POLITECNICO DI TORINO

Master degree course in Computer Engineering,
Artificial Intelligence and Data Analytics

Master Degree Thesis

Exploring Self-supervised Learning for PPG-based Heart-Rate Estimation

Supervisors
Prof. Daniele Jahier Pagliari
Dr. Alessio Burrello
Dr. Kasnesis Panagiotis

Candidate
Flavio Patti

October 2023

Abstract

In recent years, Heart Rate (HR) monitoring is becoming increasingly widespread in wrist-worn devices where low-cost photoplethysmography (PPG) sensors are installed. On the other hand, the accuracy of PPG-based HR tracking is often compromised by Motion Artifacts (MAs), which result from movements of the subject's arm, and cause degradation in the quality of the PPG signal gathered. To mitigate this issue, the PPG signal is commonly combined with acceleration measurements obtained from an inertial sensor. In the state-of-the-art, many traditional methods based on temporal and frequency filters and, more recently, deep learning algorithms have been exploited to combinate the information from these two sensors. In this thesis, driven by the recent achievements of self-supervised learning, we investigate its application for PPG-based heart rate tracking. The main disadvantage of using this kind of approach is that it requires a large amount of data to process. This problem has only been overcome in recent years, with the introduction of large datasets such as PPG-Dalia and WESAD. Once data have been made available, self-supervised learning could represent an alternative and innovative solution to obtain a good level of accuracy in model predictions, allowing better generalization. Adopting Masked Autoencoders as reference model, we exploit self-supervised learning to teach a neural network how to reconstruct PPG signals both in time and in frequency. During this pre-training step, the layers of the network learn to extract good features for the task. Subsequently, we fine-tune our Masked Autoencoder, substituting the decoder with a regressive tail and leveraging the knowledge acquired from the previous phase to estimate the heart rates of the patients. We reach an average of Mean Absolute Error (MAE) over all patients of 6.14 Beats Per Minute (BPM) for PPG-Dalia and 5.41 BPM for WESAD (better than state-of-the-art of 2.06 BPM). Applying an additional post-processing step, the MAE is further reduced to 5.72 BPM and 5.19 BPM, respectively.

Furthermore, we analyze the advantages of using Transfer Learning during the self-supervised pre-training step. This involves pre-training our reference model on one dataset and then fine-tuning it on the other, and vice versa. Through this approach, we demonstrate that, for the majority of the patients, it is possible to improve performance and further decrease MAE.

Acknowledgements

I would like to express my sincere gratitude to Prof. Daniele Pagliari, Dr. Kasnesis Panagiotis and Dr. Alessio Burrello for giving me the invaluable opportunity to undertake this challenging yet exciting journey. This experience allowed me to test myself, mature, and develop new knowledge and awareness that I will carry with me as priceless luggage throughout my life. For all this, I will always be grateful.

A special mention to Dr. Alessio Burrello, who followed me during these eight months and guided me, step by step, to the successful completion of this thesis. Thank you for all the advice you gave me and for your immense availability. Today, I don't just regard you as a supervisor but as a friend, and I hope to meet you even outside the university context.

To my grandmother, who has always been close praying for every single exam I've done in these 5 years.

To my father, who, in his own way, always supported me in all my choices.

To my mother, who, during difficult moments of my life, always provided me the right words, enabling me to face and overcome them.

To all my family, you are my strength.

Contents

List of Tables

List of Figures

Acronym

AI Artificial Intelligence

BPM Beats Per Minute

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

ECG Electrocardiogram

HR Heart Rate

ICA Independent Component Analysis

LED Light-Emitting Diode

MA Motion Artifacts

MAE Mean Absolute Error

MAE Masked Autoencoder

MCU Microcontrollers

MLP Multi-Layer Perceptron

MSA Multi-head Self-Attention

MSE Mean Square Error

NLP Natural Language Processing

PPG Photoplethysmography

RNN Recurrent Neural Network

TCN Temporal Convolutional Network

ViT Vision Transformer

Chapter 1

Introduction

Nowadays, wrist-worn devices [8] have gained significant popularity due to their convenience, portability, and ability to collect and process various types of data. In particular, this devices are a class of wearable technology designed to be worn on the wrist and they offer a range of functionalities such as activity-tracking and health-monitoring aimed at improving daily lives of people. Indeed, heart-rate (HR) monitoring is fundamental for clinical purposes and precise activity tracking. Early wrist-worn HR tracking devices rely on a separate chest band, equipped with a simple 1-3 leads Electrocardiogram (ECG) sensor. While provide accurate results, this solution was expensive and users often found it uncomfortable to wear in their daily lives. However, in recent years, a more convenient and cost-effective alternative has emerged and the ECG chest bands have been progressively replaced by photoplethysmography (PPG) sensors, which allow the direct measurement of HR and blood oxygenation (SpO2) from wrist-worn devices. A prominent example of commercial devices embracing this technology are the Apple Watch [9] and some Fitbit models [41].
PPG sensors are composed in one or more Light-Emitting Diodes (LEDs) that periodically emit light onto the skin and a photodetector (i.e., a photodiode) that measures the variations of light intensity caused by blood flow [10], [11]. More specifically, the larger the blood volume variation, the greater the attenuation of the light emitted by the LED, resulting in a reduced current output on the photodiode. This relationship allows peaks in the PPG signal to be associated with the user's HR [12], making it a reliable approach

for heart rate tracking. The main problem in PPG sensors is constituted by motion artifacts (MA), i.e., signal artifacts caused by movements of the user arm and hand, which produce variability in the sensor pressure on the skin or the light coming from the surrounding environment that infiltrates between the photodiode and the wrist and this inevitably leads to inaccuracies in the measurements. Numerous studies and investigations have been conducted to compare the performances of electrocardiogram (ECG) chest straps and photoplethysmogram (PPG)-based heart rate (HR) tracking systems [42] [43]. These studies have consistently demonstrated that the former generally achieve better levels of accuracy, particularly when encountering motion artifacts (MAs). Consequently, ECG-centered solutions continue to be upheld as the established benchmark for wearable HR tracking [44]. On the other hand, PPG sensors are nowadays the standard sensor for consumer products, given their non-invasiveness.

To cope, however, with the problems of MAs, some solutions have been proposed which employ the correlation between acceleration data and the PPG signal to cancel the noise and remove the MAs. The majority of these methodologies rely on traditional signal processing algorithms, including Independent Component Analysis (ICA), Wiener Filters, and Spectral Peak Detection [45]-[48]. Notably, seminal contributions to this domain include TROIKA [12] and its subsequent iteration, JOSS [25]. These works employ adaptive filtering to estimate the interference stemming from motion artifacts (MAs), followed by the use of spectral peak tracking to discern the heart rate frequency within the photoplethysmogram (PPG) signal. A fundamental constraint inherent in these methodologies regards to the substantial quantity of unconstrained hyper-parameters, a characteristic that frequently curtails their capacity for broad generalization.

Driven by the notable achievements of deep learning in various bio-signal applications, researchers have turned their attention to exploring the potential of deep Neural Networks (NNs) for PPG-based heart rate tracking. Furthermore, the publication of large datasets for heart rate monitoring in the presence of MAs such as PPG-Dalia and WESAD has further increased the rapid expansion of this new category of PPG-based HR prediction algorithms that relies on deep learning approaches. PPG-Dalia is a large dataset for motion compensation and heart rate estimation during daily life activities. There are in total 15 subjects who participated in the data collection: 8 female and 7 male participants with age ranging from 21 to 55 years old. WESAD,

instead, is a multimodal dataset for wearable stress and affect detection. The data collection involved 15 participants, aged between 24 and 35 years. The primary goal behind this dataset was to collect a wide amount of data to identify and differentiate various affective states, such as neutral, stress, and amusement. In addition to the two mentioned datasets, other smaller datasets have also been published such as IEEE_Train & IEEE_Test. These datasets comprise 12 sessions, each session is recorded from a distinct subject aged between 18 and 58 years. The sessions encompass activities like treadmill running at varying speeds, forearm and upper arm exercises (such as arm rehabilitation exercises), and intensive arm movements (such as boxing). Each session has an approximate duration of five minutes.
Within the family of algorithms related to Deep Learning, an interesting and promising solution is given by self-supervision learning. Self-supervision is a learning approach where a model is trained to learn from unlabelled data by automatically generating training labels. This approach is useful in a variety of contexts where obtaining large amounts of labeled data might be expensive or difficult. In particular, it has been proven to enable AI systems to recognize and understand generalizable patterns in fields such as natural language processing (NLP), computer vision and speech recognition.

In this thesis, we exploit the advantages of self-supervised learning to predict the heart rates from PPG signals. Particularly, this goal is reached in two successive steps, according to the self-supervised learning approach. The first is the pre-training phase, in which we use Masked Autoencoders (MAE) [7] as reference model to test their ability in extracting good features from data by reconstructing the input signals both in time domain and in frequency domain (i.e., spectrograms). Subsequently, in the finetuning phase, we slightly change the last part of the network in order to adapt it to the final task in which we are really interested in, using the knowledge acquired from the previous step to estimate the heart rates of the various patients present in the mentioned dataset. In the end, we compare the obtained results applying also a further step of post-processing since the dynamics characterizing the human heart rate impose an upper bound on the permissible variation of the estimation over time, in normal conditions. To improve the results, we explored Transfer Learning. A possible disadvantage of using deep NNs is, indeed, that they often require big amounts of labeled data for training, which can be expensive and time-consuming to collect in this context. Motivated by this observation, we investigate transfer learning since it mitigates this

problem by allowing models to exploit knowledge gained from one dataset and apply it to another, often related, dataset.

Within this thesis you can find the following main contributions:

1. We demonstrate that **Masked Autoencoders** are a powerful model that can be employed also for PPG signals. Indeed, we used a Masked Autoencoder with 12 attention blocks to **reconstruct PPG signals** in time and a smaller Masked Autoencoder with 4 attention blocks to reconstruct PPG signals in frequency.
   This procedure is explained in details in Section 4.3 and Section 4.5 where Figures 4.7, 4.8 and 4.9 show some examples of reconstructed signals in the two different domains.

2. We analize the behavior of Masked Autoencoders for the **estimation of heart-rates**, focusing on two different datasets: **PPG-Dalia** and **WESAD**. Respectively, we reach an average of Mean Absolute Error (MAE) over all patients of 6.14 and 5.41 Beat Per Minute (BPM). Applying an additional post-processing step, the MAE is further reduced to 5.72 and 5.19 BPM.
   Particularly, for WESAD dataset it represents a new state-of-art since in the previous work [13], the authors proposed a model based on CNNs reaching a reference MAE of 7.47 BPM.

3. We investigate **Transfer Learning** between the two mentioned datasets. Thanks to this approach, on the majority of the patients it is possible to generalize better and further decrease MAE. This is particularly evident on patient 1 (S1) where the application of transfer learning on PPG-Dalia using WESAD as a pre-training dataset helps the model to improve accuracy by 0.56 BPM, smoothing the error from 5.86 BPM to 5.30 BPM. Instead, by looking at the overall results on WESAD using PPG-Dalia as a pre-training dataset, we are able to further decrease MAE to 5.09 BPM.

Chapter 2

Background

2.1 Artificial Intelligence

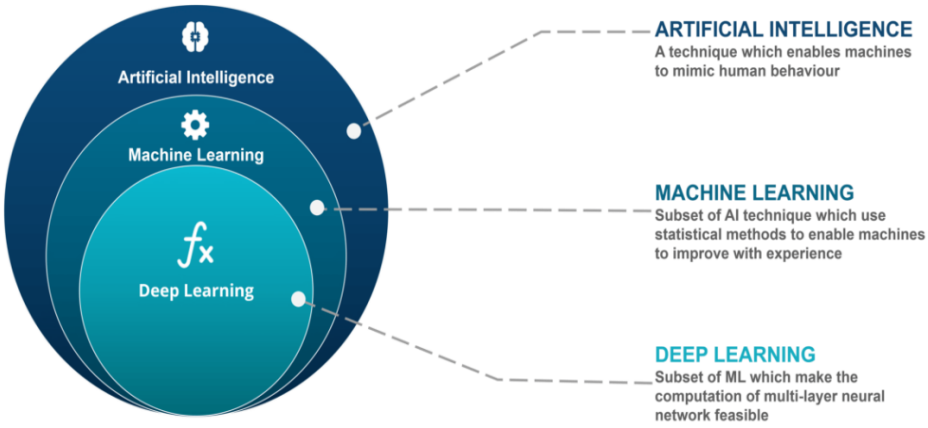


Figure 2.1: Distinction between Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL).

Artificial intelligence (AI) is a field of computer science that focuses on creating systems, programs, and machines that can perform tasks that normally require human intelligence. The main goal of AI is to develop algorithms and models that allow machines to learn from data, draw conclusions, adapt to new situations and make autonomous decisions as a human being. The applications of AI are vast and span across industries such as healthcare, finance,

manufacturing, transportation, and more. In healthcare sector, AI can help in disease diagnosis and predicting patient prognosis. In finance, it aids in fraud detection, risk assessment, and algorithmic trading. In manufacturing, AI-powered robots streamline production processes. Self-driving cars rely on AI to navigate and make real-time decisions on the road.

2.1.1 Learning algorithms: Supervised, Unsupervised, Self-supervised

Within Artificial Intelligence there are fundamental paradigms. The main difference between these approaches lies in the type of data they use for training and the learning objectives they aim to achieve:

- **Supervised Learning**: is a Learning approach where training data consists of input-output pairs, where the inputs (features) are the data, and the outputs (labels) are the corresponding target values. The goal of supervised learning is to learn a function that maps the input in the corresponding output in order to make accurate predictions mainly for unseen data.
  Examples of this kind are: classification, regression, object detection, etc...

- **Unsupervised Learning**: is a Learning approach that, on the other hand, deals with data without their labels. In this scenario, the algorithm is presented with a dataset containing only input features and is not provided with explicit output labels or target values. The goal of unsupervised learning is to find patterns, structures, or representations within the data without any predefined notion of what the output should be.
  Examples of this kind are: clustering, dimensionality reduction, etc...

- **Self-supervised Learning**: in this context, we use again data without the labels as in the case of unsupervised learning. The goal, this time, is try to extract features and knowledge from the data in order to use this kind of information in a transfer learning step (downstream task).
  Examples of this kind are: relative positioning and contrastive learning.

Since this thesis mainly focuses on the third scenario, below you can find a short additional sub-chapter in order to better understand how the latter is exploited.

Self-supervised Learning

In many real-world applications, obtaining labeled data can be expensive, time-consuming, or even infeasible. Conversely, there are frequently a huge amount of unlabeled data accessible. Self-supervised learning is designed to tackle this particular challenge by converting an unsupervised problem (lack of labels) into a supervised one through the design of pretext tasks. Indeed, in self-supervised learning, the first step is to create a pretext task. This task involves generating pseudo-labels from the available unlabeled data. The pseudo-labels are derived from specific properties or transformations of the data, which are generated without human intervention. The goal of the pretext task is to create a supervised learning setup using the unlabeled data and training the model in order to predict these pseudo-labels. After the model is trained on the pretext task, it has learned useful representations from the data. These representations capture meaningful information about the data's underlying structure, even though the model was not explicitly given any labels during the pretext task. The acquired information are then fine-tuned on a downstream task. A downstream task is a specific task that we are truly interested in, such as image classification, object detection, or semantic segmentation. In this step, the model's learned representations are used as a starting point, and the model is further fine-tuned on a labeled dataset for the downstream task. The fine-tuning step is exploited to further training the model with the labeled data to adapt the previously learned representations to the specifics of the target task.

Nowadays, the success of self-supervision is strictly correlated to the use of a new deep learning architecture called *Transformer* [18], which can handle large quantities of data in parallel. In our work, we will exploit indeed Transformer-based Masked Autoencoders [7].

2.1.2 Machine Learning and Deep Learning

Machine Learning is a branch of AI and refers to a wide range of algorithms and techniques that enable machines to learn from data without being explicitly programmed. It relies on discovering patterns and structures in the data, bringing machines to make predictions, classifications, and decisions autonomously. Some of the most common algorithms include Support Vector Machines [49], Random Forests [50], Linear Regression [51], and more.
In recent years, however, Deep Learning has garnered particular attention. Deep Learning is a subset of Machine Learning that leverages neural networks with multiple layers to autonomously extract features and acquire high-level representations from data. Deep neural networks consist of layers of interconnected neurons. Each neuron plays a crucial role in the learning process of the network. The general data processing flow of a neuron is described in Figure 2.2.

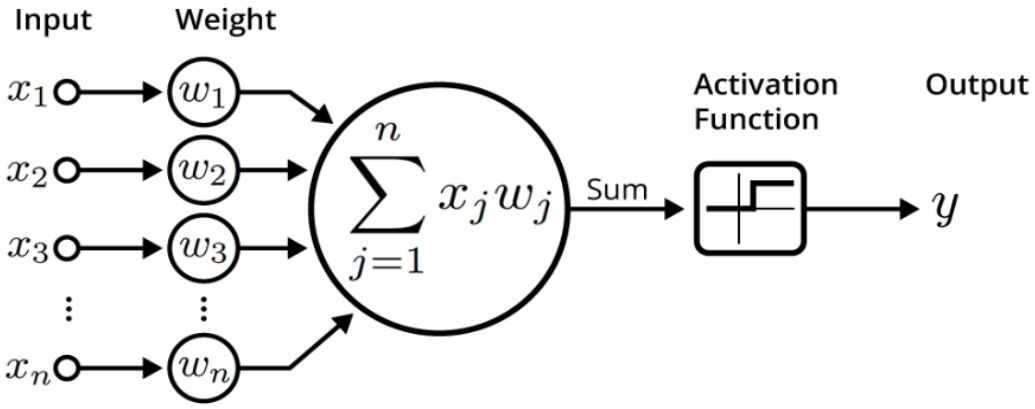


Figure 2.2: Artificial neuron.

