen.

In `pyVHR` the Normality and Heteroskedasticity conditions are automatically checked *via* the Shapiro–Wilk Normality test and, depending on the Normality with Levene's test or Bartlett's tests for homogeneity of the data.

In the case of multiple comparisons (ANOVA/Friedman), a proper post-hoc analysis is required in order to establish the pairwise differences among the pipelines. Specifically, the Tukey post-hoc Test is adopted downstream to the rejection of the null hypothesis of ANOVA (the means of the populations are equal), while the Nemenyi post-hoc Test is used after the rejection of the Friedman's null hypothesis of equality of the medians of the samples.

Besides the significance of the differences, it is convenient to report their magnitude, too. The *effect size* can be computed *via* the Cohen's d in case of Normal of populations; the Akinshin's γ is used otherwise.

The two populations case

`pyVHR` automatically handles the above significance testing procedure within the `StatAnalysis()` class, by relying on the *Autorank* Python package (*Herbold, 2020*). `StatAnalysis()` ingests the results produced at the previous step and runs the appropriate statistical test on a chosen comparison metric:

```
from pyVHR.analysis.stats import StatAnalysis

st = StatAnalysis("/path/to/results.h5")
# -- box plot statistics (medians)
st.displayBoxPlot(metric='CCC')

#testing
st.run_stats(metric='CCC')
```

The output of the statistical testing procedure is reported as follows:

> The Shapiro–Wilk Test rejected the null hypothesis of normality for the populations POS ($p < 0.01$) and GREEN ($p < 0.01$). (...) the Wilcoxon's signed rank test has been chosen to determine the differences in the central tendency; median (MD) and median absolute deviation (MAD) are reported for each population. The test rejected the null hypothesis ($p < 0.01$) that population POS ($MD = 1.344 \pm 1.256$, $MAD = 0.688$) is not greater than population GREEN ($MD = 2.297 \pm 3.217$, $MAD = 1.429$). Hence, we assume that the median of POS is significantly larger than the median of GREEN with a large effect size ($\gamma = -0.850$).

As it can be observed, the appropriate statistical test for two non-normal populations has been properly selected. The Concordance Correlation Coefficient (CCC) for the method POS turned out to be significantly larger than the CCC of the method GREEN. Besides being significant, such difference is substantial, as witnessed by the *large* effect size.

The more-than-two populations case

Suppose now to structure the above `.cfg` in order to run three methods instead of two. This would be as simple as extending the **"BVP"** and **"Methods"** blocks as follows:

```
    ###BVP###
    [BVP]
    methods = ['POS', 'GREEN', 'CHROM']
    ...
    ###METHODS###
    [POS]
    ...

    [GREEN]
    ...

    [CHROM]
    ...
    name = CHROM
    pre_filtering = ['BPFILTER']
    post_filtering = ['BPFILTER']
```

Re-running the statistical analysis would yield the following output:

> The Shapiro–Wilk Test rejected the null hypothesis of normality for the populations CHROM ($p < 0.01$), POS ($p < 0.01$), and GREEN ($p < 0.01$). Given that more than two populations are present, and normality hypothesis has been rejected, the non-parametric Friedman test is chosen to inspect the eventual significant differences between the medians of the populations. The post-hoc Nemenyi test is then used

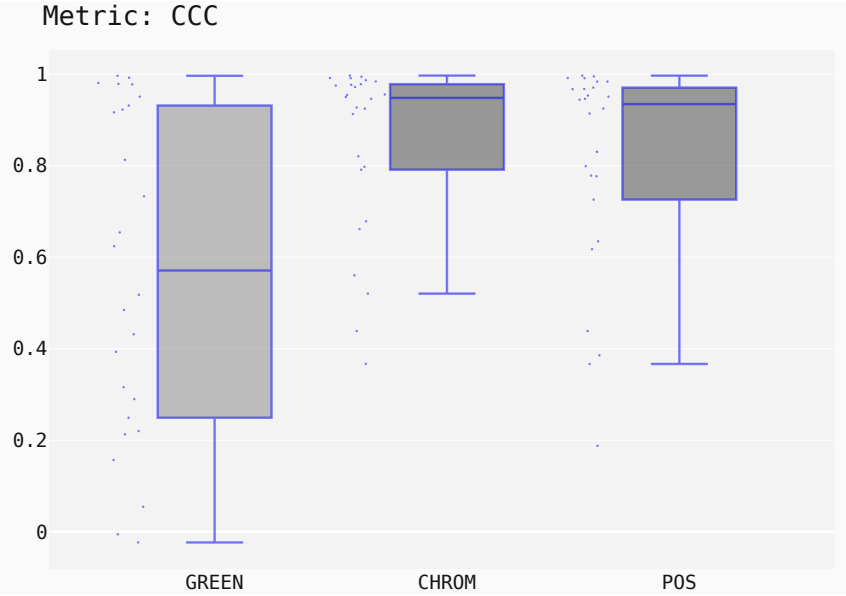

Figure 13 **Box plots showing the CCC values distribution for the POS, CHROM and GREEN methods on the UBFC2 dataset.**

Full-size DOI: 10.7717/peerjcs.929/fig-13

to determine which differences are significant. The Friedman test rejected the null hypothesis ($p < 0.01$) of equality of the medians of the populations CHROM ($MD = 1.263 \pm 1.688$, $MAD = 0.515$, $MR = 1.385$), POS ($MD = 1.344 \pm 1.513$, $MAD = 0.688$, $MR = 1.769$), and GREEN ($MD = 2.297 \pm 4.569$, $MAD = 1.429$, $MR = 2.846$). (…) the post-hoc Nemenyi test revealed no significant differences within the following groups: CHROM and POS, while other differences are significant.

