ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CVL Computer Vision Lab

Robust rPPG Extraction from Unstable Handheld Videos in Real-World Conditions

Evaluating Stabilization and Postprocessing Pipelines for Physiological Signal Recovery

Master's Thesis

Florian Moll
Department of Information Technology and Electrical Engineering

Advisors: Sara Sangalli, Luzian Bieri, Michael Rusterholz
Supervisor: Prof. Dr. Ender Konukoglu

August 30, 2025

Abstract

Remote Photoplethysmography (rPPG) enables non-contact estimation of a person's physiological signals from standard video recordings, offering valuable potential for time-critical applications such as emergency response. In this work, we address the challenge of extracting reliable remote Photoplethysmography (rPPG) signals from unstable, handheld video footage, as commonly encountered in real-world scenarios. We propose a stabilization-based preprocessing pipeline to reduce motion artifacts and improve robustness under camera shake. Several neural network architectures, including spatio-temporal convolutional models and temporal shift modules, are evaluated for their performance on both stable and unstable video sequences. We further investigate the impact of spatial compression, demonstrating that the proposed approach maintains high signal fidelity even under reduced spatial resolution and aggressive video encoding. In addition, we incorporate uncertainty estimation into the prediction pipeline, enabling the assessment of model confidence and thus enhancing reliability in safety-critical settings. The system is validated in the context of an emergency first responder scenario, where a smartphone camera is used to record an individual lying on the ground, and the heart rate is estimated from the extracted rPPG signal. Results indicate that the combination of video stabilization, robust neural architectures, and uncertainty-aware predictions substantially improves rPPG extraction performance in uncontrolled, real-world environments.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Sara Sangalli and Luzian Bieri, for their inspiring guidance, continuous support, and valuable feedback throughout my research journey. Their expertise and encouragement have been instrumental in shaping this thesis.

My heartfelt thanks also go to Professor Dr. Ender Konukoglu, whose insights and academic leadership provided crucial direction and depth to my work.

Furthermore, I gratefully acknowledge Mr. Pieter-Jan Toye for providing access to the Vital Videos dataset—an invaluable resource for this research.

Lastly, I extend my appreciation to my family and friends for their unwavering support, patience, and encouragement throughout this process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The reliable extraction of physiological signals from video recordings, known as remote photoplethysmography (rPPG), has emerged as a promising approach for non-contact health monitoring. rPPG algorithms estimate vital signs such as heart rate from subtle color variations in the skin caused by blood flow. While this technique eliminates the need for dedicated contact sensors, it faces a variety of challenges in real-world scenarios, including subject motion, lighting changes, and video compression artifacts [10] [21]. These factors can significantly degrade signal quality and, in turn, the accuracy of downstream analyses. Furthermore, rPPG pipelines can be computationally demanding [33], which limits their applicability on devices with constrained resources, such as mobile phones or embedded systems.

This thesis addresses these challenges by systematically evaluating and improving a standard rPPG processing pipeline. First, different video stabilization techniques are tested to mitigate motion artifacts, with the best-performing method identified through validation experiments. Second, the study explores uncertainty estimation in rPPG models and their conformalization, providing calibrated measures of prediction reliability. Third, the robustness of the pipeline is assessed under varying degrees of video compression, comparing the two most widely used codecs, H.264 and H.265. Finally, computational efficiency improvements are investigated by varying the input resolution and analyzing the trade-off between model accuracy and processing cost.

The remainder of this thesis is organized as follows:

- **Chapter 2** reviews the theoretical background of rPPG signal extraction, video stabilization, uncertainty estimation, and relevant deep learning concepts.
- **Chapter 3** describes the datasets used, including both synthetic and real-world recordings, as well as the preprocessing steps. Here also the processing pipeline and the model architectures developed for this study are introduced.
- **Chapter 4** outlines the experimental methodology for stabilization, uncertainty calibration, compression robustness, and computational efficiency analysis.
- **Chapter 5** discusses some experimental results and the experiments conducted.
- **Chapter 6** concludes the work with a summary of findings, their implications, and directions for future research.
- **Chapter Appendix** presents the detailed experimental results and also introduce the data format, that was used to record datasets in this work.

Chapter 2

Related Work

rPPG extraction has been an active research topic for several years. Since rPPG extraction relies on subtle color changes in the skin caused by blood volume variations, the choice of color channel can strongly influence signal quality. Early works demonstrated that standard Red Green Blue (RGB) cameras are suitable for this task and that the green channel contains the most useful information for extracting pulse signals [29]. This effect arises from the fact that light with different wavelengths penetrates the skin to different depths [2]. Blue light does not penetrate deep enough to reach the capillaries beneath the skin surface, while red light penetrates too deeply and thus passes through more tissue, where it becomes dominated by noise [31]. Green wavelengths, in contrast, hit a "sweet spot": they penetrate deeply enough to capture the pulsatile blood volume changes in the capillaries but not so deep as to accumulate excessive noise [31]. Furthermore, the absorption spectrum of hemoglobin peaks in the green spectral range [29], which further amplifies the physiological signal. Another practical reason why the green channel is especially effective might be the use of the Bayer filter in most consumer cameras, where each pixel has twice as many green detectors as red or blue ones [3].

Over time, a wide range of methods have been developed for rPPG extraction, which can be grouped into the following categories:

- **Deterministic algorithms:**
  - **Color-difference**: [10] Color-difference tracking methods that separate signal from noise.
  - **Eulerian Motion Magnification**: [35] based on image pyramids to enhance subtle temporal color changes.
- **Machine learning approaches:**
  - **Vision Transformers:** [39] introduced temporal difference self-attention, replacing image patches with video patches aggregated across consecutive frames. [40] extended this with hierarchical temporal structures to capture both local and global dependencies.
  - **3D-CNN architectures:** 3D-CNNs were successfully applied in [38], but they are computationally heavy. Efficiency can be improved with temporal shift modules [18], which approximate 3D convolutions at lower cost.
  - **Two-branch networks:** [17] proposed a dual-branch model with one branch for appearance (e.g., skin segmentation) and one for motion (e.g., rPPG extraction).

Of course these approaches will not work without proper preprocessing. In [19] a standardized preprocessing pipeline for videos was outlined that will be used throughout this work.

Studies have shown that several unique challenges arise in the task of rPPG signal extraction. One important factor is the skin tone of the subject. Darker skin tones—which can be objectively measured using the Fitzpatrick scale [13]—reduce the strength of the reflected light signal that carries physiological information. Prior works have often used datasets such as the VitalVideos dataset [28], which contains a diverse set of participants spanning all Fitzpatrick skin types, to study this effect. As a result, extracting the subtle color changes caused by blood volume variations becomes more difficult. This challenge has been observed in both classical signal processing approaches [35] and modern deep learning models [22], indicating that it arises from the optical properties of human skin rather than from a specific algorithm.

Another well-known difficulty is video instability. Unstable or jittery camera motion introduces strong artifacts into the extracted signal, often overwhelming the small physiological variations of interest. Previous studies [10] have shown that even minor camera movements can significantly reduce signal quality, highlighting the importance of stabilization or robust preprocessing steps.

Finally, noise in the video stream poses an additional challenge. High levels of sensor noise, compression artifacts, or low-light recording conditions add fluctuations to pixel intensities, which can mask the desired physiological signal. It has been demonstrated that such noise substantially degrades the accuracy of rPPG extraction methods [35] and that data-driven approaches can benefit from artificial motion augmentation to improve robustness [23].

Together, these factors emphasize that robust rPPG extraction requires careful consideration of real-world variability in skin properties, camera motion, and recording quality.

It is known that face detection is a helpful building block in the rPPG extraction task, because it removes unnecessary parts of the video, but it also introduces challenges [4]: The predicted face bounding boxes are not always perfectly aligned, which causes additional jitter on top of the inherent instability of the video camera [4]. To address this issue, researchers have proposed a temporal median filter applied to the detected face boxes. While this approach reduces the jitter, it does not resolve the underlying instability of the video recording itself.

The general video stabilization problem (independent of rPPG extraction) has been extensively studied, leading to a wide variety of approaches:

- **Optical flow with affine motion models:** A classic stabilization approach that aligns frames based on dense motion estimation [8].

- **Feature matching techniques:** Keypoints are detected and matched across frames to estimate transformations [37]. While effective, this method is computationally demanding.

- **Efficient feature-matching via convolution-FFT equivalence:** Recently, researchers have proposed stabilization methods based on the equivalence between convolution and multiplication in the Fourier domain. This allows efficient template matching and reduces the computational cost significantly [6].

- **Online template adaptation:** In addition, updating the feature template online during video playback has been shown to improve robustness to changes in appearance, lighting, or occlusion [6].

Another important task in contactless heart rate measurement is to accually find the heart rate from a PPG signal. Most rPPG algorithms focus on getting an accurate measurement of the PPG signal [38] [39] [40]. Studies have already shown that Convolutional Neural Networks (CNNs) are suitable for extracting the heart rate from PPG signals. For example [11] has shown that 1D-CNNs show an error rate of up to 4.63 on the PPG-DaLiA dataset.

Chapter 3

Materials and Methods

3.1 The Data

When working with artificial intelligence, having high-quality data is crucial. Both the quality and quantity of the data largely determine what a model can learn and how well it will perform. Poor data quality or insufficient data can severely limit the achievable results, regardless of the model's complexity. The aim of this section is to provide the reader with an overview of the datasets used in this work, including details on how the data was collected, the types of recordings, and any preprocessing or annotation steps that were performed. This context is essential for understanding the subsequent experiments and the limitations of the models trained on these datasets.

3.1.1 Datasets

We utilize several datasets in this study:

1. **Vital Videos** [28]: Consists of 202 videos, each 30 s long, accompanied by synchronized PPG measurements. Additional participant statistics include blood pressure, Fitzpatrick skin type, age, and gender. Data was collected across seven locations (Figure 3.1). This dataset is used for training, validation, and testing (see Section 3.1.2).

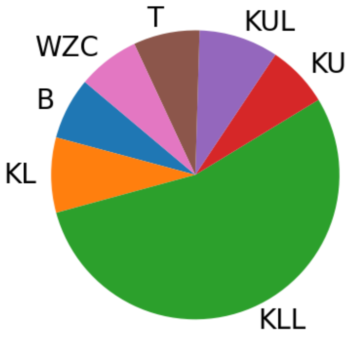

Figure 3.1: Location distribution of the Vital Videos dataset.

2. **UBFC** [5]: A public dataset for rPPG extraction, consisting of 44 videos of varying lengths with synchronized PPG recordings. It does not include Fitzpatrick skin type labels, though most participants appear to belong to lighter skin types.

3. **Ad Banner Videos**: Collected for this project, consisting of 17 videos of advertisement banners recorded with a Google Pixel 8 smartphone at Bahnhofstrasse and the Altstetten railway station in Zurich. The goal is to assess model behavior when presented with invalid input data (i.e., no heartbeat signal).

4. **Distribution Shift Videos**: Also collected for this project, comprising 13 videos of three subjects recorded with a Google Pixel 8 and a Samsung Galaxy S10 under different camera settings and lighting conditions. The camera was placed on a stable surface, and subjects were instructed to remain still. No ground-truth PPG measurements were recorded. This dataset tests model robustness to distribution shifts.

5. **Realistic Videos**: Recorded under natural sunlight with a handheld mobile phone camera, without stabilization. Ground-truth PPG measurements were collected. This dataset is used to assess performance under realistic conditions.

6. **Deepstab** [34]: Contains pairs of stabilized and unstabilized videos recorded side by side. The dataset is used here to extract realistic camera motion, which can then be overlaid on stable videos to generate artificially unstable recordings.

3.1.2 Data split

When training on a dataset, it should be large enough to allow the model to generalize effectively while also providing accurate ground truth labels. According to these criteria, in this work, two datasets are suitable for training, validation, and testing: the VitalVideos dataset and the UBFC dataset. The UBFC dataset is merged into the training set due to its lack of participants with darker Fitzpatrick skin types. We consider it important to ensure that the validation and test sets, in particular, contain participants with darker skin types.

Furthermore, to prevent data leakage between the training, validation, and test sets, the VitalVideos dataset is split by recording location. Specifically, recordings from the KLL, KL, and B locations are used for training; recordings from WZC and T are used for validation; and recordings from KUL and KU are used for testing.

3.1.3 Recording procedure

Accurate ground-truth acquisition requires precise synchronization between the recorded video and the corresponding PPG signal. Without synchronization, the PPG and the rPPG signals may be phase-shifted, leading to incorrect analysis.

This synchronization problem has already been taken into account by most public datasets for rPPG extraction, but for the datasets, that were specifically recorded for the purpose of this work, a proper solution needs to be found. In the data we recorded, synchronization is achieved by recording both a video and a PPG signal, with each video frame and each PPG sample carrying an associated timestamp. The PPG signal can then be resampled to match the timestamps of the video frames.

Video recordings are captured with a Google Pixel 8 smartphone using the Android application *Timestamp Camera*, which overlays the current timestamp on each video frame. The PPG signal is recorded using a Beurer PO 80 pulse oximeter connected to a computer via a USB cable. The USB communication between the oximeter and the computer is intercepted using Wireshark, allowing extraction of timestamps for each PPG sample.

