Class-Agnostic Repetitive Action Counting Using Wearable Devices

Duc Duy Nguyen, Lam Thanh Nguyen, Yifeng Huang, Cuong Pham *, Minh Hoai

Abstract—We present Class-agnostic Repetitive action Counting (CaRaCount), a novel approach to count repetitive human actions in the wild using wearable devices time series data. CaRaCount is the first few-shot class-agnostic method, being able to count repetitions of any action class with only a short exemplar data sequence containing a few examples from the action class of interest. To develop and evaluate this method, we collect a large-scale time series dataset of repetitive human actions in various context, containing smartwatch data from 10 subjects performing 50 different activities. Experiments on this dataset and three other activity counting datasets namely Crossfit, Recofit, and MM-Fit show that CaRaCount can count repetitive actions with low error, and it outperforms other baselines and state-of-the-art action counting methods. Finally, with a user experience study, we evaluate the usability of our real-time implementation. Our results highlight the efficiency and effectiveness of our approach when deployed outside the laboratory environments.

Index Terms—Class-agnostic, few-shot learning, action counting, multivariate time series, smartwatch, wearable

1 Introduction

Repetitive actions are common in everyday life, from daily routines (e.g., hair combing and teeth brushing) and physical exercises (e.g., weight lifting and rope skipping) to construction activities (e.g., sawing and nailing) and manufacturing activities (e.g., packing and attaching labels). The need to count repetitive actions arises in various situations and for many reasons, including exercise logging [5, 16, 17, 22, 31], fatigue monitoring [10, 12, 34], and efficiency improvement [38].

Unfortunately, there exists no class-agnostic counter for repetitive actions. There are only algorithms and mobile applications for specific action classes such as walking steps [9, 25], physical exercises [22, 33, 35, 41] and swimming laps [4]. These algorithms and apps are often based on traditional signal processing techniques or detection and recognition models, which require specific expert knowledge for each action category or offline supervised training data containing thousands of examples for each supported action class. As a result, it is difficult to scale existing algorithms to handle a large number of action classes.

In this paper, we introduce a novel approach for counting class-agnostic repetitive actions in the wild by posing it as a few-shot learning-then-inferring task as shown in Fig. 1. In this few-shot setting, the inputs for the counting task are a query time series along with a short sequence containing a few exemplar instances for the action class of interest. The desired output is the number of repetitions contained in the query sequence. The number of examples in the exemplar sequence is also provided; this can be as small as three in our experiments. Both the query and exemplar sequences are multivariate time-series data and the query may contain other actions in addition to the actions of interest. The actions of interest in the exemplar sequence form a contiguous block in the exemplar sequence, but the repetitive actions of interest in the query sequence may not form a contiguous segment.

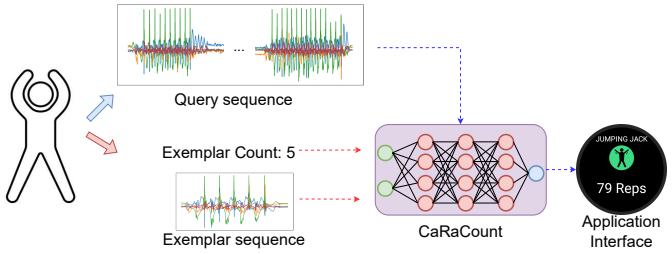


Fig. 1: **Application setting.** Given an exemplar sequence containing a few repetitions of the action class of interest, our system can count the number of repetitions performed in the query sequence.

We name our method CaRaCount, which stands for **C**lass **a**gnostic **R**epetitive **a**ction **Count**er. CaRaCount has three key components: (1) a feature extraction module, (2) a phase assignment module, and (3) a count regression module. The feature extraction module is the combination of a CNN and an adapted GRU [39] that encode each temporal window into a feature vector. The phase assignment module leverages Connectionist temporal classification (CTC) [11] characteristics to assign a phase label to each embedding feature vector for each temporal window. This module is designed to be agnostic to the action category. The count regression module analyzes the sequence of phase assignments and outputs the final counting result.

Corresponding author: Cuong Pham; Email: cuongpv@ptit.edu.vn

- *Duc Duy Nguyen and Minh Hoai are with the Australian Institute for Machine Learning, the University of Adelaide, Adelaide, Australia.*
- *Lam Thanh Nguyen, Cuong Pham, and Minh Hoai are with VinAI, Hanoi, Vietnam.*
- *Yifeng Huang is with the Department of Computer Science, Stony Brook University, Stony Brook, NY.*
- *Cuong Pham is with the Faculty of Information Technology, Posts and Telecommunications Insitute of Technology, Hanoi, Vietnam.*

To address the lack of a times series dataset for developing and evaluating the performance of few-shot counting methods, we collect a large-scale dataset of wearable device. This dataset contains more than 1700 samples, with 35,787 repetitions from 10 subjects performing 50 different action categories. We name this dataset Class Agnostic Repetitive Action (CaRa) Dataset. The code and dataset is available for research purpose and the link can be downloaded at https://github.com/duyddwcs/Action-Counting.

To summarize, the major contributions of this paper are:

1) We introduce a novel and practical application for class-agnostic repetitive action counting.
2) We develop a novel method for this application setting which outperform other state-of-the-art action counting methods.
3) We introduce a large-scale time series dataset containing repetitive actions collected under real-world settings.

The remaining of the paper is structured as follows. Section 2 reviews the related work, and Section 3 introduces our proposed class-agnostic repetitive action counting method. Section 4 provides details about our dataset named CaRa, while Section 5 presents our empirically experimental results on the CaRa dataset and three other public datasets. An user study is provided in Section 6.

2 Related Work

2.1 Vision-based repetition counting

Existing methods for vision-based repetition counting typically represent a video as a one-dimensional signal and subsequently aims to recover the repetitive structure. The repetitive structure is then extracted by Fourier analysis [2, 7, 24], peak detection [36], or singular value decomposition [6]. However, the above approaches assume the repetition is periodic and therefore are unable to deal with the non-stationary repetitions in the real world. Runia *et al.* [30] address the non-stationary situation by leveraging the wavelet transform based on the flow field. Dwibedi *et al.* [8] used temporal self-similarity between video frames for repetition estimation. The authors chose the frame rate sampling of the input video by picking the one with the maximum periodicity classification score. Zhang *et al.* [42] proposed a method developed from [21] to estimate and refine the length of two consecutive repetitions, which is called double-cycle. The counting result later can be obtained by averaging all the cycle lengths in the video. Inspired by this idea, we propose to split each repetition of an action class into multiple phases and count the number of repetitions by detecting the ordered sequence of constituent phases.

2.2 Sensor-based repetition counting

All previous sensor-based repetition counting methods are customized for specific types of activity, especially physical exercises [22, 33, 35, 41]. Zelman *et al.* [41] proposed to filter out high-frequency noise and then tune a threshold to add one count whenever the signal value crosses the line in the positive direction. Morris *et al.* [22] proposed RecoFit, a system for automatically tracking repetitive exercises. The extracted features for each 5-second sliding window are classified using a multi-class support vector machine. After the recognition phase, the data is represented by a one-dimensional signal using PCA, and the final counting result is obtained by autocorrelation combined with amplitude statistics. Soro *et al.* [33] used two identical CNN structures for a recognition module and a counting module. The raw data in a temporal window is fed into the counting module, which corresponds to the exercise that was previously detected by the recognition module. These methods require hundreds to thousands of annotated training data instances, and they also need prior knowledge about the activity from the recognition phase to achieve good counting results. In this work, we are interested in counting the number of repetitions contained in the query sequence using only a few exemplar repetitions.