Notably, the presence of more than two non-normal populations leads to the choice of the non-parametric Friedman Test as omnibus test to determine if there are any significant differences between the median values of the populations.

The box-plots showing the distributions of CCC values for all methods on the UBFC dataset is provided in Fig. 13, while the output of the post-hoc Nemenyi test can be visualized through the Critical Difference (CD) diagram (*Demšar, 2006*) shown in Fig. 14; CD Diagrams show the average rank of each method (higher ranks meaning higher average scores); models whose difference in ranks does not exceed the CD_α ($\alpha = 0.05$) are joined by thick lines and cannot be considered significantly different.

Comparing deep and traditional pipelines

How does a given DL-based rPPG method compares to the above mentioned traditional approaches? The following code snippet allows to run both the traditional and deep pipelines. The results are saved to the same folder, which is then fed as input to the `StatAnalysis` class; the `join_data =True` flag allows to merge the results yielded by the two pipelines, thus enabling the statistical comparison between the chosen methods.

```
from pyVHR.analysis.stats import StatAnalysis
```

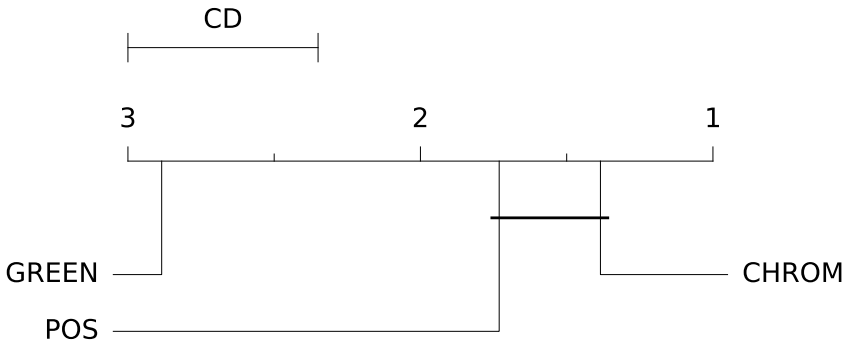

Figure 14 Results of the statistical assessment procedure. CD diagram displaying the results of the Nemenyi post-hoc test on the three populations (POS, CHROM and GREEN) of CCC values on the UBFC2 dataset.

Full-size DOI: 10.7717/peerjcs.929/fig-14

```
from pyVHR.analysis.pipeline import Pipeline, DeepPipeline

#Pipeline for Traditional Methods
traditional_pipe = Pipeline()
traditional_results = traditional_pipe.run_on_dataset('/path/to/trad_config.cfg')
traditional_results.saveResults("/path/to/results_folder/traditional_results.h5")

#Pipeline for Deep Methods
deep_pipe = DeepPipeline()
deep_results = deep_pipe.run_on_dataset('/path/to/deep_config.cfg')
deep_results.saveResults("/path/to/results_folder/deep_results.h5")

#Statistical Analysis
st = StatAnalysis("/path/to/results_folder/", join_data=True)
# -- box plot statistics
st.displayBoxPlot(metric='SNR')
#Significance testing on the SNR metric
st.run_stats(metric='SNR')
```

In this case, the Signal-to-Noise Ratio (SNR) has been chosen as comparison metric; Fig. 15 qualitatively displays the results of the comparison of the above mentioned traditional methods with the MTTS-CAN DL-based approach (*Liu et al., 2020*). The outcome of the statistical assessment is shown in the CD diagram of Fig. 16.

EXTENDING THE FRAMEWORK

Besides assessing built-in methods on public datasets included in the framework, the platform is conceived to allow the addition of new methods or datasets. This way, it is possible to assess a new proposal, comparing it against built-in methods, and testing it on either already included datasets or on new ones, this exploiting all the pre- and post-processing modules made available in pyVHR . The framework extension can be achieved following simple steps as described in the subsequent subsections.

Adding a new method

In this section we show how to add to the pyVHR framework either a new traditional or learning-based method called MY_NEW_METHOD.