1. **Input**: each neuron receives an input vector representing the data to be processed. These inputs can be the feature values of an image, the words in a sentence, or the time values in a time series.

2. **Weight**: the input is associated with a weight. The weights determine the relative importance of each input for the neuron. During the training phase of the network, the weights are updated so that the network can learn from the data.

3. **Weighted Sum**: each neuron performs a sum between its inputs multiplied by the corresponding weights.

4. **Activation Function**: after computing the weighted sum of the inputs, the neuron applies an activation function to the result. Usually, the activation function introduces non-linearities into the neuron, allowing the network to model complex relationships between input data.

5. **Output**: after the activation function, the neuron produces the output. This output can be sent to other neurons in successive layers of the network or constitute the final output of the network.

In deep learning models, neurons are organized into layers: input layer, hidden layers and the output layer, see Figure 2.3. The input data pass through each neuron in sequence, and learning occurs through the process of back propagation [17], in which the network adjusts weights to reduce the objective loss which is a function that measures the error between expected and actual outputs. Thanks to this complex process, a neural network can learn from data and become able to solve a wide variety of complex problems, such as image recognition and natural language processing, often achieving better performance compared to traditional Machine Learning techniques. Despite promising results, Deep Learning also presents significant challenges: training deep neural networks can require large training datasets and considerable computational resources.

2.2 Transformers and ViT

A Transformer is a type of deep learning model introduced in 2017 by Vaswani et al. [4]. Transformers were a significant innovation in the field of Natural Language Processing (NLP) and have subsequently proven to be highly effective in a wide range of machine learning applications, including automatic translation, text generation, speech synthesis, and much more. The main feature of Transformers is the "*attention*" mechanism, which allows the model to weigh the importance of different parts of an input during processing. This attention capability enables Transformers to handle long sequences of data effectively, improving their ability to capture long-term relationships

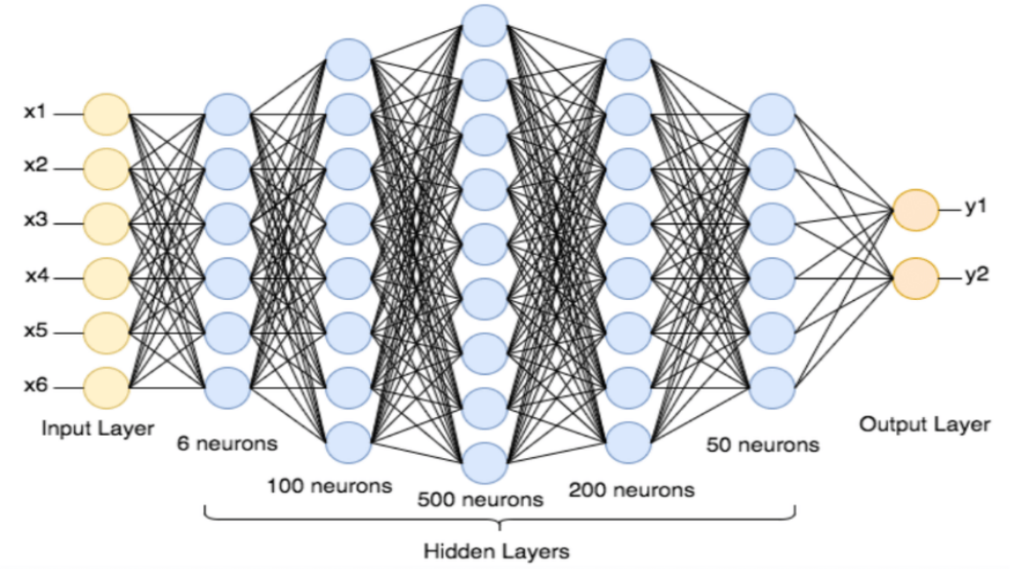


Figure 2.3: Deep Neural Network architecture.

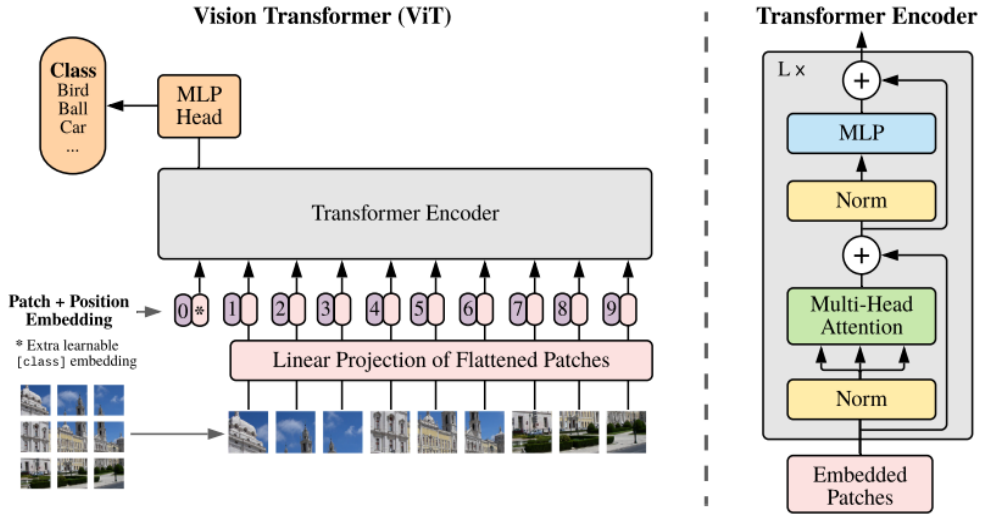


Figure 2.4: Model summary.

between words or features in the data. A "*Vision Transformer*" (or ViT), instead, is a variant of the Transformer designed specifically for image processing. Initially, Transformer models were primarily developed for natural language processing (NLP), but the idea of applying the same architecture to visual data proved to be very promising. Vision Transformers replace sequences of words or tokens in NLP models with a grid of image patches and use the attention mechanism to process these patches in the same way that NLP Transformers process words. The use of Vision Transformers has led to outstanding results in computer vision tasks such as object recognition, semantic segmentation, and generating descriptive text for images. These models have become an important part of the computer vision field and are used in applications requiring advanced understanding of images. Furthermore, the main advantage of use this kind of architectures is that, unlike the RNNs counterparts, they can process the entire sequence of input data in parallel focusing only to sub-parts of the input instead of the entire input.
An overview of the Vision Transformer model is depicted in Figure 2.4. The conventional Transformer takes as input a 1D sequence of token embeddings. To handle 2D images, as written in [18], the authors of the model did in this way: *"we reshape the image* $\boldsymbol{x} \in \mathbb{R}^{H \times W \times C}$ *into a sequence of flattened 2D patches* $\boldsymbol{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$*, where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and* $N = H \times W/P^2$ *is the resulting number of patches, which also serves as the effective input sequence length for the Transformer. The Transformer uses constant latent vector size D through all of its layers, so we flatten the patches and map to D dimensions with a trainable linear projection* (2.2)*, where* $\boldsymbol{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}$ *and* $\boldsymbol{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$*. We refer to the output of this projection as the patch embeddings."*
Then we prepend a learnable embedding to the sequence of embedded patches ($\mathbf{z}_0^0 = \mathbf{x}_{class}$), whose state, at the output of the Transformer encoder, ($\mathbf{z}_L^0$) is used as the image representation $\mathbf{y}$ (2.5). A classification head is added to the end of the model and to $\mathbf{z}_L^0$ and it is implemented by a single linear layer. Position embeddings are incorporated into the patch embeddings to preserve positional information. There are a lot of possible position embeddings, the main one is 2D sine-cosine position embedding which consists in:

$$PE(pos, 2i) = \sin(\frac{pos}{n^{2i/d_{model}}}); \quad PE(pos, 2i+1) = \cos(\frac{pos}{n^{2i/d_{model}}}); \quad (2.1)$$

where *pos* is the position in the input sequence, d_{model} is the output dimension of the embedding, n is a scalar and i is the dimension.

The sequence of resulting embedding vectors serves as input to the encoder. The Transformer encoder is composed of alternating layers of multiheaded self-attention (MSA) and MLP blocks (2.3), (2.4). Layer-normalization (LN) is applied before every block, and residual connections after every block. The MLP contains two linear layers with a GELU non-linearity.

$$\mathbf{z}_0 = [x_{class};\ \mathbf{x}_p^1\mathbf{E};\ \mathbf{x}_p^2\mathbf{E};\cdots;\ \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos} \tag{2.2}$$

$$\mathbf{z}'_\ell = MSA(LN(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1...L \tag{2.3}$$

$$\mathbf{z}'_\ell = MLP(LN(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1...L \tag{2.4}$$

$$\mathbf{y} = LN(\mathbf{z}_L^0) \tag{2.5}$$

2.2.1 Multihead self-attention

Standard **qkv** self-attention (SA) is a popular building block for neural architectures. For each element in an input sequence $\mathbf{z} \in \mathbb{R}^{N\times D}$, we compute a weighted sum over all values $\mathbf{v}$ in the sequence. The attention weights A_{ij} are based on the pairwise similarity between two elements of the sequence and their respective query $\mathbf{q}^i$ and key $\mathbf{k}^j$ representations.

$$[\mathbf{q}, \mathbf{k}, \mathbf{v}] = \mathbf{z}\mathbf{U}_{qkv}, \qquad \mathbf{U}_{qkv} \in \mathbb{R}^{(D\times 3D_h)} \tag{2.6}$$

$$A = softmax(\mathbf{q}\mathbf{k}^T/\sqrt{D_h}), \qquad A \in \mathbb{R}^{(N\times N)} \tag{2.7}$$

$$SA(\mathbf{z}) = A\mathbf{v} \tag{2.8}$$

Multihead self-attention (MSA) is an extension of SA in which we run k self-attention operations, called "heads", in parallel, and project their concatenated outputs. As highlighted previously, each head focuses on a relevant detail within the input and, therefore, having several heads within the same layer, it is possible to distinguish different definitions of relevance. For example, in NLP tasks, one head might focus on the next word, one on the subject, and another on the verb of the entire sentence. To keep compute and number of parameters constant when changing k, D_h (2.6) is typically set to D/k.

2.3 Masked Autoencoder

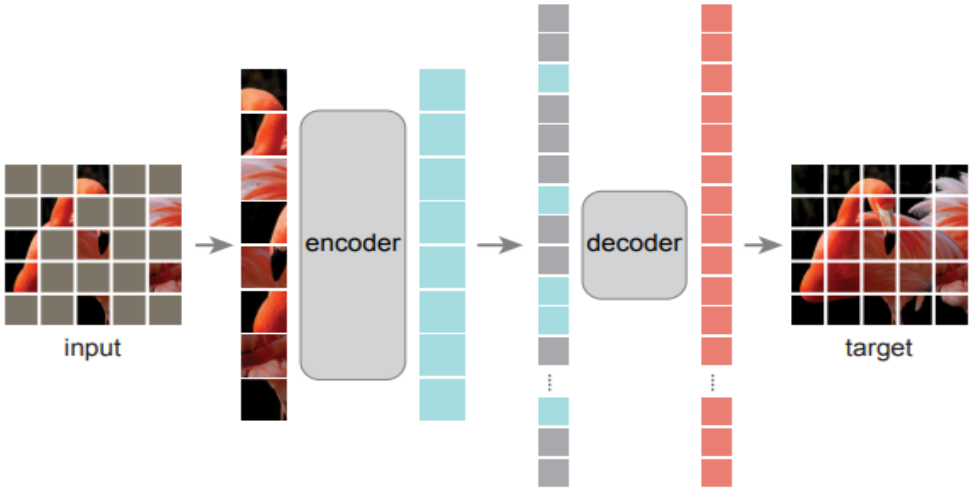


Figure 2.5: MAE architecture.

With the maturity of deep learning, modern networks are becoming more and more deep, since they try to solve complex tasks, which necessitate of deep and width neural network architectures. On the other hand, training such architectures is not trivial, given problems such as vanishing gradient, exploding gradient or simply underfitting. One of the possible solution to these issues are Masked autoencoders (MAE), a form of more general denoising autoencoders [3]. The concept behind this innovative architecture is fundamentally straightforward: it involves eliminating a portion of the data and training the model to predict the missing content. The concepts of masked autoencoders are very intuitive and applicable both in NLP and computer vision, enabling the training of models containing over one hundred bilion parameters.

Kaiming He et al. [7] introduces a efficient and scalable version of a masked autoencoder (MAE) designed for visual representation learning. Their MAE masks random patches from the input image and subsequently reconstructs the absent patches in the pixel space. The architecture features an asymmetrical encoder-decoder design. The encoder exclusively processes the visible subset of patches, excluding the mask tokens. The decoder is lightweight and reconstructs the input using the latent representation and the mask tokens (Figure 2.5). Furthermore, the transfer of mask tokens to the smaller

decoder results in a notable reduction in computational requirements. Furthermore, the transfer of mask tokens to the smaller decoder results in a notable reduction in computational demands. This design allows for the use of a significantly high masking ratio, such as 75%, which yields a beneficial outcome: it improves accuracy while enabling the encoder to process only a small fraction, e.g., 25%, of patches. As a result, the overall pre-training time can be reduced by at least 3 times or more, and memory consumption is also minimized, facilitating the efficient scaling of the MAE to larger models.

Since this is the main architecture used for the different experiments conducted in this thesis, I would like to focus more on explaining the various implementation details:

- **Masking.** Following ViT [6], the authors partition an image into non-overlapping patches. Then they sample a subset of patches and mask (i.e., remove) the remaining ones. They sample random patches without replacement, following a uniform distribution and refer to this strategy as "random sampling". Employing random sampling with a *high* value of masking ratio effectively reduces redundancy, resulting in a task that cannot be solved in a easy way by extrapolating from neighboring visible patches (see Figure 2.6). The uniform distribution ensures the avoidance of potential center bias, where more masked patches are concentrated near the image center.

- **MAE encoder.** The encoder in this architecture is based on the Vision Transformer (ViT) [6] saw in previous sub-section, but it operates solely on the visible, unmasked patches. Just like in a standard ViT, the encoder projects the patches linearly and adds positional embeddings to the result. It then processes the set of patches through a sequence of Transformer blocks. However, unlike the traditional ViT, this encoder only works with a small subset (e.g., 25%) of the complete patch set. Masked patches are excluded, and no mask tokens are utilized. Thanks to this approach is possible to train very large encoders using only a fraction of the computational power and memory.

  The input of the MAE decoder includes (i) encoded visible patches and (ii) mask tokens (refer to Figure 2.4). Each mask token [2] represents a shared, learned vector indicating the presence of a missing patch that needs to be predicted. Positional embeddings are added to ensure that

mask tokens have information about their location in the image. Similar to the encoder, the decoder consists of a series of Transformer blocks. It is important to note that the MAE decoder is exclusively employed for the purpose of image reconstruction during the pre-training phase. As a result, the decoder's architecture can be flexibly designed *independently* of the encoder's design. Using this design, the full set of tokens is solely processed by the lightweight decoder, leading to a significant reduction in pre-training time.

- **Reconstruction target.** Their MAE reconstructs the input by predicting the pixel values for each masked patch. Each element in the decoder's output is a vector of pixel values reprenting a patch. The final layer of the decoder is a linear projection that has the same number of output channels as there are pixel values in a patch. The decoder's output is reshaped to form a reconstruced image. The loss function computes the distance in pixel between the original and reconstructed images through the *Mean Squared Error* (MSE) metric. The loss is computed only on masked patches.

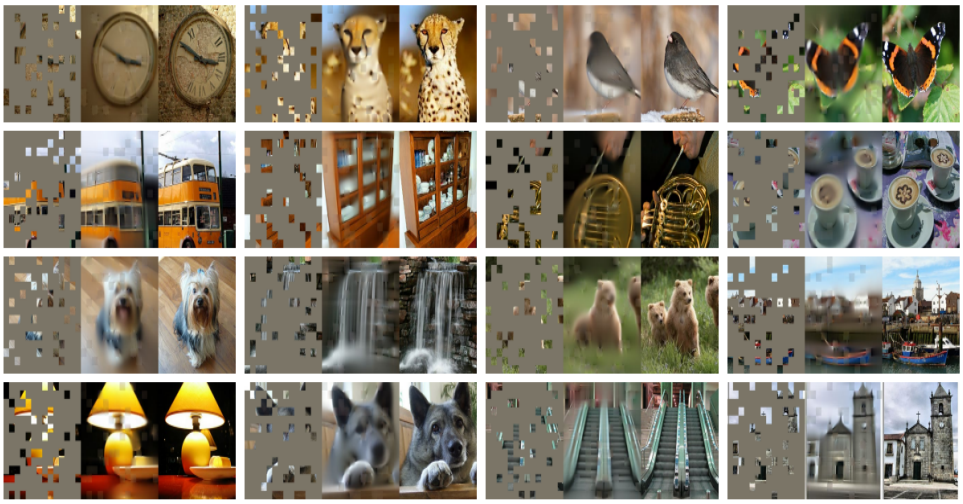


Figure 2.6: Results on ImageNet *validation* images. Left: masked image. Middle: MAE reconstruction. Right: Ground-truth. The masking ration is 80%, leaving only 39 out of 196 patches.

