This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Souza Leite, Clayton; Xiao, Yu

Improving Resource Efficiency of Deep Activity Recognition via Redundancy Reduction

Published in:
HotMobile 2020 - Proceedings of the 21st International Workshop on Mobile Computing Systems and Applications

DOI:
[10.1145/3376897.3377859](https://doi.org/10.1145/3376897.3377859)

Published: 03/03/2020

Document Version
Peer reviewed version

Please cite the original version:
Souza Leite, C., & Xiao, Y. (2020). Improving Resource Efficiency of Deep Activity Recognition via Redundancy Reduction. In *HotMobile 2020 - Proceedings of the 21st International Workshop on Mobile Computing Systems and Applications* (pp. 33-38). ACM. <https://doi.org/10.1145/3376897.3377859>

Improving Resource Efficiency of Deep Activity Recognition via Redundancy Reduction

Clayton Frederick Souza Leite
Aalto University
Espoo, Finland
clayton.souzaleite@aalto.fi

Yu Xiao
Aalto University
Espoo, Finland
yu.xiao@aalto.fi

Abstract

Compression methods for deep learning have been recently used to port deep neural networks into resource-constrained devices - such as digital gloves and smartwatches - for human activity recognition (HAR). While the results have been in favor of utilizing compressed models, we envision that the current paradigm of long and fixed-size overlapping sliding windows that permeate the literature of HAR contributes negatively toward the goal of more resource-efficient systems, as it induces redundancies in memory and computation. In this work, we provide a different perspective by demonstrating that memory footprint, computational expense, and possibly energy consumption can be dramatically spared by modifying the architecture of the neural networks and their training. It is achieved by enabling non-overlapping short sliding windows and skipping fine-grained features in favor of rough ones on certain occasions, thus reducing the demand for more powerful hardware. Compared with the state-of-the-art, our method is able to achieve comparable performance far more efficiently in terms of resource use.

CCS Concepts

• Computing methodologies → Machine learning.

Keywords

Human activity recognition; resource-constrained devices; deep learning

ACM Reference Format:

Clayton Frederick Souza Leite and Yu Xiao. 2020. Improving Resource Efficiency of Deep Activity Recognition via Redundancy Reduction. In *Proceedings of the 21st International Workshop on Mobile Computing Systems and Applications (HotMobile '20)*, March 3–4, 2020, Austin, TX, USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3376897.3377859>

1 Introduction

Human activity recognition (HAR) is a key element in wearable applications such as fitness tracking, stroke rehabilitation, and employee training. It consists in inferring from sensor data the type of actions a person is executing. Currently, shallow machine learning algorithms are being overshadowed by deep learning (DL)

in HAR for two reasons. First, DL algorithms provide significant leaps in performance. Second, with feature learning, DL drops the requirement for hand-engineered features obtained from expert knowledge.

High performance and feature learning are attained by using several stacked layers with innumerable computations that gradually transform the raw data into the desired output. The number of parameters in the layers can reach beyond millions. Hence, DL models possess a high memory footprint, computation expense, and energy consumption. As an example, a simple CNN-LSTM model for HAR can require above 400KB of memory and 300MFLOPs of computation per prediction, which is beyond the specifications of popular off-the-shelf microcontrollers that are commonly used for wearable devices due to their small size and low cost.

The memory footprint of DL networks limits their suitability on memory-scarce appliances. Also, despite microcontrollers having reached higher computation capabilities over the past years, the execution time of deep models can still take long enough to bring delays - even in the order of seconds - to the end-user and harm the overall user experience. Furthermore, minimizing energy consumption is primordial to satisfy end-users, since devices with short autonomy can be seen as burdensome. Running the DL algorithms on the cloud is an alternative, but not always desired, by cause of privacy-related issues and additional overhead due to latency in communications. To overcome these issues, much research attention has been drawn to porting DL algorithms into wearable devices that are resource-constrained.