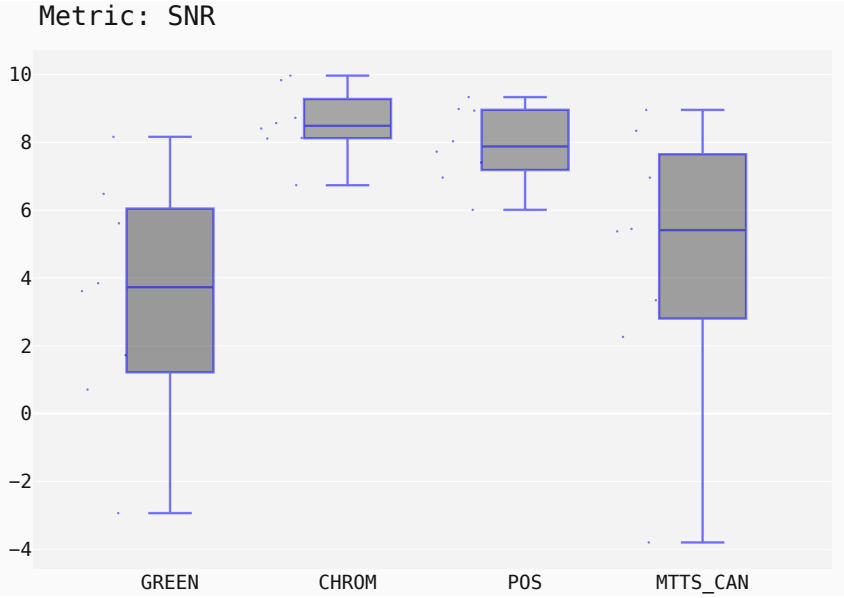

Figure 15 **Box plots showing the SNR values distribution for the POS, CHROM, MTTS-CAN and GREEN methods on the UBFC1 dataset.**

Full-size DOI: 10.7717/peerjcs.929/fig-15

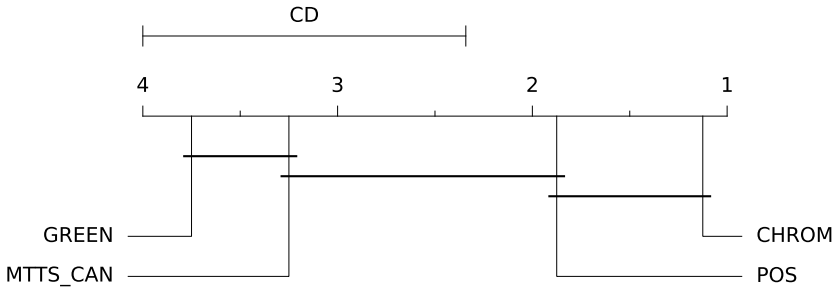

Figure 16 **Results of the statistical assessment procedure.** CD diagram displaying the results of the Nemenyi post-hoc test on the four populations (POS, CHROM, MTTS-CAN and GREEN) of SNR values on the UBFC1 dataset.

Full-size DOI: 10.7717/peerjcs.929/fig-16

In the first case, to exploit the pyVHR built-in modules the new function should receive as input a `signal` in the shape produced by the built-in pre-processing modules, together with some other parameters required by the method itself. Specifically, this results in a signature of the form:

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

where `signal` is a Numpy array in the form `(P, 3, K)`; `P` is the number of considered patches (it can be 1 if the holistic approach is used), 3 is the number of RGB Channels and `K` is the number of frames. `**kargs` refers to a dictionary that contains all the parameters required by the method at hand. A proper function implementing an rPPG method must return a BVP signal as a Numpy array of shape `(P,K)`.

In case of DL-based method, the new function should receive as input the raw `frames` as a Numpy array in the form `(H,W,3,K)`, where `H,W` denote the frame dimensions. The output of the new method could be either a BVP signal or the HR directly.

Accordingly, the signature becomes:

```
MY_NEW_METHOD(frames, fps)
```

Both for traditional and DL-based method, the function call `MY_NEW_METHOD` can now be embedded into the proper `Pipeline`, and assessed as described earlier. In order to do so, the `.cfg` file should be tweaked as follows:

```
[MY_NEW_METHOD]
path = 'path/to/module.py'
name = 'MY_NEW_METHOD'
...
```

Moreover, the `methods` block of the `.cfg` file is supposed to contain a specific listing describing `MY_NEW_METHOD`, providing the path to the Python module encoding the method and its function name.

Adding a new dataset

Currently `pyVHR` provides APIs for handling five datasets commonly adopted for the evaluation of rPPG methods, namely LGI-PPGI (*Pilz et al., 2018*), UBFC (*Bobbia et al., 2019*), PURE (*Stricker, Müller & Gross, 2014*), MAHNOB-HCI (*Soleymani et al., 2011*), and COHFACE (*Heusch, Anjos & Marcel, 2017a*). However, the platform allows to add new datasets favoring the method assessment on new data. A comprehensive list of the datasets that are typically employed for rPPG estimation and evaluation is reported in Table 3.

The framework conceives datasets as a hierarchy of classes (see Fig. 17) that allows to describe a new dataset by inheriting from the `Dataset` base class and implementing few methods for loading videos and ground truth PPG data.

Specifically, the following two functions should be supplied:

- a `loadFilenames()` function to load video files in a Python list; this function has no inputs and defines two class variables, namely `videoFilenames` and `BVPFilenames`. These are both Python lists containing, respectively, video and ground-truth BVP filenames from the dataset);
- a `readSigfile(filename)` function loading and returning the ground-truth BVP signal given a video filename.