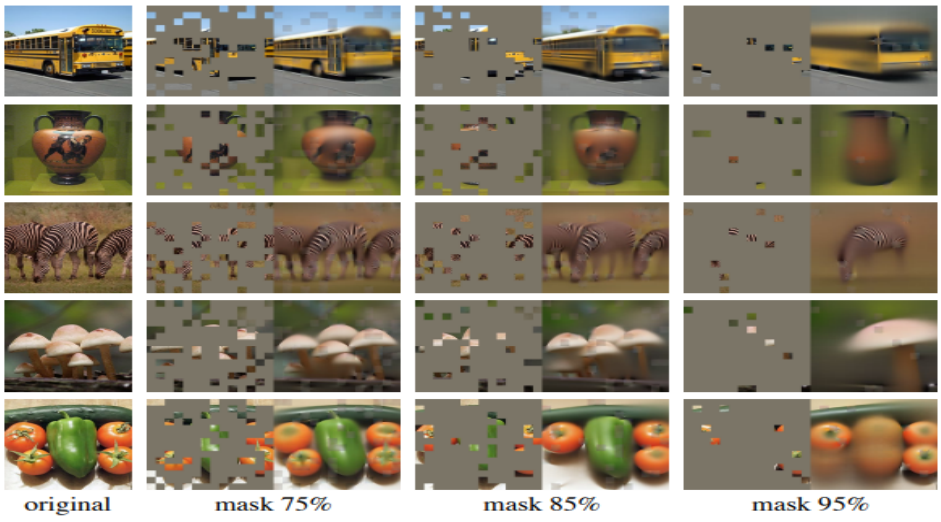


Figure 2.7: Reconstructions of ImageNet validation images using a MAE pre-trained with a masking ratio of 75% but applied on inputs with higher masking ratios. The predictions differ plausibly from the original images, showing that the method can generalize.

2.3.1 Hyperparameters

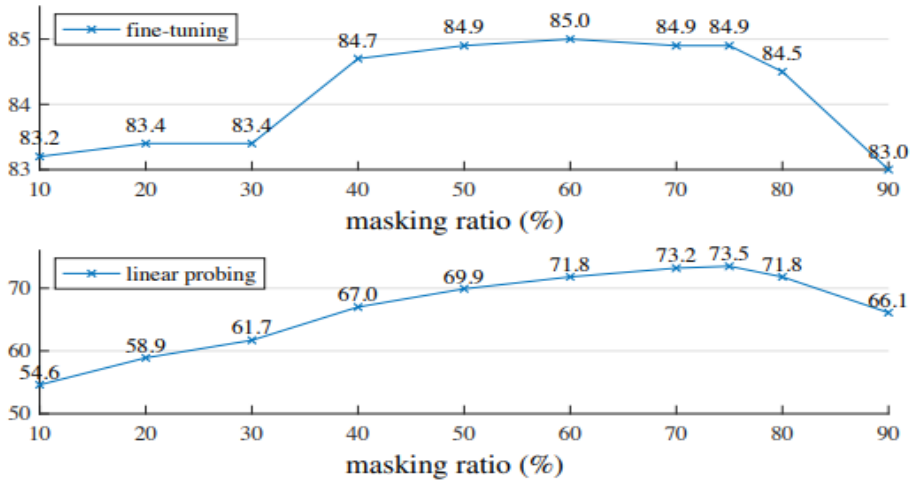


Figure 2.8: Masking ratio. A high masking ratio (75%) works well for both fine-tuning (top) and linear probing (bottom). The y-axes are ImageNet-1K validation accuracy (%).

It is important to note that from [7] several interesting properties can be observed:

- Their MAE pre-training approach is implemented efficiently and, in particular, does not require any sparse operations. Initially, they create a token for each input patch by implementing a linear projection in conjunction with an added positional embedding. Next, they *randomly shuffle* the list of tokens and *remove* a portion of the list based on the chosen masking ratio. This process results in a smaller subset of tokens, which serves as input for the encoder and effectively correspond to sampling patches without replacement. After encoding, they add a list of mask tokens to the list of encoded patches and then *unshuffle* the full list (reversing the random shuffle operation) to align all tokens with their respective targets. The decoder operates on this full list with positional embeddings included. This methodology introduces minimal overhead, as the shuffling and unshuffling operations are fast and straightforward.

- Figure 2.8 illustrates the impact of the masking ratio on the model's performance. Surprisingly, the optimal ratios are relatively high. For both linear probing and fine-tuning, a ratio of 75% proves to be effective. This is in contrast to BERT [2], which typically uses a much lower masking ratio of around 15%. The model *deduces* missing patches to generate various yet reasonable outputs. (as seen in Figure 2.7). The authors hypothesize that this behavior is strictly connected to the learning of useful representations.

2.4 Heart-rate estimation

Heart rate estimation and monitoring is one of the most important challenges in the field of Deep Learning as it has the dual purpose of tracking fitness-related metrics and monitoring cardiovascular well-being. Heart rate refers to the number of times the heart beats per minute (bpm), and it is an essential physiological measurement that can provide insights into cardiovascular health and overall well-being of people. In practice, the heart rate is an indicator of how effectively the heart is pumping blood throughout the body. For instance, it is customary for the majority of adults to maintain an HR range of 60 to 100 beats per minute (bpm) [19]. This rate can be influenced by various factors including stress, anxiety, hormonal balance, medications, and physical activity levels. Indeed for example, athletes or individuals engaged in higher levels of activity might have a resting heart rate as low as 40 bpm. In the context of resting heart rate, a lower value is typically advantageous, as it often indicates a better cardiac muscle condition and decreased effort to maintain a constant heart rate level. Furthermore, research underscores a direct correlation between elevated resting heart rate, diminished physical fitness, and increased blood pressure and body weight. The process of HR estimation commonly involves the computation of the temporal gap between two consecutive heartbeats. These heart rate measurements are derived from various types of biological signals, such as those obtained from electrocardiograms or photoplethysmographic signals. However, the presence of noise can obscure the authentic heart rate signal. As a result, the accuracy of the heart rate estimation depend on the quality of the underlying waveform [20], which is intricately linked to the degree of noise present in the signal. Initially, electrocardiograms predominantly served as the primary

source for obtaining heart rate data. This approach entails the positioning of electrodes on specific areas of the body (such as the chest, wrists, arms, and legs) to record electrical impulses and subsequently calculate heartbeats. This approach proves to be rapid, uncomplicated, and efficacious; however, it simultaneously entails higher costs and necessitates direct skin contact, rendering it non-portable and uncomfortable (as the wearing of an ECG chest band could hinder an individual's daily activities).

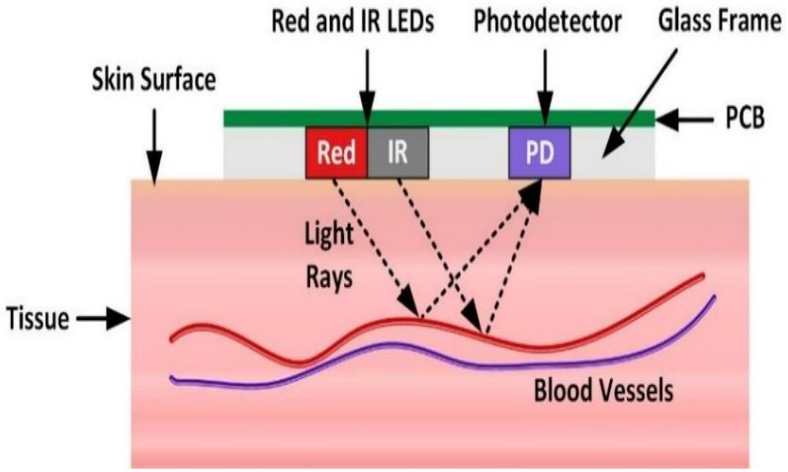


Figure 2.9: The principle behind a PPG sensor. The pulse signal obtained from a PPG sensor comprises an AC (pulsatile) and a DC (slowly varying) component. The AC component is attributed to changes in the blood volume synchronous with each heartbeat, whereas the DC component is related to respiration, tissues, and average blood volume. The two most common LEDs are red and infrared (IR), which exhibit distinct absorption properties in the bloodstream. The photodetector captures light and it is used to estimate blood volume changes. [22]

To surmount this challenge, recent years have witnessed an increasingly widespread use of PhotoPlethysmoGraphic (PPG) signals for heart rate computation in individuals utilizing smartwatches. In contrast to ECG, the acquisition of PPG signals is non-intrusive and more comfortable. Specifically, PPG leverages LED lights in conjunction with a photo-detector as a receiver to discern volumetric variations within blood streams [21]. Figure 2.9 illustrates the operational logic underlying a PPG sensor's functionality. The rhythm of the light pulses corresponds to the heart rate.

2.4.1 PPG signal

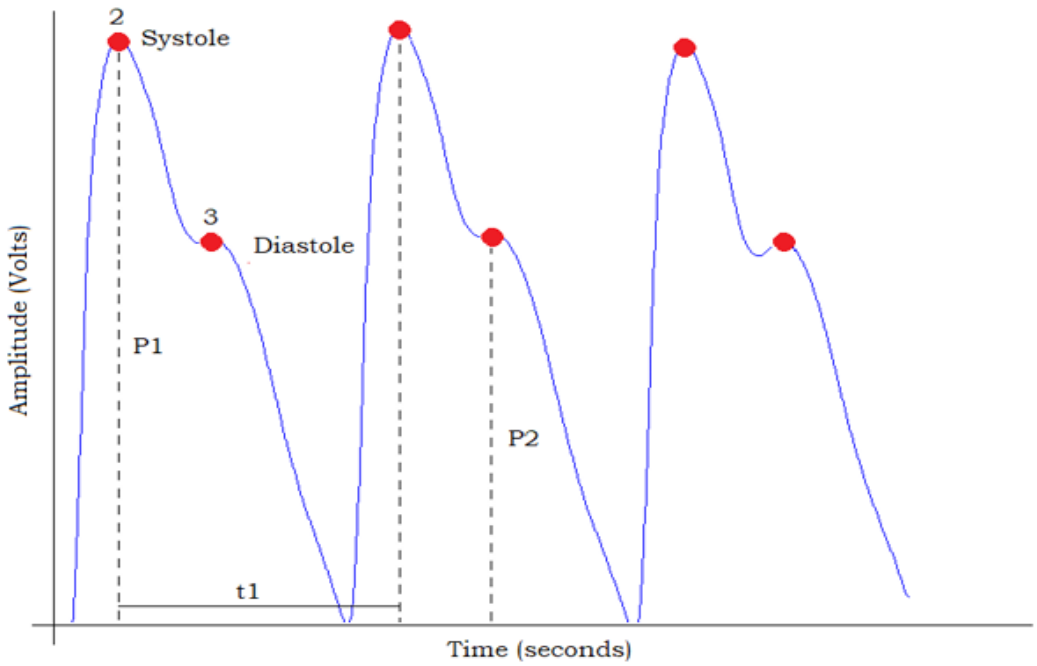


Figure 2.10: PPG waves.

A PPG (Photoplethysmogram) signal represents variations in blood volume in the blood vessels of a tissue. It is commonly employed to monitor and record physiological data related to the cardiovascular system. PPG signals are typically obtained from a small sensor placed on the skin's surface and provide valuable information about a person's heart rate and blood flow. This signal can be divided into two main components:

- **Systolic Peak**: is the highest point in the PPG signal and is associated with the contraction of the heart, known as systole. During systole, the heart contracts and pumps a greater amount of blood into the arteries. This increase in blood volume in the peripheral blood vessels is detected by the PPG sensor as a sudden increase in the signal's amplitude. The systolic peak represents the moment when blood pressure is highest in the arteries.

- **Diastolic Peak**: is the lowest point in the PPG signal and is associated with the relaxation phase of the heart, known as diastole. During diastole, the heart relaxes, and blood flow in the arteries decreases. This is reflected in the PPG signal as a reduction in the signal's amplitude. The diastolic peak represents the moment when blood pressure is lowest in the arteries.

The PPG signal is typically represented as a cyclic waveform in which these systolic and diastolic peaks repeat in synchrony with the cardiac cycle. The time interval between one systolic peak and the next systolic peak corresponds to the interval between two consecutive heartbeats, known as the RR interval and identified with *t1* in Figure 2.10. This interval can be used to calculate heart rate.

2.4.2 Motion Artifacts

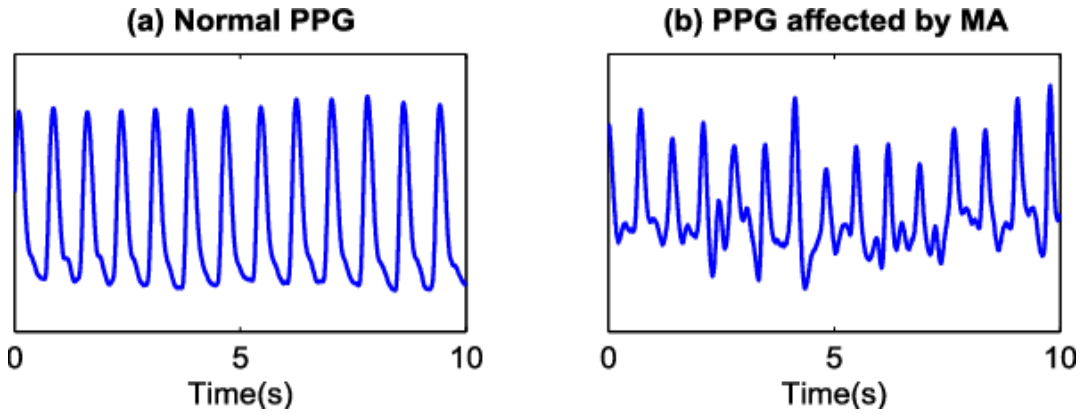


Figure 2.11: Photoplethysmography signals. (a): PPG signals in normal conditions. (b): PPG signals influenced by MAs during movement.

The existence of distinct motion artifacts, characterized as signal distortions, frequently encountered in waveform signals acquired via wrist-worn devices, can compromise the signal's accuracy [23]. These artifacts occur when the PPG signal is distorted or contaminated due to body movement. Any movement of the body part being monitored (e.g., the wrist, finger, or earlobe) can introduce motion artifacts. This includes actions like walking, talking, or even subtle muscle contractions. Moreover, if the PPG sensor is not securely attached to the skin or if it moves during measurement, it can result

in artifacts. Motion artifacts can lead to irregularities in the PPG waveform, including fluctuations in the signal amplitude and shape. These irregularities can make it difficult to accurately detect the systolic and diastolic peaks and measure physiological parameters such as heart rate and oxygen saturation and, in severe cases, can render the PPG signal unusable for clinical or research purposes. Figure 2.11 illustrates a visual example of the difference between clean and dirty PPG signal. Specifically, as depicted in Figure 2.11 (b), it is evident that identifying peaks accurately becomes more challenging from PPG affected by MA, as the signal undergoes changes in shape and amplitude. This can potentially result in erroneous heart rate estimations.

Chapter 3

Related work

Research in recent years has witnessed considerable interest from both academic and industrial sectors in the examination of HR monitoring solutions that rely on wearable devices incorporating PPG sensors. While monitoring people in steady positions (e.g., while sleeping or watching the TV) is a relatively straightforward and resolved task, the introduction of movement often hinders the effectiveness of this process, introducing interference into the PPG signal. Consequently, the principal hurdle lies in maintaining a consistently high level of accuracy, even during activities involving vigorous movements. In contemporary practice, the methodology to mitigate these distortions involves the application of sensor fusion techniques combining PPG signals with accelerometer data. The algorithms designed to address such data can be categorized into two primary groups. Firstly, there are *classical approaches*, based on time or frequency related features, as well as filtering and peak detection functions. The second category relies on *deep learning approches*.

3.1 Classical approches

An exemplary case is illustrated by **TROIKA** [24], a framework including three distinct components that lend their names to the system: decomposiTion, sparse signal Re-cOnstructIon, and spectral peaK trAcking. The first

module partially mitigates the impact of motion artifacts (MA) on PPG data, dispersing its spectrum within the [0.4 - 5] Hz range. The second module bolsters TROIKA's resilience against noise interference by computing a high-resolution spectrum of the PPG signal. Finally, the spectral peak tracking module, a fondamental aspect of the framework, try to find peaks corresponding to heart rate (HR) values. For validation, the authors gathered data samples from 12 subjects and demonstrated that TROIKA reached notably improved outcomes, with a standard deviation error of 3.07 beats per minute (BPM), in contrast to previous jobs. Other studies such as Independent Component Analysis (ICA) [26] and Kalman filtering [27], explored only scenarios with minimal motion artifacts. To address this limitation and reach a satisfying accuracy in case of high MAs, researchers from [25] introduce a novel approach called JOint Sparse Spectrum reconstruction (**JOSS**). This approach is inspired on the notion that PPG and acceleration signals may share common spectral structure characteristics. JOSS employs a model known as the multiple measurement vector (MMV) to jointly estimate spectrum (a less powerful version applied to single spectrum was also presented in TROIKA). This MMV model identifies spectral peaks linked to motion artifacts within PPG spectrum by leveraging spectral peaks extracted from acceleration spectrum. The method focuses around a common sparsity constraint, promoting alignment between the frequency locations of motion artifacts within PPG spectrum and corresponding locations in acceleration spectrum. The author assessed the algorithm across 12 distinct patients, each comprising PPG signals, accelerometer data, and an ECG data channel. As an initial data preprocessing step, the raw data was subjected to bandpass filtration, covering the frequency range from 0.4 to 4 Hz. The paper [25] provided a direct comparison with the results obtained from TROIKA using the same dataset, demonstrating improvements in the Mean Absolute Error (MAE) values. To be specific, JOSS achieved an MAE of 1.28 beats per minute (BPM), in contrast to TROIKA's 2.42 BPM across all datasets.
Following JOSS, several other algorithms have been developed with the objective of estimating heart rate from PPG signals that are affected by motion artifacts. In this section, I'll mention the two most recent algorithms. First, the approach introduced by Salehizadeh et al [55] called **SpaMa**. This method initially calculates the power spectral density of both the PPG and accelerometer signals within each sample (8-second segment). Subsequently, it identifies the highest peaks in each spectrum. The logic behind this approach is that the peaks in the acceleration spectrum correspond to motion. Therefore, by removing these motion-related peaks from the PPG spectrum,

it effectively eliminates major motion artifacts. The highest peak remaining in the PPG spectrum is then taken as the heart rate. Recognizing that rapid changes in heart rate are physiologically constrained, it is advisable to rely on estimates from preceding segments. However, the SpaMa approach considers only the last segment in its heart rate tracking step. To enhance estimation robustness, **SpaMaPlus** extends the heart rate tracking step by incorporating a mean filter over the last *six* heart rate estimates (with the value six determined empirically). It then identifies the peak in the current segment's PPG spectrum closest to the heart rate frequency predicted by the mean filter. Another issue addressed by SpaMaPlus is the potential propagation of errors in heart rate estimation. To mitigate this, it resets the heart rate tracking if the difference between the current and previous segment's heart rate estimation surpasses a trainable threshold (e.g., 10 beats per minute) for a set number of consecutive times (e.g., three times). This way, heart rate estimation and tracking can recover every few seconds, even if spectral filtering fails over certain periods.
Second, the approach introduced by Schaeck et al. [56] called **Schaeck2017**. This method, in contrast to SpaMa and SpaMaPlus, supports the utilization of multiple PPG signal channels (e.g., the IEEE datasets include two PPG channels). The algorithm initiates by applying correlation functions, both auto- and cross-correlation, to the time series signal. This step aims to reduce noise. Subsequently, it computes the spectrum of the resulting time series. Similar to the SpaMa approaches, spectral filtering is applied to diminish motion artifacts by considering the acceleration spectra. Finally, it utilizes a linear least squares fit on the preceding three segments for heart rate tracking. To compare performance, tested on WESAD, SpaMa reaches a MAE over all the subjects of 11.51, SpaMaPlus of 9.45 and Schaeck2017 of 19.97. Tested on PPG-Dalia, the overall MAE reached are 15.56, 11.06 and 20.45, respectively.