Removing redundancies is the essence in porting DL algorithms for HAR into devices of limited resources. Generic compression methods [2, 3, 5, 8, 11] have been extensively used to make deep models meet the memory, computation, and energy consumption requirements of small devices by reducing the redundancy in the parameters of the neural network. However, we envision that the current paradigm of HAR systems presents three additional overlooked redundancies that are yet to be addressed and, in combination with compression methods (such as weight pruning, quantization, and factorization), can potentially reach even higher levels of resource efficiency than the sole use of compression methods. These overlooked redundancies are presented as follows.

1) Overlapping sliding windows. The best-performing methods on various HAR benchmarks (as InnoHAR [13]) still utilize overlapping sliding windows. When training, segmenting the dataset with overlapping windows serves as data augmentation and concretely helps the neural network in generalizing better. However, when testing, having overlapping windows leads to computational redundancy since the same portion of data is processed more than once.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HotMobile '20, March 3–4, 2020, Austin, TX, USA

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-7116-2/20/03.

<https://doi.org/10.1145/3376897.3377859>

2) Long sliding windows. While long sliding windows provide a wider temporal context leading to more accurate predictions, they also significantly increase the number of parameters in certain types of layers (such as recurrent and fully-connected (FC) layers) in a neural network. This is memory redundancy. As we will see in Section 3, long sliding windows do not help in obtaining quicker predictions.

3) Consecutive predictions of the same activity. If the same activity is performed for a long period, the DL algorithm will be redundantly extracting the same features for the whole duration of the activity. Removing this computational redundancy can lead to an appreciable reduction of resource utilization.

In this paper, we investigate to what extent the aforementioned redundancies affect resource utilization and the potential improvement gained by addressing them with a novel method proposed by us. First, we drop the need for overlapping windows by a change in the training of the network, and we significantly shorten the sliding windows to minimize memory footprint and to detect activities as soon as possible. Second, we utilize rough features and skip fine-grained ones for generating predictions during the execution of long-lasting activities. With slight performance decrease, our solutions are able to achieve comparable performance with the state-of-the-art with significantly fewer network parameters and computational cost.

The rest of the paper is organized as follows. Section 2 introduces the related work of HAR. Section 3 presents the proposed methods. Section 4 describes the datasets, with the preliminary experimental results presented in Section 5 along with a discussion about them and future work. Section 6 presents the conclusion.

2 Related Work

HAR with DL. Ordonez and Roggen [10] combined convolutional and recurrent Long Short-Term Memory (LSTM) layers in a single network for performing HAR. Guan *et al.* [7] worked with ensembles of LSTM networks for HAR. They have shown that significant performance improvement (up to 10%) can be achieved using ensembles of recurrent networks instead of a single one. Long *et al.* [9] employed residual networks with convolutional and LSTM layers to capture spatial and fine temporal features, respectively. Their model reached higher performance compared to several simpler ones when tested on public datasets with activities like standing up, lying down, running, etc. Xu *et al.* [13] have achieved state-of-the-art performance on the Opportunity [4] and PAMAP2 [12] datasets by proposing a neural network with Inception-like convolutional layers followed by gated recurrent unit (GRU) layers. Overall, DL has been applied to HAR with an enormous variety of methods, with each of which delivering ever-increasing performance scores. However, these methods still utilize long and fixed-size overlapping sliding windows that render their implementation on resource-constrained devices difficult.

DL in resource-constrained devices. Lane *et al.* [8] created a toolkit that applies to any existing DNN or CNN model 5 different types of compression (i.e., sparse factorization, weight factorization, precision reduction, convolution separation, and parameter cleaning) generating a new compressed model with slightly lower

accuracy but significant improvements in memory footprint, execution time and energy consumption. To reduce execution times of deep neural networks on embedded devices, Yao *et al.* [14] developed a framework that spots layers that take longer to run and concentrates existing compression algorithms on them, instead of treating all layers equally. Their method provided shorter execution time and smaller energy consumption, compared to the state-of-the-art compression algorithms.