The new dataset can then be included in the testing *via* the `.cfg` file as described in the paragraph *Dataset* of section 'The configuration (.cfg) file'. As for the addition of new method, also in case of adding a new dataset the `.cfg` file should be completed by specifying the path pointing to the new dataset class:

```
        [DATASET]
        dataset     = DatasetName
        path = /path/to/datasetClass/
        videodataDIR= /path/to/videos/
        BVPdataDIR  = /path/to/gtfiles/
        ...

```

Table 3 A list of datasets commonly used for rPPG. The left-most column collects the dataset names and introducing papers; second column, the number of subjects involved; third column, the task or condition under which data have been collected (Stationary: subject are asked to sit still; Interaction: emulation of a human–computer interaction scenario via a time sensitive mathematical game; Multiple: more than one condition has been considered while recording subjects, such as Steady, Talking, Head Motion *etc*; Physical Activities: subjects are recorded while performing activities such as speaking, rowing, exercising on a stationary bike *etc*; Stress Test: participants are subject to tasks with different levels of difficulty inspired by the Trier Social Stress Test; Emotion Elicitation: participants were shown fragments of movies and pictures apt at eliciting emotional reactions). In the last column, datasets whose handling APIs are currently available in pyVHR have been checked.

Dataset	Subjects	Task/Condition	pyVHR
UBFC1 (*Bobbia et al., 2019*)	8	Stationary	✓
UBFC2 (*Bobbia et al., 2019*)	42	Interaction	✓
PURE (*Stricker, Müller & Gross, 2014*)	10	Multiple	✓
LGI-PPGI (*Pilz et al., 2018*)	25 (6 available)	Multiple	✓
MAHNOB-HCI (*Soleymani et al., 2011*)	27	Emotion elicitation	✓
COHFACE (*Heusch, Anjos & Marcel, 2017b*)	40	Stationary	✓
UBFC-Phys (*Sabour et al., 2021*)	56	Stress test	×
AFRL (*Estepp, Blackford & Meier, 2014*)	25	Multiple	×
MMSE-HR (*Zhang et al., 2016*)	140	Simulating facial expressions	×
OBF (*Li et al., 2018*)	106	Multiple	×
VIPL-HR (*Niu et al., 2018*)	107	Multiple	×
ECG-Fitness (*Špetlík, Franc & Matas, 2018*)	17	Physical activities	×

CASE STUDY: DEEPFAKE DETECTION WITH PYVHR

DeepFakes are a set of DL based techniques allowing to create fake videos by swapping the face of a person by that of another. This technology has many diverse applications such as expression re-enactment (*Bansal et al., 2018*) or video de-identification (*Bursic et al., 2021*). However, in recent years the quality of deepfakes has reached tremendous levels of realism, thus posing a series of treats related to the possibility of arbitrary manipulation of identity, such as political propaganda, blackmailing, and fake news (*Mirsky & Lee, 2021*).

As a consequence, efforts have been devoted to the study and the development of methods allowing to discriminate between real and forged videos (*Tolosana et al., 2020*; *Mirsky & Lee, 2021*). Interestingly enough, one effective approach is represented by the exploitation of physiological information (*Hernandez-Ortega et al., 2020*; *Ciftci, Demir & Yin, 2020*; *Qi et al., 2020*) . Indeed, signals originating from biological action such as heart beat, blood flow, or breathing are expected to be (in large part) disrupted after face-swapping. Therefore, methods such as remote PPG can be adopted in order to evaluate their presence.

In the following, it is shown how pyVHR can be effectively employed to easily perform a DeepFake detection task. To this end, we rely on the FaceForensics++[4] dataset (*Rössler et al., 2019*) consisting of 1,000 original video sequences (mostly frontal face without occlusions) that have been manipulated with four automated face manipulation methods.

Each video, either original or swapped is fed as input to the pyVHR pipeline; then, the estimated BVPs and the predicted BPMs can be analyzed in order to detect DeepFakes.

[4]Available at GitHub: https://github.com/ondyari/FaceForensics.

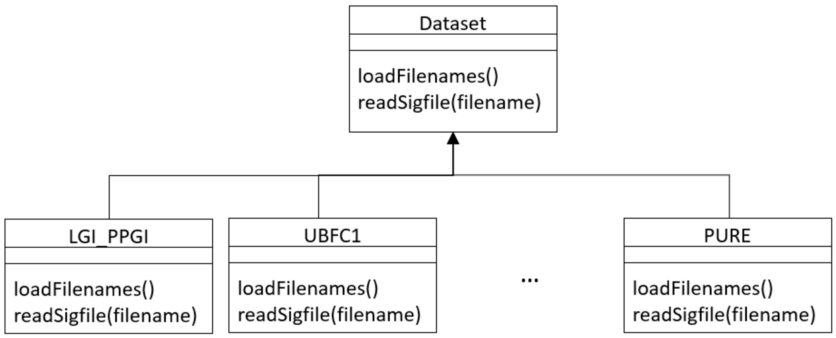

Figure 17 **Class diagram of dataset hierarchy of classes.**
Full-size DOI: 10.7717/peerjcs.929/fig-17

It is reasonable to imagine that the BVP signals estimated on original videos would have much lower complexity if compared with the swapped ones, due to the stronger presence of PPG related information that would be possibly ruled out during swapping procedures. As a consequence, BVP signals from DeepFakes would perhaps exhibit higher levels of noise and hence more complex behaviour.