2.3 Few-shot learning

The goal of few-shot learning is to learn a model from a few training examples. There are many promising studies in the computer vision literature; and the most relevant to our work are class-agnostic object counting [15, 23, 26, 29] following a density-based estimation approach (e.g., [1, 27, 28]). In contrast, few-shot learning for sensor-based counting tasks has not received much attention. To the best of our knowledge, we are the first to address the few-shot class-agnostic sensor-based action counting problem.

3 Class-agnostic Repetition Counting

We propose a novel approach for counting repetitions of unseen action classes in the wild, by posing it as a few-shot learning-then-inferring task as shown in Fig. 1. In this few-shot setting, the inputs for the counting task are: (1) a query sequence of length n, $\mathcal{Q} \in \mathfrak{R}^{d \times n}$, where d is the number of features at each time step; (2) an exemplar sequence of length m, $\mathcal{E} \in \mathfrak{R}^{d \times m}$, and (3) the scalar r for the number of repetition instances in the exemplar sequence. Both the query and exemplar sequences are multivariate time-series data of d dimensions, and the query may contain other actions in addition to the instances of the action of interest. The desired output is the number of repetitions of the action of interest contained in the query sequence.

The process for few-shot learning-then-inferring consists of two stages. In the first stage, we use the exemplar sequence and the known number of repetitions in this sequence to train the model. In the second stage, we use this model to predict the number of repetitions for the specified action of interest in the query sequence. In the remaining of this section, we will describe the architecture and training of this model. We name the model CaRaCount, which stands for Class-agnostic Repetitive Action Counter.

3.1 Network architecture

CaRaCount consists of three key modules: (1) a feature extraction module, (2) a phase assignment module, and (3) a counting module. These modules and the processing pipeline are illustrated in Fig. 2. Although our method is applicable to various types of time series data, we will specifically describe our method for data collected with wearable devices in this section for brevity.

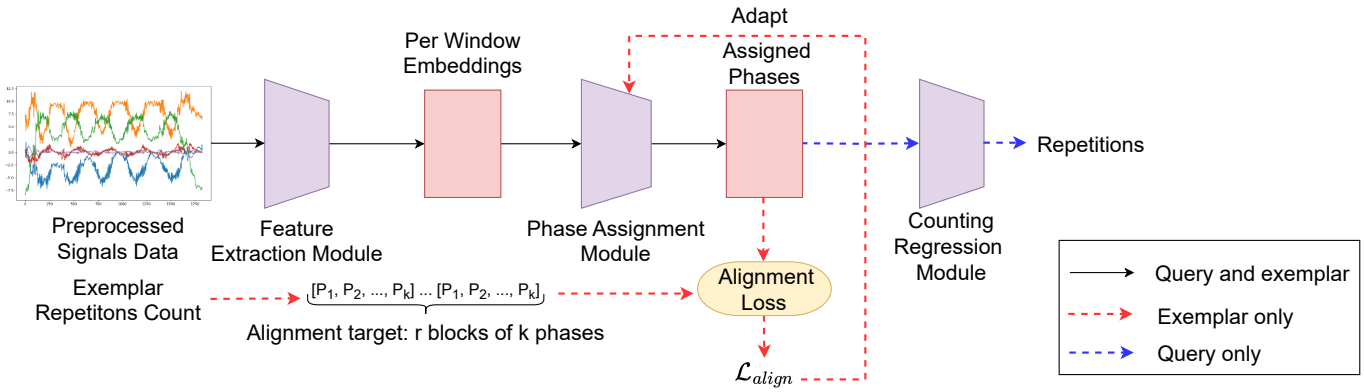


Fig. 2: **Overview of the CaRaCount.** CaRaCount takes the query sequence along with an exemplar sequence containing a few example repetitions from the activity of interest as an input. The final result is achieved by the count regression module, a pretrained model with synthetic data. The adaptation loss is measured based on the aligned repetition count from the exemplar sequence.

3.1.1 *Feature extraction module*

This module inputs raw time series data from a wearable device and outputs a sequence of feature vectors, each vector corresponding to a temporal window of the original time series. As the raw signals from a smartwatch might be noisy due to environmental fluctuation, sensor variety, and sensor misplacement, we apply some pre-processing techniques as follows. First, we use a Butterworth low-pass filter for noise removal. Second, we apply cubic spline interpolation to resample signal data, with the sampling frequency being 100Hz. This means, for each second, we have 100 samples of 6-dimensional values constructed from a 3-axis accelerometer and a 3-axis gyroscope. Finally, we divide the time series data into overlapping sliding windows. The size of the sliding window is a crucial parameter for the effectiveness of our method, and we determine the window size as follows. We first project the exemplar sequence onto its first principal component direction to obtain a 1D signal and then smooth the data using a third-degree polynomial filter. We then segment the sequence using window size n/r and overlapping duration 250ms, where n is the length of the exemplar sequence and r is the number of repetitions represented in the exemplar sequence. For each window, we calculate the autocorrelation coefficients and take all the peaks as a candidate for the length of one repetition. Considering the range of all candidate lengths, we divide this range into ten bins and put each candidate into its corresponding bin. We choose the bin that contains the largest number of candidate values and average the values to get the estimated repetition duration. The final window size is computed by dividing the repetition duration by the number of phases and the overlapping duration is 50% of the window size.

We subsequently embed each window into a high-dimensional representation turning the original exemplar sequence into a sequence of 200-dimensional embedding vectors using ConvGru adopted from [39]

3.1.2 *Phase assignment module*

The phase assignment consists of a three-layer MLP with sizes 100, 50, and $k+2$, where k is a tunable hyper-parameter for the number of phases for the action of interest and two additional dimensions are the background and the blank filter for CTC. We insert a dropout layer $(p = 0.5)$ after each MLP to improve model generalizability. Finally, a CTC transcription layer is used to output a discrete class index from 0 to k from eaach embedding vector. Our key assumption is that each action of interest contains exactly k phases: $P_1, P_2, \ldots, P_k$ in this order. Since the exemplar sequence may contain noise from undesirable actions at both ends, to improve the model's alignment precision, we concatenate the k+1 index to the start and end of the alignment target to indicate the background. Once the phase assignment module has been trained using the exemplar sequence, we feed the query sequence into this module to get a sequence of phase labels.

3.1.3 *Count regression module*

In theory, we can obtain the total count by determining the number of periodic blocks $[P_1, P_2, \ldots, P_k]$ in the obtained sequence of phase labels. In practice, due to noisy data and imperfect phase assignment, an action instance might not correspond to this exact periodic block. For robustness, we use a count regression module to obtain the final count. We use three-layer MLP, each layer followed by a dropout layer to create a regression head. Before feeding into this regression head, output from the phase assignment module is zero-padded to have a fixed length.

3.2 Model training

A CaRaCount model has multiple learnable parameters. Some parameters are class independent, and they are learned once and fixed for all classes. Some parameters are class dependent, and they are learned specifically for each action class in a few-shot setting when the exemplar action sequence is provided.