Alippi *et al.* [1] resorted to approximate computing techniques to reduce computational load and memory footprint of CNN networks on embedded devices. Fang *et al.* [6] developed a tool that continuously monitors resource availability in systems that run multiple applications concurrently and switches between different compressed versions of the same neural network depending on the resource scarcity.

Except for utilizing compressed networks, resource consumption can potentially be further reduced by removing the redundancies mentioned in Section 1, which has not been studied previously. Motivated by this, we draw our attention to these redundancies and propose ways to address them.

3 Methods

We propose two approaches: short non-overlapping sliding windows and rough features for predicting long-lasting activities. The former is used to address the first and second redundancies discussed in Section 1. The latter addresses the third one. These two approaches are independent of each other, hence, can be either used separately or together.

3.1 Short non-overlapping sliding windows

During the training phase, using overlapping windows serves as an effective data augmentation process, as it results in classification algorithms being less biased to the exact position where certain characteristics (e.g., peaks and valleys in the sensory readings) appear in the data. During the testing phase, they serve the neural network with a broader temporal context to predict the activity more accurately. However, this signifies that the same portion of data is processed more than once, which causes a waste of computational resources.

Recurrent layers as LSTM or GRU have been extensively used to capture temporal context in HAR. However, authors limited the training of these layers to grasp the temporal context only contained inside the sliding window, disregarding the temporal context existent across sliding windows. We call the former **intra-window temporal context** and the latter **inter-window temporal context**.

We propose an architecture composed of convolutional layers that learn spatial and intra-window temporal context, followed by an LSTM layer limited to grasping inter-window temporal context. To achieve this, we collapse the three-dimensional feature maps given by the last convolutional data to one dimension (instead of two dimensions). These feature maps go through the LSTM layer which is trained using backpropagation through the feature maps of M previous windows, where M is an integer hyper-parameter of the training. This means that the LSTM layer is trained to be able to recall information from previous windows, thus removing the

Figure 1: The approach of short non-overlapping sliding windows (on the right) compared to the literature's approach (on the left). x_k is the k -th sliding window since the beginning of the data stream. In our method, a prediction $\hat{y}_k$ is ignored if its confidence level is below a certain threshold. In both methods, for every sliding window, a prediction is given.

need for repeating previously seen information - i.e. removing the need for overlapping windows.

The need for using a long sliding window comes from the idea that having an appreciable amount of data is beneficial for producing more accurate predictions since the classifier is fed with a broader temporal context. Having established that our LSTM layer is required to learn inter-window temporal context, the need for long sliding windows is removed. We can shorten the window size and appreciably decrease the number of parameters in the LSTM and FC layers. Fig. 1 illustrates this approach compared to how it is currently done in the literature.

Shorter sliding windows also help in predicting new incoming activities quicker. Each sliding window can be labeled either with the activity that is present in the largest portion of the window or with that seen at the end of it. The former approach is more widely used. When using long windows, even if a new activity is already showing up at the end of the window, the neural networks will be trained to ignore it in favor of the one that still dominates the window. With this, the prediction of new incoming activities is delayed. The latter approach introduces noisy labels affecting the performance adversely since most of the information contained in the window is not related to the upcoming activity. The existence of noisy labels is more pronounced when using longer windows, as the odds of having more than one activity in a single window is higher.

Using shorter windows reduces the incidences of noisy labels and, also importantly, helps in our goal of having convolutional layers that learn the intra-window temporal context. This is true because convolutional layers rely on the concept of locality and have difficulties learning distant dependencies in the data.

With shorter sliding windows, predictions may be erroneous right at the moment of transition - that is, the beginning of a new activity - since the neural network may not have been presented with enough information to accurately recognize the upcoming activity. Therefore, we propose to disregard predictions with low confidence scores (which is calculated by taking the maximum value

Figure 2: Network architecture. During inference, the input x_{k+n} and, consequently, the prediction $y_k = y_{k+n}$ are not present.

in the vector output by the network). This signifies that prediction $y_{t,k}$ is ignored if $y_{t,k} < P_{th}$, with P_{th} being a hyper-parameter referred to as the confidence threshold. In a certain sense, this is equivalent to having a dynamic sliding window, since predictions may come at non-fixed intervals of time.