There exist many ways of measuring the complexity of a signal; here we choose to compute the Fractal Dimension (FD) of BVPs; in particular the Katz's method (*Katz, 1988*) is employed.

The FD of the BVP estimated from the *i*th patch on the *k*-th time window (D_i^k) can be computed as *Katz (1988)*:

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

where L is the sum of distances between successive points, a is their average, and d is the maximum distance between the first point and any other point of the estimated BVP signal.

The FD associated to a given video can then be obtained *via* averaging:

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

Similarly, one could consider adopting the average Median Absolute Deviation (MAD) of the BPM predictions on a video as a predictor of the presence of DeepFakes:

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

Figure 18 shows how the FaceForensics++ videos lie in the 2-dimensional space defined by the average Fractal Dimension ($\hat{FD}$) of predicted BVPs using the `POS` method and the average MADs of BPM predictions ($\hat{MAD}$), when considering the original and swapped videos with the *FaceShifter* method.

It is easy to see how adopting these simple statistics on `pyVHR`'s predictions allows to discriminate original videos from DeepFakes. In particular, learning a baseline Linear

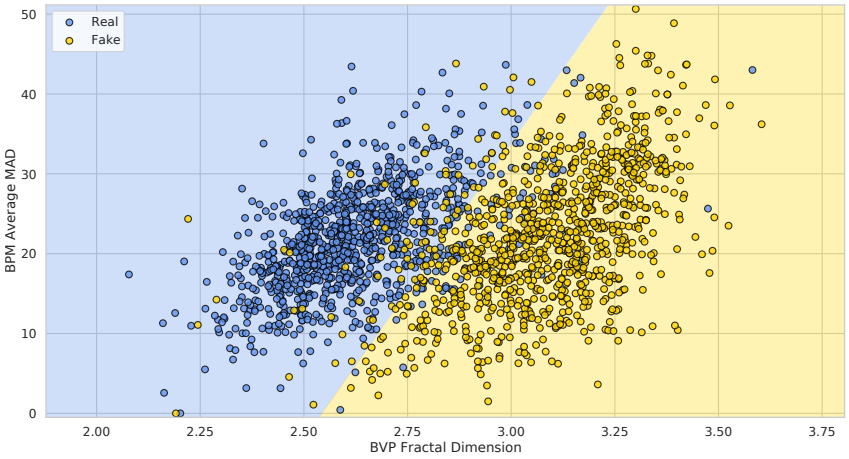

Figure 18 **Deepfake detection results.** The 1,000 FaceForensics++ original videos (blue) and their swapped versions (yellow) represented in the 2-D space of BVP Fractal Dimension *vs.* BPMs average MAD. The green and red half-spaces are simply learned *via* a linear SVM.
Full-size DOI: 10.7717/peerjcs.929/fig-18

SVM for the classification of Real *vs.* Fake videos generated by the *FaceShifter* method, yields an average 10-fold Cross-Validation Accuracy of 91.41% ± 2.05. This result is comparable with state of the art approaches usually adopting much more complex solutions.

CONCLUSIONS

In recent years, the rPPG-based pulse rate recovery has attracted much attention due to its promise to reduce invasiveness, while granting higher and higher precision in heart rate estimation. In particular, we have witnessed the proliferation of rPPG algorithms and models that accelerate the successful deployment in areas that traditionally exploited wearable sensors or ambulatory monitoring. These two trends, combined together, have fostered a new perspective in which advanced video-based computing techniques play a fundamental role in replacing the domain of physical sensing.

In this paper, in order to allow the rapid development and the assessment of new techniques, we presented an open and very general framework, namely `pyVHR` . It allows for a careful study of every step, and no less important, for a sound comparison of methods on multiple datasets.

`pyVHR` is a re-engineered version of the framework presented in *Boccignone et al. (2020a)* but exhibiting substantial novelties:

- Ease of installation and use.
- Two distinct pipelines for either traditional or DL-based methods.
- Holistic or patch processing for traditional approaches.
- Acceleration by GPU architectures.
- Ease of extension (adding new methods or new datasets).

The adoption of GPU support allows the whole process to be safely run in real-time for 30 fps HD videos and an average speedup ($time_{seq}/time_{parall}$) of around 5.

Besides addressing the challenges of remote Heart Rate monitoring, we also expect that this framework will be useful to researchers and practitioners from various disciplines when dealing with new problems and building new applications leveraging rPPG technology.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

This work was supported by the University of Milan through the APC initiative. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the authors:
The University of Milan through the APC initiative.

Competing Interests

The authors declare there are no competing interests.