A challenge with this setup is that the video recording device (the smartphone) and the PPG recording device (the computer) use independent clocks. To compensate, timestamps from the smartphone are periodically retrieved via Android Debug Bridge (ADB), while timestamps from the computer are obtained via the Python API. Both are logged to a file for later alignment. The detailed storage format of the dataset is described in Appendix A.

Since the training data only contains ground truth for the PPG signal and not directly for the heart rate, we performed manual labeling of the PPG signals. For this purpose, a Python tool was used, which first initializes a peak-finding algorithm from the `scipy` package and then allows the user to manually adjust the peak labeling. In this way, a new ground truth is created for the positions of the peaks in the PPG signals, from which the heart rate can be easily inferred.

3.2 Training device and software

All training and testing are performed on a desktop computer equipped with an Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz and two built-in Graphics Processing Units (GPUs), namely the NVIDIA GeForce GTX TITAN X and NVIDIA TITAN RTX. For training, the RPPG-Toolbox [19], with some modifications, is utilized.

3.3 The Processing Pipeline

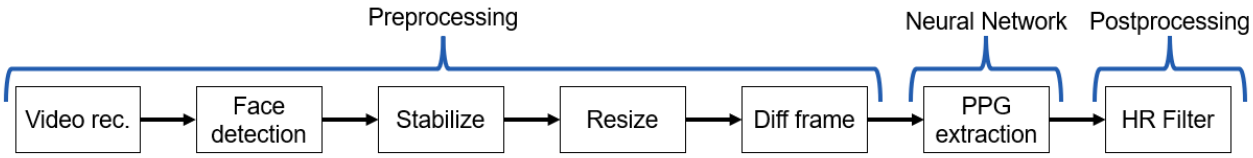

Figure 3.2: A visualization of the steps of the processing pipeline.

The rPPG processing pipeline, which is illustrated in figure 3.2, extracts the photoplethysmographic signal from video in a sequence of stages: first, the face is detected and cropped; next, the face region is stabilized to reduce jitter and motion; the frames are then resized to meet model input specifications; optionally, difference frames are computed and the data is standardized; a neural network extracts the PPG signal. This pipeline has been described that way in [19]. In our implementation, two additional steps are included: postprocessing filters to enhance signal quality, and the final calculation of the heart rate in beats per minute from the PPG signal. Each stage is executed sequentially to progressively refine the input for accurate PPG estimation.

3.3.1 Face Detectors

A face detector is an essential component of the processing pipeline for rPPG extraction. Since the face contains the relevant information, throwing away the background, while keeping the face, removes unnecessary regions and therefore improves the algorithm's performance. This is the first stage of the processing pipeline used in this work. Three different face detection algorithms, outlined below, are compared in this study.

1. **Haar Cascade** [30] – One of the earliest face detection methods based on Haar-like features and a cascade of classifiers. It is computationally efficient and suitable for mobile devices with limited

processing power, but generally performs worse than modern AI-based detectors, especially under challenging conditions [15].

2. **YuNet** [36] – A lightweight CNN-based detector optimized for resource-constrained devices. It provides a good balance between detection accuracy and inference speed, making it suitable as a baseline for evaluations and real-time mobile deployment.

3. **YOLO5Face** [24] – Based on the YOLOv5 architecture, this detector achieves high detection accuracy and fast inference on powerful hardware. It is more computationally demanding than Haar Cascade and YuNet, limiting its practicality on devices with limited processing capabilities.

3.3.2 Stabilizers

After face detection, the bounding boxes are often subject to jitter, and the video itself may not be perfectly stable in the first place. To address this, we compare four stabilization approaches:

1. **Baseline (Temporal Mean)** – As used in [40], this method computes the temporal mean of all detected face boxes and applies the resulting average box to every frame. It reduces jitter from the face detector but does not compensate for instability in the video itself. This serves as a minimal-stabilization baseline.

2. **Template Matching** – This approach initializes with the face detector output from the first frame and tracks it in subsequent frames using template matching with the normalized correlation coefficient method provided by OpenCV (TM_COEFF_NORMED). It allows precise alignment based on the initial face template.

3. **Optical Flow** – The optical flow approach estimates motion using the Gunnar Farnebäck algorithm [12]. The average flow within the face region is used to compute the mean displacement of the face, enabling alignment of each subsequent frame to the current one. This method can handle both detector jitter and small video movements.

4. **MOSSE** – The Minimum Output Sum of Squared Error (MOSSE) method [6] is similar to the template matching approach, but it leverages the FFT for improved efficiency. Additionally, the template is updated online to account for small changes of the object over time.

3.3.3 Resize

After the stabilization step, further preprocessing is required to adapt the videos to the input requirements of the Artificial Intelligence (AI) models. This includes both spatial and temporal adjustments:

- **Spatial resizing:** Each frame must be resized to a fixed spatial resolution, as specified for each model in Table 3.1. During this step, frames are converted to floating-point values rather than integers, in order to preserve the subtle intensity variations that carry the physiological signal.

- **Temporal resizing:** The models expect a fixed temporal input length. Following the motivation from [40], we use a sequence length of 160 frames, which has shown to provide consistently good results. Since the videos in our datasets do not always contain a multiple of 160 frames, any remaining frames at the end that do not form a complete sample are discarded.

3.3.4 Difference Frames/Standardization

After resizing, the pipeline applies Z-score standardization to the video, which involves subtracting the mean and dividing by the standard deviation of the pixel values. Optionally, the relative difference between consecutive frames can be calculated before this step to enhance the rPPG signal and suppress irrelevant information. The data format without this difference step is referred to as Standardized, while the format with the difference step is referred to as DiffNormalized. The specific format used for each model is listed in Table 3.1.

Similarly, the ground truth labels, i.e., the PPG signals from the dataset, are also processed with the DiffNormalization procedure to match the input data.

3.3.5 PPG extraction

The PPG signal is extracted from the video using neural networks. In this work, we compare five architectures: EfficientPhys [18], PhysFormer [39], PhysNet [38], RythmFormer [40], and TSCAN [17].

Model	Learning Rate	Input Format	Resolution	Train Epochs
EfficientPhys	9e-3	Standardized	72×72	30
PhysFormer	1e-4	DiffNormalized	128×128	10
PhysNet	9e-3	DiffNormalized	72×72	30
RythmFormer	1e-4	Standardized	128×128	10
TSCAN	9e-3	Both	72×72	30

Table 3.1: Hyperparameters and input specifications for the different models

3.3.6 Postprocessing Filter

Postprocessing

The following postprocessing strategies are considered:

1. **None:** No processing is applied.
2. **Cumsum:** Computes the cumulative sum of the PPG signal to reverse the differencing operation described in Section 3.3.4, which is required for some models as can be seen in table 3.1.
3. **Detrending:** Removes slow-varying trends using the algorithm in [27] with a smoothness prior $\lambda = 100$.
4. **Butterworth filter:** A first-order Butterworth bandpass filter with corner frequencies of $0.6\,\mathrm{Hz}$ and $3.3\,\mathrm{Hz}$ is applied [7], corresponding to plausible human heart rates (36–198bpm).

Heart Rate Estimation

The final pipeline stage converts the PPG signal into a heart rate estimate using one of the following methods:

1. **Peak Finding:** Detects peaks, computes median inter-peak distance, and calculates heart rate. Parameters include minimum peak distance, height, and prominence.
2. **Fourier Transform:** Computes the FFT of the signal and identifies the strongest spectral component.

3. **Custom:** This method was developed us and aims to combine the advantages of peak detection in the time domain with the Fourier Transform in the frequency domain. The motivation stems from the observation that, in the FFT spectrum of PPG signals, the first or second harmonic can sometimes appear more prominent than the fundamental frequency (ground tone).

   To address this, the FFT spectrum is first resampled to a fixed frequency resolution and smoothed with a Gaussian blur, where both the resolution and the blur's standard deviation are parameters of the method. Next, the time axis of the spectrum is compressed by integer factors (1, 2, 3, ...). Each of these scaled spectra is then multiplied with the original spectrum, which reinforces peaks corresponding to the fundamental frequency while suppressing spurious harmonics. The number of such iterations is a hyperparameter.

   In parallel, the heart rate is also estimated using a peak-finding method, which provides a prior probability distribution. This prior is modeled as a Gaussian centered at the peak-finding estimate, with a tunable standard deviation. The prior is then multiplied with the modified FFT spectrum, and the maximum of the resulting product yields the final heart rate estimate.

   The method is applied in a sliding-window fashion across the entire PPG signal, with the window length being another hyperparameter.

4. **Autocorrelation:** Computes the autocorrelation and finds the first peak using standard peak finding parameters.

5. **Fourier Transform of Autocorrelation:** In this method one would calculate the autocorrelation as described above and then calculate the FFT of that signal. Then we find the first peak in the FFT spectrum and by knowing the frequency resolution, we can get the heart rate.

6. **Neural Network:** Based on a ResNet [14] architecture, the network takes as input the PPG signal along with auxiliary features: PPG uncertainty, FFT magnitude and phase, and autocorrelation. The model uses four blocks of 1D convolution, batch normalization, and ReLU with skip connections, followed by a fully connected network. Linear layers embed the FFT magnitude and phase. The network is shown in figure 3.3 and 3.4.

3.4 Data Augmentation

Data augmentation plays a crucial role in improving the robustness and generalization capabilities of machine learning models, particularly in computer vision tasks. By artificially introducing variability into the training data, we can expose the model to conditions that it is likely to encounter in real-world scenarios but that may not be fully represented in the original dataset. This reduces the risk of overfitting and helps the model learn invariant features. The data augmentation techniques used in this study are outlined below.

1. **Artificial Destabilization:** To generate unstable videos from stable ones, we use the DeepStab dataset [34]. First, we detect good features to track in the stable frames using the Harris corner detector [9]. We then locate these features in the corresponding unstable frames.

   By analyzing the displacement of these features, we can deduce the camera motion path throughout the unstable video. This path can then be applied to augment a data sample recorded with a stable camera, effectively creating an artificially destabilized version by shifting the video frames along the estimated motion trajectory.