We utilize the usual cross-entropy loss for training the network. We emphasize that the training, validation and test sets don't possess overlapping windows. Also, the windows can be as short as 0.1 seconds.

3.2 Prediction with rough features

Denoting as x_k a sliding window at instant k and y_k its label, suppose an activity has been predicted with a high confidence score at instant k , the odds are high that subsequent windows contain the same activity. As the neural network continues to predict this same activity across successive sliding windows, the same features are redundantly extracted. This leads to a misuse of computation.

To address this issue, we propose an additional neural network of low-complexity referred to as the **secondary** network. During inference time, given a window x_k , the main network extracts its rough features $N_R(x_k)$ which are concatenated in the channel dimension with $N_R(x_{k-1})$. The result of the concatenation is passed through the secondary network that tells whether or not $y_k = y_{k-1}$. In case this is true and the confidence of this prediction is higher than a certain confidence threshold, fine-grained features and their subsequent classification are skipped. In the opposite case, the main network N follows its execution until the classification to obtain $\hat{y}_k$. Thus, the secondary network consists in a binary classifier that outputs 1 if $\hat{y}_k = \hat{y}_{k-1}$ or, otherwise, 0. We denote this network as a function $D(\cdot)$ and $N_R(x_k)$ as the rough features extracted by the main network N . One of the loss functions for the training of the secondary network is given in Eq. 1.

$$\mathcal{L}_{CECW} = -\frac{1}{B-1} \sum_{k=2}^B g_{k,k-1} \log(\hat{g}_{k,k-1}) - (1 - g_{k,k-1}) \log(1 - \hat{g}_{k,k-1}), \quad (1)$$

where B is the number of windows in the training set, $\hat{g}_{k,k-1} = D(N_R(x_k) \oplus N_R(x_{k-1}))$ (here the symbol $\oplus$ denotes concatenation of the 3D features along the channel dimension) and $g_{k,k-1} = 1$ if $y_k = y_{k-1}$ or $g_{k,k-1} = 0$ otherwise.

Training the secondary network only with $\mathcal{L}_{CECW}$ leads to a biased classifier, since having $g_{k,k-1} = 1$ is much more common

than the opposite. Therefore, to balance the training, we propose to minimize an additional loss function (Eq. 2).

$$\mathcal{L}_{\text{CERW}} = -\frac{1}{B} \sum_{k=1}^B g_{k,k+n} \log(\hat{g}_{k,k+n}) - (1 - g_{k,k+n}) \log(1 - \hat{g}_{k,k+n}), \quad (2)$$

where n is an arbitrary integer defined such that $y_{k+n} \neq y_k$ to balance the classification.

Naturally, the benefits come only when the secondary network predicts that the same activity is occurring since the need for performing more complex computations (as those in the LSTM layer) is avoided. Overall, since changes in activities occur with low frequency in HAR, the use of the secondary network helps to reduce the computational load as long as its costs are less than that of extracting more fine-grained features and classifying them. To help towards this goal, we design the secondary network with convolutions of short kernel sizes and a sole convolutional layer.

Finally, the entire training of the networks (depicted in Fig. 2) can be summarized by minimizing the loss function $\mathcal{L} = \mathcal{L}_{\text{CE}} + \mathcal{L}_{\text{CECW}} + \mathcal{L}_{\text{CERW}}$, where $\mathcal{L}_{\text{CE}}$ is the usual cross-entropy loss function.

4 Datasets

To evaluate our method, we utilize the PAMAP2 [12] and Opportunity [4] datasets. All samples in both datasets are normalized to zero mean and unit variance. We follow previous works in the selection of training, validation and test sets (detailed as follows) to provide means of easily comparing our proposal with theirs.