Author Contributions

- Giuseppe Boccignone conceived and designed the experiments, performed the experiments, authored or reviewed drafts of the paper, and approved the final draft.
- Donatello Conte and Giuliano Grossi analyzed the data, performed the computation work, authored or reviewed drafts of the paper, and approved the final draft.
- Vittorio Cuculo performed the experiments, performed the computation work, prepared figures and/or tables, and approved the final draft.
- Alessandro D'Amelio conceived and designed the experiments, performed the computation work, prepared figures and/or tables, and approved the final draft.
- Raffaella Lanzarotti conceived and designed the experiments, analyzed the data, authored or reviewed drafts of the paper, and approved the final draft.
- Edoardo Mortara performed the experiments, performed the computation work, prepared figures and/or tables, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

The code is available at GitHub: https://github.com/phuselab/pyVHR.

The UBFC data used in the experiments is available at: https://sites.google.com/view/ybenezeth/ubfcrppg.

The FaceForensics++ dataset used in the case study is available at GitHub: https://github.com/ondyari/FaceForensics.

The LGI-PPGI dataset is available at GitHub: https://github.com/partofthestars/LGI-PPGI-DB.

REFERENCES

Aarts LA, Jeanne V, Cleary JP, Lieber C, Nelson JS, Oetomo SB, Verkruysse W. 2013. Non-contact heart rate monitoring utilizing camera photoplethysmography in the neonatal intensive care unit A pilot study. *Early Human Development* **89(12)**:943–948 DOI 10.1016/j.earlhumdev.2013.09.016.

Balakrishnan G, Durand F, Guttag J. 2013. Detecting pulse from head motions in video. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Piscataway: IEEE, 3430–3437.

Bansal A, Ma S, Ramanan D, Sheikh Y. 2018. Recycle-gan: unsupervised video retargeting. In: *Proceedings of the european conference on computer vision (ECCV)*. 119–135.

Benavoli A, Corani G, Demšar J, Zaffalon M. 2017. Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. *The Journal of Machine Learning Research* **18(1)**:2653–2688.

Benezeth Y, Li P, Macwan R, Nakamura K, Gomez R, Yang F. 2018. Remote heart rate variability for emotional state monitoring. In: *2018 IEEE EMBS international conference on biomedical & health informatics (BHI)*. Piscataway: IEEE, 153–156.

Blazek V, Schultz-Ehrenburg U. 1996. *Quantitative Photoplethysmography: basic facts and examination tests for evaluating peripheral vascular funktions*. Düsseldorf: VDI-Verlag.

Bobbia S, Macwan R, Benezeth Y, Mansouri A, Dubois J. 2019. Unsupervised skin tissue segmentation for remote photoplethysmography. *Pattern Recognition Letters* **124**:82–90 DOI 10.1016/j.patrec.2017.10.017.

Boccignone G, Conte D, Cuculo V, DAmelio A, Grossi G, Lanzarotti R. 2020a. An open framework for remote-PPG methods and their assessment. *IEEE Access* **8**:216083–216103 DOI 10.1109/ACCESS.2020.3040936.

Boccignone G, de'Sperati C, Granato M, Grossi G, Lanzarotti R, Noceti N, Odone F. 2020b. Stairway to Elders: bridging space, time and emotions in their social environment for wellbeing. In: *ICPRAM*. 548–554.

Bursic S, D'Amelio A, Granato M, Grossi G, Lanzarotti R. 2021. A quantitative evaluation framework of video de-identification methods. In: *2020 25th international conference on pattern recognition (ICPR)*. 6089–6095.

Chen W, McDuff D. 2018. Deepphys: video-based physiological measurement using convolutional attention networks. In: *Proceedings of the european conference on computer vision (ECCV)*. 349–365.

Cheng C-H, Wong K-L, Chin J-W, Chan T-T, So RH. 2021. Deep learning methods for remote heart rate measurement: a review and future research agenda. *Sensors* **21(18)**:6296 DOI 10.3390/s21186296.

Ciftci UA, Demir I, Yin L. 2020. 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)*. Piscataway: IEEE, 1–10.

De Haan G, Jeanne V. 2013. Robust pulse rate from chrominance-based rPPG. *IEEE Transactions on Biomedical Engineering* **60(10)**:2878–2886 DOI 10.1109/TBME.2013.2266196.

De Haan G, Van Leest A. 2014. Improved motion robustness of remote-PPG by using the blood volume pulse signature. *Physiological Measurement* **35(9)**:1913–1926 DOI 10.1088/0967-3334/35/9/1913.

Demšar J. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* **7**:1–30.

Eisinga R, Heskes T, Pelzer B, Te Grotenhuis M. 2017. Exact p-values for pairwise comparison of Friedman rank sums, with application to comparing classifiers. *BMC Bioinformatics* **18(1)**:68 DOI 10.1186/s12859-017-1486-2.

Estepp JR, Blackford EB, Meier CM. 2014. 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)*. Piscataway: IEEE, 1462–1469.

Gideon J, Stent S. 2021. 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*. Piscataway: IEEE, 3995–4004.

Lugaresi C, Tang J, Nash H, McClanahan C, Uboweja E, Hays M , et al, Grundmann M. 2019. Mediapipe: a framework for building perception pipelines. ArXiv preprint. arXiv:1906.08172.