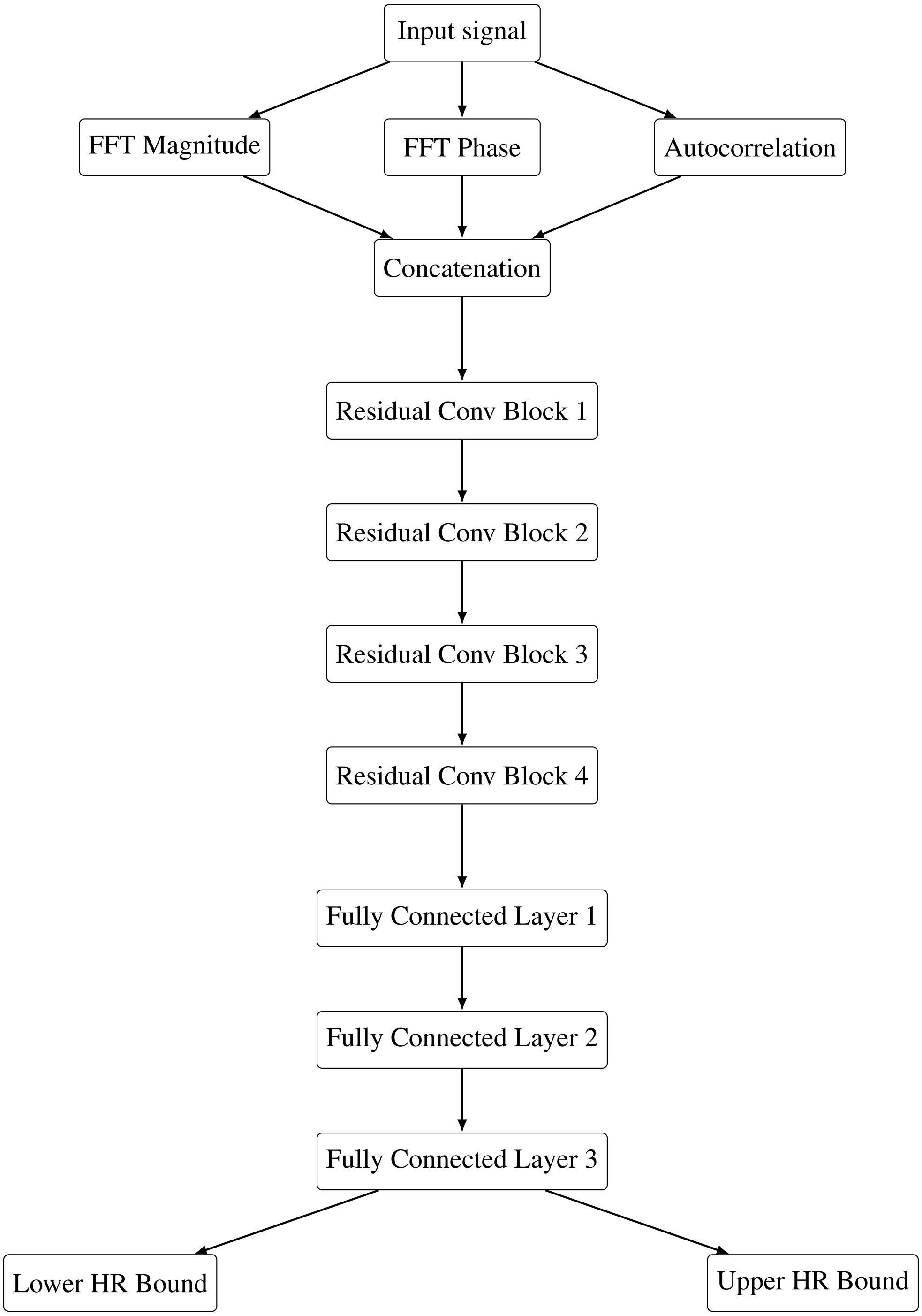

Figure 3.3: A visualization of the Heart rate classifier network

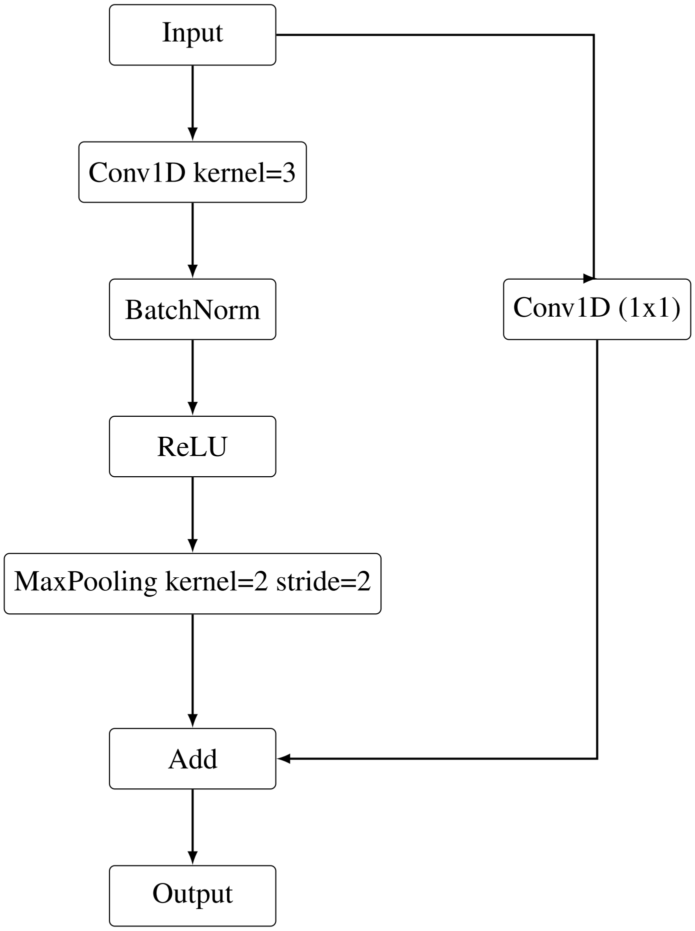

Figure 3.4: A visualization of the Residual Conv Block

2. **Gaussian Noise:** Instead of using real camera motion, this approach shifts the frame by a random ammount in x and y direction. This helps to increase data diversity and might make the model generalize better to unseen videos

3. **Random Affine:** It is also interesting to not only shift the frame in x and y direction, but also shear it. The idea is that this would simulate a rotation of the camera and that the model would generalize to this kind of motion.

4. **Compression:** To test how the model reacts on video compression, we added an option to compress and decompress a video before passing it into the preprocessing pipeline. The hope is, that a model trained on compressed videos, might learn to generalize better and show better performance on compressed videos.

   One can choose here from two codecs, namely H.264 and H.265. Also the CRF value can be freely set and one can choose to use a constant quantization parameter. The CRF controls the trade-off between video quality and compression efficiency: lower values lead to higher visual quality at the cost of larger file sizes, while higher values yield stronger compression but with increased loss of detail [1]. In practice, each increment of about six in CRF corresponds roughly to a halving or doubling of the resulting bitrate.

   By contrast, the Constant Quantization Parameter (CQP) mode fixes the quantization level for every frame, regardless of content or complexity. This gives the encoder no flexibility to adapt the bitrate to scene motion or detail, which results in predictable but often inefficient compression. While CRF adapts dynamically to preserve perceptual quality at varying bitrates, CQP enforces strict uniformity in quantization and therefore mainly serves as a tool for testing or controlled comparisons [1].

3.5 Computation speedup measures

Due to limited computational power, it is infeasible to process the entire dataset for every training epoch. Therefore, the pipeline is divided into three main parts: preprocessing, PPG extraction, and postprocessing. The preprocessing stage includes all steps up to and including the standardization, following the procedure described in 3.3. The readily preprocessed videos after the standardization step will be stored as NumPy tensors for fast access. We always store the Standardized and the DiffNormalized frames next to each other in the same tensor, which helps in case both of them are needed, because the preprocessing then has to be done only once.

We also added the option to store the resulting videos as compressed NumPy tensors to save disk space. These tensors are uncompressed into a cache directory as soon as they are needed for model training. Additionally, we allow preprocessing of multiple video sizes simultaneously. This accelerates computation, as the resize and standardization steps are performed for each input size, while video parsing, face detection, and stabilization are executed only once. In most cases, this is used to output both 72×72 and 128×128 pixel videos, since different models require different input sizes.

To enable unattended computation, we provide a script that executes the pipeline for all configuration files in a folder sequentially. A detailed description of these options is provided in the appendix section A.2.

Finally, to speed up training of the neural network described in section 3.3.6, the PPG predictions from the pipeline (up to and including the postprocessing step) are stored in a flatbuffer for fast access.

3.6 Uncertainty estimation

The concept of uncertainty estimation is important for understanding the reliability of the predicted PPG signals. There are several approaches to quantify this uncertainty. The following subsections will provide a detailed overview of these methods, explaining how they work, and how they can be integrated into the overall heart rate estimation pipeline.

3.6.1 Ensemble

The Ensemble method [25] estimates uncertainty by leveraging the predictions of multiple models trained for the same task. During inference, all models generate predictions, and the final output is taken as the mean of these predictions. The uncertainty is quantified as the standard deviation across the predictions. The underlying idea is that if the models "hallucinate" or make uncertain predictions, they are likely to do so in different ways, resulting in a higher standard deviation. Conversely, when the models are confident and accurate, the predictions will be consistent, yielding a lower standard deviation. In this work, we use the five models that were compared in the earlier stages of the pipeline.

3.6.2 Quantile regression

In quantile regression, the network is modified to produce two output neurons for each prediction, which are then interpreted as the lower and upper bounds of a predicted interval. For this approach, we adapt the PhysNet architecture [38]. We extend it by duplicating the upsampling stage and applying it to the output of the first stage, producing an additional set of outputs with the same shape as the original upsampling stage. To stabilize training, particularly during the initial epochs, we incorporate the negative Pearson loss from the original PhysNet as a regularization term. The final loss function is defined as in equation 3.1.

$$\text{Loss} = \sum_{i=1}^{N} \max\big(\alpha(l_i - \hat{x}_i), (1-\alpha)(l_i - \hat{x}_i)\big) + \sum_{i=1}^{N} \max\big(\alpha(\hat{y}_i - l_i), (1-\alpha)(\hat{y}_i - l_i)\big) + \lambda P \tag{3.1}$$

- N: Number of datapoints
- $\hat{x}_i$: Model prediction for the lower interval bound of datapoint i
- $\hat{y}_i$: Model prediction for the upper interval bound of datapoint i
- l_i: Ground truth label for datapoint i
- $\alpha \in (0, 1)$: Desired quantile
- P: Negative Pearson loss as described in [38]
- λ: Regularization weight, empirically chosen as 1

3.6.3 Data Likelihood

This method basically uses a similar idea as the quantile regression method. We also use two output neurons for each prediction but this time we interpret the predictions as the mean and standard deviation of a gaussian [25]. The loss function in this case is described in equation 3.2

$$\text{Loss} = -\log\left[\prod_{i=1}^{N}\frac{1}{\sqrt{2\pi\hat{\sigma}_i^2}}\exp\left(-\frac{(\hat{y}_i - l_i)^2}{2\hat{\sigma}_i^2}\right)\right] + \lambda P = \sum_{i=1}^{N}\left(\frac{(\hat{y}_i - l_i)^2}{2\hat{\sigma}_i^2} + \log\hat{\sigma}_i\right) + \lambda P \tag{3.2}$$

- N: Number of datapoints
- $\hat{y}_i$: Predicted mean for datapoint i
- $\hat{\sigma}_i$: Predicted standard deviation (uncertainty) for datapoint i
- l_i: Ground truth label for datapoint i
- P: Negative Pearson loss as described in [38]
- λ: Regularization weight, empirically chosen as 0.1

3.6.4 Data Augmentation

Empirical observations revealed that the uncertainty models—in particular the data likelihood variant—are highly susceptible to overfitting. To mitigate this issue, we apply a random brightness shift during training. Specifically, for each video a brightness factor is sampled uniformly from the interval $[0.8, 1.2]$ and applied consistently across all frames of that video. This augmentation preserves temporal consistency while introducing variability in illumination conditions, thereby improving the robustness of the models. We employ this method for training both the quantile regression model and the data likelihood model.

3.6.5 Conformalization

Conformalization [25] is a powerful technique for calibrating uncertainty predictions and obtaining well-defined confidence intervals. The core idea is to first perform inference on a validation set to assess how accurate the model's predictions are. Then, when a new datapoint is encountered, the prediction is adjusted based on the observed performance on the validation set. This ensures that the resulting confidence intervals are statistically valid and reflect the true uncertainty in the predictions.

3.6.6 Lifeness prediction

Lifeness is a method introduced in this work to train a model that predicts whether an rPPG signal is present in a video. We use the PhysNet embedding vectors (see Section 3.6.2) as input to a fully connected network with a single output neuron, which estimates the presence of an rPPG signal.

During training, half of the samples are temporally shuffled to remove the rPPG signal, while the other half remain intact. Since the rPPG signal is a sinusoid in the time dimension, random shuffling the frames, will destroy the signal, while not beeing detectable assuming a relatively static setting. for clearity this was illustrated in figure 3.5. For the unshuffled samples, the model is trained using the standard negative Pearson loss, whereas the shuffled samples are excluded from this loss term. Additionally, for all samples, a binary cross-entropy loss is applied on the single output neuron, using the "shuffled vs. unshuffled" label as ground truth.

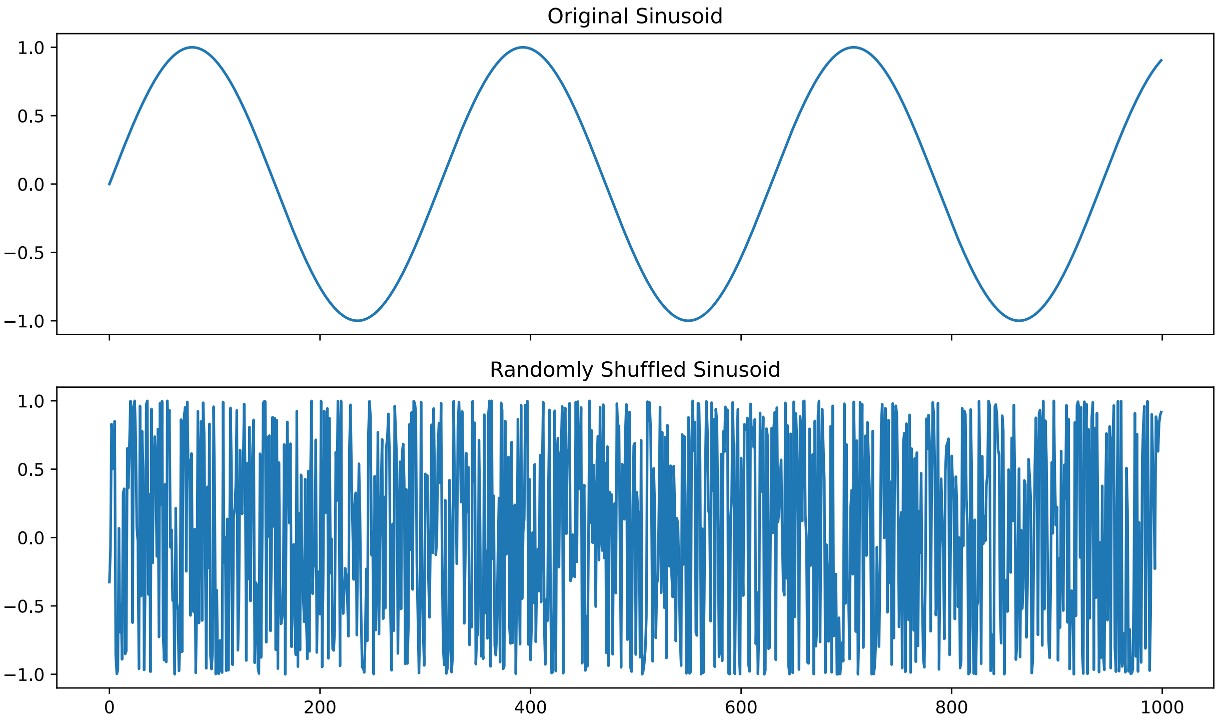

Figure 3.5: A comparison of a sinusoid signal with a random shuffled version of it

3.7 Visualization

To gain insight into what the network actually learns, we employ saliency maps [26]. Saliency maps are a visualization technique that highlights which parts of the input data contribute most strongly to the model's predictions. In our case, for each predicted point in the PPG signal, we compute the gradient of the output with respect to the input video frames. This gradient essentially measures how sensitive the prediction is to changes in each input pixel or region. Regions with higher gradient values indicate areas that the model considers more important for producing the corresponding PPG value. By visualizing these regions, we can interpret which spatial and temporal features in the video the network relies on, providing a more intuitive understanding of the model's decision-making process and helping to detect potential biases or failure modes.