PAMAP2. The PAMAP2 dataset contains 18 different physical activities performed by 9 participants wearing a heart monitor and three IMUs attached to the chest, hand, and ankle, respectively. Out of the 18 activities, 6 are rarely present in the data. To avoid having a heavily imbalanced dataset, only the remaining 12 activities are considered in our experiments: lying quietly, sitting, standing, ironing, vacuum cleaning, ascending stairs, descending stairs, walking, Nordic walking, bicycling, running, and rope jumping. The mean duration is 1 minute for ascending and descending stairs, 2 minutes for rope jumping and 3 minutes for the remaining activities. Missing values on the data (NaN values) are discarded. Also, the sampling rate is decimated from 100Hz to 33.3Hz. The entire data from subject 5 and subject 6 were used, respectively, for the validation set and the testing set. This entire protocol is followed in diverse works, as in [7].

Opportunity. This dataset incorporates 18 kitchen-related activities (the mean duration of the activities is given in parentheses): cleaning a table (10s), opening/closing the fridge (3s), opening/closing the dishwasher (3s), opening/ closing 3 different drawers (2s), opening/closing 2 different doors (4s), toggling lights on and off (1s), and drinking from a standing and sitting position (10s). The data collection was carried out with 4 participants using 23 body-worn sensors. In our experiments, the sampling rate was reduced to 30Hz and only the sensory readings from the upper limbs, the back, and both feet were considered (as it was also done in [7]). Also following [7], the validation set is composed of session 1 from subject 2, whereas the testing set consists of sessions 2 and 3 from subjects 4 and 5.

Method	F1	NPA	NPR	TCE	PDNA
<i>PAMAP2 dataset</i>					
Baseline CNN-LSTM	0.876	83.65K	3141	33.77	6.16
CNN-LSTM SN	0.912	51.75K	7827	4.60	1.68
CNN-LSTM SNR	0.882	53.08K	7904	3.99	2.13
InnoHAR [13]	0.935	34.91M	3141	1887.58	-
<i>Opportunity dataset</i>					
Baseline CNN-LSTM	0.895	118.46K	7416	35.58	1.03
CNN-LSTM SN	0.959	84.91K	14178	10.07	0.58
CNN-LSTM SNR	0.893	87.06K	15341	9.06	0.61
InnoHAR [13]	0.946	6.20M	9890	873.54	-

Table 1: Experimental results. NPA, NPE and TCE for the InnoHAR method are approximations based on the architecture of the network.

5 Evaluation

5.1 Methods and hyper-parameters

We tested a baseline CNN-LSTM network and variants of our method: CNN-LSTM SN and CNN-LSTM SNR. The first variant only employs backpropagation through feature maps of previous sliding windows and short sliding windows. The SNR variant differs from the previous as it also encompasses the classification of long-lasting activities with rough features. For brevity, a variant of only the approach of Section 3.2 is not considered.

The baseline. The baseline CNN-LSTM utilizes overlapping sliding windows of 5.12 seconds with 1-second step size (78% overlapping) for PAMAP2, whereas for Opportunity the windows have a length of 1 second with 0.5 seconds overlapped (50%). Previous works [7, 9, 10, 13] have also used these settings. The baseline network consists of 3 CNN layers with 3x3 kernel size, with each CNN layer followed by a 2x2 max-pooling layer. The output of the CNN is flattened from 3D to 2D before being fed to the LSTM with 64 hidden units. Following, there are two FC layers of 64 and C (number of classes) neurons, respectively, before the final softmax layer.

Our methods. The shorter the windows, the lower the memory footprint and computation expense, but also the longer the inter-window temporal context that must be learned by the LSTM layer. There is a point at which further shortening of the windows significantly degrades the performance due to the inability of the LSTM layer in modeling the longer dependencies it is required to. We have chosen windows sizes of 0.25 and 0.18 seconds for the PAMAP2 and Opportunity datasets, respectively. This selection was made solely aiming at minimizing the window size without negative impact on the validation performance. We have not considered or investigated the effect of the mean duration of the activities in a dataset in this selection. The neural network has the same hyper-parameters as the baseline, with the exception that the output of the convolutions is flattened from 3D to 1D before the LSTM layer. The confidence threshold was set to 0.9. M was set by trial and error to 15 and 8 for PAMAP2 and Opportunity, respectively. The secondary network (in the CNN-LSTM SNR variant) is composed of one convolutional layer with 2x2 convolutions accompanied by a max-pooling layer of 2x2 kernel size. The output is flattened to 1D before an FC layer with softmax that outputs 2 real numbers. Also for the secondary

network, the confidence threshold was set to 0.95. In our methods, only the training set is segmented into overlapping windows. Each window has 78% and 50% overlapped with the previous one for the PAMAP2 and Opportunity datasets, respectively.