3.2 Deep Learning-based approaches

Deep Learning (DL) is relatively less explored for this task. Only in recent years researchers are embarking on an exploration of Deep Learning (DL) techniques for this task, identifying a series of problems and challenges that

necessitate a solution. The primary problem is memory since highly accurate DL models typically involve millions of parameters, which exceed the memory available in most Microcontrollers (MCUs) commonly found in wrist-worn devices. Second, DL solutions necessitate large amounts of labeled data to be trained efficiently and to obtain good performances, which were not available for this task, since signal annotation is a laborious and time consuming procedure. About this last point, things improved recently with the introduction of PPG-Dalia [13], a very large dataset for heart rate tracking using PPG in presence of MAs, which includes recordings from 15 subjects performing different daily activities. When the publication of this dataset was available, various researchers have tried to delve into this new task. The authors of [13] also introduced the first DL solution based on Convolutional Neural Networks (CNNs), which was shown to outperform state-of the-art algorithms. Following their example, other researchers have then proposed different DL models for this task, such as CorNET [15] and Binary CorNET [16], which combine convolutional and recurrent layers. CorNET was evaluated on the TROIKA and JOSS dataset having 22 PPG records collected during various physical activities and achieving a mean absolute error of 1.47 BPM for HR estimation. BinaryCorNET, instead, achieves a MAE of 6.67 BPM when evaluated on the mentioned 22 PPG records. Lastly, Burello et al. focus in particular on Temporal Convolutional Networks(TCNs). With their proposed methodology [14] on PPGDalia they achieves a Mean Absolute Error (MAE) of 4.36 BPM (Beats Per Minute), and with an additional fine-tuning step, the MAE is further reduced to 3.61 BPM.

3.2.1 ActPPG

Risso et al. [30] introduced a collection of *Temporal Convolutional Networks* (TCNs) for the estimation of heart rate (HR) using raw photoplethysmogram (PPG) signals and acceleration data. TCNs represent a form of 1D Convolutional Neural Networks (1D-CNN) that incorporate *causality* and *dilation* parameters within their convolutional layers. The inclusion of causality ensures that the output of the layer $\mathbf{y_t}$ solely relies on inputs $\mathbf{x_{\tilde{t}}}$ with $\tilde{t} \leq t$. Dilation, on the other hand, involves introducing a gap (d) between the input samples processed by the convolution. This gap increases the receptive field without necessitating the addition of new parameters. Thus, a convolutional layer in a TCN can be expressed through the following function:

$$y_t^m = \sum_{i=0}^{K-1} \sum_{t=0}^{C_{in}-1} x_{t-di}^l \cdot \mathbf{W}_i^{l,m} \tag{3.1}$$

Where x and y represent the input and feature maps respectively,t corresponds to the output time-step, and m denotes the output channel. $\mathbf{W}$ stands for the filter weights, $C_{i,n}$ is the number of input channels, d represents the dilation factor, and K denotes the filter size. Within their paper, the authors propose an adaptation of a well-known TCN named TEMPONet, originally designed for gesture recognition. Their customized TEMPONet receives raw data from a PPG sensor and the raw input from a three-axis accelerometer sensor as inputs. Furthermore, the final classification layer is substituted with a single neuron for regression purposes. The training loss employed is *LogCosh*. To optimize the network's size and complexity, enabling its deployment in embedded systems, the authors employ MorphNet, an algorithm for Neural Architecture Search (NAS). This algorithm automatically prunes channels within each layer, significantly reducing the occupation of the seed network, TEMPONet in this instance. The authors also apply full-integer post-training quantization, converting the outputs from *float32* to *int8*.
In [31], the work presented in [30] is extended to introduce two primary contributions: *TimePPG* and *ActPPG*. TimePPG comprises a series of TCN architectures designed to predict heart rate by utilizing raw PPG values and tri-axial accelerometer data as inputs. The diverse architectures are generated through the MorphNet NAS algorithm [32], using TEMPONet as the seed TCN. MorphNet takes the training dataset and the original TEMPONet as inputs, initiating the architecture exploration process. This initial TCN optimization produces multiple refined models, each reaching a good balance between heart rate predictions and model complexity. This set of optimized models collectively forms what the authors refer to as *TimePPG*. In addition, a smoothing post-processing step is introduced to further enhance accuracy. This process entails the application of a threshold (P_{th}) to the maximum variation in predicted heart rate concerning the averaged heart rate estimated during the preceding N stages, in order to limit eventual incorrect predictions of the model and is motivated by the fact that during heart rate (HR) tracking, a subset of these errors can be readily filtered out by taking into account the human physiological parameters. The results indicate that the largest model, referred to as TimePPG-Big, produces a Mean Absolute Error (MAE) of 4.88 BPM with approximately 232k parameters. Even the smallest mode, in normal conditions, comprising only 5.09k parameters, achieves

promising results with an MAE of 5.63 BPM.

The second significant contribution presented in [31] is a framework called **ActPPG**. This framework strategically integrates various TimePPG models based on the quality of the PPG waveform, hence on the high presence of Motion Artifacts (MAs), leveraging the accelerometer-derived movement data. ActPPG is structured around two core modules:

- Movement Detector: This component employs a lightweight Random Forest model, comprising 8 trees. It is fed with accelerometer data to categorize movement levels on a scale of $[0, N-1]$, with N being set to 2. Although accelerometer values do not inherently offer insights into heart rate estimation, they have recently emerged as the state-of-the-art signal for mitigating MAs in PPG values arising from device movement, especially in wrist-worn devices.

- Predictors: These are the models responsible for actual heart rate estimation. This module receives inputs including the previously calculated window difficulty (w_d), training data, and optionally accelerometer data. Based on w_d, a specific predictor is selected, with higher w_d values favoring more performant predictors. It's important to note that the predictor ordering is established offline.

The framework is based on two assumptions. Firstly, as the occurrence of movements increases, MAs escalate, consequently reducing the accuracy of heart rate predictions. Secondly, the disparity between larger and smaller models primarily depend on their capacity to correctly predict the HR even in presence of MAs.
Drawing inspiration from big-little neural networks, *ActPPG* enhances both small and large models by deploying the more suitable model according to the given context. Despite the notable improvements in Mean Absolute Error (MAE) values, the substantial reduction in complexity, size, and subsequent energy consumption is a pivotal outcome.

3.2.2 Q-PPG

Following the methodology introduced in [30], the authors extended their research to further reduce the complexity of models used for estimating heart rate (HR) based on PPG in [33]. Their extended work encompasses three main contributions:

- The optimization scope of Neural Architecture Search (NAS) is broadened to encompass *dilation* parameters for convolutional layers. This expansion serves to further reduce model complexity with a marginal compromise in accuracy.
- To trim down the model size, enhancing the Pareto frontier, hardware-friendly *quantization* is introduced.
- The researchers deployed their results on a real embedded smartwatch device powered by an STM32WB55 MCU from ST Microelectronics.

As with previous research efforts, TEMPONet serves as the seed network for the NAS. The outcome of this process yields a collection of quantized Temporal Convolutional Networks (TCNs), thus giving rise to the methodology's name: *Quantized*-PPG (Q-PPG). More precisely, the method employed is linear quantization, which transforms the floating-point tensor from the range $[\alpha_t, \beta_t]$ into N-bit integer tensor $\hat{t}$, as expressed by the following equation:

$$\hat{t} = round\Big(\frac{t - \alpha_t}{\epsilon_t}\Big) \tag{3.2}$$

where $\epsilon_t = (\beta_t - \alpha_t)/(2^N - 1)$ is the smallest value the quantizer tensor can assume.
In [33], the authors initiate their process by implementing *uniform quantization*. Subsequently, quantization-aware training is executed across various formats, including `int2`, `int4`, and `int8`. This iterative procedure is followed by the identification of the optimal data format for each layer. In the final stage, a post-processing step is introduced to mitigate the inherent and unpredictable errors associated with data-driven models like TCNs. This

post-processing phase involves the application of a filter based on the natural dynamics of the heart, which establishes a reasonable range for heart rate estimations over time.
The quantization optimization results in networks spanning a size spectrum, ranging from the largest at 1MB (in floating-point representation) to the smallest at less than 1kB. The most extensive model that can accommodate the target embedded device, STM32WB55, requires approximately $\approx$ 412kB, yielding a Mean Absolute Error (MAE) of 4.41 beats per minute (BPM).

Chapter 4

Methods

This section constitutes the core of this thesis: here, I'll describe how the extraction of the heart rate values from ECG-signals commonly occurs. These values will be our targets (ground-truth) during the fine-tuning phase where a regression task is performed. As often happens during the training of DNNs, a pre-processing step is applied to normalize input values, reducing the risk of possible outliers. For this reason, before starting the effective training loop, raw data (PPG signal and acceleration data) are first preprocessed with a *sliding window* of 8 seconds and *shift* of 2 seconds, and the label is calculated accordingly, as the average of the signal peaks in each time window. This is a commonly used methodology in heart rate tracking via PPG signals. Furthermore, as we concentrate on conducting two distinct experiments, we must take into account both the time and frequency domains. Consequently, when working in the frequency domain, the Fast Fourier Transform (FFT) is employed to extract *spectrograms* from the raw signal. During the fine-tuning step we also apply a *post-processing* at runtime to prevent eventual errors in the predictions of the model, mainly linked to data distributions different from those processed during the training phase. In the last part of this chapter, I'll discuss the advantages of using *transfer learning* and how it is exploited in the specific case of heart rate estimation through different datasets, to further reduce patients' *Mean Absolute Error* (MAE).

4.1 Label computing

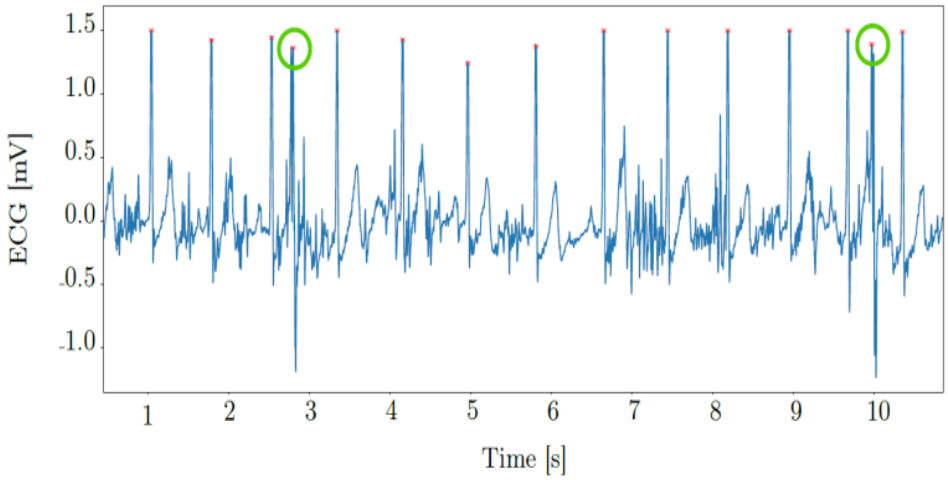


Figure 4.1: Example ECG-signal snippet from the data recording of subject S1. The two encircled R-peaks were falsely identified by the R-peak detector, and were thus manually removed during heart rate ground truth generation.

Reliable ground truth information can be derived from the ECG signal. The procedure for heart rate extraction is quite simple: initially, an R-peak detector [39] was employed. Subsequently, the identified R-peaks were manually reviewed and adjusted when necessary. Such adjustments were needed in a few instances for each subject due to strong motion artifacts in the ECG signal. R-peak correction encompassed both the elimination of erroneous peaks and the retrieval of R-peaks that were overlooked in the initial stage. Shown in Figure 4.1 there is an illustrative segment of the ECG signal from participant S1 of PPG-Dalia extracted from [13], where motion artifacts induced during the table soccer activity resulted in the erroneous identification of R-peaks. Based on these corrected R-peaks, the instantaneous heart rate was computed (in BPM) through the following equation:

$$instant_{HR} = 60 \times (\frac{f_s}{\delta_t}) \tag{4.1}$$

where f_s is the sampling rate and δ_t is the time difference between two consecutive corrected peaks.
Ultimately, the ECG signal was segmented using a shifted window approach (window length: 8 seconds, window shift: 2 seconds). Ground truth heart

rate was consequently defined as the average instantaneous heart rate within each 8-second window. Thus, the first value gives the calcualted heart rate ground-truth in the first 8 seconds, while the second value in gives the calculated heart rate ground-truth from the 3rd second to the 10th second and so on. Utilizing a sliding window of 8 seconds with a 2-second shift is a common practice within the literature of PPG-based heart rate estimation. [37], [40].

4.2 Pre-processing

The sliding window described in Section 4.1 can be applied directly in time to the raw PPG signals or in frequency to the corresponding spectrograms. If we work in the time domain, PPG signals are divided into time windows for analyzing heart rate in specific intervals and these windows are passed directly to the algorithm described in Section 4.1. This means each window will be 8 seconds in duration and the next window will start 2 seconds after the end of the previous one. Otherwise, if we work in the frequency domain, PPG signals are always divided into time windows but, this time, we apply an additional step to extract from each window the corresponding spectrum through the **Fast Fourier Transform** (FTT). This process converts the signals from the time domain to the frequency domain, showing how energy in different frequencies varies over time. Heart rate can also be calculated from spectrogram data by looking for peaks in the frequencies associated with the heart rate, often in the range of heart frequencies (e.g., between 0 and 4 Hz). The dominant frequency in each time window will represent the estimated heart rate for that window.
Furthermore, researches have indicated that humans don't perceive frequencies on a linear scale. Our sensitivity to distinctions in lower frequencies surpasses that in higher frequencies. For instance, we can effortlessly distinguish between 500 and 1000 Hz, but perform a distinction between 10,000 and 10,500 Hz proves challenging, despite the interval range is equal. Back in 1937, Stevens, Volkmann, and Newmann introduced a pitch unit that aimed to make equal pitch intervals sound equally perceptible to the listener. This unit is known as the **mel scale**. We perform the following mathematical operation to convert frequencies to the mel scale:

$$mel(f) = \begin{cases} f, & \text{if } f \leq 1\text{kHz} \\ 2595 \times \log(1 + \frac{f}{700}), & \text{if } f > 1\text{kHz} \end{cases} \tag{4.2}$$

4.2.1 Spectrogram extraction

From the PPG signal is possible to retrive the corrispondent spectogram thought the **Fourier Transform** (FT).