3.8 Evaluation

For all evaluations, we define a standard processing pipeline, modifying only a single block at a time. The default configuration for each pipeline stage is given in Table 4.1.

To quantify performance, we use two metrics: the mean absolute error (MAE) for heart rate estimation, and the signal-to-noise ratio (SNR) for the PPG signal. The SNR [10] is defined as follows. First, a bandpass filter with corner frequencies of $0.6\,\mathrm{Hz}$ and $3.3\,\mathrm{Hz}$ is applied, following [7]. From the ground-truth PPG signal, we identify its most prominent frequency f_0. In the predicted signal, we then select a frequency band of ± 6 bpm around f_0 and around its first harmonic $2f_0$. The SNR is computed as the ratio between the energy contained in these two bands and the energy in the remaining spectrum. The exact formula is given in equation 3.3. In figure 3.6 an example prediction is plotted and the two energy bands are marked. The

SNR can be calculated after reassembling the PPG predictions, which were sliced before in the resize step of the pipeline 3.3.3, but in this study, we decided to do the evaluation for each video slice (160 samples) separately and then calculating the mean. The idea of that is to become more robust against subjects with changing heart rates.

$$\mathrm{SNR} = 10\log_{10}\left(\frac{\sum\limits_{f\in B_{f_0}\cup B_{2f_0}} |X(f)|^2}{\sum\limits_{f\notin B_{f_0}\cup B_{2f_0}} |X(f)|^2}\right), \tag{3.3}$$

where:

- $X(f)$ – Fourier transform of the predicted PPG signal after bandpass filtering.
- f_0 – dominant frequency of the ground-truth PPG signal.
- $2f_0$ – first harmonic of f_0.
- $B_{f_0} = \{f \mid |f - f_0| \leq \Delta f\}$ – frequency band around f_0.
- $B_{2f_0} = \{f \mid |f - 2f_0| \leq \Delta f\}$ – frequency band around the first harmonic.
- Δf – half-bandwidth, here $\Delta f = 6/60$ Hz corresponding to ±6 bpm.

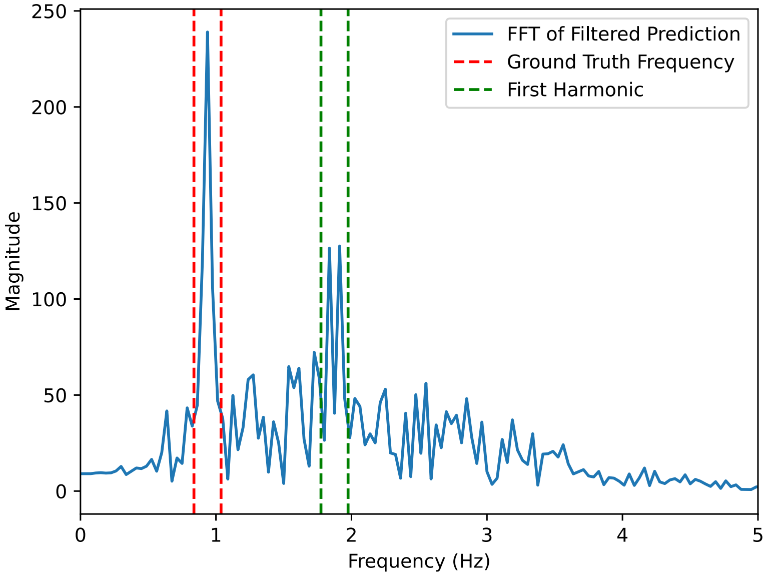

Figure 3.6: The FFT of a predicted PPG signal. The frequency bands around the first and second harmonic, that are used for the SNR calculation are marked with red and green dashed lines

Chapter 4

Experiments and Results

This chapter presents the experiments conducted to evaluate the algorithm's performance in real-world scenarios and to assess its robustness. Each section first describes the experiment and its underlying rationale, and then highlights key results. Detailed figures are provided in the appendix and are referenced where appropriate.

4.1 Pipeline stage evaluation

In this section, we aim to identify the pipeline that achieves the best performance on our available data. To do this, we start with the standard pipeline described in table 4.1 and iteratively modify one block at a time, allowing us to systematically evaluate the impact of each component on overall performance. For every experiment, a new network is trained on the dataset described in Section 3.1.2 using the loss functions and training procedures specified in the respective papers. For optimization, we employ the Adam optimizer without weight decay for all models, except for PhysFormer, which uses a weight decay of 0.00005, based on the reference implementation. The learning rates and number of training epochs for each model are summarized in Table 3.1.

Model	**Resolution**
Face Detector	YUNET
Stabilizer	Median Face Box
PPG extractor	PhysNet
Postprocessing Filter	Butterworth
Heart Rate filter	Peak detection

Table 4.1: The standard pipeline used for further evaluations

4.1.1 Face Detectors

In this experiment, we compare the face detectors introduced in Section 3.3.1. For each detector, we retrain the model using the dataset preprocessed with that specific detector. We then evaluate the resulting SNR on both the validation and test sets. Additionally, we measure the average execution time of each face detector by running it 10 times on the first video file in the VitalVideos dataset. The results are summarized in Table 4.2.

Detector	Validation SNR	Test SNR	Execution Time
YOLO5Face	-0.27	-0.83	0.34
YUNET	-0.30	-0.99	0.20
HC	-0.84	-3.95	0.78

Table 4.2: The mean SNRs of the pipeline with different face detectors

4.1.2 Stabilizers/PPG extraction

In this experiment, we compare the four stabilizers introduced in Section 3.3.2 and examine their performance on both stable and artificially destabilized videos, as described in Section 3.4. The destabilization is performed using realistic camera motion in this experiment. Additionally, all five models from Section 3.3.5 are tested. For each sub-experiment, the model is retrained, resulting in a total of $2 \times 4 \times 5 = 40$ training processes. To reduce computation time, the optimization strategies described in Section 3.5 were employed. The parameters, we used for the optical flow calculation, are documented in table 4.3. The results are summarized in Tables 4.4 and 4.5, with the test set results provided in the appendix (Tables A.2 and A.3).

Parameter	**Value**
`pyr_scale`	0.5
`levels`	3
`winsize`	15
`iterations`	3
`poly_n`	5
`poly_sigma`	1.2

Table 4.3: Farneback optical flow parameters

Model	Median Face Box	Template Matching	Optical Flow	MOSSE
EfficientPhys	-2.44	-2.3	-2.18	-2.34
Physnet	-1.13	-0.94	-0.91	-1.09
TSCAN	-2.69	-1.95	-2.09	-2.25
RythmFormer	-0.78	-0.9	-0.99	-1.1
PhysFormer	-1.54	-1.68	-1.33	-1.51

Table 4.4: The mean SNRs of the models stable videos and different stabilizers on the validation set

Model	Median Face Box	Template Matching	Optical Flow	MOSSE
EfficientPhys	-8.36	-3.13	-4.44	-2.68
Physnet	-2.82	-1.32	-2.51	-0.91
TSCAN	-10.07	-9.12	-9.57	-9.46
RythmFormer	-8.69	-1.82	-9.88	-1.71
PhysFormer	-5.45	-2.91	-4.74	-1.74

Table 4.5: The mean SNRs of the models on destabilized videos and different stabilizers on the validation set

4.1.3 PPG Postprocessing/Heart Rate Filter

For the final pipeline step, we compare the postprocessing methods introduced in Sections 3.3.6 and 3.3.6. To determine the optimal parameters for each method, we employ Bayesian estimation on the validation set. The selected parameters are then evaluated on the test set to assess generalization performance. In this experiment, we systematically explore the entire search space spanned by the four postprocessors and the six heart rate filters. The neural network is trained for 100 epochs using the Adam optimizer (learning rate 9e-5), with either the quantile loss or negative data likelihood loss (see Section 3.6). Computational optimizations are described in Section 3.5. The results for the validation set are summarized in Tables 4.6 and 4.7.

Preprocessing	Peaks	Custom	FFT	Autocorrelation	FFT+Autocorrelation	DNN
Raw	7.36	1.73	13.14	3.25	14.67	2.78
Cumsum	12.16	6.05	3.63	1.67	22.97	2.61
Detrend	9.08	6.23	3.72	1.83	25.01	3.41
Butter	8.12	4.22	3.95	2.32	36.22	2.91

Table 4.6: The MAEs output by the postprocessing hyperparameter search on the validation set with the likelihood model

Preprocessing	Peaks	Custom	FFT	Autocorrelation	FFT+Autocorrelation	DNN
Raw	2.8	1.66	21.18	4.05	13.09	3.05
Cumsum	11.59	6.69	3.72	1.63	20.56	7.64
Detrend	4.66	8.16	3.71	2.05	21.1	2.93
Butter	7.99	5.6	3.66	1.92	38.23	3.64

Table 4.7: The MAEs output by the postprocessing hyperparameter search on the validation set with the quantile regression model

4.2 Skin tones

In this experiment, we aim to evaluate which model generalizes best to darker skin tones. All available models are assessed on the combined validation and test sets, and the resulting SNRs are compared across different skin types. This analysis is possible because the VitalVideos dataset provides skin tone information for each participant using the Fitzpatrick scale. The results are summarized in Table 4.8.

Model	Type 1	Type 2	Type 3	Type 4	Type 5
EfficientPhys	-0.67	-1.17	-3.19	-2.62	-3.22
Physnet	0.41	0.29	-0.39	-1.79	-2.29
TSCAN	-1.15	-2.41	-2.25	-2.71	-3.22
RythmFormer	0.75	0.78	-1.14	-4.12	-1.68
PhysFormer	0.37	0.11	-0.04	-1.16	-1.55
Physnet_Quantile	0.27	-0.09	0.36	-0.46	-1.5
Physnet_NegLogLikelihood	0.54	0.11	0.32	-2.14	-1.79

Table 4.8: The mean SNR of the standard pipeline by fitzpatrick value

4.3 Real World Evaluation

To test how well the pipeline performs under real-world conditions, we use the Realistic Videos dataset described in Section 3.1.1. For stabilization, we compare the baseline method and the best performing method on the validation set. For both we train first on stable and then on unstable videos and then do inference on the Realistic Videos dataset. The results are presented in Table 4.9. For this experiment, we also measure the mean and standard deviation of the ground truth heart rates in the Realistic Videos dataset. The result is 58.81 for the mean and 7.15 for the standard deviation.

Model	Standard	Standard Destabilized	MOSSE	MOSSE Destabilized
EfficientPhys	-7.46	-8.23	-5.45	-5.76
Physnet	-7.68	-7.28	-5.24	-5.59
TSCAN	-6.48	-8.62	-4.74	-9.1
RythmFormer	-8.65	-11.7	-7.08	-7.21
PhysFormer	-6.89	-7.9	-6.48	-6.31
Physnet_Quantile	-7.56	-9.91	-6.38	-5.8
Physnet_NegLogLikelihood	-6.81	-8.08	-5.78	-5.2

Table 4.9: The mean SNR for the standard pipeline versus the best performing stabilizer for both stable and unstable videos on the Realistic Videos dataset

4.4 Conformalization

To evaluate how well the uncertainty models can be conformalized, we execute the process described in Section 3.6.5. Here, we conformalize for different coverages and then plot the mean coverage on the validation set and on the test set. On the validation set, the actual coverage will always equal the desired coverage, since it is used for calibration. It serves as a reference that we hope to approximate closely on the test set. The results are plotted in Appendix A.4.7.

4.5 Uncertainty

In this experiment, we execute the models that output a measure of uncertainty on the datasets described in Section 3.1.1. We calculate the average uncertainty outputted for each dataset and normalize the values on the validation set to 1. Finally, we include an entry for the *RealisticVideos* dataset stabilized with the best-performing stabilizer from Section 4.1.2. The results are presented in Table 4.10.

4.6 Lifeness

To test whether the Lifeness version of Physnet learns to detect the presence of an rPPG signal, we first train it as described in Section 3.6.6 and then evaluate it on the datasets introduced in Section 3.1.1. We then compare the average predicted lifeness for each dataset. The results are presented in Table 4.11.