Few-shot learning of the phase assignment module. The phase assignment module can be trained using only one exemplar sequence $\mathcal{E}$ and the known number of action count r for this exemplar sequence. We adopt the CTC loss as Alignment loss for training with the objective is minimizing

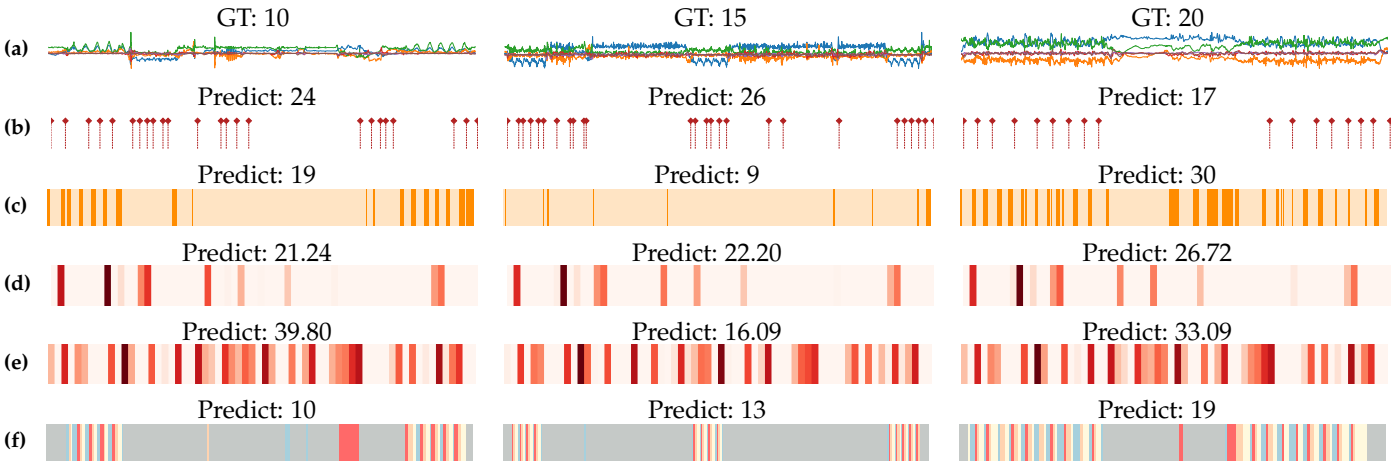


Fig. 3: **Difficulties in counting human actions include variability in form, irrelevant periodic actions, and inconsistencies in duration and appearance.** The breakpoints predicted from [22] and [33] are presented in (b) and (c). Predicted density maps from modified RepNet [8] and TransRAC [14] are shown in the (d) and (e), respectively. The predicted label sequence in the (f) demonstrates the repetitions accurately detected by CaRaCount.

the negative log-likelihood of the CTC conditional probability:

$$P(Y \mid X) = \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p_t(a_t \mid X) \tag{1}$$

In the above equation, Y is the ground truth target being r blocks of k phases with background at both ends: $[P_0], [P_1, P_2, \ldots, P_k], \ldots, [P_1, P_2, \ldots, P_k], [P_0]$ while X denotes the sequence of embedded vector. $\mathcal{A}$ is the set of valid alignments and $A = [a_1, a_2, \ldots, a_T]$ is a member of $\mathcal{A}$. The phase assignment module will classify each 200-dimensional feature embedding vector into $k + 2$ classes, with k classes corresponding to the k phases, one class indicates the background and one special class for the "blank" filler. We trained the phase assignment module along with the feature extractor until the loss converges with Adam optimizer[20] and a learning rate of 5×10^{-3}.

One-off learning of the count regression module. This count regression head is trained to count the number of periodic blocks with synthetically generated data as follows. We first generate a large number of initial sequences by concatenating blocks of k phases, with the number of blocks ranging from 1 to 150, which is the highest number of repetitions performed in our CaRa dataset. Then, for each initial sequence, we randomly remove, repeat, swap some indexes or mix those operations to create a noisy sequence (shown in Fig. 4). We set the threshold equal 30% length of the sequence for the maximum number of operations that can perform on the initial sequence to keep the original periodic characteristic of the sequence. Using this simple yet effective technique, we generate hundreds of thousands of synthetic data. We then use this synthetic data to train our count regression module by minimizing the Mean Squared Error loss between the predicted number produced by the module and the ground truth repetition count used to generate the initial perfect sequence. This regression module is trained for 200 epochs with a learning rate of 3×10^{-4} and a 0.1 exponential learning rate decay every 50 epochs.

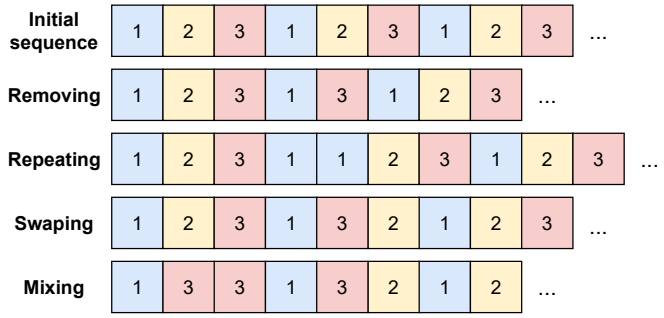


Fig. 4: **Our synthetic data generation technique produces noisy periodic sequences with k = 3.** We randomly perform removing, repeating, swaping some indexes, or mixing those operations on the initial sequence to create a synthetic noisy sequence.

3.3 Intuition of our approach.

Given the ability to measure the discrepancy between two sequences of different lengths, the CTC loss has been widely used for training sequences labeling tasks like speech recognition [13, 18] or optical character recognition [32]. Utilizing the CTC for counting human action in a few-shot manner is the key technical novelty of our network architecture and it is developed based on our intuition about the characteristics of human action.

Human actions can be performed at different speeds and vary in form, resulting in a difference in the pace of repetition or the temporal "shape" of the movement. By adopting the action counting to the sequence labeling task and leveraging the alignment-free characteristic of CTC, we can just assume the order of the phases, the length of each phase is neither determined nor important. The wide variation in arm posture is also a challenge. The less familiar the action to the subject, the more level of variability. This variability is even more challenging between different subjects. Other than large-scale training data collection with annotation, which is time- and cost-consuming, CaRaCount

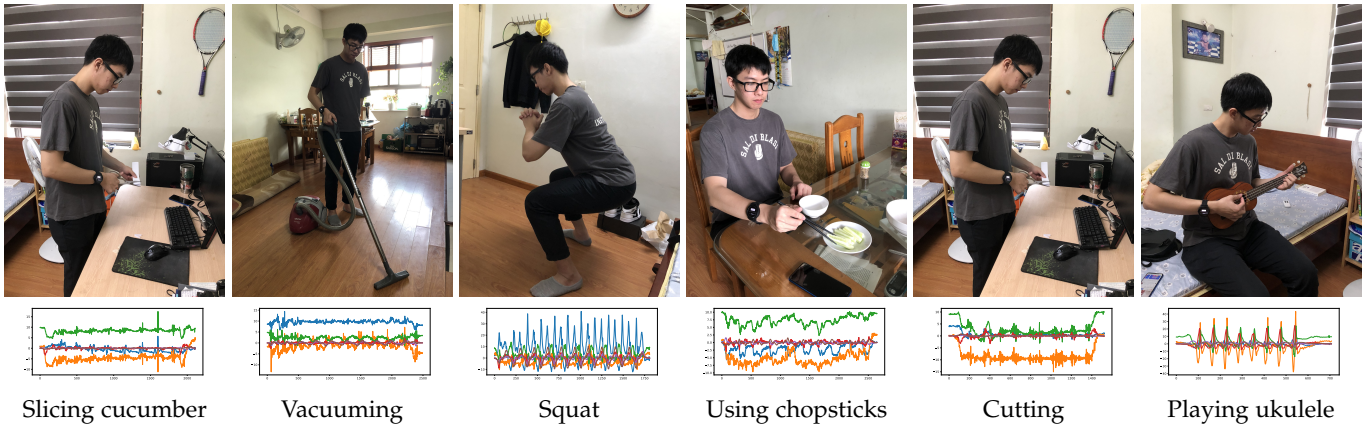


Fig. 5: **Examples of activities and the associated wearable data from the proposed CaRa dataset**. The number of repetitions in each sample ranges widely, from a few to hundred repetitions.

can address the problem of variation by creating a few-shot counter customized for each action of the subject.