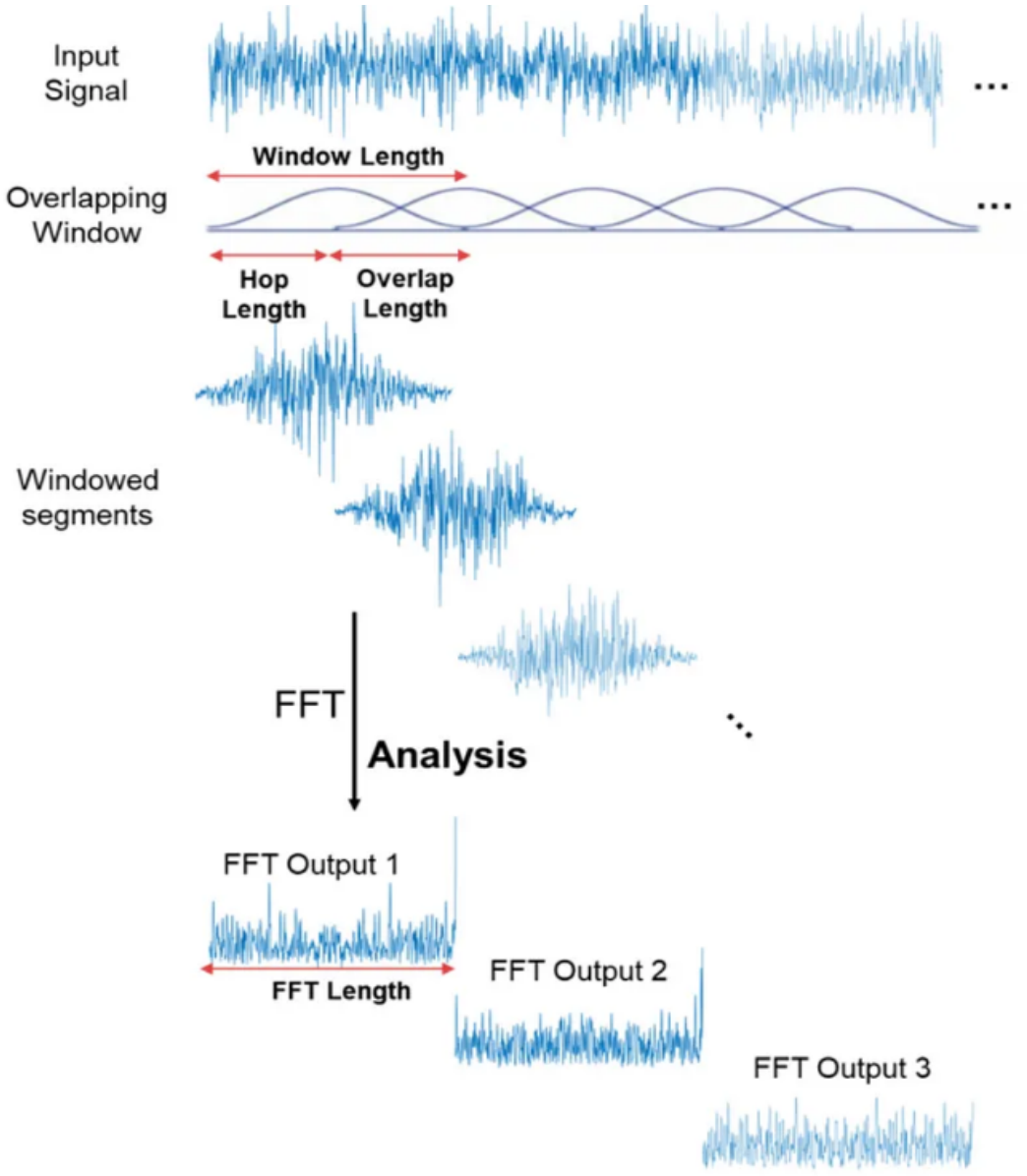


Figure 4.2: Spectogram from a raw signal using FFT on overlapping windowed segments.

The Fourier Transform is a mathematical formula that gives us the possibility to decompose a signal into its constituent frequencies and their corresponding amplitudes (see Figure 4.3), in this way, it transforms the signal from its time-domain representation into the frequency-domain representation, the so called **spectrum**. The Fast Fourier Transform (FFT), instead, is an algorithm well-suited for efficiently computing the Fourier transform and it is widely used in signal processing. It is a technique that allows to analyze the frequency content of a signal, but in numerous applications, it can prove valuable to comprehend how the frequency components of our signal change across time. This holds true for various audio signals, including music and speech. The FFT is computed on overlapping windowed segments of the raw signal and in this way we get what is called the **spectrogram**. Figure 4.2 can help to understand the process with a visual example which now which I'll describe in greater detail with a code example.

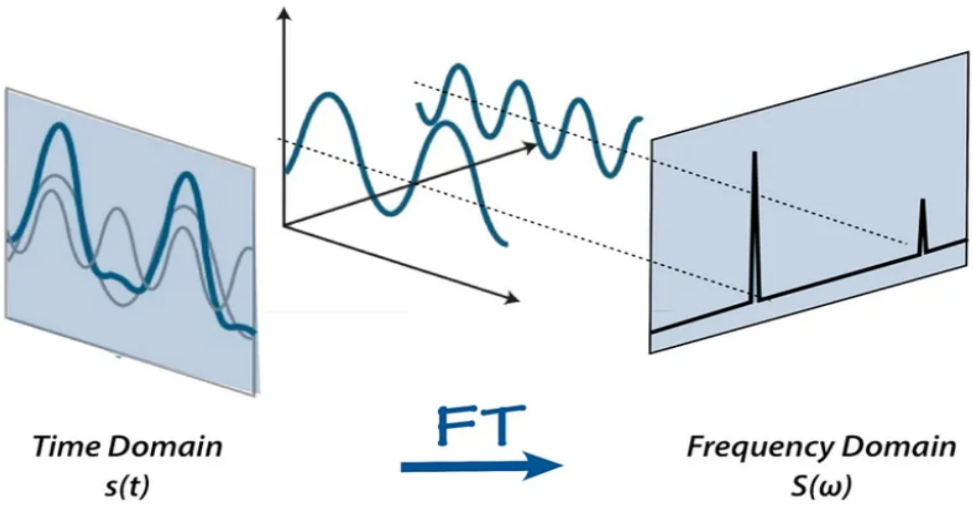


Figure 4.3: Fourier Transform from a raw signal.

There are several transformations that allow to compute the FTT from the raw signals. In this thesis, I investigated **MelSpectogram** from the TorchAudio library. After careful analysis, the parameter configurations were set in the following way:

```
#spectogram trasformation and relative parameters

spectrogram_transform = torchaudio.transforms.MelSpectrogram(
    sample_rate = 32,
    n_fft=510
    win_length=32
    hop_length=1
    center=True,
    pad_mode="reflect",
    power=2.0,
    normalized=True,
    f_min = 0,
    f_max = 4,
    n_mels =  64
)

```

Listing 1: MelSpectogram transformation.

- **sample_rate**: this parameter specifies the audio sample rate in Hz (samples per second). It indicates how many times per second the audio signal is sampled. Working with PPG signals sampled at 32 Hz, I set the parameter accordingly.
- **n_fft**: it represents the size of the analysis window used to calculate the spectrogram of an audio signal (*FTT Length* in Figure 4.2). Setting the parameter to 510 means that the analysis window will have a size of 510 audio samples. This, indeed, is the number of samples that will be considered in each analysis window. A larger *n_fft* value can capture finer temporal details but requires more computational power.
- **win_length**: this parameter defines the length of the analysis window in time-domain samples (*Window Length* in Figure 4.2). This window is used to divide the audio signal into small temporal segments, called frames, on which the spectrogram will be computed. In other words, *win_length* specifies how many audio samples are included in each analysis frame.
- **hop_length**: it represent the amount of overlap between successive windows in samples (*Overlap Length* in Figure 4.2). A smaller value

captures finer temporal details but requires more computation.

- **center**: this flag indicates whether the analysis window should be centered with respect to the audio frame. Setting this parameter to *True* positions the window at the center of each frame, while *False* positions it at the beginning of each frame.
- **pad_mode**: if *center=True*, this argument is passed to *np.pad* for padding the edges of the signal. By default *pad_mode = "reflect"* is set. This means that signal is padded on both sides with its own reflection, mirrored around its first and last sample respectively.
- **power**: this parameter specifies the power to which the magnitude of the Mel spectrogram should be raised. It is often set to 2 to compute the power spectrogram.
- **normalized**: it indicates whether the Mel spectrogram should be normalized. When set to True, the spectrogram values are normalized to a range between 0 and 1.
- **f_min**: it represents the minimum frequency (in Hertz) that we want to include in the Mel spectrogram. The Mel filters will start capturing spectral energy from this frequency.
- **f_max**: it represents the maximum frequency (in Hertz) that you want to include in the Mel spectrogram. The Mel filters will stop at this frequency and will not capture spectral energy beyond this point.
- **n_mels**: it controls the number of Mel filters used during the computation of the Mel spectrogram. This parameter is essential in the process because it determines the amount of spectral information that will be extracted from the audio representation. Setting *n_mels* to a higher value, you'll have more Mel filters at our disposal. This means that the extraction of spectral details will be finer, and we'll have a more detailed representation of the spectral components of the signal. Otherwise, setting *n_mels* to a lower value, we'll have fewer Mel filters. This results in a coarser spectral representation, where spectral components are grouped into a smaller number of frequency bands. In general, it's common to use values between 20 and 128.

In the particular context of this thesis, I set **n_fft** to 510 and **win_length** to 32. It means we are using a very short analysis window (32 samples)

compared to the total size of the FTT length (510 samples). From these parameter configurations is important to highlight some considerations:

1. **Zero Padding**: To adapt the analysis window (*win_length*) to the size of *n_fft*, zero padding is required. In other words, the 32-sample analysis frame is extended with zeros to reach a total length of 510 samples.

2. **Temporal Resolution**: Using such a short analysis window produce a very high temporal resolution. That is, our Mel spectrogram will capture very small temporal changes in the audio signal.

3. **Spectral Resolution**: The spectral resolution, i.e., the ability to distinguish frequencies in the signal, will still be determined by the size of *n_fft*, which is 510 samples. This means that despite the short analysis window, we will still have good spectral resolution.

4.2.2 Patchifying and Masking strategies

The process of patchifying divides input into smaller parts called "patches", allowing for more flexible data handling and more efficient data processing. A "patch" represents a small portion of the image associated to the 1D raw PPG signal or the corrisponging 2D spectrogram. The number of patches obtained from splitting the original image in small portions is given by:

$$N = \frac{H \times W}{P^2} \tag{4.3}$$

where H is the length and W is the width of the original image, P is the size of each square patch. This partitioning process can be done sequentially, moving from left to right and top to bottom, or with overlap, where patches can have overlap with adjacent ones. In this thesis we don't focus on the latter. Indeed, following [35] and [36], we transform PPG signals into Melspectrograms (in frequency experiments) and divide them into non-overlapped regular grid patches. These patches are then flattened and embedded by a linear projection. In the end, a fixed sinusoidal positional embeddings is added to the embedded patches. Positional embeddings are vectors that are added to each patch of the spectrogram. These vectors encode information about the position or relative position of the patches in the spectrogram.

Adding these embeddings is important because in this way we can preserve information about order of patches inside the encoder of our Masked Autoencoder.
Furthermore, Masked Autoencoders remove a significant subset of spectrogram patches. Since a spectrogram can be viewed as a 2D representation of time and frequency components of a sound, it is reasonable to explore treating time and frequency differently during masking. In [34] the authors investigate both the unstructured (i.e., applying random masking without considering time or frequency distinctions) and structured (i.e., applying random masking over a portion of time, frequency, or time+frequency) of a spectrogram but, in this thesis, we focus only on the structured one. Illustrative examples are shown in Figure 4.4 in which masked regions are shown with dark overlay.

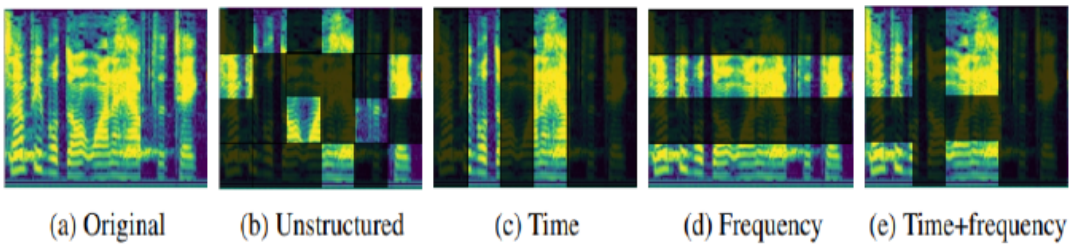


Figure 4.4: MAE's masking strategies on Mel-spectrograms.

4.3 Masked Autoencoders

Our proposed method for the various experiments conducted in this thesis relies on the Masked Autoencoder architecture. It is made up of two main parts: *encoder* and *decoder*. The encoder consists of **Transformer Encoder** blocks, whose details are shown in the Figure 4.5. As shown in Figure 4.5, indeed, it processes the input through attention layers and multi-layer perceptron (MLP). Before going into these blocks the input passes through a *layer normalization* that apply normalization over a mini-batch of inputs [53]. The encoder is responsible for extracting features from the input. For this purpose, and following patchifying explained in Section 4.2.2, the input is divided into patches and, subsequently, these patches are transformed into embedding vectors through a linear projection. Patch embedding are added with positional embedding to keep information about relative position of the

patches inside the encoder. Before going into the encoder, some of these patches are masked (i.e., removed) following the structured strategy as mentioned in Section 4.2.2. The quantity of patches to remove from the image is establishes accordingly to a parameter called **mask_ratio** (Useful examples illustrating the results of signal reconstruction with variations in this parameter are presented in the Section 4.5). The encoder only works with a small subset of the complete patches set. Mask tokens are then restored after the encoder. The full set of encoded masks, and a set of "random-noise" patches (i.e., patches composed by random numbers that fill the space left by the masked patches) are then processed by the decoder. The decoder also consists of Transformer Encoder blocks as those represented in Figure 4.5 and, starting from the output of the encoder, tries to reconstruct the original input from the extracted features.

To achieve a good level of accuracy in the recostruction process of the inputs, a fine-tuning step of the hyperparameters was performed. The hyperparameters which I paid attention to are *depth*, *embed_dim* and *heads*, in order to build Masked Autoencoders (MAEs) of different orders of magnitude and analyze their behavior. In particular, small, medium and large MAEs were created for both time and frequency experiments. In Table 4.1 you can find a summary comparing these architectures to each other as well as to Vision Transformers (ViTs) [6], described in Section 2.2. From Table 4.1 it is clear that Masked Autoencoders perform much fewer operations and have many fewer parameters than ViTs, and this aspect turns out to be an essential advantage in training large quantities of data, significantly reducing waiting times and resources required.

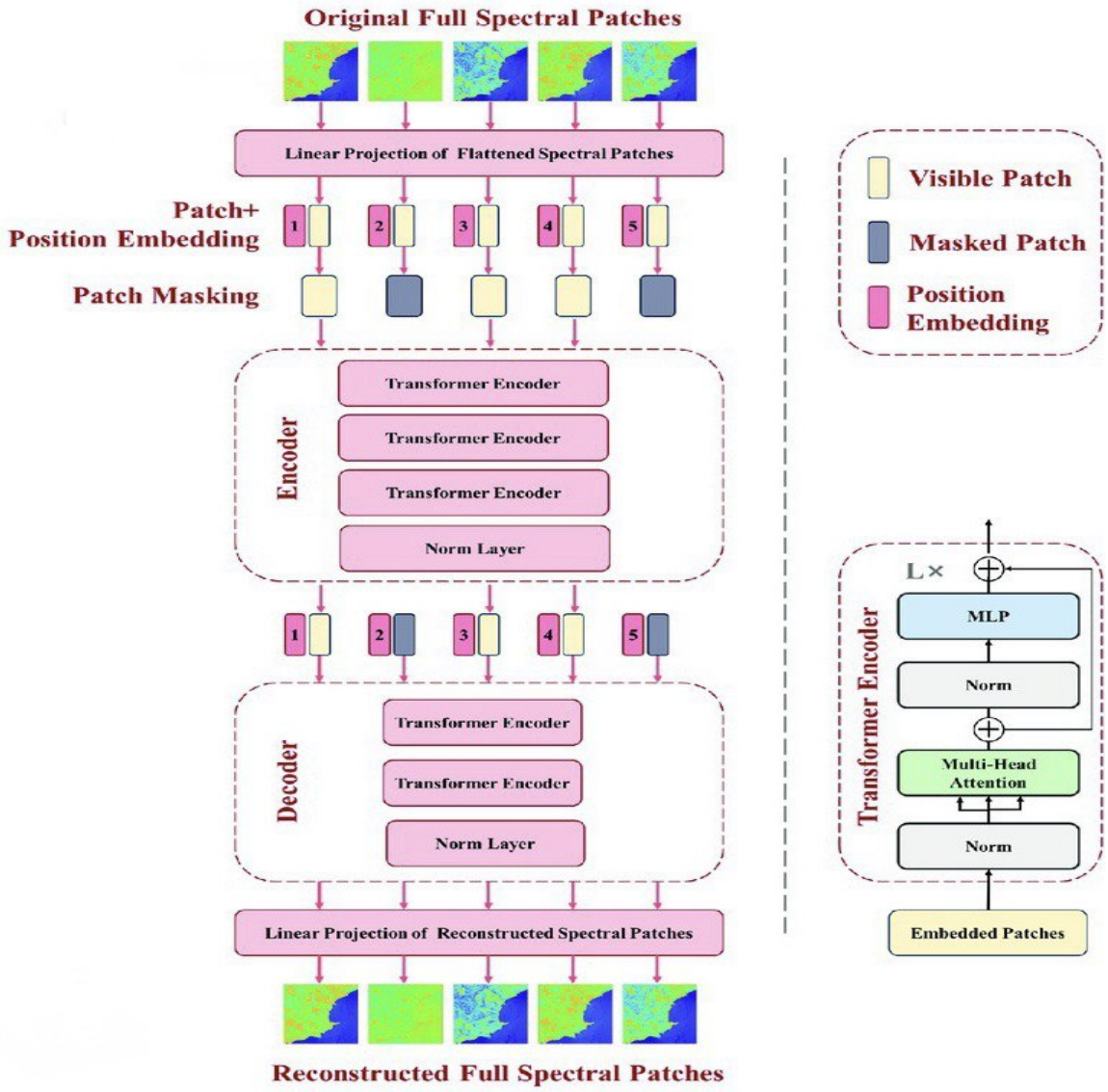


Figure 4.5: Detailed description of the architecture of our Masked Autoencoder for manage PPG signals in frequency domain. Inputs, indeed, are patches extracted from the PPG-spectograms.

Model Name	**Type**	**Depth**	**Heads**	**Embed dim**	**# params**	**# ops**
ViT-Base	-	12	12	768	86M	5G
ViT-Large	-	24	16	1024	307M	7G
ViT-Huge	-	32	16	1280	632M	49G
My MAE - small	Time	4	4	64	364k	53M
My MAE - medium	Time	8	16	64	564k	104M
My MAE - big	Time	12	16	256	9M	2G
My MAE - small	Freq	4	16	64	380k	57M
My MAE - medium	Freq	8	4	128	1M	419M
My MAE - big	Freq	12	16	256	9M	2G

Table 4.1: Comparison between different models.