Graczyk M, Lasota T, Telec Z, Trawiński B. 2010. Nonparametric statistical analysis of machine learning algorithms for regression problems. In: Setchi R, Jordanov I, Howlett RJ, Jain LC, eds. *Knowledge-based and intelligent information and engineering systems*. Berlin, Heidelberg: Springer, 111–120.

Herbold S. 2020. Autorank: a python package for automated ranking of classifiers. *Journal of Open Source Software* **5(48)**:2173 DOI 10.21105/joss.02173.

Hernandez-Ortega J, Tolosana R, Fierrez J, Morales A. 2020. Deepfakeson-phys: deepfakes detection based on heart rate estimation. ArXiv preprint. arXiv:2010.00400.

Hertzman AB. 1937. Photoelectric plethysmography of the fingers and toes in man. In: *Proceedings of the society for experimental biology and medicine*. 529–534 DOI 10.3181/00379727-37-9630.

Heusch G, Anjos A, Marcel S. 2017a. A reproducible study on remote heart rate measurement. ArXiv preprint. arXiv:1709.00962.

Heusch G, Anjos A, Marcel S. 2017b. A reproducible study on remote heart rate measurement. ArXiv preprint. arXiv:1709.00962.

Humphreys K, Ward T, Markham C. 2007. Noncontact simultaneous dual wavelength photoplethysmography: a further step toward noncontact pulse oximetry. *Review of Scientific Instruments* **78(4)**:044304 DOI 10.1063/1.2724789.

Katz MJ. 1988. Fractals and the analysis of waveforms. *Computers in Biology and Medicine* **18(3)**:145–156 DOI 10.1016/0010-4825(88)90041-8.

Koelstra S, Muhl C, Soleymani M, Lee J-S, Yazdani A, Ebrahimi T, Pun T, Nijholt A, Patras I. 2011. Deap: a database for emotion analysis; using physiological signals. *IEEE Transactions on Affective Computing* **3(1)**:18–31.

Lawrence I, Lin K. 1989. A concordance correlation coefficient to evaluate reproducibility. *Biometrics* **45**:255–268.

Lewandowska M, Rumiński J, Kocejko T, Nowak J. 2011. 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)*. 405–410.

Li X, Alikhani I, Shi J, Seppanen T, Junttila J, Majamaa-Voltti K, Tulppo M, Zhao G. 2018. 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)*. Piscataway: IEEE, 242–249.

Liu X, Fromm J, Patel S, McDuff D. 2020. Multi-task temporal shift attention networks for on-device contactless vitals measurement. *Advances in Neural Information Processing Systems* **33**:19400–19411.

Liu X, Jiang Z, Fromm J, Xu X, Patel S, McDuff D. 2021. MetaPhys: few-shot adaptation for non-contact physiological measurement. In: *Proceedings of the conference on health, inference, and learning*. 154–163.

McDuff D. 2021. Camera measurement of physiological vital signs. ArXiv preprint. arXiv:2111.11547.

McDuff D, Blackford E. 2019. 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)*. Piscataway: IEEE, 6521–6524.

McDuff DJ, Estepp JR, Piasecki AM, Blackford EB. 2015. A survey of remote optical photoplethysmographic imaging methods. In: *2015 37th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*. Piscataway: IEEE, 6398–6404.

McDuff D, Gontarek S, Picard R. 2014. Remote measurement of cognitive stress via heart rate variability. In: *2014 36th annual international conference of the IEEE engineering in medicine and biology society*. Piscataway: IEEE, 2957–2960.

Mirsky Y, Lee W. 2021. The creation and detection of deepfakes: a survey. *ACM Computing Surveys (CSUR)* **54(1)**:1–41.

Ni A, Azarang A, Kehtarnavaz N. 2021. A review of deep learning-based contactless heart rate measurement methods. *Sensors* **21(11)**:3719 DOI 10.3390/s21113719.

Niu X, Han H, Shan S, Chen X. 2018. VIPL-HR: a multi-modal database for pulse estimation from less-constrained face video. In: *Asian conference on computer vision*. 562–576.

Niu X, Shan S, Han H, Chen X. 2019. Rhythmnet: end-to-end heart rate estimation from face via spatial-temporal representation. *IEEE Transactions on Image Processing* **29**:2409–2423.

Nowara E, McDuff D, Veeraraghavan A. 2020. The benefit of distraction: denoising remote vitals measurements using inverse attention. ArXiv preprint. arXiv:2010.07770.

Pilz C. 2019. On the vector space in photoplethysmography imaging. In: *Proceedings of the IEEE/CVF international conference on computer vision workshops*. Piscataway: IEEE.

Pilz CS, Zaunseder S, Krajewski J, Blazek V. 2018. 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*. Piscataway: IEEE, 1254–1262.

Poh M-Z, McDuff DJ, Picard RW. 2010. Non-contact, automated cardiac pulse measurements using video imaging and blind source separation. *Optics Express* **18(10)**:10762–10774 DOI 10.1364/OE.18.010762.