In the real-world scenario, besides the action of interest, data collected from the wearable devices might include arbitrary actions as well. To distinguish the target action from these irrelevant actions, the periodic nature of the action can be utilized. However, this leads to a challenge, some actions like walking or dynamic stretching, though not our desired action, are extremely periodic. Additionally, the analysis of repetitive actions faces the hurdle of temporal irregularities. These challenges are illustrated in Fig. 3. The first two columns illustrate the difficulties posed by variations in form and the periodicity of actions that are not relevant. The action squat, for example, varies significantly when executed by different individuals due to differences in movement, speed, or the orientation of the device. Furthermore, the periodic nature of irrelevant actions in these scenarios complicates the task to the extent that traditional methods such as peak detection, threshold-crossing, or Fourier analysis are incapable of producing a reasonable counting result without recognizing the action first. The last column represents a very challenging case in which the action (rolling flour) is extremely irregular in duration and form. To address these challenges, [22, 33, 35] classify each window to distinguish the action of interest from irrelevant actions and then utilize a traditional approach to count the number of repetitions but large-scale training data is required for the generalization of the model. In contrast, our proposed approach is motivated by the fact that we do not have to precisely detect and segment the repetitions to count them. We can simply count the number of internal transitions between the phases of each repetition. For example, to count the number of walk cycles, we do not have to detect them. We only need to count the number of transitions from a left strike to a right strike. Furthermore, the difference between the action of interest and the irrelevant actions or the difference among phases of each repetition can be distinguished through the training process on the exemplar sequence.

4 CaRa Dataset

Our goal is to develop a class-agnostic repetitive action counting method for wearabe devices that works for many action categories. Unfortunately, existing sensor-based datasets are not suitable for our purpose because they are either contain small number of action classes [33, 41] or action repetitions per sample [35]. Moreover, those datasets only contain actions that belong to the physical exercise category, and the repetition count also lacks variety. This prompted us to collect and annotate a dataset ourselves. Our dataset consists of 50 activity classes from six superclasses: kitchen activities, household chores, physical exercises, factory activities, daily routines, and instrument-related activities. The number of repetition counts in our dataset varies widely, from 1 to 150 repetitions, with an average of 21.03 repetitions per sample. The difference between CaRa and other counting datasets is reported in Table 1. The categories as well as the statistics are shown in Fig. 6. In the sections that follow, we will provide a detailed explanation of the data collection process, data statistics, and how the data was split into disjoint sets.

4.1 Hardware sensors and data logging

Signal data was collected from a Ticwatch E2 worn on the wrist of the subject. The smartwatch had several sensors, but only the accelerometer and gyroscope sensors were relevant for this action counting task. We developed a smartwatch app to collect and synchronize the signal data from a 3-axis accelerometer and a 3-axis gyroscope. The accelerometer measured the acceleration (including gravity) along the x, y, and z axes in m/s^2. The axes were defined in relation to the device, where the z-axis was perpendicular to the screen. The gyroscope measured the rotational velocity, in rad/s, around the x, y, and z axes, where the coordinate system was the same as for the accelerometer. During the data collection process, the data was first saved on the smartwatch at the sampling rate of 100Hz and later transferred out via the internet to a computation server for further processing.

4.2 Data Collection

The collected data comprises of 50 activity classes from six superclasses. A total of ten subjects aged between 18 and 25

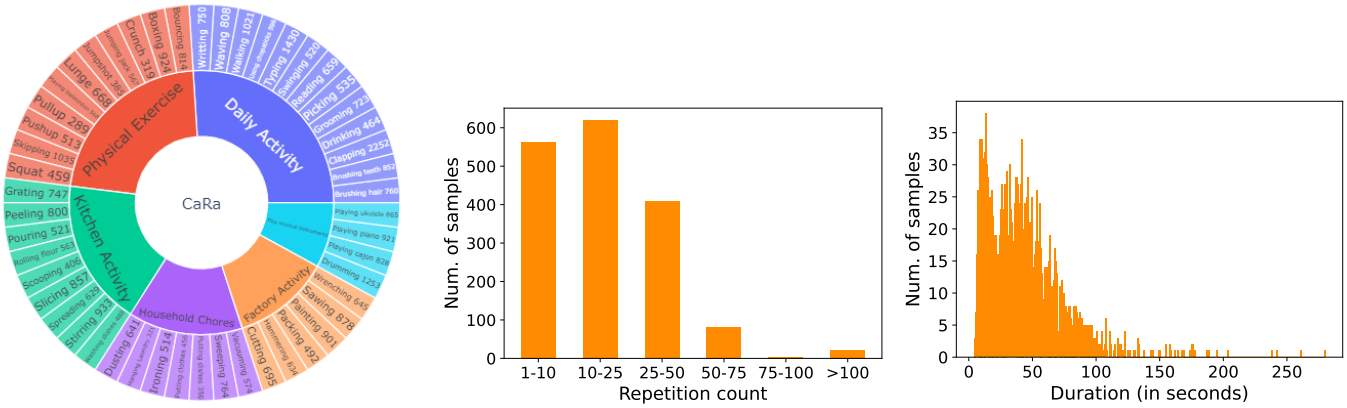


Fig. 6: **CaRa dataset's statistics.** The left figure displays the action categories and the proportion of samples for each category in CaRa. The two figures on the right display the quantity of samples across different ranges of repetition counts and durations.

who signed the consent form were asked to wear Ticwatch E2 on their dominant hand. The subjects were given a list of activities, which they can perform in any order along with instructions that included an illustration and a short textual description of each exercise. Fig. 5 shows examples of six activities performed by a subject along with the collected data. For diversity, a wide range of repetitions was suggested for each activity, and subjects were not required to complete any specific number of repetitions. All subjects were asked to perform the activities as naturally as possible without any further instruction from the experimenters.

The data collection process for a subject and an activity consisted of two sessions. In the first session, the subject was asked to perform the action with a fixed number of repetitions continuously. In the second session, the subject was asked to perform the action with any number of repetitions in a suggested range. The subject did not have to perform all the repetitions at once; in fact, they were instructed to take a break or do other actions in between the contiguous blocks of repetitions. The subject was asked to perform the actions consistently between the two sessions. Session 1 will later be used as the exemplar sequence, and Session 2 will be used as the query sequence. We manually inspected the collected data and annotated each sample with the number of repetitions contained in the sample.

4.3 Dataset Statistics

The CaRa dataset contains 1702 sequences, with a total of 35, 787 repetitions. The number of sequences collected in Session 1 and Session 2 are 494 and 1199, respectively. The number of repetitions in Session 1 and Session 2 are 3780 and 32007. The average number of repetitions per sample is 21.03. The minimum and the maximum number of repetitions for one sample are 1 and 150, respectively. The three activities with the highest total number of repetitions count are: Drumming (1253), Typing (1430), and Clapping (2252). The three activities with smallest numbers of repetitions count are: Pulling up (289), Crunching (319), and Hanging Laundry (321).