We used varying learning rates from $1e-4$ to $5e-4$ with Adam optimization algorithm. The code was written in Python 3 with the Tensorflow 1.12.

5.2 Metrics

In our experiments, we measured 5 different characteristics described below.

F1-Score (F1). The F1-Score is a more meaningful performance metric than the accuracy when dealing with imbalanced datasets, as it is usually the case in HAR.

Number of parameters (NPA). The total number of parameters (weights and biases) in the neural network.

Number of predictions (NPR). The total number of predictions over the entire test set. For the SN and SNR variants, predictions are discarded if the confidence level is below the confidence threshold. These discarded predictions are not counted to the number of predictions, however, their computation cost is counted to the total computational effort.

Total computational effort (TCE). Total computational expense (in GFLOPs) in predicting activities on the entire test set.

Prediction delay of new activities (PDNA). The average number of seconds of data needed to accurately predict a new activity once it has started. It is counted from the moment the new activity is started up to the moment when it is accurately predicted.

While the execution time and energy consumption of a deep neural network do not solely depend on its number of parameters and total FLOPs, these measures can still be used as reasonably good indicators for this purpose.

5.3 Discussion

Table 1 presents the results of the described methods for both datasets. We also include InnoHAR [13] that currently ranks on the top of HAR benchmarks for the same datasets.

CNN-LSTM Baseline vs. CNN-LSTM SN. Significant computational effort can be spared by learning inter-window temporal context via an LSTM layer instead of resorting to overlapping windows. Shortening the sliding windows considerably decreases the number of network parameters of the LSTM layer, as the input dimensionality of the input of the LSTM layer also decreases. Also, we obtain the benefit of shorter delays in recognizing new activities. The CNN-LSTM SN only outputs predictions whose confidence level is above the determined confidence threshold. Classes that are harder to be predicted are predicted less often than the easier ones since reaching a high enough confidence level for the harder classes requires more data. This results in an increased performance score.

CNN-LSTM SN vs. CNN-LSTM SNR. It is possible to take advantage of rough features to predict long-lasting activities. Naturally, the number of parameters increases with the addition of the secondary neural network, however, this also leads to a fairly substantial reduction in computations since the LSTM layer is skipped on several occasions. Predictions are seen more often as the secondary network outputs predictions with confidence above the threshold

without difficulty, as a binary classification problem is usually simpler than a multiclass problem. The performance, however, suffers a negative impact, in special for the Opportunity dataset, whose activities last less than 10s.

Our method vs. InnoHAR. Our method - either the SN or SNR variant - possesses a massively fewer number of parameters and reduces computational expense. Performance-wise, our method is able to rank higher (with the SN variant) in the Opportunity dataset, while having similar F1-score for PAMAP2. The PDNA metric for InnoHAR was not calculated, however, it should be near the baseline as long overlapping sliding windows are also used in InnoHAR. We believe that combining either of the methods proposed here with the inception-like structure of the CNN layers of InnoHAR could bring even better performance with much-improved resource efficiency.

Additional resource efficiency could be gained by utilizing different levels of features for predicting different activities. Rougher features could be used to predict easy-to-predict activities, while more fine-grained features would be used for the harder ones.

We would also like to investigate to what extent our methods can improve resource efficiency on actual embedded devices, by evaluating the effects on CPU and memory separately, instead of counting only the number of parameters and the total FLOPs during inference. Finally, we are interested in comparing our method with conventional compression methods and even finding ways to combine them for a further increase in resource efficiency.