Experiment	Ensamble Model	Physnet_NegLogLikelihood	Physnet_Quantile
Validation	1	1	1
Test	0.92	0.93	0.94
Own Videos	0.98	1.19	1.21
Validation Shuffled	1.64	1.68	1.8
Test Shuffled	1.63	1.72	1.88
Dead Videos	1.4	1.87	2.24
Emergency Videos	1.28	1.85	1.84
Emergency Videos Stable	1.21	1.44	1.62

Table 4.10: The mean uncertainty predicted by different models on different datasets. All values normalized to the validation set

Dataset	Mean Lifeness
Test set	0.8701
Validation set	0.9065
Ad Banner Videos	0.2502
Distribution Shift Videos	0.6924
Realistic Videos	0.3791

Table 4.11: The average lifeness predicted by the lifeness version of Physnet described in section 3.6.6 on different datasets

4.7 Optimizations

4.7.1 Video Compression

Video compression is an effective method to reduce the bandwidth requirements of an rPPG algorithm. We compare the two most widely used codecs in video compression, namely H.264 and H.265. In this experiment, we first apply each codec with a given CRF value to the videos, then retrain the model on compressed videos with CRF values ranging from 0 to 50, and finally execute the complete rPPG pipeline. The resulting model is then tested on compressed videos with CRF values from 0 to 50. In our experiments, we use the constant quantization parameter mode as described in section 3.4. To illustrate the potential bandwidth savings, we compress one data sample and compare its size to that of the original video. The resulting plots are presented in figure 4.1 for the validation set and in figure A.1 for the test set.

4.7.2 Computation efficiency improvement

We investigate whether the computational requirements of the standard pipeline can be reduced by changing the input size in the resize stage. In this experiment, we measure the number of MACC operations using the Torch Profiler and evaluate the pipeline's accuracy using the SNR. The neural network, in this case PhysNet, does not require architectural changes because it is fully convolutional. Its final stage contains an adaptive average pooling layer, which aggregates the activations from the preceding layer regardless of their spatial dimensions. The results are presented in Appendix A.4.8.

Figure 4.2 shows the effect of input spatial resolution on the extracted rPPG signal. We observe that the signal quality, as measured by SNR, begins to drop significantly when the input size falls below approximately 30x30 pixels. This indicates that maintaining a minimum spatial resolution is essential for capturing the subtle color variations caused by blood volume changes.

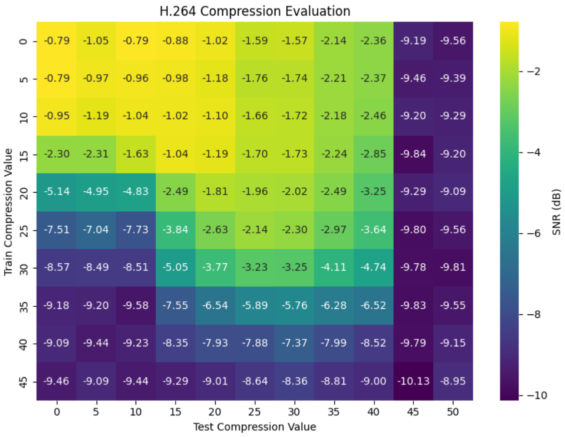

Figure 4.1: The evaluation on the compressed videos of the validation dataset with H.264 on the left and H.265 on the right

Additionally, the computational demand decreases approximately with $\mathcal{O}(n^2)$ as the input size is reduced, which is also illustrated in figure 4.2. This demonstrates the trade-off between processing efficiency and signal quality, highlighting that excessively small input sizes can lead to a considerable loss in physiological signal fidelity.

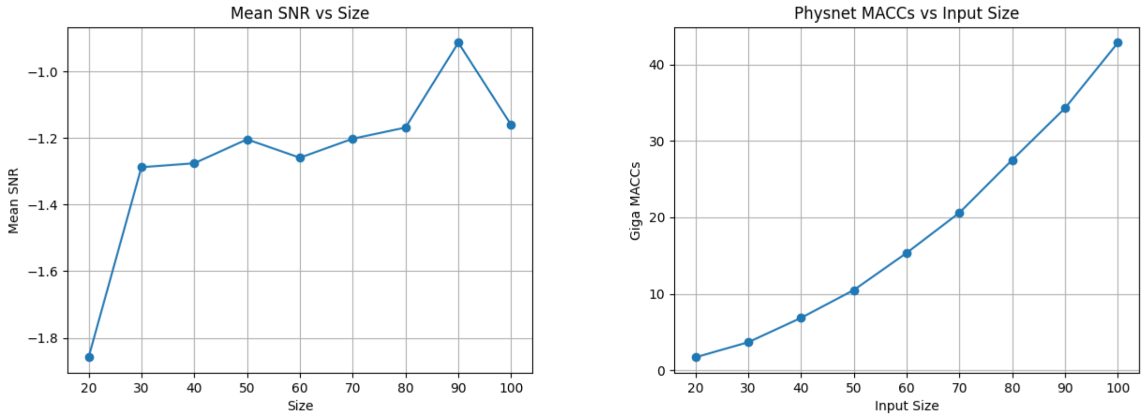

Figure 4.2: The Mean SNR resulting from the evaluation of the Physnet on images with different size on the left hand side and the needed MACCs needed for every size on the right

4.8 Visualization

To visualize what the network learns, we use the standard pipeline and apply it on a datasample from the VitalVideos dataset. Unfortunately, due to data protection legislation, we are only allowed to show one datasample from this dataset in the report. Therefore we execute the pipeline on this datasample and evaluate the saliency maps as described in section 3.7. Then we plot the first frame of the video in the leftmost image and then the three color channels with its respective saliency maps overlayed. The result is shown in figure 4.3.

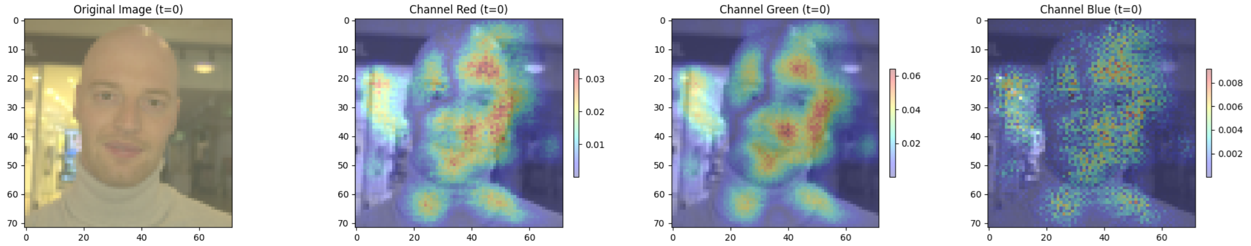

Figure 4.3: The results with the Saliency maps overlayed

Chapter 5

Discussion

First, we note that the YOLO-based face detector appears to be the most effective among the four evaluated detectors for rPPG tasks. Additionally, AI-based methods generally achieve higher performance while requiring less inference time, as summarized in Table 4.2.

For video stabilization, the MOSSE method shows the best results (see Table 4.5). This is likely due to its ability to adapt the feature template online, allowing better generalization to changing objects. Presumably it will, compared to the other approaches, perform better in cases where the test subject slightly rotates their head. Considering the PPG-extraction stage, RythmFormer seems to be the best performing network under stable conditions, as expected since it is the most recent model and optimized for standard rPPG tasks (compare Table 4.4). However, RythmFormer does not generalize as well to unstable videos, whereas PhysNet consistently outperforms in these scenarios, likely because it aggregates consecutive frames locally without relying on global information. This aggregation seems particularly beneficial for stabilization.

The final steps of the pipeline concern heart rate extraction. The results indicate that Quantile Regression with the cumulative sum postprocessor, combined with the autocorrelation extractor, yields the best performance (see Table 4.7). Autocorrelation is robust to noise and can handle signal harmonics effectively. In practice, when the model is uncertain about the exact location of the next peak, it may skip it or predict two peaks, which challenges peak-finding algorithms. Autocorrelation mitigates this issue by aligning on the other peaks it detects, explaining its superior robustness. Interestingly, the neural method does not outperform autocorrelation. The best-performing pipeline, combining the optimal choices for each step, is summarized in Table 5.1.

Pipeline Step	**Best Method**
Face Detector	YOLO5Face
Stabilizer	MOSSE
rPPG Extraction Model	PhysNet
Postprocessor	Cumsum
Heart Rate Filter	Autocorrelation

Table 5.1: Best-performing methods for each pipeline step.

Our study also replicates the findings of prior work, which showed that darker skin types make rPPG extraction more challenging [22]. This effect is evident in Table 4.8, where performance decreases for subjects with higher Fitzpatrick skin types. Note that the RythmFormer model, while performing well overall, shows a more pronounced drop in accuracy for darker skin tones compared to PhysNet. This suggests that models optimized for ideal or balanced datasets may still struggle under real-world variability, highlighting the importance of diverse training data and robust preprocessing steps to mitigate skin tone-related biases.

The most important finding of this study is that stabilization can, to some extent, help rPPG models to perform better on unstable videos. We observed that, when executed on the Realistic Videos dataset, the pipeline with MOSSE stabilization achieves significantly lower SNR than the standard pipeline as seen in table 4.9. Also note that augmenting videos with realistic camera motion during training does not seem to decrease SNR significantly. However, the performance on the Realistic Videos dataset is still lower than on the VitalVideos dataset, which can be attributed to other distribution shifts, such as differences in camera type, subject posture (e.g., lying on the ground), motion blur from rapid camera movement, and video enhancement algorithms running on the recording device. Another part of the story is probably also that the subjects in the Realistic Videos dataset have pretty low heart rates, which makes the task also harder.

The impact of specific distribution shifts on the pipeline is further illustrated by the uncertainty experiments. The network output similar uncertainties on the Distribution Shift videos as on the validation set, but much higher uncertainties on the Realistic Videos, as shown in Table 4.10. Since the Distribution Shift and Realistic Videos share the same camera, with the main difference being camera stability, we can conclude that unstable camera motion triggers high uncertainty. Therefore, the performance drop from the validation set to the Realistic Videos dataset is primarily driven by motion blur, which cannot be fully mitigated by simple frame alignment or shifting.

Examining the Ad Banner Videos, the network produces the highest uncertainties, suggesting that the model has learned a meaningful notion of uncertainty when trained with the quantile or data likelihood loss. This conclusion is further supported by the results of the lifeness experiment (Table 4.11), where the network correctly outputs lower lifeness scores for the Ad Banner Videos. Together, these findings indicate that the network effectively captures both the reliability of its predictions and the inherent challenges of real-world video conditions.

Additionally, we observe that conformalization performs best when combined with quantile regression, as illustrated in Figure A.7. Interestingly, the conformalized graphs generated using the entire rPPG signal are noticeably smoother than those produced when conformalizing only the extracted heart rate. This is likely because using the full signal provides a larger number of data points, which helps to average out local fluctuations and noise, leading to a more stable uncertainty estimate.

Another important finding of this study is that the proposed pipeline shows a certain degree of resilience to video compression. As illustrated in Figure 4.1, videos encoded with the H.265 codec tend to preserve the rPPG signal more effectively, likely due to the codec's improved compression strategies that maintain signal fidelity at smaller file sizes.

We also observe differences in training strategies for the two codecs. For H.264, it appears advantageous to train on the lowest compression factor to maximize generalization, whereas for H.265, training on the same compression level that will be used during testing provides a small performance benefit. This insight has practical implications: for instance, when deploying a server that processes incoming videos from multiple subjects, it may be necessary to maintain multiple models for H.264 videos, each optimized for different compression levels, to ensure optimal rPPG extraction performance.

By examining the network visualization in Figure 4.3, we can see that the model primarily focuses on the skin regions of the face, which is expected since these areas carry the strongest rPPG signal. The eyes appear to be largely ignored, consistent with their limited relevance for rPPG extraction.

Interestingly, the green channel exhibits the highest activation magnitudes, which aligns with previous findings indicating that the green channel is most informative for rPPG tasks. The blue channel shows much noisier activations, suggesting that the network may be using it to estimate and compensate for sensor or camera noise, before relying on the red and green channels for the physiological signal. This strategy parallels classical deterministic algorithms that often subtract the blue channel to reduce noise in the extracted signal.

Chapter 6

Conclusion

In this work, we presented a comprehensive study on robust rPPG extraction from handheld videos under real-world conditions. We identified several key challenges, including varying skin tones, video instability, sensor noise, and compression artifacts, and systematically evaluated their impact on modern deep learning models.

Our experiments show that preprocessing steps such as face detection, stabilization, and resizing are critical for achieving reliable results. Among the methods evaluated, the YOLO-based face detector, the MOSSE stabilizer, and the PhysNet model demonstrated the best trade-off between accuracy and robustness, depending on the scenario. For heart rate extraction, quantile regression combined with cumulative sum postprocessing and autocorrelation proved to be the most effective approach.

We further showed that artificial destabilization during training can improve model generalization to unstable videos and that conformalization with quantile regression produces reliable uncertainty estimates across different datasets. Additionally, the pipeline exhibits a degree of resilience against video compression, particularly when the training and test codecs are aligned.