	Number of				
Dataset	Superclasses	Classes	Samples	Reps	Max Reps
Crossfit [33]	1	10	603	5470	31
Recofit [22]	1	26	1348	28807	100
MM-Fit [35]	1	10	616	6160	14
CaRa	**6**	**50**	**1702**	**35787**	**150**

TABLE 1: **Comparison with other action counting datasets.**

4.4 Dataset split

Given that our few-shot learning-then-inferring scenario requires only the exemplars for learning before directly inferring on the query, we deviate from the conventional practice of dividing the dataset into training and testing sets. Rather, we have randomly chosen 35 action classes to tune the hyper-parameters, referred to as the tuning set, while the remaining 15 action classes are allocated for the test set, on which we will present our experimental results.

5 Experiments

This section describes experiments on the proposed CaRa dataset and three other activity counting datasets. We compare the performance of CaRaCount and several other counting methods and observe that CaRaCount clearly outperforms all other methods. Specifically, we consider four simple baselines and implement the following algorithms: threshold crossing [41], peak detection [22], deep learning repetition counter [33] and adaption of video-based repetition counter for signal data [8, 14]. We also present a set of ablation studies in this section.

5.1 Existing activity counting datasets

Crossfit [33] is a dataset collected from 54 subjects. Their data collection process was divided into two sessions. The first session was designed to give an accurate label for the start of each repetition by a smartwatch vibration signal. This reduced the data annotation effort needed, but also significantly decreased the natural execution speed of the

Methods	CARA		Crossfit		Recofit		MM-Fit	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Mean	18.51	24.15	7.39	7.50	11.38	12.53	4.90	4.92
Median	15.74	26.37	7.03	7.15	11.36	12.52	4.90	4.94
Average time	13.25	20.56	2.16	2.62	1.87	3.23	4.01	5.98
Frequency	13.73	21.06	3.05	3.44	3.07	4.97	5.77	7.74
Threshold crossing [41]	15.76	21.60	2.35	3.69	2.65	5.08	4.97	7.54
Peak detection [22, 35]	10.12	19.04	2.67	3.42	3.09	5.67	1.33	2.62
Soro *et al.*[33]	10.86	18.22	0.55	1.19	2.23	3.77	-	-
Basic transformer	10.70	12.80	0.89	1.04	5.60	7.46	0.99	1.27
RepNet [8]	10.37	12.46	0.63	0.79	5.38	7.20	0.34	0.76
TransRAC [14]	8.42	11.53	0.71	0.85	6.54	8.64	0.39	0.77
CaRaCount (ours)	**6.26**	**9.78**	**0.13**	**0.42**	**1.48**	**2.97**	**0.32**	**0.68**

TABLE 2: **Performance of various counting methods** on four datasets: CARA, Crossfit, Recofit, and MM-Fit. CaRaCount outperforms the others on all datasets.

exercises. The second session, collected data for evaluation following two schemes, a 10-rep scheme and a 1-2-3 rep scheme.

Recofit [22] is a dataset that used an armband worn on the right forearm, containing a 3-axis accelerometer and a 3-axis gyroscope. They collected data for 26 exercises plus walking and stretching from 94 subjects with a sampling rate of 50Hz.

MM-Fit [35] is a collection of inertial sensor data from smartphones, smartwatches and earbuds worn by 10 participants while performing 10 full-body workout. The collected data was then time-synchronized with a multi-viewpoint RGB-D video, with 2D and 3D pose estimates. The sampling rate varies between devices, ranging from 30Hz to 500Hz.

5.2 Methods for comparison

Simple baselines. We consider four simple baseline methods: (1) always predicts a constant number, which is the mean repetition count, (2) always outputs the median repetition count, (3) estimating the average time to perform a single rep from the exemplar sequence and then utilizing it to compute the final count in the query sequence, and (4) estimating the dominant frequency of the action of interest from the exemplar sequence and then utilizing it to compute the final count in the query sequence. The mean and median counts are calculated from all the exemplar sequences.

Threshold crossing. Zelman *et al.*[41] set a threshold line positioned at two-thirds of the range between the minimum and maximum values. When the signal goes from below to above the threshold line, the count will be increased by one. After adding each count, there will be a refractory period of 0.1 seconds to prevent double counting.

Peak detection. Morris *et al.*[22] and David *et al.*[35] use similar approaches to tackle the counting task. They first computed a set of candidate peaks from the signal and then utilized autocorrelation as well as amplitude statistics to eliminate unqualified candidates.

Deep learning repetition counter. Soro *et al.*[33] proposed a CNN-based repetition counting network evaluated on their self-collected Crossfit dataset. For this method, we have to specify a different set of hyperparameters for each activity, which is very time-consuming. Theoretically, the Crossfit method cannot work on other datasets besides the Crossfit itself as it requires annotation for the beginning of each repetition to train the model. To study the performance of Crossfit on the CaRa and Recofit datasets, we provide additional manual annotation. Even with this unfair advantages, Crossfit is still outperformed by our method, as will be seen in the result section below. Furthermore, to address the lack of data when fine-tuning, we adapt the augmentation techniques inspired by [37].

Super-classes	Predictions		Ground truth	
	MAE	RMSE	Mean	IQR
Household chores	3.88	5.72	19.60	10
Physical exercises	4.92	7.69	22.07	14
Kitchen activities	7.63	10.57	24.51	15
Factory activities	6.34	9.37	26.21	18
Daily routines	9.98	16.54	32.12	20
Instrument-related	10.56	15.74	33.40	30

TABLE 3: **Counting results of CaRaCount for each action superclass within the CaRa dataset. The Interquartile Range (IQR), calculated as the difference between the 75th and 25th percentiles, represents the spread of the ground truth repetition numbers. A greater spread makes predicting the correct count more challenging, explaining the higher errors observed in certain superclasses.**

Methods	Parameters	MACs	Inference Time
Soro *et al.* [33]	36.93M	11.34G	121.89ms
Basic transformer	6.56M	2.68G	8.11ms
RepNet [8]	26.34M	5.95G	11.64ms
TransRAC [14]	26.53M	5.97G	10.28ms
CaRaCount (proposed)	3.81M	0.41G	1.96ms

TABLE 4: **Number of trainable parameters, MACs, and inference time of various counting methods.** The inference time is measured on a GPU using an NVIDIA GTX 4060, averaged over 100 runs while processing a 30-second length sequence. For the CaRaCount method, the inference time on a CPU is 17.79 ms.

Number of Phases	MAE	RMSE
2	10.79	15.54
3	9.01	13.62
4	6.26	9.78
5	6.43	9.61

TABLE 5: **Performance of CaRaCount** on the validation set with varying the number of phase assignment settings.

Adaption of video-based repetition counters. We also compare our methods with some state-of-the-art visual repetition counters, namely basic transformer, RepNet [8], and TransRAC [14]. Given the difference between video and sensor signals, we needed to make some modifications. For the basic transformer, we use a transformer encoder as the feature extractor, a 1D adaptative pooling layer to output a fixed-length feature vector, and an MLP to regress the count. For RepNet, instead of ResNet 50, we use a one-layer transformer encoder as the feature extractor, then calculate the temporal similarity map in the same way. Since we don't have the label of the time, we directly regress the final count. For TransRAC, we use a multi-size sliding window to resample the signal and then a transformer to extract the feature and regress the count with an MLP regression head. Again, given the absence of a temporal density map, we directly regress the count.