6 Conclusions

We have seen that the most common approach in HAR is to use long and fixed-size overlapping sliding windows, which affects negatively resource efficiency. By studying the overlooked redundancies of this approach, we proposed to use short non-overlapping sliding windows and to skip fine-grained features during long-lasting activities. We use a CNN-LSTM neural network, whose LSTM layer is trained to learn inter-window temporal context (dropping the need for overlapping windows) and the CNN layers are designated to learn spatial and intra-window temporal context. With this, our proposal is able to obtain up to 35% less number of parameters in the network and 8x less computation with respect to a baseline, while maintaining high performance and predicting new activities as soon as possible. In summary, our proposal delivers comparable performance with the state-of-the-art while being greatly improved in terms of resource efficiency.

Acknowledgments

This work was funded by Business Finland (grant No. 1660/31/ 2018) and the European Union's Horizon 2020 Research and Innovation Programme (grant No. 777222).

References

- [1] C. Alippi, S. Disabato, and M. Roveri. 2018. Moving Convolutional Neural Networks to Embedded Systems: The Alexnet and VGG-16 Case. In *IPSN '18* (Porto, Portugal). IEEE Press, Piscataway, NJ, USA, 212–223.
- [2] M. Alizadeh and N. D. Lane. 2018. Using Pre-trained Full-Precision Models to Speed Up Training Binary Networks For Mobile Devices. In *MobiSys '18* (Munich, Germany). ACM, New York, NY, USA, 528–528.

- [3] J. Chauhan, J. Rajasegaran, S. Seneviratne, A. Misra, A. Seneviratne, and Y. Lee. 2018. Performance Characterization of Deep Learning Models for Breathing-based Authentication on Resource-Constrained Devices. *2018 ACM IMWUT* 2, 4, Article 158 (Dec. 2018), 24 pages.
- [4] R. Chavarriaga, H. Sagha, A. Calatroni, S. T. Digumarti, G. Tröster, J. del R. Millán, and D. Roggen. 2013. The Opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognition Letters* 34, 15 (2013), 2033 – 2042.
- [5] M. Edel and E. Köppe. 2016. Binarized-BLSTM-RNN based Human Activity Recognition. In *IPIN'16*. 1–7.
- [6] B. Fang, X. Zeng, and M. Zhang. 2018. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision. In *MobiCom '18* (New Delhi, India). ACM, 115–127.
- [7] Y. Guan and T. Plotz. 2017. Ensembles of Deep LSTM Learners for Activity Recognition Using Wearables. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 2, Article 11 (June 2017), 28 pages.
- [8] N. D. Lane, S. Bhattacharya, A. Mathur, C. Forlivesi, and F. Kawsar. 2016. DXTK: Enabling Resource-efficient Deep Learning on Mobile and Embedded Devices with the DeepX Toolkit. In *MobiCASE'16*. 98–107.
- [9] J. Long, W. Sun, Z. Yang, O. I. Raymond, and B. Li. 2019. Dual Residual Network for Accurate Human Activity Recognition. *CoRR* abs/1903.05359 (2019). arXiv:1903.05359
- [10] F. J. Ordóñez and D. Roggen. 2016. Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. *Sensors* 16, 1 (2016).
- [11] D. Ravi, C. Wong, B. Lo, and G. Yang. 2016. Deep learning for human activity recognition: A resource efficient implementation on low-power devices. In *2016 IEEE BSN*. 71–76.
- [12] A. Reiss and D. Stricker. 2012. Introducing a New Benchmarked Dataset for Activity Monitoring. In *ISWC'12*. 108–109.
- [13] C. Xu, D. Chai, J. He, X. Zhang, and S. Duan. 2019. InnoHAR: A Deep Neural Network for Complex Human Activity Recognition. *IEEE Access* 7 (2019), 9893–9902.
- [14] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher. 2018. FastDeepIoT: Towards Understanding and Optimizing Neural Network Execution Time on Mobile and Embedded Devices. In *Sensys '18* (Shenzhen, China). ACM, New York, NY, USA, 278–291.