By looking at Table 4.1, all the MAE configurations were tested and, to achieve better results, as well as those contained in the Results Chapter, in the end the following configurations were chosen:

- **MAE-big** for **time** experiments:
  - *depth = 12*
  - *heads = 16*
  - *embed_dim = 256*
  - *decoder_embed_dim = 256*
  - *decoder_num_heads = 16*
- **MAE-small** for **frequency** experiments:
  - *depth = 4*
  - *heads = 16*
  - *embed_dim = 64*
  - *decoder_embed_dim = 64*

– *decoder_num_heads = 16*

Moreover, is important to underline that the implementation choice of using a MAE-small for the frequency experiments is motivated by the fact that increasing excessively the number of layers the performance of the model decreases, incurring overfitting.

4.3.1 Training Protocol

Following a self-supervised approach, the model is trained in two successive steps: pre-training and fine-tune, which will be discuss in details in their respective Sections 4.5 and 4.6. A simplified pseudo-code of the training protocol is shown below in few lines.

```
for patient in range(patients):
    train_set, val_set, test_set = apply_kfold(patient)
    apply_200epochs_pretraining(train_set)
    apply_200epochs_finetuning(train_set,val_set,test_set)

```

Listing 2: Pseudo-code of the training protocol.

Since both PPG-Dalia and WESAD contain data extracted from a total of 15 different patients, for each of them, the validation step is performed by means of *k-fold cross validation*: train over 11 subjects, validate over 3 and test on the remaining one. Training loop of each patient consists of 200 *pre-training* and 200 *fine-tuning* epochs. These values only represent an upper bound since, to prevent overfitting, training can be stopped when the loss starts to increase or no longer decreases consistently through an early stop on the validation fix to 20 epochs. The pre-training only involves training data because, following an unsupervised learning approach, we don't care about labels in this step. For this reason we don't pass *val_set* and *test_set* to *apply_200epochs_pretraining* function. We test model performance only at the end of the loop during the finetuning phase.

To get a clearer idea of how self-supervised learning is performed, a more realistic code example (Listing 3) regarding the frequency domain is shown

below, in order to explain better the concepts summarized in the pseudo-code and underline the main functionalities.

Some general considerations to understand the code are the following ones:

1. `get_data` returns a tuple of Pytorch Datasets. The number of returned datasets is 3 (train, validation and test). The argument of the function is directly the link to download the entire data zip of *Dalia* or *WESAD*.

2. `build_dataloaders` returns a tuple of Pytorch Dataloaders. Takes as inputs the dataset returned by `get_data` and constants such as the *batch-size* and the *number of workers*. The number of elements of the returned tuple is 3 (train,validation, test) according to `get_data` implementation.

3. `get_reference_model` returns an istance of *Masked Autoencoder* as described in Section 4.3. The number of layers inside the encoder and decoder such as the number of attention head contained in each layer depends on the string passed as input.

4. `get_default_criterion` returns the *Mean Square Error* (MSE) criterion if the task is pretrain otherwise the *LogCosh* criterion if the task is finetune.

5. `get_default_optimizer` returns *AdamW* optimizer if the task is pretrain otherwise the *Adam* optimizer if the task is finetune.

6. `train_one_epoch_masked_autoencoder_freq` implements one epoch of training for to reconstruct the input signal. It takes as input an integer specifying the current *epoch*, the *model* to be trained, the *criterion*, the *optimizer*, the *train* and *val* dataloaders and finally the *device* to be used for the training. It returns a dictionary of tracked metrics.

7. `train_one_epoch_hr_detection_freq` implements one epoch of training and validation for predict heart rate estimation. For the validation part it directly calls the `evaluate_freq` function. It takes as input an integer specifying the current *epoch*, the *model* to be trained, the *criterion*, the *optimizer*, the *train* and *val* dataloaders and finally the *device* to be used for the training. It returns a dictionary of tracked metrics.

8. `evaluate_freq` implement an evaluation step of the model. This step can be both of validation or test depending on the specific dataloader provided as input. It takes as input the *model*, the *criterion*, the *dataloader* and the *device*. It returns a dictionary of tracked metrics.

9. `DATASET_PRETRAIN` and `DATASET_FINETUNING` can only be the strings *DALIA* or *WESAD*. The two variables **must** have the same dataset name, since in this simplified code the transfer learning step is not contemplated.

10. The experiments in the *time* domain are structured in the same way.

The full implementation of PPG Masked Autoencoders described in the various chapters of this thesis is currently private. The work will be made available as soon as possible and published in our official GitHub group: https://github.com/eml-eda.

```
# Set flags for experiments
N_PRETRAIN_EPOCHS = 200
N_FINETUNE_EPOCHS = 200
DATASET_PRETRAIN = "DALIA"
DATASET_FINETUNING = "DALIA"

print(f"=> Running frequency experiment with dataset = {DATASET_PRETRAIN}")

# Check CUDA availability
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("Training on:", device)

# Get data and perform cross-validation
data_gen = hrd.get_data(dataset_name = DATASET_PRETRAIN, augment = False)
for datasets in data_gen:
    train_ds, val_ds, test_ds = datasets
    test_subj = test_ds.test_subj
    dataloaders = hrd.build_dataloaders(datasets)
    train_dl, val_dl, test_dl = dataloaders

    # Get the Model => Masked Autoencoder ViT (encoder + decoder)
    model = utils.get_reference_model('vit_freq_pretrain')
    if torch.cuda.is_available():
      model = model.cuda()

    # Get Training Settings
    criterion = utils.get_default_criterion("pretrain")
    optimizer = utils.get_default_optimizer(model, "pretrain")
    best_loss = sys.float_info.max

    print(f"=> Starting pretrain for {N_PRETRAIN_EPOCHS} epochs...")
    #Pretraining for recostruct input signals
    for epoch in range(N_PRETRAIN_EPOCHS):

      train_stats = hrd.train_one_epoch_masked_autoencoder_freq(
```

```
          model, train_dl, criterion,
          optimizer, device, epoch)

      loss = train_stats['loss']
      if loss < best_loss:
        best_loss = loss
        #Save checkpoint
        checkpoint = {'state_dict': model.state_dict()}
        utils.save_checkpoint_pretrain(checkpoint)

    #Get the Model => Masked Autoencoder (only encoder + CNNs)
    model = utils.get_reference_model('vit_freq_finetune')
    if torch.cuda.is_available():
        model = model.cuda()

    # Get Training Settings
    criterion = utils.get_default_criterion("finetune")
    optimizer = utils.get_default_optimizer(model, "finetune")

    #Load checkpoint from pretrain if exists
    utils.load_checkpoint_pretrain(model,
    ↪  torch.load("./checkpoint_pretrain"))

    print(f"=> Starting finetuning for {N_FINETUNE_EPOCHS}
    ↪  epochs...")
    for epoch in range(N_FINETUNE_EPOCHS):
      train_metrics = hrd.train_one_epoch_hr_detection_freq(
            epoch, model, criterion, optimizer, train_dl, val_dl,
            ↪  device)

      test_metrics = hrd.evaluate_freq(model, criterion, test_dl,
      ↪  device)
    print(f" => Frequency experiment completed")

```

Listing 3: Simplified code example for the training protocol.

4.3.2 Time-based Masked Autoencoder

As described in Section 4.3, for the time experiments we choose a **big-MAE** architecture. The following code example returns an instance of our Time-based Masked Autoencoder that is able to handle raw PPG signals. Each parameter is described below.

```
return MaskedAutoencoderViT_time(
        img_size = 256, in_chans = 4, mask_2d=False,
        patch_size=1, embed_dim=256, depth=12, num_heads=16,
        decoder_embed_dim=256, decoder_num_heads=16,
        mlp_ratio=4, norm_layer=partial(nn.LayerNorm, eps=1e-6) )
```

Listing 4: Time-based Masked Autoencoder.

- **img_size**: size of the raw PPG signal.
- **in_chans**: input channels (PPG signal + 3-axis accelerations).
- **mask_2d**: flag that indicates whether the mask ratio should be applied to both time and frequency or only to one of them.
- **patch_size**: size of the patches.
- **embed_dim**: output size of each layer in the encoder.
- **depth**: number of layers of the encoder.
- **num_heads**: number of attention heads contained in each layer of the encoder.
- **decoder_embed_dim**: output size of each layer in the decoder.
- **decoder_num_heads**: number of attention heads contained in each layer of the decoder.

In particular, we deal with signals which have a shape of *256*. This means each signal is an array containing 256 elements representing time intervals (from 0 to 8 seconds). As mentioned in Section 2.1.2, we split the array into patches of size *(1,1)* in order to reach a number of patches of 256.

Working with a mask ratio of 75% means that 75% of the 256 patches will be removed during the pre-training, therefore, the model will only learn from the remaining 64 visible patches and from these it will predict the remaining 192 hidden ones.

4.3.3 Frequency-based Masked Autoencoder

For the frequency experiments, we exploit the **small-MAE** architecture. The following code example returns an instance of our Frequency-based Masked Autoencoder that is able to handle signals in the frequency domain (spectrograms).

```
return MaskedAutoencoderViT_freq(
        img_size = (64,256), in_chans = 4, mask_2d=True,
        patch_size=8, embed_dim=64, depth=4, num_heads=16,
        decoder_embed_dim=64, decoder_num_heads=16,
        mlp_ratio=4, norm_layer=partial(nn.LayerNorm, eps=1e-6) )
```

Listing 5: Frequency-based Masked Autoencoder.

This time, focusing in the frequency domain, we deal with signals which have a shape of *(64, 256)*. This means each signal is a two-dimensional array with 64 rows and 256 columns, where each element represents a data point in the signal, 64 represent the frequencies (which are bounded between 0 Hz and 4 Hz) after applying the spectrogram transformation and 256 are the time intervals (from 0 to 8 seconds). As mentioned in Section 2.1.2, we split the two-dimensional array into patches of size *(8,8)* in order to reach a number of patches of 256, as for Time-based Masked Autoencoder.

4.4 Post-processing

One other important step is the post-processing that, in our case, is applied directly at runtime on the output of our Masked Autoencoder. This step is independent from what has been described up to now and is motivated by the fact data-driven models like Masked Autoencoders, although generally reaching a high level of accuracy, can occasionally produce significant and unpredictable errors, particularly when the processed inputs radically change from those seen during the training phase. Fortunately, in the specific context of heart rate (HR) tracking, a subset of these errors can be readily filtered out by considering the predictions of the model and human physiological parameters. Hence, during the process of continuous heart rate (HR) tracking, the presence of a prediction that markedly diverges from all preceding estimations is indicative of a potential model error. Based on these insights, the post-processing procedure implements a simple filtering mechanism on the outputs. Specifically, the latest Masked Autoencoder prediction HR_n is compared with the average of the previous N, $E_{n,N} = E[HR_{n-1}, ..., HR_{n-N}]$. If the difference between these two values is larger than a threshold P_{th}, the estimate is clipped to $HR_n = E_{n,N} \pm P_{th}$. We set N to 10 and $P_{th} = E_{n,N}/10$, identical for all patients.

4.5 Pre-training

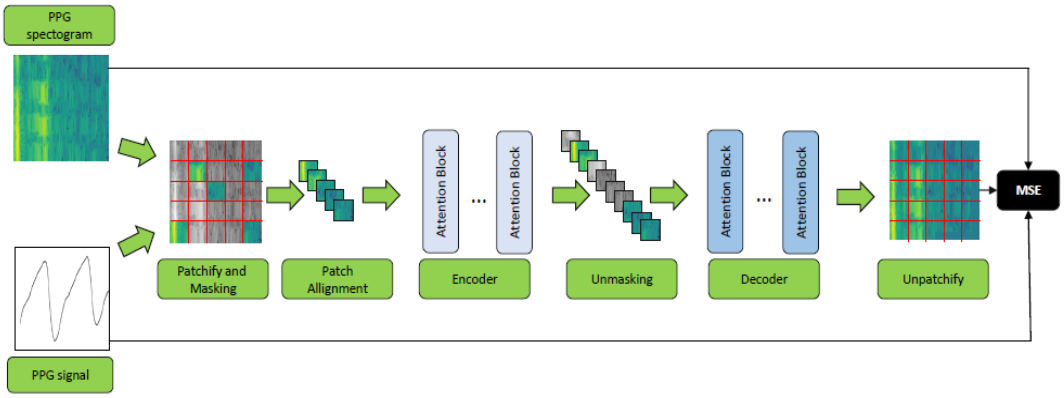


Figure 4.6: Pre-training pipeline.

The pre-training phase is described by Figure 4.6. The pipeline begins with an image that can represents the raw PPG signal or the corresponding spectrogram. The image is divided into patches and multiplied with a mask (composed by 0s and 1s) that randomly removes some of these patches. The encoder only works with a small subset of the complete patches set. Mask tokens are then restored after the encoder, during the unmasking. The full set of encoded masks, and a set of "random-noise" (see Section 4.3) are then processed by the decoder. The goal of the decoder is to reconstruct the original signal starting from the non-masked patches minimizing an appropriate metric. We use the Mean Square Error (MSE) as a reference metric to compute the distance in pixels between targets (input signals) and predictions (reconstructed signals).
Following [34], for the frequency the structured strategy (time+frequency) was chosen using a mask ratio of 15% for time and 15% for frequency. Figure 4.9 shows the reconstruction process. For the time domain, instead, following the information extracted from [7], the mask ratio parameter is first tried with a low value of 15% (Figure 4.7) and then with a high value of 75% (Figure 4.8) to undeline the differences.

4.5.1 Pre-training results.

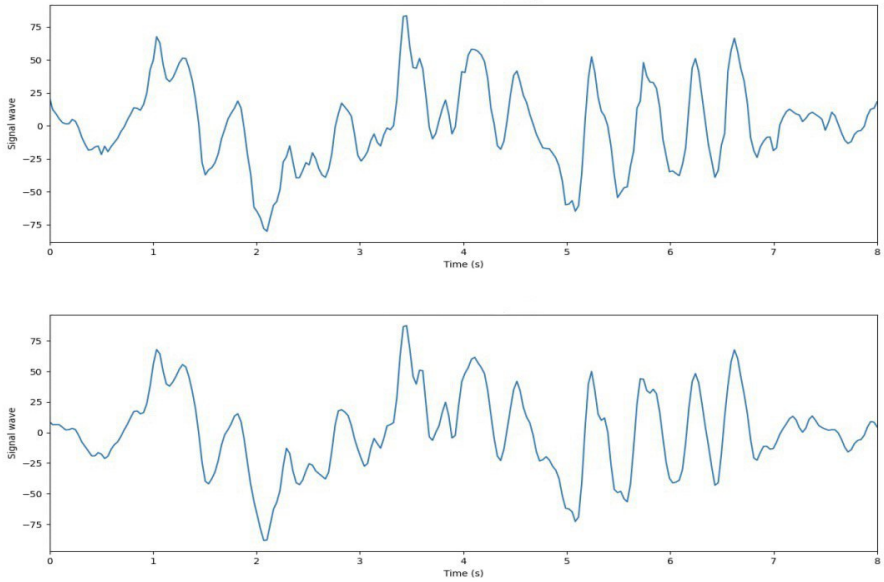


Figure 4.7: Time experiment - Mask ratio 15%. Up: raw PPG signal, Down: PPG signal reconstruction.

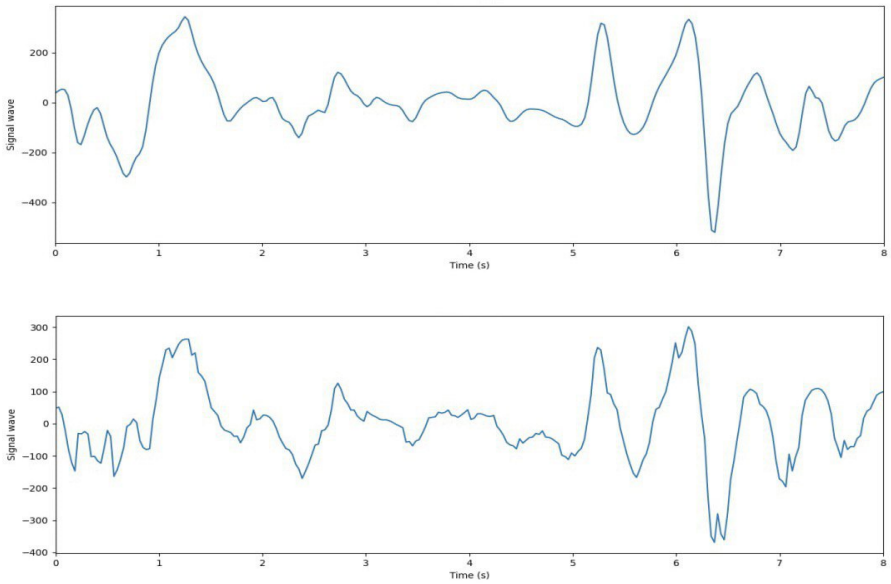


Figure 4.8: Time experiment - Mask ratio 75%. Up: raw PPG signal, Down: PPG signal reconstruction.