Qi H, Guo Q, Juefei-Xu F, Xie X, Ma L, Feng W, Liu Y, Zhao J. 2020. DeepRhythm: exposing deepfakes with attentional visual heartbeat rhythms. In: *Proceedings of the 28th ACM international conference on multimedia*. 4318–4327.

Ramírez GA, Fuentes O, Crites SL, Jimenez M, Ordonez J. 2014. Color analysis of facial skin: detection of emotional state. In: *2014 IEEE conference on computer vision and pattern recognition workshops*. Piscataway: IEEE, 474–479.

Rössler A, Cozzolino D, Verdoliva L, Riess C, Thies J, Nießner M. 2019. FaceForensics++: learning to detect manipulated facial images. In: *International conference on computer vision (ICCV)*.

Rouast PV, Adam MT, Cornforth DJ, Lux E, Weinhardt C. 2017. Using contactless heart rate measurements for real-time assessment of affective states. In: *Information Systems and Neuroscience*. Cham, Switzerland: Springer, 157–163.

Rouast PV, Adam MTP, Chiong R, Cornforth D, Lux E. 2018. Remote heart rate measurement using low-cost RGB face video: a technical literature review. *Frontiers of Computer Science* **12(5)**:858–872 DOI 10.1007/s11704-016-6243-6.

Sabour RM, Benezeth Y, De Oliveira P, Chappe J, Yang F. 2021. Ubfc-phys: a multimodal database for psychophysiological studies of social stress. *IEEE Transactions on Affective Computing* Epub ahead of print Feb 3 2021 DOI 10.1109/TAFFC.2021.3056960.

Soleymani M, Lichtenauer J, Pun T, Pantic M. 2011. A multimodal database for affect recognition and implicit tagging. *IEEE Transactions on Affective Computing* **3(1)**:42–55.

Špetlík R, Franc V, Matas J. 2018. Visual heart rate estimation with convolutional neural network. In: *Proceedings of the british machine vision conference, Newcastle, UK*. 3–6.

Stricker R, Müller S, Gross H-M. 2014. Non-contact video-based pulse rate measurement on a mobile service robot. In: *23rd IEEE international symposium on robot and human interactive communication*. Piscataway: IEEE, 1056–1062.

Tarassenko L, Villarroel M, Guazzi A, Jorge J, Clifton DA, Pugh C. 2014. Non-contact video-based vital sign monitoring using ambient light and auto-regressive models. *Physiological Measurement* **35(5)**:807–831 DOI 10.1088/0967-3334/35/5/807.

Tolosana R, Vera-Rodriguez R, Fierrez J, Morales A, Ortega-Garcia J. 2020. Deepfakes and beyond: a survey of face manipulation and fake detection. *Information Fusion* **64**:131–148 DOI 10.1016/j.inffus.2020.06.014.

Torralba A, Efros AA. 2011. Unbiased look at dataset bias. In: *CVPR 2011*. 1521–1528.

Unakafov AM. 2018. Pulse rate estimation using imaging photoplethysmography: generic framework and comparison of methods on a publicly available dataset. *Biomedical Physics & Engineering Express* **4(4)**:045001 DOI 10.1088/2057-1976/aabd09.

Verkruysse W, Svaasand LO, Nelson JS. 2008. Remote plethysmographic imaging using ambient light. *Optics Express* **16(26)**:21434–21445 DOI 10.1364/OE.16.021434.

Wang W, den Brinker AC, Stuijk S, de Haan G. 2016. Algorithmic principles of remote PPG. *IEEE Transactions on Biomedical Engineering* **64(7)**:1479–1491.

Wang W, Stuijk S, De Haan G. 2015. A novel algorithm for remote photoplethysmography: spatial subspace rotation. *IEEE Transactions on Biomedical Engineering* **63(9)**:1974–1984.

Wieringa FP, Mastik F, Steen AFWvd. 2005. Contactless multiple wavelength photoplethysmographic imaging: a first step toward "SpO2 Camera"Technology. *Annals of Biomedical Engineering* **33(8)**:1034–1041 DOI 10.1007/s10439-005-5763-2.

Yu Z, Li X, Niu X, Shi J, Zhao G. 2020. Autohr: a strong end-to-end baseline for remote heart rate measurement with neural searching. *IEEE Signal Processing Letters* **27**:1245–1249 DOI 10.1109/LSP.2020.3007086.

Yu Z, Shen Y, Shi J, Zhao H, Torr P, Zhao G. 2021. PhysFormer: facial video-based physiological measurement with temporal difference transformer. ArXiv preprint. arXiv:2111.12082.

Yu C, Wang J, Peng C, Gao C, Yu G, Sang N. 2018. Bisenet: bilateral segmentation network for real-time semantic segmentation. In: *Proceedings of the European conference on computer vision (ECCV)*. 325–341.

Zhang Z, Girard JM, Wu Y, Zhang X, Liu P, Ciftci U, Canavan S, Reale M, Horowitz A, Yang H. 2016. Multimodal spontaneous emotion corpus for human behavior analysis. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Piscataway: IEEE, 3438–3446.