5.3 Quantitative results

To measure the performance, we use Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE), which are commonly used to evaluate counting methods [27, 29], and they are defined as follow. $MAE = \frac{1}{n}\sum_{i=1}^{n}|c_i - \hat{c}_i|$; $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(c_i - \hat{c}_i)^2}$, where n is the number of query sequences, and c_i and $\hat{c}_i$ are the ground truth and predicted counts. Table 2 reports the performance of all methods on four datasets. As can be seen, CaRaCount outperforms all the other methods on all datasets. Table 3 presents the counting errors and the average along with the IQR ground truth counts for each of the six superclasses within the CaRa dataset. The findings indicate that CaRaCount is proficient at counting repetitions across a wide range of action classes, including those not included in the tuning set, thereby highlighting the effectiveness of our method.

To further assess the efficiency of our method, we present the number of trainable parameters and Multiply-Accumulate Operations (MACs) for various learning models in Table 4. MACs, a widely used metric for evaluating efficiency, quantify the total number of multiplication and addition operations in a neural network. Additionally, we compare inference times across models, highlighting the exceptional efficiency of CaraCount. The inference time is calculated as the average of 100 runs when processing a sequence of 30 seconds in length. Notably, the results on CPU demonstrate its practicality for real-world applications, particularly when deployed on wearable devices with limited computational resources.

Components	Combinations			
Window length estimation	✗	✓	✓	✓
Count regression module	✗	✗	✓	✓
Label zero-phase addition	✗	✗	✗	✓
MAE	13.43	11.62	8.65	6.26
RMSE	18.40	14.81	12.33	9.78

TABLE 6: **Analyzing the components of CaRaCount.**

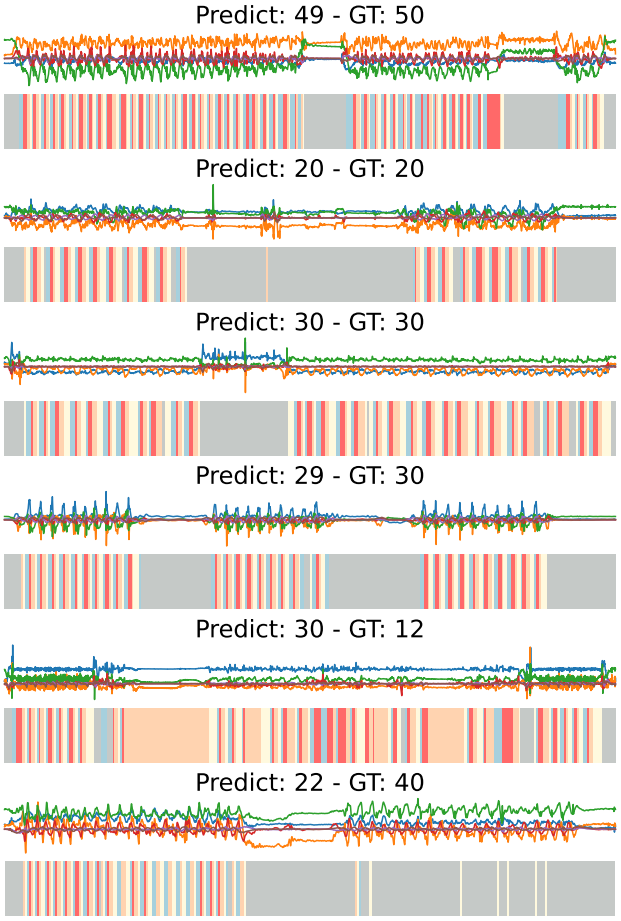


Fig. 7: **Predicted sequence labels and counts of CaRaCount.**

5.4 Ablation Studies

We conduct several ablation studies on the CaRa dataset to analyze: (1) how the counting performance changes as the number of exemplars increases, and (2) the benefits of different components of CaRaCount.

In Table 5, we analyze the performance of CaRaCount as k, the parameter for the number of phases is varied between two to five. As can be seen, using higher settings for k, four or five, yields better performance than lower settings, two or three. In general, the higher the number of phases, the more context and information the model can capture, hence making better predictions. However, there is no clear benefit of using $k = 5$ over $k = 4$, and the former would be less efficient. Therefore, we propose to use $k = 4$.

In Table 6, we analyze the importance of the key components of CaRaCount: the window length estimation

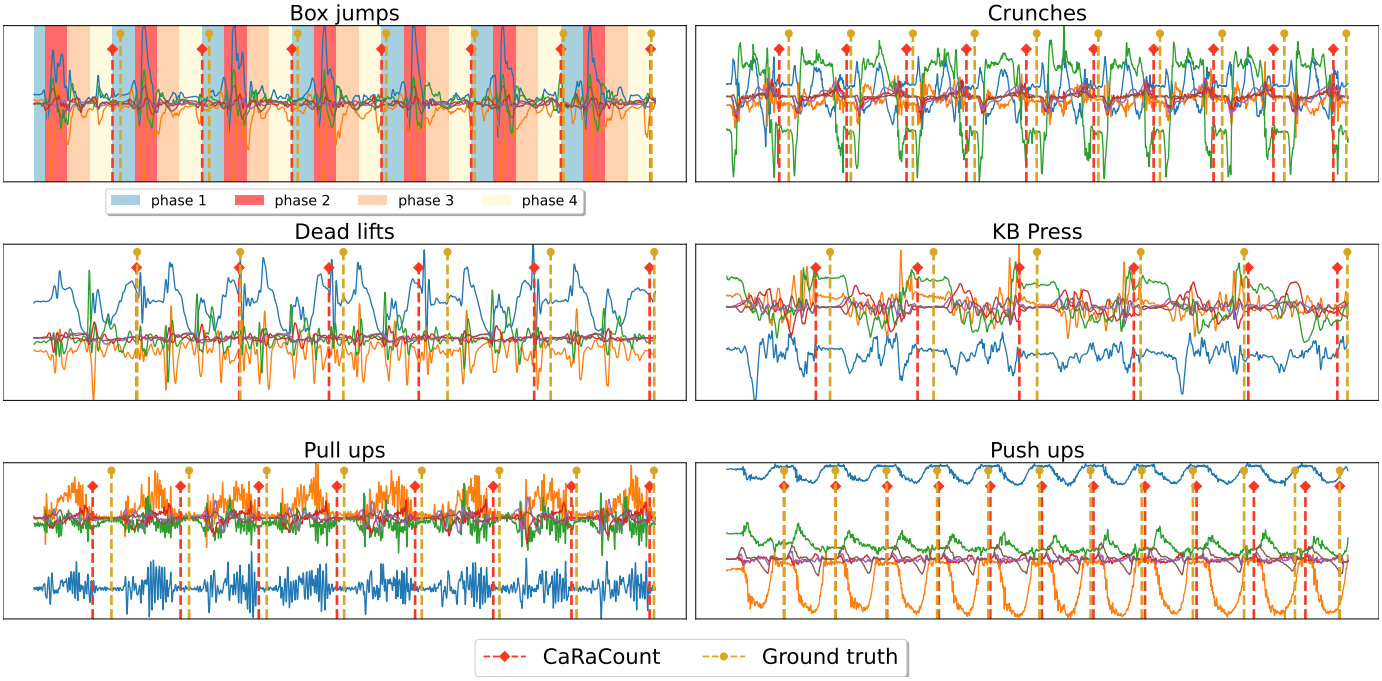


Fig. 8: **The results of the Phase assignment module are used as a breakpoint detector across various activities in the Crossfit dataset.** Ground truths are represented by gold circles, while detected breakpoints using our approach are marked by red diamonds. Only the sequence label result from the first sample is displayed for clarity.

process, the count regression module, and the zero-phase addition on the ground truth. We train models without a few or all of these components and notice that all of the components of CaRaCount are important, and adding each of the components leads to improved results. For the setting without window length estimation, the window size is fixed to 50ms with a 25ms overlapping ratio. This window size is estimated by diving the average repetition duration of the exemplar in the dataset by the number of phases, where the number of phases is set by 4 in our ablation studies. For the setting without the Count regression module, CaRaCount estimates the number of repetitions by counting the number of fully repeated blocks. And for setting without the zero-phase addition on ground truth, the target of exemplar sequence is only r blocks of k phases: $[P_1, P_2, \ldots, P_k], \ldots, [P_1, P_2, \ldots, P_k]$.