Finally, visualizations of the network's attention confirm that the models focus primarily on skin regions and exploit channel-specific information, supporting both the physiological and algorithmic rationale behind the design choices.

Overall, our results provide practical guidance for building robust rPPG systems and highlight the remaining challenges in real-world applications, particularly regarding motion blur and distribution shifts that cannot be fully mitigated through preprocessing alone.

Although the findings of this study are promising, future work should focus on extending the Realistic Videos dataset and retraining the models on it. This would allow the algorithms to better learn how to handle motion blur in rPPG tasks. In addition, future research could investigate alternative data augmentation techniques to simulate motion blur in unstable videos, thereby improving model robustness under real-world conditions.

Appendix A

The First Appendix

A.1 Dataset recording format

This section documents the format of a recording in the datasets created for this project. The datasets without ground truth are quite simple, because they are just a list of videos. The Realistic Videos dataset defined in Section 3.1.1 on the contrary has a synchronized ground truth and therefore has a more complex format. A readily postprocessed datasample consists of four files, which are documented below:

timestamps.csv

This file contains a csv Table of the timestamps from the android phone and the corresponding timestamps from the computer. Both are in the format of linux timestamps (number of milliseconds since Jan 01 1970. (UTC)). This file is created automatically during the recording process

recording.pcapng

This is the wireshark recording of the PPG measurement. Some reverse engineering showed that the data is sent via the standard Universal Serial Bus (USB)-protocol. A PPG data sample has 48 bit in total and starts with a fixed initialization sequence of 8 bits, which is 0xeb. Afterwards we see 16 bits that are probably used as status indicator, current amplifier gain or something similar. They are not very relevant for the data recording, so they were not investigated further. The trailing 24 bits transport the information of one PPG sample as an unsigned integer. The device bundles three data samples into a single USB packet, which means that the packet in the wireshark recording will carry 144 bit or 18 byte of information. The device pads it with zeros to a lenght of 64 data bytes and with the header information from the USB protocol it has 91 bytes in total. The protocol also contains some other communication, which is not investigated in detail, since it is sufficient to search for USB packets that match the pattern described above.

video.mp4

This file contains the video from the participant's face with the android timestamps printed on it. This file is pulled from the phone automatically during the recording process.

waveform.npy

This file is generated by the postprocessing step from the other three files. It contains two vectors of the same length, which have one datasample for each frame in the video. The first vector contains the timestamps of the windows clock and the second vector the PPG samples.

A.2 Code documentation

The code is basically a modified version of the code used in [19] and it was published in a github repository [20]. The purpose of this section is to introduce the reader to the code, such that he can use it and reproduce the results.

The rPPG-Toolbox [19] uses YAML Ain't a Markup Language (YAML) configuration files to define properties that can be set in the training engine. Each config file will train or test one model and serialize the outputs to the disk. For a detailed description of how to setup a config file, please refer to the README of the repository.

The study uses 9 different experiments where each experiment consists of multiple configuration files. The outputs for each experiment are stored in a separate folder named by the experiment in the runs directory. A detailed description what each experiment includes is given below:

1. `compression`
   In this experiment the PhysNet model is trained on compressed videos with CRF values ranging from 0 to 50 and codecs. Afterwards the models are tested on CRF values from 0 to 50.

2. `dataAug`
   This experiment trains PhysNet models with different Data augmentation strategies as described in section 3.4. The models are then executed on all available datasets.

3. `deadalive`
   This experiment first trains all models used in this study on the input data they require as described in section 3.3 and then test on all available datasets, on the shuffled validation, shuffled test set and the MOSSE stabilized version of the RealisticVideos dataset.

4. `deepstab`
   Here all the models are first trained first on the standard inputs with different stabilisation backends and then the same thing with the destabilized frames as described in 3.4.

5. `emergency`
   This experiment executes models pretrained on VitalVideos and UBFC on the RealisticVideos dataset.

6. `facedetection`v Here models are trained with different face detectors and then tested and validated.

7. `lifeness`
   This experiment trains the lifeness version of the PhysNet as described in section 3.6.6 and then tests it on all available datasets, on the shuffled validation and shuffled test set.

8. `physnet_in_size`
   Trains tests and validates the PhysNet with different input sizes ranging from 20 to 100.

9. `uncertainty`
   Finally, in the uncertainty experiment the data likelihood and quantile regression version of the PhysNet, as described in section 3.6.2 and 3.6.3. Also the heart rate classifier network is trained on the respective outputs with the different postprocessing methods described in section 3.3.6. Then everything is executed on the test and validation set.

All python scripts, that are meant to be executed directly from the command line, are stored in the root folder of the repository. The following list will introduce the reader to each one of them:

1. `compare_facedetectors.py`
   Will output a performance comparison of the three face detectors used in this study. As a prerequisite, the facedetection experiment needs to be run.

2. `compare_physnet_sizes.py`
   outputs a comparison of the performance of the PhysNet when feeded with different input sizes. As a prerequisite, the physnet_in_size experiment needs to be run.

3. `compare_skin_colors.py`
   outputs a comparison of the performance of the PhysNet on the fitzpatrick skin types in the validation and test set. As a prerequisite, the deadalive experiment needs to be run.

4. `compare_stabilizers.py`
   outputs a comparison of the performance of the PhysNet when different stabilizers are used. As a prerequisite, the deepstab experiment needs to be run.

5. `comparison_emergency.py`
   outputs a comparison of the performance of the PhysNet on the RealisticVideos dataset. As a prerequisite, the emergency experiment needs to be run.

6. `compression_evaluation.py`
   outputs a comparison of the performance of the PhysNet when feeded with compressed videos. As a prerequisite, the compression experiment needs to be run.

7. `compression_video_size_evaluation.py`
   This script applies video compression with the H.264 and H.265 codecs using CRF values ranging from 0 to 50 and compare the size reduction of the compressed video when compared to the original video. The results are outputed in the form of a plot.

8. `config_generator.py`
   Execute this script once in the beginning to generate the configs for all the experiments. The script will run a few sanity checks and output all configuration files for each experiment to the directory configs/all_experiment_configs. This is done for convenience, since it is easy to make mistakes in generating configuration files for experiments.

9. `conformalization_evaluation.py`
   Conformalizes all uncertainty outputs by calibrating on the validation set. It will plot validation and test coverage next to each other in one plot for each method.

10. `construct_uncertainty_tables.py`
    Outputs a table that compares the uncertainty outputs on the datasets used in this study.

11. `dataset_statistics.py`
    Outputs plots for all relevant statistics for the VitalVideos dataset.

12. `deepstab_preprocess.py`
    This script takes the Deepstab dataset and extracts the ammount of pixels by which every frame was shifted in x and y direction. This data is then exported to .npy files on the disk.

13. `http_client.py`
    For testing purposes, a Hyper Text Transfer Protocol (HTTP) client has been developed to match http_server.py. It will load one datasample from the disk and send it to the server for processing.

14. `http_server.py`
    This is a demo server application, that was developed for the purpose of showing how a real world example could look like. It is designed to match the demo android application, that is documented in section A.3.

15. `hyperparameter_search_hrfilter.py`
    This script will apply bayesian estimation to determine the best parameters for the methods in the postprocessing and heart rate extraction step.

16. `lifeness_evaluation.py`
    Produces a table with the lifenesses outputed by the lifeness version of the PhysNet on the datasets in this study.

17. `main.py`
    This script is adapted from [19]. It can be supplied with a YAML configuration file and it will train or test the model with the given properties. Most of the time this script is not executed directly, but through run_pipeline.py

18. `main_PyEVM.py`
    This script will execute either color [35], motion [16] or phase based [32] magnification algorithms

19. `main_export_model.py`
    This script exports a model to a format that can be read by the demo android application, which is documented in section A.3.

20. `ppg_editor.py`
    This script is used for the peak labeling in the ground truth of the dataset as described in section 3.1.3.

21. `prepare_experiment.py`
    This script can be supplied with the name of one of the experiments documented above or 'all'. It will take the configuration files associated with the given experiment and put them into the configs/pipeline folder.

22. `run_pipeline.py`
    This script will execute all YAML files it can find in the directory configs/pipeline. The ones that run successfully (with return code 0) it will move to configs/pipeline_results/success, while the failed ones (return code not 0) will be put to configs/pipeline_results/failure. Next to each configuration file, the script will also put a .log file with the exact output of the process, which is useful for debugging.

23. `run_pipeline_check.py`
    This script performs some sanity checks on all YAML files in the configs/pipeline directory. This can help to find errors e.g. two configuration files overwriting each others outputs, non existent pretrained models for test configurations.

24. `setup.sh`
    Execute this script in the beginning to setup the python environment with the uv package manager.

The following section outlines the project's folder structure. For each folder, we provide a short description of its contents and its role within the project.

1. `config_generation`
   This folder contains the scripts that were used for generating config files. They are automatically executed by `config_generator.py`

2. `configs`
   This folder contains a few example YAML configuration files and also the automatically generated ones.

3. `csv`
   Some scripts export data in excel format. These tables are stored in here.

4. `dataset`
   Contains the logic for reading and processing datasets. To add support for a new dataset, add a dataloader here as described in README.md

5. `dataset_recorder`
   This folder contains its own README.md and requirements.txt files. It is an independent entity that can be executed to record a dataset with ground truth, as described in section 3.1.1.

6. `evaluation`
   Contains some utility methods for evaluation after the AI-algorithm has run.

7. `figures`
   Contains some example figures from the original rPPG-Toolbox implementation

8. `final_model_release`
   Contains some example network weights from the original rPPG-Toolbox implementation

9. `graphics`
   Contains the figures produced in this work. They are automatically generated by the scripts above.

10. `mosse`
    Contains the code from the work about MOSSE [6].

11. `neural_methods`
    Here the neural network algorithms are defined. To add support for a new neural network, do it in here as described in README.MD

12. `opencv_zoo`
    Contains a github repository that hosts a lot of different models, that can be used together with opencv.

13. `PyEVM`
    This is a module that can execute the either color [35], motion [16] or phase based [32] magnification algorithms. It was produced in this work from phase_based GitHub repository and PyEVM GitHub repository. Due to licence restrictions, this module is not included in the public repository.

14. `rppg_buffers`
    Here the readily postprocessed PPG signals output by the AI-algorithms are stored. This is used primarily for training the heart rate detection network.

15. `runs`
    Contains the results of the experiments when they are available. For each run, the system will store some plots, the model outputs and the weights from each epoch.

16. `template-matching-android`
    This folder contains the demo application for android. It has its own README.md file.

17. `tools`
    Contains some jupyter notebooks that can be used to visualize data, generate the images used in the presentation for this work and do some processing.

18. `unsupervised_methods`
    Unsupervised is a bit misleading here, but the name was taken over from rPPG-Toolbox. It contains all the non-AI-algorithms, that can be used to extract PPG signals.

19. `wip`
    Contains some dataset comparisons from rPPG-Toolbox.

A.3 Demo Application

For the purpose of this study a demo application for android has been created to demonstrate how the results could be used in a real world setting. The application uses the front camera to record a video, apply the YUNET face detection together with the template matching stabilization. In the end the frames will be resized to 72x72 pixels and sent to the server. The server needs to be started with the http_server.py. There is also an option to execute the whole pipeline directly on the phone, but this part is currently not working.