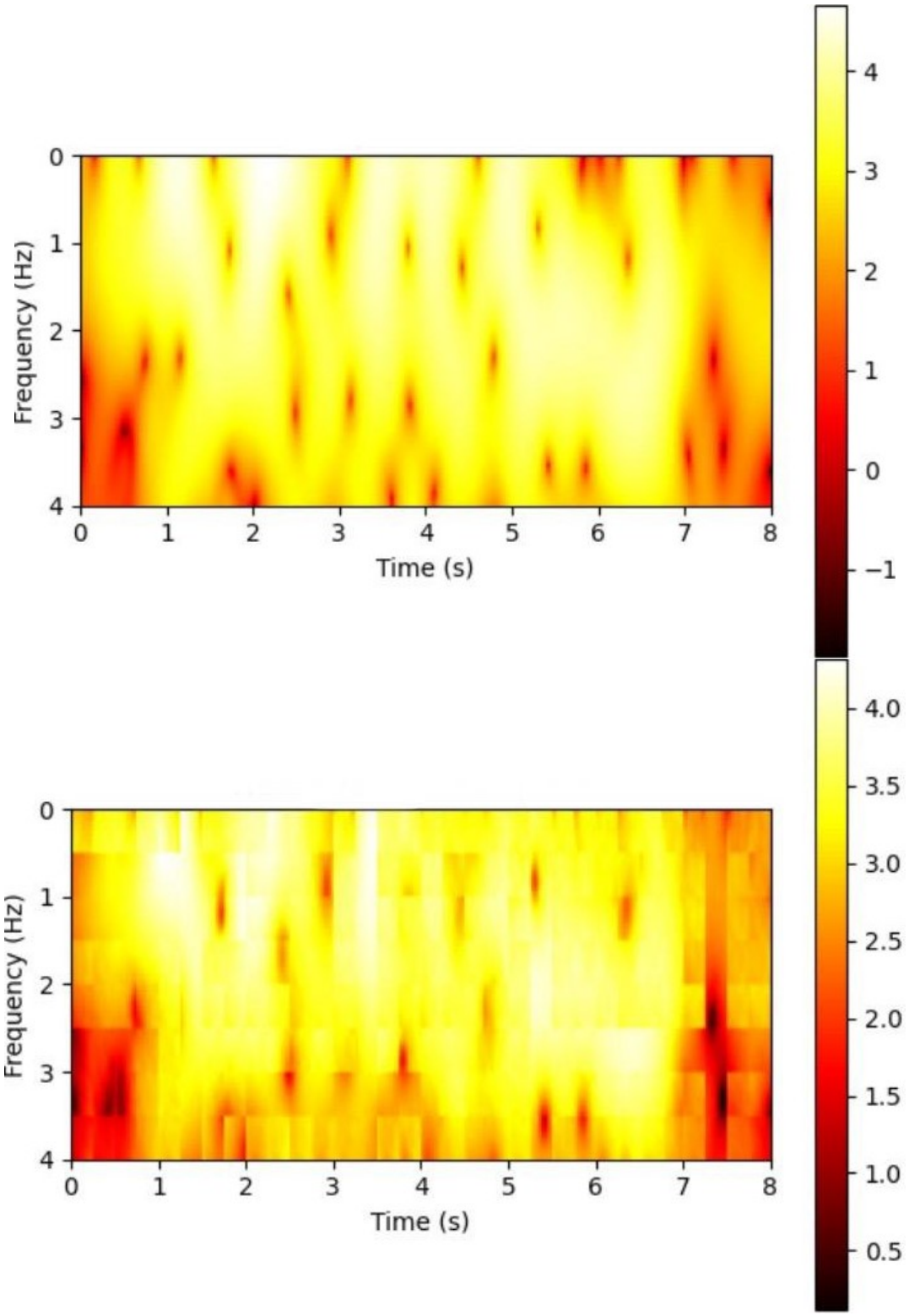


Figure 4.9: Frequency experiment - Mask ratio 15%. Up: PPG heatmap, Down: PPG heatmap reconstruction.

4.6 Fine-tuning

Once the pre-training phase is over, we preserve the information about the extracted features by saving the model weights. This checkpoint is then used as a starting point for the finetuning phase where the goal is to predict the heart rates. To perform this task, we replace the decoder from the original Masked Autoencoder with a regression tail comprising *2 convolutional layers*, *a pooling layer*, and *a final linear layer*.

The implementation is the following one:

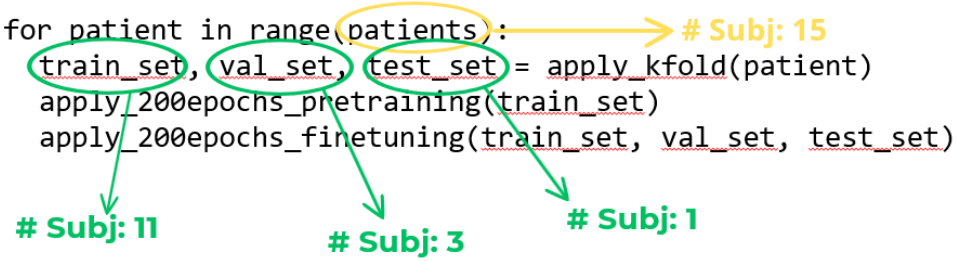
- **First istance of regression**:
  - nn.Conv1d(in_channels=256, out_channels=128, kernel_size=4, stride=4)
  - nn.ReLU()
  - nn.BatchNorm1d(num_features=128)
- **Second istance of regression**:
  - nn.Conv1d(in_channels=128, out_channels=64, kernel_size=4, stride=4)
  - nn.ReLU()
  - nn.BatchNorm1d(num_features=64)
- **Linear layer for predict HR**
  - nn.AvgPool1d(int(embed_dim/16))
  - nn.Linear(in_features=64, out_features=1)

The final task is the HR prediction, which is performed by the regressive tail starting from the hidden representation created by the encoder. We measure the accuracy of our model through the *Mean Absolute Error* (MAE).
Note that the model has been made flexible to fit any **correct** configuration of *embed_dim*, *depth* and *heads* parameters.

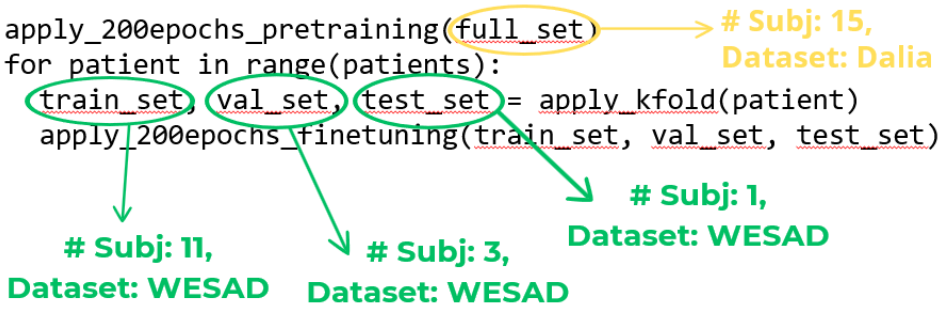
4.7 Transfer Learning

Transfer learning is a powerful technique in the field of deep learning that enables the transfer of knowledge learned from one task to another. In traditional machine learning approaches, models are built from scratch for each specific task. However, deep learning models, especially neural networks, often require large amounts of data and substantial computational resources for training, making them challenging to train from scratch for every new task. Transfer learning addresses this challenge by allowing a pre-trained model, which has been trained on a large dataset and a related task, to be fine-tuned or adapted to a new task with a smaller dataset. This approach leverages the knowledge and features learned by the model during its initial training, enabling it to perform well on the new task with fewer data and computation resources. Applying transfer learning, pre-trained models have learned abstract features that can be relevant across a range of tasks. By transferring this knowledge, the model can generalize better to new tasks, especially when the tasks share similar underlying patterns or characteristics. Driven mainly by this motivation, in this thesis, we investigate a transfer learning approach. In particular, our transfer learning implementation uses the full data set during the pre-training (instead of split the latter into training, validation and test subsets and then exploit only the one relating to the training subset). This dataset is specified by the variable `DATASET_PRETRAIN`. As soon as a new minimum of the loss function is found we save model parameters: doing so we are able to preserve the acquired information in extracting good features coming from *all* the patience, therefore, we are able to have a greater amount of data available during the pre-training which can be used for reach a greater generalization. This information is then exploits during the fine-tuning step, where the goal is to predict heart rates for each test patient present in dataset specified by the variable `DATASET_FINETUNING`. Next page shows with greater details what is described here, highlighting the comparison between the normal k-fold cross validation approach and our transfer learning implementation through a pseudo-code. Furthermore, the differences between transfer learning on WESAD and that on PPG-Dalia are also the highlighted.

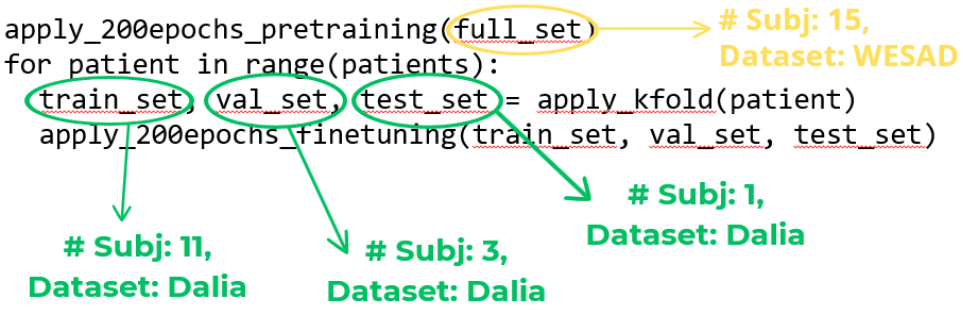
K-fold implementation:



Transfer Learning implementation on **WESAD**:



Transfer Learning implementation on **Dalia**:



Chapter 5

Results

The masked autoencoders described previously has been tested on *4* different datasets which I will describe in more detail one by one in this Chapter. The datasets are: PPG-Dalia, WESAD, IEEE_Train & IEEE_Test. A necessary observation to make is that the model is not able to correctly fit data from IEEE_Train & IEEE_Test, reaching only an average MAE over all the patiens of $\simeq$ 10 BPM. IEEE_Train & IEEE_Test are considerably smaller datasets compared to PPG-Dalia and WESAD and this behavior is probably related to the fact that the model is too complex and has too many parameters for the amount of training data available (see Table 4.1), trying to memorize training data rather than generalize correctly (overfitting). Since results for these datasets are not significant, they will not be reported in this thesis.

5.1 Datasets

5.1.1 PPG-Dalia

PPG-Dalia [13] is a large dataset for motion compensation and heart rate estimation during daily life activities. There are in total 15 subjects who participated in the data collection: eight female and seven male participants

with age between 21 and 55 years old. In the dataset we can find also further information about each subject such as height, weight and general fitness level. The collection of data was performed employing a chest-worn device and a wrist-worn device. The first was used to measure ECG signals, respiration capture with an inductive plethysmograph sensor, and 3D-accelerometer data. The wearable device, instead, provided a PPG sensor with four kinds of LEDs (two red and two green), and an inertial three-axis acceleration sensor sampling at 32Hz. As previously mentioned, the data collection protocol includes different activities, eight in particular, which are typically performed in daily life. In their study, the authors [13] incorporated activities that covered different intensity levels, considering tasks of low intensity such as driving, moderate intensity like walking, and involving high-intensity arm movements similar to playing table soccer, in order to generate motion artefacts at diverse amplitudes, see Table 5.1.

Table 5.1: Data collection protocol on PPG-Dalia: activities and their duration.

Activity	***Duration (min)***
Sitting still	10
Ascending/Descending stairs	5
Table Soccer	5
Cycling	8
Driving car	15
Lunch break	30
Walking	10
Working	20

An additional condition was the presence of both activities with periodic (like walking or descending stairs) and non-periodic patterns (like eating or table soccer). Furthermore, to ensure a wide range of heart rates, activities requiring varying levels of physical effort were selected (e.g., driving versus ascending stairs).

5.1.2 WESAD

WESAD is a multimodal dataset for wearable stress and affect detection. The data was captured through two wearable devices: a wrist-worn one (with sensors like PPG, accelerometer, electrodermal activity, and body temperature) and a chest-worn device (incorporating sensors like ECG, accelerometer, respiration and body temperature). The data collection involved 15 participants, aged between 24 and 35 years, each contributing approximately 100 minutes of data. The primary goal behind this dataset was to collect a wide amount of data in order to identify and differentiate various affective states, such as neutral, stress, and amusement. Consequently, WESAD predominantly includes data recorded during sedentary activities. PPG and accelerometers are sampled at 32Hz while ECG signal at 700Hz. The labels are different from those of PPG dataset, in fact they don't refer directly to heart rate values but are discrete numbers(e.g., 0 = not defined / transient, 1 = baseline, 2 = stress, 3 = amusement, 4 = meditation...). For this reason it is necessary to follow the methodology described previously in the Methods section to extract the mean of the instant heart rates from the ECG signal. Additional insights about this dataset are available in [37].

5.2 Experimental Results

In this section, we present the main results obtained in this thesis using the proposed method on PPG-Dalia and WESAD datasets. We measure accuracy of our model through the *Mean Absolute Error* (MAE). The MAE is a commonly used metric for evaluating the performance of a regression model. It measures the average of the absolute differences between the predicted values by the model (*HR_pred*) and the actual values from the test dataset (*HR_true*):

$$MAE = |HR_{true} - HR_{pred}| \tag{5.1}$$

Therefore, MAE provides a measure of the "average accuracy" of the model: a lower MAE indicates that the model is better at predicting the target values.

5.2.1 Experiments on time-domain input data

PPG-Dalia

Compared to the corresponding frequency experiment, the time results on PPG-Dalia are significantly worse. In particular, from Table 5.2, we can see how the worst result is given by patient 5 (S5), where the MAE is 24.86 BPM against 5.41 BPM of the reference. This behavior is explained by the fact that the model we are using is too large and is unable to extract correctly good features from the raw data in the time domain, running into overfitting. This phenomenon is also highlighted by Figure 5.2, which shows the trends of *train_mae* and *test_mae* of patient 5, during the first 10 epochs: the first drops reasonably until reaching a MAE of 4 BPM, while the second fluctuates continuously around a value of MAE of 27 BPM. From this figure, it can be deduced how the model performs well on training data, but poorly on test data and therefore learns to memorize the training data instead of generalizing from them. This gave us further proof that by applying a simple FFT and, therefore, working on spectrograms rather than on raw PPG signals as in [34], it is possible to teach better to the layers of the network learn how to extract good features for the task. Working in the frequency domain, indeed, we teach the model to give greater priority to the frequency component of the input, which provide more details about the heart rate. This is the primary reason why time experiments are confined exclusively to the use of PPG-Dalia and are not subjected to further investigation.

Figure 5.1: Comparison between Target, Output and Ouput Post-processing on 100 samples from S5 of PPG-Dalia.

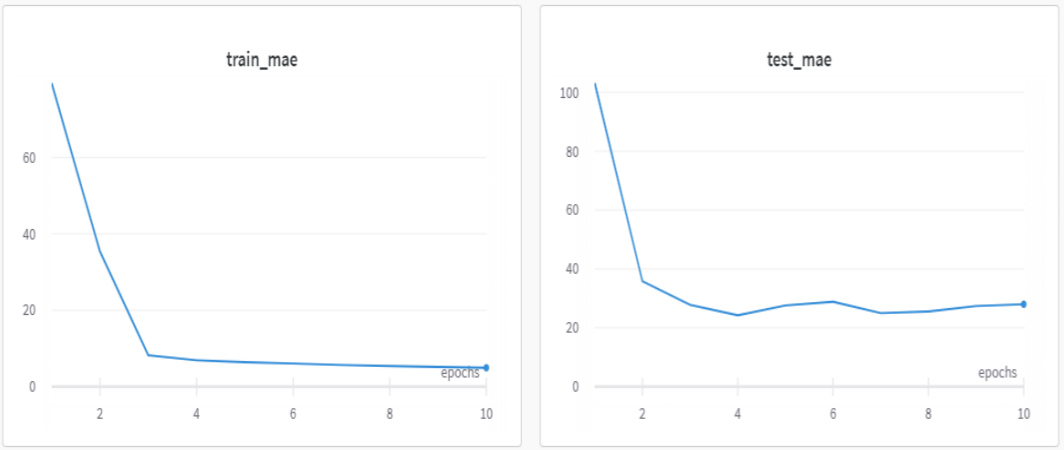


Figure 5.2: Comparison between Training MAE and Testing MAE from S5 of PPG-Dalia.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**
Reference MAE	3.25	2.55	2.66	4.21	5.41	4.11	2.06	5.07
T: Dalia, PT: Dalia	7,76	5,59	4,92	6,72	24,87	13,45	4,32	8,68
+ Post-processing	7,05	4,96	4,55	6,12	24,86	12,31	4,18	8,18
	S9	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**	**AVG**
Reference MAE	7.15	3.04	3.07	3.39	2.13	3.13	2.96	3.61
T: Dalia, PT: Dalia	9,26	4,94	10,15	9,97	5,10	5,13	5,70	8,43
+ Post-processing	8,11	4,42	9,86	9,44	4,98	4,70	5,08	7,92

Table 5.2: Time experiment results on PPG-Dalia. "T" is Target dataset, "PT" is Pre-Training dataset.

Despite these considerations, on some patients we are still able to be comparable with the state-of-the-art. For example, on patient 10 (S10) we reach a value of 4.42 BPM (after applying post-processing) against 3.04 BPM for the reference. Looking at the overall results, we reach a average MAE over all the patients of 7.92 BPM against the 3.61 BPM of the reference.