5.5 Qualitative Results

Fig. 7 shows a few query sequences and the CaRaCount predictions. CaRaCount successfully distinguishes between background and action of interest in the first four cases and classifies windows of action into their corresponding phases. Some background windows were misclassified, but thanks to the generalization of the count regression module, our model did not recognize those false negative windows as a repetition. In the fifth case, CaRaCount mistakes the background for the action of interest because of the similarity in appearance between them. In the sixth case, the action of interest on the right differs from the exemplar (or the action of interest on the left), so the phase assignment module is unable to identify the target action.

Fig. 8 further illustrates the capability of the Phase Assignment module as a reliable breakpoint detector. These breakpoints are visualized as transitions from the end of Phase 4 to the beginning of Phase 1, emphasizing the module's precision in capturing critical activity shifts. Notably, CaRaCount effectively identifies these transitions without the need for individual annotations, leveraging the alignment-free nature of the CTC framework.

Action	MAE	RMSE	# of Users
Squat	0.95	1.39	20
Snapping finger	0.66	1.15	3
Pirouetting	2.00	2.00	1
Jumping jack	0.40	0.63	5
Using hammer	1.00	1.00	1
Clapping	1.25	1.50	4
Jogging	0.50	0.70	2
Situp	0.00	0.00	1
Battle rope training	2.00	2.00	1
Bottle shaking	2.00	2.00	1
Sliding fruit	0.66	1.15	3
Arm swing	0.25	0.50	4
Front raise	0.50	0.70	2
Hook punch	0.33	0.57	3
Thanks in ASL	1.00	1.00	1

TABLE 7: **Counting results in the real-time user experience study.**

6 User Study

Based on our framework, we implemented a real-time system and evaluate it via a user study. Fig. 9 shows the real-time system watch interface design for the application. The system has two working modes, few-shot exemplar training,

Fig. 9: **Real-Time Action counting System Watch Interface Design.** From the main screen, the subject can choose to provide an example sequence to create a new counter or use the existing counter to count the number of repetitions.

and query inference. In the first mode, the participants provide an exemplar sequence along with an exemplar count. CaRaCount will process these inputs to create a new counter. In the second mode, the participants can use existed counter to count the action of interest. We evaluate the real-time counting performance as well as the system's usability with both objective results and subjective feedback from the participants.

6.1 Participants

We recruited 20 participants, 12 men and 8 women. Participants were on average 23.6 years old. 5 participants are left-handed. Each participant was compensated with a gift card for their participation in the 25-minute-long study. We employ the Ticwatch S2 for the user study. All participants wore the smartwatch in their dominant hands. Data from the watch is streamed to an ASUS laptop. The laptop did the model training and repetition counting and sent the results back to the watch for real-time interaction.

6.2 Study Procedure

After a brief introduction to the framework and the interface, the participants went through the following stages:

1) Participants first tried to perform 5 repetitions of Squat for the system as an exemplar sequence. After processing, participants can turn on the inference mode and use the squat counter to count the number of squat repetitions. The purpose of this step is to familiarize the user with the system.
2) Participants were asked to create one or more counters of any action of interest they have in mind by providing the exemplar sequence and exemplar count to the system and test them in the inference mode.
3) Participants were told to use the system freely till the end the session, after which they flled in the System Usability Scale (SUS) questionnaire [3].

During the study, participants could ask the experimenter anytime if they have any questions.

6.3 Results

Following other works evaluated the system on smartwatch [19, 40], we demonstrate the user study result with both objective results and subjective feedback.

Objective Result. The real-time performance of the system is equivalent to the offline results reported in Section 5. The average counting MAE and RMSE from the participants are 0.80 and 1.21. The actions chosen by participants in the study were diverse, including actions such as snapping fingers, clapping, jogging, and jumping jack. Table 7 summarizes the counting results in the real-time user experience study. For each of the actions, participants provide an exemplar sequence with the exemplar count ranging from 3 to 5. MAE and RMSE indicate the counting error of each action while # of Users indicates the number of participants who chose to perform that action in the user study. The results indicate the usability and robustness as well as the class-agnostic capability of our framework.

Subjective Feedback. In addition to the objective results, the user study also suggests positive feedback from participants. Our system receives the average overall SUS score of 81.6$\pm$7.4 out of 100, which indicates the high overall

usability of CaRaCount. Especially for question 4th, 'I think I don't need the support of a technical person to be able to use this systems, the score was 92.3±4.1 indicating that the system is really easy to use. The SUS questionnaires' results demonstrate the good usability of our real-time system.

7 Conclusions and Future Work

In this paper, we posed activity counting as a few-shot learning-then-inferring task. Given the non-existence of a proper dataset for the few-shot activity counting task, we collected a large-scale sensor-based activity counting dataset CaRa with a large number of activities and repetitions. We have presented CaRaCount, an effective matching strategy suitable for the few-shot action counting task. This method successfully counts the number of repetitions of the query sequence given only an exemplar sequence. The proposed method is evaluated and compared with several baselines and state-of-the-art deep learning-based activity counting and achieved better results in all of the existing as well as our newly proposed datasets. These results opened the opportunity to investigate counting in an interactive way for devices with different functionalities, intentions, and platforms.

There are some directions for future work. One potential approach involves expanding this study to not only count repetitions but also recognize the action of interest in a few-shot context. Additionally, exploring a multimodal approach that incorporates data from vision and sound, in addition to accelerometer and gyroscope readings from a wrist-worn device, represents another promising direction.

References

[1] Shahira Abousamra, Minh Hoai, Dimitris Samaras, and Chao Chen. Localization in the crowd with topological constraints. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2021.

[2] Ousman Azy and Narendra Ahuja. Segmentation of periodically moving objects. In *Proceedings of the International Conference on Pattern Recognition*, 2008.

[3] Aaron Bangor, Philip T. Kortum, and James T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human–Computer Interaction*, 24:574 – 594, 2008.

[4] Gino Brunner, Darya Melnyk, Birkir Sigfússon, and Roger Wattenhofer. Swimming style recognition and lap counting using a smartwatch and deep learning. In *ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2019.

[5] Keng-hao Chang, Mike Y. Chen, and John F. Canny. Tracking free-weight exercises. In *ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2007.

[6] Dmitry Chetverikov and Sándor Fazekas. On motion periodicity of dynamic textures. In *Proceedings of the British Machine Vision Conference*, 2006.

[7] R. Cutler and L.S. Davis. Robust real-time periodic motion detection, analysis, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):781–796, 2000.

[8] Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. Counting out time: Class agnostic video repetition counting in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[9] Vincenzo Genovese, Andrea Mannini, and Angelo M. Sabatini. A smartwatch step counter for slow and intermittent ambulation. *IEEE Access*, 5:13028–13037, 2017.