A.4 Detailed Results

This in this section the results of the work are shown for a detailed discussion have a look at chapter 4

A.4.1 Face detection

Detector	Validation MAE	Test MAE	Execution Time
Y5F	2.63	3.67	0.34
YUNET	2.42	4.99	0.20
HC	3.20	8.89	0.78

Table A.1: The MAEs of the models with different face detectors on the validation and test set

A.4.2 Stabilization/PPG Extraction

Model	Median Face Box	Template Matching	Optical Flow	MOSSE
EfficientPhys	-2.33	-2.3	-2.18	-2.32
Physnet	-0.48	-0.06	-0.34	-0.21
TSCAN	-2.85	-2.15	-2.21	-2.25
RythmFormer	-0.37	-0.85	-0.97	-0.87
PhysFormer	-0.34	-0.64	-0.32	-0.33

Table A.2: The mean SNRs of the models on stable videos and different stabilizers on the test set

Model	Median Face Box	Template Matching	Optical Flow	MOSSE
EfficientPhys	-10.28	-2.6	-5.09	-2.83
Physnet	-1.76	-0.55	-1.58	-0.59
TSCAN	-10.87	-10.74	-10.64	-10.87
RythmFormer	-10.83	-2.94	-11.06	-1.77
PhysFormer	-5.76	-1.97	-4.95	-0.63

Table A.3: The mean SNRs of the models on destabilized videos and different stabilizers on the test set

Model	Median Face Box	Template Matching	Optical Flow	MOSSE
EfficientPhys	11.69	11.46	9.95	9.7
Physnet	3.64	3.34	3.82	5.04
TSCAN	14.82	10.51	11.45	12.45
RythmFormer	2.89	4.19	2.94	3.58
PhysFormer	6.9	7.82	5.24	5.6

Table A.4: The MAEs of the models on stable videos and different stabilizers on the validation set

Model	Median Face Box	Template Matching	Optical Flow	MOSSE
EfficientPhys	23.65	13.91	19.63	12.23
Physnet	6.96	4.03	8.37	3.17
TSCAN	38.66	24.72	26.94	23.66
RythmFormer	12.69	4.99	17.47	5.67
PhysFormer	14.32	12.45	16.71	6.7

Table A.5: The MAEs of the models on destabilized videos and different stabilizers on the validation set

Model	Median Face Box	Template Matching	Optical Flow	MOSSE
EfficientPhys	6.74	5.93	6.11	5.27
Physnet	2.37	2.16	2.72	2.63
TSCAN	9.61	6.92	7.02	9.05
RythmFormer	1.88	3.8	4.05	3.66
PhysFormer	2.34	3.78	2.44	3.41

Table A.6: The MAEs of the models on stable videos and different stabilizers on the test set

Model	Median Face Box	Template Matching	Optical Flow	MOSSE
EfficientPhys	19.19	6.72	16.4	6.48
Physnet	5.39	2.92	5.47	2.5
TSCAN	25.14	19	21.79	23.73
RythmFormer	14.43	5.85	15.6	5.21
PhysFormer	10.79	7.43	11.75	3.28

Table A.7: The MAEs of the models on destabilized videos and different stabilizers on the test set

A.4.3 PPG Postprocessing/Heart Rate Filter

Preprocessing	Peaks	Custom	FFT	Autocorrelation	FFT+Autocorrelation	DNN
Raw	2.47	2.58	18.68	5.36	14.19	3.47
Cumsum	3.56	3.77	4.6	2.82	23.44	3.78
Detrend	2.95	3.82	4.59	2.61	23.76	3.94
Butter	2.34	3.22	4.6	2.39	30.45	4.1

Table A.8: The MAEs of the postprocessing hyperparameter search on the test set with the likelihood model

Preprocessing	Peaks	Custom	FFT	Autocorrelation	FFT+Autocorrelation	DNN
Raw	2.8	2.3	18.74	7.22	10.99	3.7
Cumsum	2.86	4.22	5.06	3.7	21.09	8.33
Detrend	2.34	4.39	4.89	3.56	23.15	4.31
Butter	1.85	3.3	4.74	2.33	32.78	4.78

Table A.9: The MAEs of the postprocessing hyperparameter search on the test set with the quantile regression model

A.4.4 Skin Tones

Model	Type 1	Type 2	Type 3	Type 4	Type 5
EfficientPhys	22.48	0.88	13.31	2.93	14.36
Physnet	0.24	0.61	0.91	2.88	0.31
TSCAN	10.55	19.86	9.75	23.94	8.82
RythmFormer	0.24	0.61	1.92	3.81	0.62
PhysFormer	1.11	0.6	1.15	2	0.51
Physnet_Quantile	0.24	0.91	0.12	1.74	0.72
Physnet_NegLogLikelihood	0.24	0.31	0.41	9.37	0.72

Table A.10: The MAEs of the standard pipeline by fitzpatrick value

A.4.5 Real World Evaluation

Model	Standard	Standard Destabilized	MOSSE	MOSSE Destabilized
EfficientPhys	27.41	19.76	25.83	28.94
Physnet	24.8	15.88	23.58	21.32
TSCAN	24.82	16.93	27.63	20.4
RythmFormer	19	13.75	25.41	23.97
PhysFormer	24.36	17.05	31.63	32.99
Physnet_Quantile	32.19	15.41	27.24	24.36
Physnet_NegLogLikelihood	22.79	8.32	24.41	24.78

Table A.11: The MAEs for standard pipeline versus the best performing stabilizer for stable and unstable videos on the RealisticVideos dataset

A.4.6 Compression

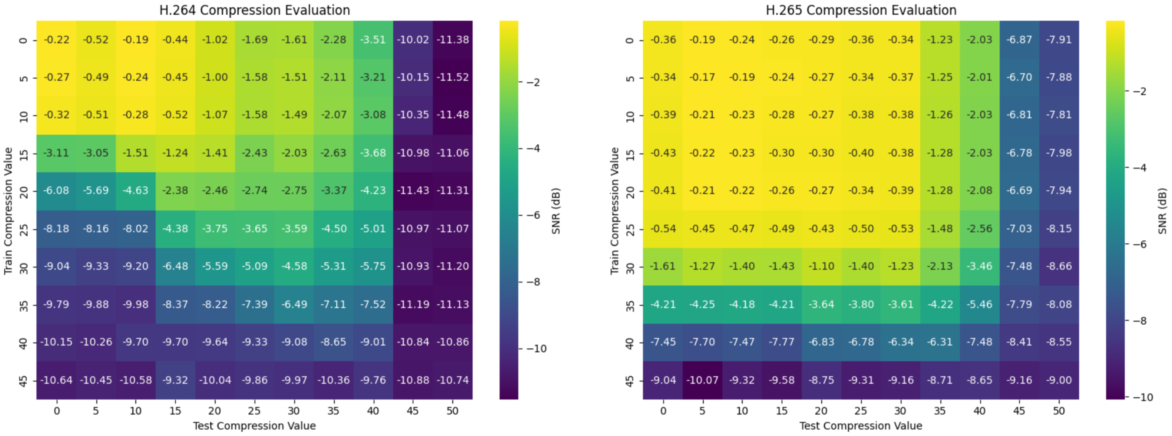

Figure A.1: The evaluation on the compressed videos of the test dataset with H.264 on the left and H.265 on the right

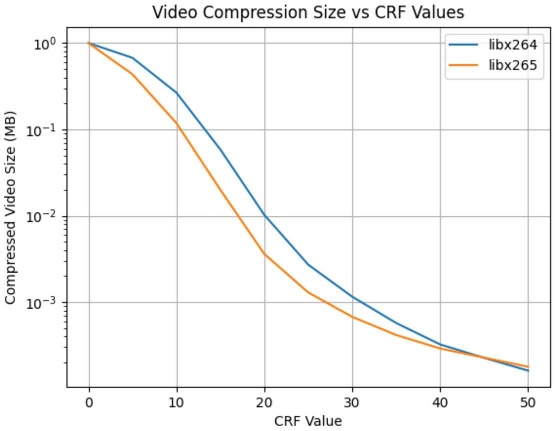

Figure A.2: The video file size versus the CRF value for the H.264 and the H.265 algorithm

A.4.7 Conformalization

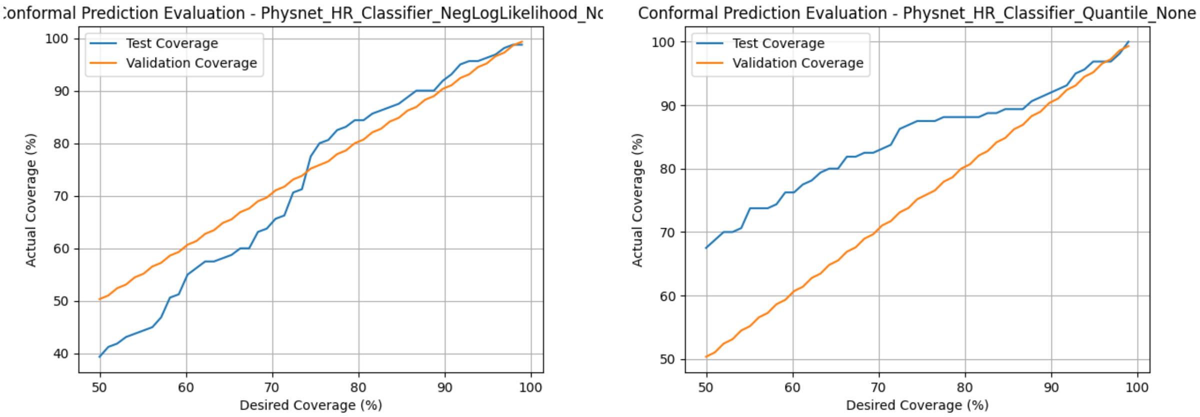

Figure A.3: Conformalization evaluation for the heart rate classifier without postprocessing with gaussian uncertainty estimation on the left and quantile regression on the right

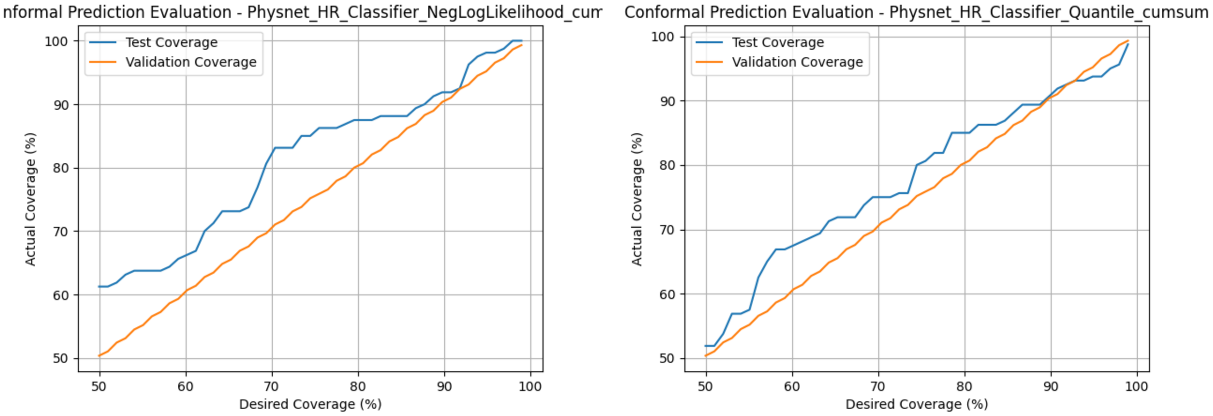

Figure A.4: Conformalization evaluation for the heart rate classifier with the cumsum filter as postprocessing with gaussian uncertainty estimation on the left and quantile regression on the right

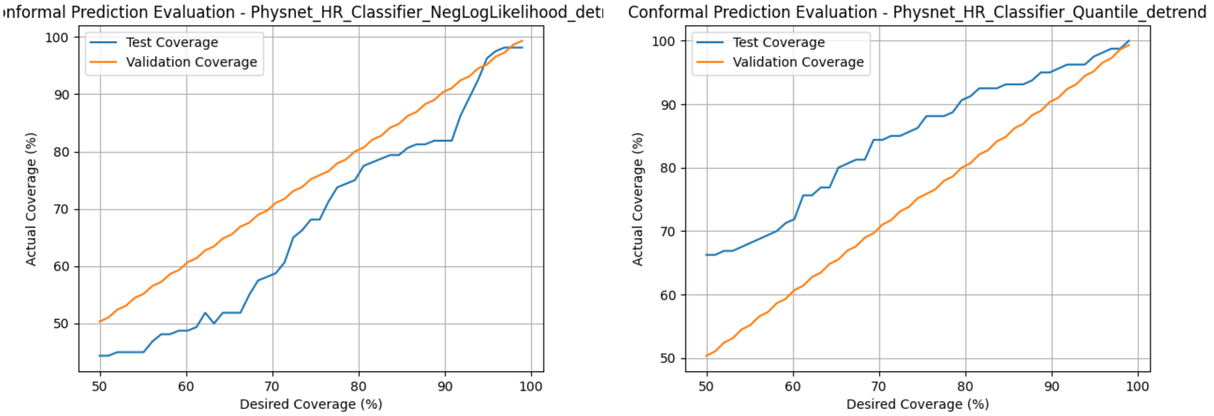

Figure A.5: Conformalization evaluation for the heart rate classifier with the detrending filter as postprocessing with gaussian uncertainty estimation on the left and quantile regression on the right

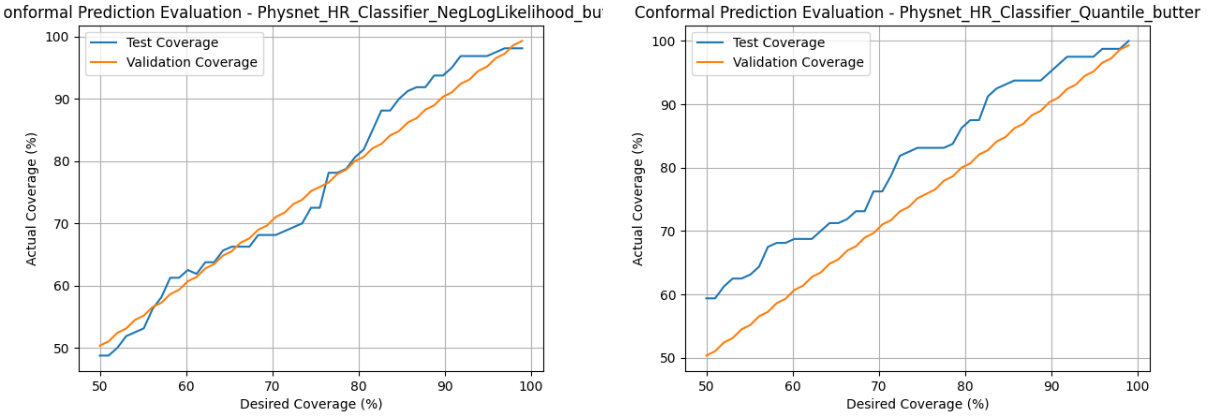

Figure A.6: Conformalization evaluation for the heart rate classifier with the Butterworth filter as postprocessing with gaussian uncertainty estimation on the left and quantile regression on the right