5.2.2 Experiments on frequency-domain input data

PPG-Dalia

As can be seen by looking at Table 5.3, experimental results on frequency domain regarding PPG_Dalia dataset are slightly worse but still comparable to the state-of-the-art. This is true, except for patient 5 (S5) whose MAE is approximately double (10.83 BPM vs 5.41 BPM). From this further evidence, we deduce that among all the patients in the dataset, patient 5 is the one on which obtaining a good level of accuracy is most challenging. We also note that applying a further post-processing step it is possible to further smooth MAE of all patients. Patients 3 (S3) and 13 (S13) are those that the model is able to fit best, going down the threshold of 4 BPM. Figure 5.3 shows in details the comparison between *target*, *output* and output after *post processing* of patient 3. From this example it is clear to note how the model

Figure 5.3: Comparison between Target, Output and Ouput Post-processing on 100 samples from S3 of PPG-Dalia.

sometimes varies its predictions excessively from one value of heart-rate to the next and how this behavior is promptly corrected by the post-processing step. Moreover, it is also possible to underline a comparison with the behavior of patient 5 (S5) in the time domain, shown in Figure 5.1. In this case, as we expected, the model is able to fit the HR trend much better, demonstrating the effectiveness of the proposed method in the frequency domain.
We can also observe how the application of Transfer Learning using WESAD as a pre-training dataset helps the model to slightly decrease MAE in the majority of the cases. This is particularly evident on patient 1 (S1) where it

was possible to improve accuracy by 0.56 BPM (5.86 BPM vs 5.30 BPM).

WESAD

When employing our proposed method on the WESAD dataset, all the patients' Mean Absolute Error (MAE) results exhibit significant enhancement compared to the reference ones. In particular, remarkable enhancement is obtained on patient 16 (S16), where, after applying post-processing, the performance improved significantly by 8.36 BPM (dropping from 12.78 BPM of the reference to 4.42 BPM). Looking at the overall results, it is possible to achieve an average of 5.19 BPM against 7.47 BPM of the reference. By taking advantage from Transfer Learning, exploiting Dalia as a pre-training dataset, it is possible to further reduce the average MAE up to 5.09 BPM. These important results are shown in Table 5.4, and demonstrate the effectiveness and general consistency of our approach.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**
Reference MAE	3.25	2.55	2.66	4.21	5.41	4.11	2.06	5.07
T: Dalia, PT: Dalia	6.30	5.67	3.69	6.28	11.09	6.15	3.48	8.37
+ Post-processing	5.86	5.19	3.63	5.90	10.83	5.76	3.47	7.42
T: Dalia, PT: Wesad	5.95	5.85	3.69	6.31	11.30	6.10	3.37	8.20
+ Post-processing	5.30	5.20	3.66	5.81	10.99	5.52	3.37	7.48
	S9	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**	**AVG**
Reference MAE	7.15	3.04	3.07	3.39	2.13	3.13	2.96	3.61
T: Dalia, PT: Dalia	9.48	4.57	7.27	6.33	3.15	5.00	5.33	6.14
+ Post-processing	8.47	4.07	6.86	5.65	3.05	4.56	5.10	5.72
T: Dalia, PT: Wesad	9.48	4.51	7.34	6.57	3.24	4.98	5.49	6.15
+ Post-processing	8.26	4.05	6.71	5.66	3.16	4.48	5.02	5.64

Table 5.3: Frequency experiment results on PPG-Dalia. "T" is Target dataset, "PT" is Pre-Training dataset.

	S2	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**
Reference MAE	5.07	14.48	7.84	7.70	3.88	6.78	4.27	3.99
T:Wesad, PT:Wesad	3.77	12.06	3.72	6.37	3.76	4.94	4.35	3.62
+ Post-processing	3.60	11.93	3.63	5.82	3.50	4.85	3.88	3.50
T: Wesad, PT: Dalia	3.77	11.96	3.79	6.20	3.59	5.06	4.14	3.45
+ Post-processing	3.62	11.73	3.67	5.76	3.33	4.94	3.89	3.37
	S10	**S11**	**S13**	**S14**	**S15**	**S16**	**S17**	**AVG**
Reference MAE	8.89	11.07	6.52	5.26	4.18	12.78	9.36	7.47
T:Wesad, PT:Wesad	4.16	9.04	4.66	3.74	3.87	4.71	8.44	5.41
+ Post-processing	4.00	8.93	4.29	3.59	3.79	4.42	8.24	5.19
T: Wesad, PT: Dalia	4.04	9.13	4.47	3.72	3.76	4.81	8.42	5.35
+ Post-processing	3.94	8.55	4.07	3.46	3.63	4.44	8.08	5.09

Table 5.4: Frequency experiment results on WESAD. "T" is Target dataset, "PT" is Pre-Training dataset.

Chapter 6

Conclusions and future works

In this thesis, we explored the application of self-supervised learning to improve heart rate estimation extracted through PPG sensor placed on wearable devices. To test the effectiveness of our proposed method we have used mainly PPG-Dalia and WESAD datasets. They are, indeed, the two largest datasets currently available in which various daily activities of the participants involved are monitored. The results obtained were promising and demonstrate the value of this innovative approach. Below, I summarize the main conclusions:

We have seen a huge improvement in heart rate estimation performance using self-supervised learning compared to traditional methods. This progress was particularly evident in the WESAD dataset, where the proposed model reaches and average of 5.19 BPM (betten than state-of-the-art of 2.28 BPM) suggesting that the use of self-learning techniques could revolutionize the field of heart rate estimation.

At the same time it is important to note that, in addition to the WESAD dataset, we examined also the DALIA dataset. However, in this case, the results obtained were comparable to traditional approaches. This suggests that the effectiveness of self-supervised learning may vary based on the specific dataset and data characteristics. So, despite WESAD was invaluable

for this research, it may be interesting to explore other datasets to further test the effectiveness of self-supervised learning. Larger or context-specific datasets, indeed, could reveal new challenges and opportunities and the presented work can be used as a basis for further studies and applications in domains such as medicine, fitness and wellness.

In conclusion, it's worth mentioning that the implementation choice of utilizing a straightforward regression tail, as described in Chapter 4.6, might not be the best solution. New configurations could be tested, including the use of attention blocks, as shown in the Transformer Encoder of Figure 2.4, even in the final phase where the heart-rate is extracted. This strategy is motivated by the fact that attention blocks are able to capture broader contexts and complex relationships between input elements, which can be difficult to achieve with traditional CNNs that rely on local convolutions.

Bibliography

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words:Transformers for image recognition at scale. In ICLR, 2021.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In NAACL, 2019

[3] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and PierreAntoine Manzagol. Extracting and composing robust features with denoising autoencoders. In ICML, 2008.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In NeurIPS, 2017.

[5] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. Neural computation, 1989.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In ICLR, 2021.

[7] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," arXiv preprint arXiv:2111.06377, 2021.

[8] N. Sviridova and K. Sakai, "Human photoplethysmogram: New insight

into chaotic characteristics," Chaos, Solitons Fractals, vol. 77, pp. 53–63, 2015

[9] Apple, "Apple watch series." Accessed: Aug. 25, 2021. [Online]. Available: https://www.apple.com/lae/watch/

[10] T. Tamura et al., "Wearable photoplethysmographic sensors-past and present," Electronics, vol. 3, no. 2, pp. 282–302, 2014.

[11] D. Castaneda et al., "A review on wearable photoplethysmography sensors and their potential future applications in health care," Int. J. Biosensors Bioelectron., vol. 4, no. 4, pp. 195–202, 2018.

[12] Z. Zhang, Z. Pi, and B. Liu, "TROIKA: A general framework for heart rate monitoring using wrist-type photoplethysmographic signals during intensive physical exercise," IEEE Trans. Biomed. Eng., vol. 62, no. 2, pp. 522–531, Feb. 2015

[13]] A. Reiss et al., "Deep PPG: Large-scale heart rate estimation with convolutional neural networks," Sensors, vol. 19, no. 14, 2019, Art. no. 3079.

[14] Burrello A, Pagliari DJ, Risso M, Benatti S, Macii E, Benini L, Poncino M. Q-PPG: Energy-Efficient PPG-Based Heart Rate Monitoring on Wearable Devices. IEEE Trans Biomed Circuits Syst. 2021 Dec;15(6):1196-1209. doi: 10.1109/TBCAS.2021.3122017. Epub 2022 Feb 17. PMID: 34673496.

[15] D. Biswas et al., "CorNET: Deep learning framework for PPG-based heart rate estimation and biometric identification in ambulant environment," IEEE Trans. Biomed. Circuits Syst., vol. 13, no. 2, pp. 282–291, Apr. 2019.

[16] L. G. Rocha et al., "Binary corNET: Accelerator for HR estimation from wrist-PPG," IEEE Trans. Biomed. Circuits Syst., vol. 14, no. 4, pp. 715–726, Aug. 2020.

[17] Hecht-Nielsen, "Theory of the backpropagation neural network," International 1989 Joint Conference on Neural Networks, Washington, DC, USA, 1989, pp. 593-605 vol.1, doi: 10.1109/IJCNN.1989.118638

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words:Transformers for image recognition at scale. In ICLR, 2021

[19] Robert Avram et al. «Real-world heart rate norms in the Health eHeart study». In: NPJ digital medicine 2.1 (2019), pp. 1–10 (cit. on p. 24).

[20] Chenggang Yu, Zhenqiu Liu, Thomas McKenna, Andrew T Reisner,

and Jaques Reifman. «A method for automatic identification of reliable heart rates calculated from ECG and PPG waveforms». In: Journal of the American Medical Informatics Association 13.3 (2006), pp. 309–320 (cit. on p. 24)

[21] Shahid Ismail, Usman Akram, and Imran Siddiqi. «Heart rate tracking in photoplethysmography signals affected by motion artifacts: a review». In: EURASIP Journal on Advances in Signal Processing 2021.1 (2021), pp. 1–27 (cit. on p. 24)

[22] Sanjeev Kumar et al. «A wristwatch-based wireless sensor platform for IoT health monitoring applications». In: Sensors 20.6 (2020), p. 1675 (cit. on p. 25).

[23] Dwaipayan Biswas et al. «CorNET: Deep Learning Framework for PPG-Based Heart Rate Estimation and Biometric Identification in Ambulant Environment». In: IEEE Transactions on Biomedical Circuits and Systems 13.2 (2019), pp. 282–291. doi: 10.1109/TBCAS.2019.2892297 (cit. on pp. 25, 28)

[24] Zhilin Zhang, Zhouyue Pi, and Benyuan Liu. «TROIKA: A General Frame- work for Heart Rate Monitoring Using Wrist-Type Photoplethysmographic Sig- nals During Intensive Physical Exercise». In: IEEE Transactions on Biomed- ical Engineering 62.2 (Feb. 2015), pp. 522–531. doi: 10.1109/tbme.2014. 2359372. url: https://doi.org/10.1109p. 25).

[25] Zhilin Zhang. «Photoplethysmography-Based Heart Rate Monitoring in Phys- ical Activities via Joint Sparse Spectrum Reconstruction». In: IEEE Transac- tions on Biomedical Engineering 62.8 (2015), pp. 1902–1910. doi: 10.1109/ TBME.2015.2406332 (cit. on p. 26)

[26] Zhilin Zhang. «Heart rate monitoring from wrist-type photoplethysmographic (PPG) signals during intensive physical exercise». In: 2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP). IEEE. 2014, pp. 698–702 (cit. on p. 26).

[27] Boreom Lee, Jonghee Han, Hyun Jae Baek, Jae Hyuk Shin, Kwang Suk Park, and Won Jin Yi. «Improved elimination of motion artifacts from a photoplethysmographic signal using a Kalman smoother with simultaneous accelerometry». In: Physiological measurement 31.12 (2010), p. 1585 (cit. on p. 26).

[28] R. M. Al-Eidan, H. Al-Khalifa, and A. M. Al-Salman, "A review of wristworn wearable: Sensors, models, and challenges," J. Sensors, vol. 2018, 2018.

[29] A. Reiss et al., "Deep PPG: Large-scale heart rate estimation with convolutional neural networks," Sensors, vol. 19, no. 14, 2019, Art. no. 307

[30] Matteo Risso, Alessio Burrello, Daniele Jahier Pagliari, Simone Benatti, Enrico Macii, Luca Benini, and Massimo Pontino. «Robust and Energy-Efficient PPG-Based Heart-Rate Monitoring». In: 2021 IEEE International Symposium on Circuits and Systems (ISCAS). 2021, pp. 1–5. doi: 10.1109/ISCAS51556. 2021.9401282 (cit. on pp. 29–31, 47).

[31] Alessio Burrello, Daniele Jahier Pagliari, Pierangelo Maria Rapa, Matilde Semilia, Matteo Risso, Tommaso Polonelli, Massimo Poncino, Luca Benini, and Simone Benatti. «Embedding temporal convolutional networks for energy-efficient PPG-based heart rate monitoring». In: ACM Transactions on Com- puting for Healthcare (HEALTH) 3.2 (2022), pp. 1–25 (cit. on pp. 30, 47– 49).

[32] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. «Morphnet: Fast and simple resource-constrained struc- ture learning of deep networks». In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2018, pp. 1586–1595 (cit. on p. 30)

[33] Alessio Burrello, Daniele Jahier Pagliari, Matteo Risso, Simone Benatti, Enrico Macii, Luca Benini, and Massimo Poncino. «Q-ppg: energy-efficient ppg-based heart rate monitoring on wearable devices». In: IEEE Transactions on Biomedical Circuits and Systems 15.6 (2021), pp. 1196–1209 (cit. on pp. 29, 31, 47).

[34] Po-Yao Huang, Hu Xu, Juncheng B Li, Alexei Baevski, Michael Auli, Wojciech Galuba, Florian Metze, Christoph Feichtenhofer. "Masked Autoencoders that Listen", 2022

[35] Y. Gong, Y. Chung, and J. R. Glass, “AST: audio spectrogram transformer,” in Interspeech 2021, 22nd Annual Conference of the International Speech Communication Association, Brno, Czechia, 30 August - 3 September 2021. ISCA, 2021, pp. 571–575.

[36] Y. Gong, C.-I. Lai, Y.-A. Chung, and J. R. Glass, “Ssast: Self-supervised audio spectrogram transformer,” ArXiv, vol. abs/2110.09784, 2021.

[37] Schmidt, P.; Reiss, A.; Duerichen, R.; Van Laerhoven, K. Introducing WESAD, a multimodal dataset for WEarable Stress and Affect Detection. In Proceedings of the 20th ACM International Conference on Multimodal Interaction, Boulder, CO, USA, 16–20 October 2018.

[38] Cup, I.S.P. Heart Rate Monitoring During Physical Exercise using Wrist-Type Photoplethysmographic (PPG) Signals. 2015. Available online: https://sites.google.com/site/researchbyzhang/ieeespcup2015 (accessed on 21 May 2019).

[39] Zhang, Z. Photoplethysmography-based Heart Rate Monitoring in Physical Activities via Joint Sparse Spectrum Reconstruction. IEEE Trans. Biomed. Eng. 2015, 62, 1902–1910. [CrossRef] [PubMed]

[40] Jarchi, D.; Casson, A. Description of a Database Containing Wrist PPG Signals Recorded during Physical Exercise with Both Accelerometer and Gyroscope Measures of Motion. Data 2017, 2, 1.

[41] Fitbit, "Fitbit charge 4." Accessed: Aug. 25, 2021. [Online]. Available: https://www.fitbit.com/global/us/products/trackers/charge4

[42] Z. Ge et al., "Evaluating the accuracy of wearable heart rate monitors," in Proc. 2nd Int. Conf. Adv. Comput., Commun. Automat., 2016, pp. 1–6.

[43] H.-Y. Jan et al., "Evaluation of coherence between ECG and PPG derived parameters on heart rate variability and respiration in healthy volunteers with/without controlled breathing," J. Med. Biol. Eng., vol. 39, no. 5, pp. 783–795, 2019.

[44] "Neurosky." 2020. Accessed: Aug. 25, 2021. [Online]. Available: 'http://neurosky.com/wp-content/uploads/2016/06/TOF-side-by-sidecompetitor- comparison.pdf'

[45] A. Temko, "Accurate heart rate monitoring during physical exercises using PPG," IEEE Trans.Biomed. Eng., vol. 64, no. 9, pp. 2016–2024, Sep. 2017.

[46] T. Schäck, M. Muma, and A. M. Zoubir, "Computationally efficient heart rate estimation during physical exercise using photoplethysmographic signals," in Proc. 25th Eur. Signal Process. Conf., 2017, pp. 2478–2481.

[47] H. Chung, H. Lee, and J. Lee, "Finite state machine framework for instantaneous heart rate validation using wearable photoplethysmography during intensive exercise," IEEE J. Biomed. Health Inform., vol. 23, no. 4, pp. 1595–1606, Jul. 2019.

[48] S. Salehizadeh et al., "A novel time-varying spectral filtering algorithm for reconstruction of motion artifact corrupted heart rate signals during intense physical activities using a wearable photoplethysmogram sensor," Sensors, vol. 16, no. 1, pp. 10–30, 2016

[49] Theodoros Evgeniou, Massimiliano Pontil, "Support Vector Machines: Theory and Applications" September 2001

[50] Gilles Louppe, "Understanding Random Forests: From Theory to Practice", July 2014

[51] Khushbu Kumari, Suniti Yadav "Linear regression analysis study", January 2018

[52] Alessio Burrello et al. Improving PPG-based Heart-Rate Monitoring with Synthetically Generated Data, 2022

[53] Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E.Hinton, "Layer Normalization" July 2016

[54] M.Zanghieri,S.Benatti,A.Burrello,V.Kartsch,F.Conti,andL.Benini, "Robust real-time embedded emg recognition framework using temporal convolutional networks on a multicore iot processor,"IEEE Transactions on Biomedical Circuits and Systems,2019.

[55] Salehizadeh, S.; Dao, D.; Bolkhovsky, J.; Cho, C.; Mendelson, Y.; Chon, K. A Novel Time-varying Spectral Filtering Algorithm for Reconstruction of Motion Artifact Corrupted Heart Rate Signals During Intense Physical Activities using aWearable Photoplethysmogram Sensor. Sensors 2016, 16, 10.

[56] Schaeck, T.; Muma, M.; Zoubir, A.M. Computationally Efficient Heart Rate Estimation During Physical Exercise using Photoplethysmographic Signals. In Proceedings of the 25th European Signal Processing Conference (EUSIPCO), Kos, Greece, 28 August–2 September 2017.