[10] Mohsen Gholami, Christopher Napier, Astrid Garc´ia Patiño, Tyler J. Cuthbert, and Carlo Menon. Fatigue monitoring in running using flexible textile wearable sensors. *Sensors*, 20(19):5573, 2020.

[11] Alex Graves, Santiago Fernandez, and Faustino Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, 2006.

[12] Tian Hao, Guoliang Xing, and Gang Zhou. Runbuddy: a smartphone system for running rhythm monitoring. In *ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 133–144, 2015.

[13] Takaaki Hori, Shinji Watanabe, and John R. Hershey. Joint ctc/attention decoding for end-to-end speech recognition. In *Annual Meeting of the Association for Computational Linguistics*, 2017.

[14] Huazhang Hu, Sixun Dong, Yiqun Zhao, Dongze Lian, Zhengxin Li, and Shenghua Gao. Transrac: Encoding multi-scale temporal correlation with transformers for repetitive action counting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[15] Yifeng Huang, Viresh Ranjan, and Minh Hoai. Interactive class-agnostic object counting. In *Proceedings of the International Conference on Computer Vision*, 2023.

[16] Yifeng Huang, Duc Duy Nguyen, Lam Nguyen, Cuong Pham, and Minh Hoai. Count what you want: Exemplar identification and few-shot counting of human actions in the wild. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2024.

[17] Rushil Khurana, Karan Ahuja, Zac Yu, Jennifer Mankoff, Chris Harrison, and Mayank Goel. Gymcam. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2:1 – 17, 2018.

[18] Suyoun Kim, Takaaki Hori, and Shinji Watanabe. Joint ctc-attention based end-to-end speech recognition using multi-task learning. *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017.

[19] Young-Ho Kim, Diana Chou, Bongshin Lee, Margaret Danilovich, Amanda Lazar, David E Conroy, Hernisa Kacorri, and Eun Kyoung Choe. Mymove: Facilitating older adults to collect in-situ activity labels on a smartwatch with speech. In *ACM Conference on Human Factors in Computing Systems*, 2022.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning and Representation*, 2015.

[21] Ofir Levy and Lior Wolf. Live repetition counting. In *Proceedings of the International Conference on Computer Vision*, 2015.

[22] Dan Morris, T. Scott Saponas, Andrew Guillory, and Ilya Kelner. Recofit: using a wearable sensor to find, recognize, and count repetitive exercises. In *ACM Conference on Human Factors in Computing Systems*, 2014.

[23] Thanh Nguyen, Chau Pham, Khoi Nguyen, and Minh Hoai. Few-shot object counting and detection. In *Proceedings of the European Conference on Computer Vision*, 2022.

[24] E. Pogalin, A.W.M. Smeulders, and A.H.C. Thean. Visual quasi-periodicity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[25] Anshul Rai, Krishna Chintalapudi, Venkata N. Padmanabhan, and Rijurekha Sen. Zee: zero-effort crowdsourcing for indoor localization. In *Proceedings of the International Conference on Mobile computing and networking*, 2012.

[26] Viresh Ranjan and Minh Hoai. Exemplar free class agnostic counting. In *Proceedings of the Asian Conference on Computer Vision*, 2022.

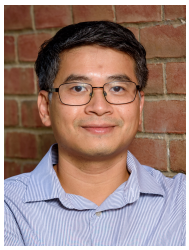
[27] Viresh Ranjan, Hieu Le, and Minh Hoai. Iterative crowd counting. In *Proceedings of the European Conference on Computer Vision*, 2018.
[28] Viresh Ranjan, Boyu Wang, Mubarak Shah, and Minh Hoai. Uncertainty estimation and sample selection for crowd counting. In *Proceedings of the Asian Conference on Computer Vision*, 2020.
[29] Viresh Ranjan, Udbhav Sharma, Thu Nguyen, and Minh Hoai. Learning to count everything. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.
[30] Tom F. H. Runia, Cees G. M. Snoek, and Arnold W. M. Smeulders. Real-world repetition estimation by div, grad and curl. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
[31] Christian Seeger, Alejandro P. Buchmann, and Kristof Van Laerhoven. myhealthassistant: a phone-based body sensor network that captures the wearer's exercises throughout the day. In *Proceedings of the International Conference on Body Area Networks*, 2011.
[32] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39: 2298–2304, 2017.
[33] Andrea Soro, Gino Brunner, Simon Tanner, and Roger Wattenhofer. Recognition and repetition counting for complex physical exercises with deep learning. *Sensors*, 19(3):714, 2019.
[34] Christina Strohrmann, Holger Harms, Cornelia Kappeler-Setz, and Gerhard Tröster. Monitoring kinematic changes with fatigue in running using body-worn sensors. *IEEE Transactions on Information Technology in Biomedicine*, 16: 983–990, 2012.
[35] David Strömbäck, Sangxia Huang, and Valentin Radu. Mm-fit: Multimodal deep learning for automatic exercise logging across sensing devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4 (4), 2020.
[36] Ashwin Thangali and Stan Sclaroff. Periodic motion detection and estimation via space-time sampling. In *Proceedings of the IEEE Workshop on Applications of Computer Vision*, 2005.
[37] Terry T. Um, Franz M. J. Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić. Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. In *Proceedings of the ACM International Conference on Multimodal Interaction*, 2017.
[38] Qingxin Xia, Atsushi Wada, Joseph Korpela, Takuya Maekawa, and Yasuo Namioka. Unsupervised factory activity recognition with wearable sensors using process instruction information. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 3:1 – 23, 2019.
[39] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the International Conference on World Wide Web*, 2017.
[40] Hui-Shyong Yeo, Wenxin Feng, and Michael Xuelin Huang. Watouch: Enabling direct input on non-touchscreen using smartwatch's photoplethysmogram and imu sensor fusion. In *ACM Conference on Human Factors in Computing Systems*, 2020.
[41] Samuel Zelman, Michael McD. Dow, Thasina Tabashum, Ting Xiao, and Mark V. Albert. Accelerometer-based automated counting of ten exercises without exercise-specific training or tuning. *Journal of Healthcare Engineering*, 2020.
[42] Huaidong Zhang, Xuemiao Xu, Guoqiang Han, and Shengfeng He. Context-aware and scale-insensitive temporal repetition counting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Duc Duy Nguyen is a Ph.D. student at the Australian Institute for Machine Learning at the University of Adelaide. His research interests lie in human behavior and motion analysis, with a focus on multimodal approaches to understanding and modeling complex human actions.

Lam Thanh Nguyen is an AI engineer at VinAI. His research focuses on Computer Vision and Machine Learning.

Yifeng Huang is a Ph.D. candidate at Stonybrook University. His research focuses on visual counting.

Cuong Pham received a PhD in Computer Science at Newcastle University in 2012. He is an Associate Professor of Computer Science at Posts & Telecommunications Institute of Technology and a Visiting Research Scientist at VinAI. His research interests include ubiquitous computing, wearable computing, human activity recognition, computer vision, and pervasive healthcare.



Minh Hoai is a Professor and Deputy Director of the Australian Institute for Machine Learning at the University of Adelaide, as well as a consulting research scientist at VinAI. Previously, he was an Associate Professor of Computer Science at Stony Brook University. He received a Bachelor of Software Engineering from the University of New South Wales in 2005 and a Ph.D. in Robotics from Carnegie Mellon University in 2012. His research interests include computer vision and machine learning, with a particular focus on human action and activity recognition and prediction.