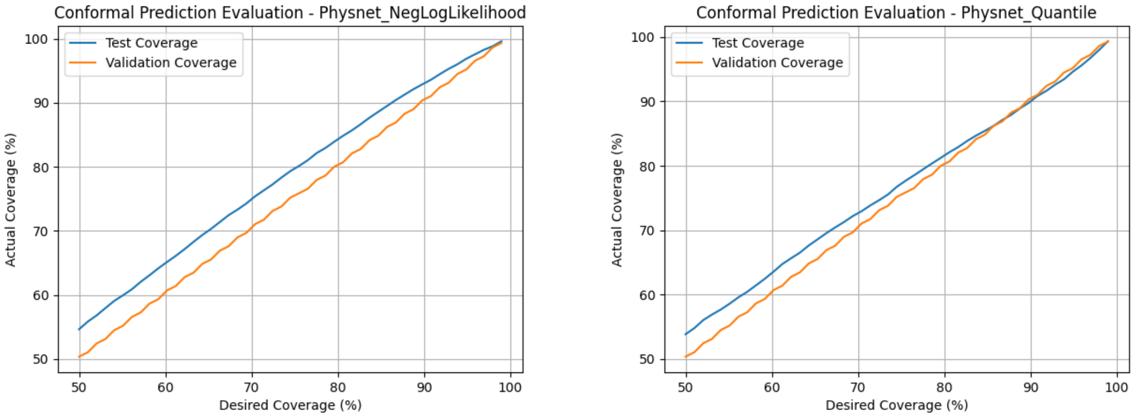

Figure A.7: The results for the conformalization evaluation of the PPG extractor with gaussian uncertainty estimation on the left and quantile regression on the right

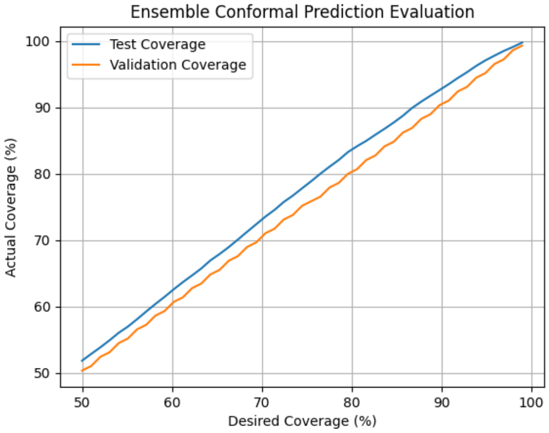

Figure A.8: The results for the conformalization evaluation of the PPG extractor using the Ensemble method

A.4.8 Computation efficiency improvement

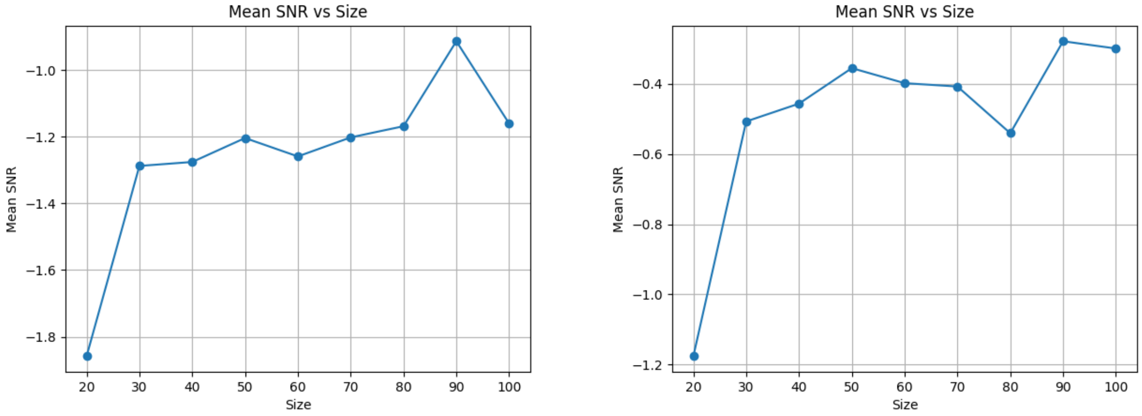

Figure A.9: The Mean SNR resulting from the evaluation of the Physnet on images with different size. On the left hand side the validation set and on the right hand side the test set

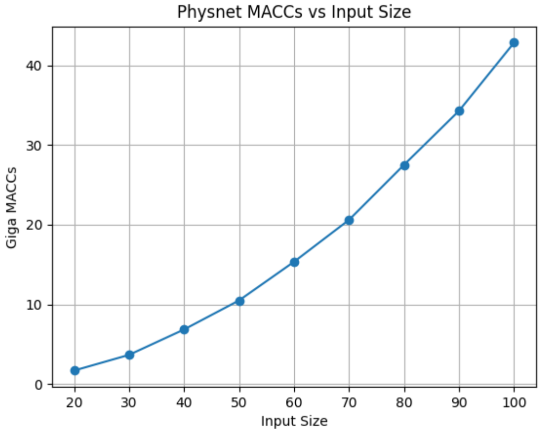

Figure A.10: The number of MACCs needed to evaluate the Physnet on a given input size

Acronyms

ADB Android Debug Bridge. 7

AI Artificial Intelligence. 8, 27, 37, 38

CNN Convolutional Neural Network. 4, 8

CQP Constant Quantization Parameter. 13

CRF Constant Rate Factor. 13, 23, 34, 35, 42, III

FFT Fast Fourier Transform. 8–10, 17, III

GPU Graphics Processing Unit. 7

HTTP Hyper Text Transfer Protocol. 35

MACC Multiply ACCumulate. 23, 24, 45, III

MAE Mean Absulute Error. 21, 38–41, V

MOSSE Minimum Output Sum of Squared Error. 8, 27, 28, 31, 34, 37

PPG Photoplethysmography. 4–7, 9, 10, 13, 14, 16, 17, 27, 33, 37, 38, 44, III

RGB Red Green Blue. 3

rPPG remote Photoplethysmography. 1, 3, 4, 6, 7, 9, 15, 22, 23, 27–29, 31

SNR Signal to Noise Ratio. 17, 19–23, 28, 38, 39, III, V

USB Universal Serial Bus. 33

YAML YAML Ain’t a Markup Language. 34, 36, 37

Bibliography

[1] Ffmpeg h.264 video encoding guide. `https://trac.ffmpeg.org/wiki/Encode/H.264`. Accessed: 2025-08-21.

[2] Caerwyn Ash, Michael Dubec, Kelvin Donne, and Tim Bashford. Effect of wavelength and beam width on penetration in light-tissue interaction using computational methods. *Lasers in Medical Science*, 32(8):1909–1918, 2017.

[3] Bryce E. Bayer. Color imaging array. (US3971065A), July 20 1976. U.S. Patent.

[4] Abdallah Benhamida and Miklos Kozlovszky. Mc-evm: A movement-compensated evm algorithm with face detection for remote pulse monitoring. *Applied Sciences*, 15(3), 2025.

[5] S. Bobbia, R. Macwan, Y. Benezeth, A. Mansouri, and J. Dubois. Unsupervised skin tissue segmentation for remote photoplethysmography. *Pattern Recognition Letters*, 2017.

[6] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550, 2010.

[7] Raymundo Cassani, Abhishek Tiwari, and Tiago Falk. Optimal filter characterization for photoplethysmography-based pulse rate and pulse power spectrum estimation. volume 2020, pages 914–917, 07 2020.

[8] Hung-Chang Chang, Shang-Hong Lai, and Kuang-Rong Lu. A robust and efficient video stabilization algorithm. In *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)*, volume 1, pages 29–32 Vol.1, 2004.

[9] Mike Stephens Chris Harris. A combined corner and edge detector. 1988.

[10] Gerard de Haan and Vincent Jeanne. Robust pulse rate from chrominance-based rppg. *IEEE Transactions on Biomedical Engineering*, 60(10):2878–2886, 2013.

[11] Karapinar Ercument and Sevinc Ender. A non-invasive heart rate prediction method using a convolutional approach. 2024.

[12] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. volume 2749, pages 363–370, 06 2003.

[13] TB Fitzpatrick. Sun and skin. *Journal de Medecine Esthetique*, 2:33–34, 1975.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[15] Ilya Kalinovskii and Vladimir Spitsyn. Compact convolutional neural network cascade for face detection, 2015.

[16] Ce Liu, Antonio Torralba, William T. Freeman, Frédo Durand, and Edward H. Adelson. Motion magnification. *ACM Trans. Graph.*, 24(3):519–526, July 2005.

[17] Xin Liu, Josh Fromm, Shwetak Patel, and Daniel McDuff. Multi-task temporal shift attention networks for on-device contactless vitals measurement, 2021.

[18] Xin Liu, Brian L. Hill, Ziheng Jiang, Shwetak Patel, and Daniel McDuff. Efficientphys: Enabling simple, fast and accurate camera-based vitals measurement, 2022.

[19] Xin Liu, Girish Narayanswamy, Akshay Paruchuri, Xiaoyu Zhang, Jiankai Tang, Yuzhe Zhang, Roni Sengupta, Shwetak Patel, Yuntao Wang, and Daniel McDuff. rPPG-toolbox: Deep remote PPG toolbox. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[20] Florian Moll. rppg-extractor. `https://github.com/flmoll/rPPG-Extractor`, 2025. Zugriff am 30. August 2025.

[21] Nhi Nguyen, Le Nguyen, Honghan Li, Miguel Bordallo López, and Constantino Álvarez Casado. Evaluation of video-based rppg in challenging environments: Artifact mitigation and network resilience, 2024.

[22] Ewa M. Nowara, Daniel McDuff, and Ashok Veeraraghavan. A meta-analysis of the impact of skin tone and gender on non-contact photoplethysmography measurements. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[23] Akshay Paruchuri, Xin Liu, Yulu Pan, Shwetak Patel, Daniel McDuff, and Soumyadip Sengupta. Motion matters: Neural motion transfer for better camera physiological measurement. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 5933–5942, January 2024.

[24] Delong Qi, Weijun Tan, Qi Yao, and Jingfeng Liu. Yolo5face: Why reinventing a face detector, 2022.

[25] Yaniv Romano, Evan Patterson, and Emmanuel J. Candès. Conformalized quantile regression, 2019.

[26] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

[27] Mika Tarvainen, Perttu Ranta-aho, and Pasi Karjalainen. An advanced detrending method with application to hrv analysis. *IEEE transactions on bio-medical engineering*, 49:172–5, 03 2002.

[28] Pieter-Jan Toye. Vitalvideos-europe: A dataset of face videos with ppg and blood pressure ground truths, 2025.

[29] Wim Verkruysse, Lars O Svaasand, and J Stuart Nelson. Remote plethysmographic imaging using ambient light. *Opt. Express*, 16(26):21434–21445, Dec 2008.

[30] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001.

[31] Vytautas Vizbara. Comparison of green, blue and infrared light in wrist and forehead photoplethysmography. In *BioMed*, 2013.

[32] Neal Wadhwa, Michael Rubinstein, Frédo Durand, and William T. Freeman. Phase-based video motion processing. *ACM Trans. Graph.*, 32(4), July 2013.

[33] Kegang Wang, Jiankai Tang, Yuxuan Fan, Jiatong Ji, Yuanchun Shi, and Yuntao Wang. Memory-efficient low-latency remote photoplethysmography through temporal-spatial state space duality, 2025.

[34] Miao Wang, Guo-Ye Yang, Jin-Kun Lin, Ariel Shamir, Song-Hai Zhang, Shao-Ping Lu, and Shi-Min Hu. Deep online video stabilization, 2018.

[35] Hao-Yu Wu, Michael Rubinstein, Eugene Shih, John Guttag, Frédo Durand, and William Freeman. Eulerian video magnification for revealing subtle changes in the world. *ACM Trans. Graph.*, 31(4), July 2012.

[36] Wei Wu, Hanyang Peng, and Shiqi Yu. Yunet: A tiny millisecond-level face detector. *Springer Nature Link*, 2023.

[37] Yeou-Min Yeh, Sheng-Jyh Wang, and Huang-Cheng Chiang. A digital camcorder image stabilizer based on gray coded bit-plane block matching. *Proceedings of SPIE - The International Society for Optical Engineering*, 40:9, 07 2000.

[38] Zitong Yu, Xiaobai Li, and Guoying Zhao. Remote photoplethysmograph signal measurement from facial videos using spatio-temporal networks, 2019.

[39] Zitong Yu, Yuming Shen, Jingang Shi, Hengshuang Zhao, Philip Torr, and Guoying Zhao. Physformer: Facial video-based physiological measurement with temporal difference transformer, 2022.

[40] Bochao Zou, Zizheng Guo, Jiansheng Chen, Junbao Zhuo, Weiran Huang, and Huimin Ma. Rhythmformer: Extracting patterned rppg signals based on periodic sparse attention, 2025.