A Public Repository to Improve Replicability and Collaboration in Deep Learning for HAR*

1st Lloyd Pellatt
University of Sussex
Brighton, United Kingdom
lp349@sussex.ac.uk

1st Marius Bock
University of Siegen
Siegen, Germany
marius.bock@uni-siegen.de

2nd Daniel Roggen
University of Sussex
Brighton, United Kingdom
daniel.roggen@ieee.org

2nd Kristof Van Laerhoven
University of Siegen
Siegen, Germany
kvl@eti.uni-siegen.de

2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) | 978-1-6654-1647-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/PerComWorkshops53856.2022.9767436

***Abstract*—Deep learning methods have become an almost default choice of machine learning approach for human activity recognition (HAR) systems that operate on time series data, such as those of wearable sensors. However, the implementations of such methods suffer from complex package dependencies, obsolescence, and subtleties in the implementation which are sometimes not well documented. In order to accelerate research and minimise any discrepancies between (re-)implementations, we introduce a curated, open-source repository which (1) contains complete data loading and preprocessing pipelines for 6 well-established HAR datasets, (2) supports several popular HAR deep learning architectures, and (3) provides necessary functionalities to train and evaluate said models. We welcome contributions from the fellow researcher to this repository, made available through: https://github.com/STRCSussex-UbiCompSiegen/dl_har_public**

***Index Terms*—human activity recognition, deep learning**

I. Introduction

Deep Learning (DL) has established itself as a state-of-the-art machine learning approach in HAR from wearable and mobile sensors [1]–[3]. The wide variety of architectures, however, combined with the fact that source code for these approaches is often unreleased, poorly documented or not maintained, has contributed to a problem of replicability in this field. We argue that this impedes the analysis of novel network architectures or approaches and transparent comparisons against existing models. Examples are mentioned in [4] and [5], where authors were unable to reproduce the results from [1] due to obsolete source code.

Another issue affecting replicability is a lack of an accepted standard in the community for the preprocessing of public datasets, leading to different results on the same dataset. This tends to occur even where the model is re-implemented as methodically as possible from published papers and hinders exact replication of research outcomes.

We here propose to address these issues by creating and maintaining a central public repository of DL models, dataset loading and preprocessing pipelines, which can be used to create replicable benchmark scores for novel and established DL models. The aim is to use this as a single entry point into the field of HAR and learning resources for all researchers. The repository has been started from the outset to facilitate collaboration between different and distributed institutions on future research projects, and we give an example of how this is already happening in section III.

The repository contains complete data loading and preprocessing pipelines for 6 popular datasets [6]–[11]. It further includes two versions of the DL model DeepConvLSTM [1], the original and a shallow version with only one LSTM layer [4], as well as the state-of-the-art model Attend and Discriminate [3], which uses novel techniques such as self- and temporal attention. We also provide functions to train and evaluate the models and encourage contributing models and data loading / preprocessing configurations for other datasets, to expand the coverage of the repository.

This work received support from the EU H2020-ICT-2019-3 project "HumanE AI Net" (project number 952026) and from the House of Young Talents at the University of Siegen.

II. Repository Structure and Modules

The repository consists of a *main* script, containing a usage example, and four GitHub submodules which serve different purposes within the deep learning pipeline. Submodules allow to manage and extend each part of the pipeline individually. Within the repository, users also can find documentation describing how to properly use the repository to perform experiments. The following describes each submodule and its functionalities.

A. Dataloader

The *dataloader* module is formatted as a python package, and provides functions which help in loading raw data and applying preprocessing techniques. This module can:

- Define parameters in YAML files, with presets for widely used datasets, and a guide for adding new datasets.
- Download public HAR datasets.
- Extract raw data and organize resulting data by subject or according to an arbitrary train/test/validation split.
- Apply preprocessing functions.

The *dataloader* module contains a class *DatasetLoader* which stores the configuration options as attributes and implements several methods which enable the loading of data from a compressed archive. A function *load_preset* is also defined, which reads the configuration options in dictionary form from the given YAML file. The preprocessing functions which are currently implemented in the repository include:

- Select sensor channels and activity labels, as well as mapping labels to integers for pytorch.

```
config = load_preset(preset_name)
loader = DatasetLoader(data_dir, **config)
preprocess_dataset(loader, args)
```

Fig. 1. A minimal example illustrating downloading and preprocessing a given dataset with the data module.

```
name: "nameofdataset"
data_files:
    subject 1:
        - "path/to/file"
        - "path/to/file"
    subject 2:
        - "path/to/file"
sensor_columns: [0, 1, 2, 3, 4, 5, 6]
```

Fig. 2. Example of YAML configuration options.

- Replace missing data.
- Downsample the data by an integer factor after applying a low-pass filter.
- Write processed data to disk using numpy.

These are all contained in a function *preprocess_dataset*. As such, an instance of the *DatasetLoader* class can be initialized with a preset or user-defined YAML config file and this can be used to download and preprocess the given dataset using the code shown in figure 1, where “preset_name" is the name of the YAML file (without extension) and “data_dir" is the directory where the raw dataset zip file can be found.

The *dataloader* module is fully configurable by means of YAML formatted configuration files. YAML (YAML Ain’t Markup Language) [12] is a data serialization language which is both human and machine readable, and as such is often used for configuration files. Configuration options for the *dataloader* module are listed in the readme file in the git repository, and include the URL from which the dataset can be downloaded, the name of the archive, the columns containing sensor data, activity labels, and user information, among others.

Figure 2 shows an example YAML file. When read using PyYAML, this will give a dictionary with keys “name", “data_files", and “sensor_columns". The value of “name" will be a string —“nameofdataset", and the value of “sensor_columns" will be a list of integers. The value of “data_files" will be a nested dictionary, with keys “subject 1" and “subject 2", where each value is a list of strings. These three configuration options specify where the processed data should be put (a folder called “nameofdataset"), what the files containing processed data should be called (“subject 1" and “subject 2") and which sensor columns should be used from the raw data.

As of submitting this paper the *dataloader* module supports 6 popular HAR datasets, namely Heterogeneity Human Activity Recognition (HHAR) [6], RealWorld HAR (RWHAR) [7], Opportunity [8], Sussex-Huawei Locomotion (SHL) [9], Skoda Mini Checkpoint (Skoda) [10], and Physical Activity Monitoring (PAMAP2) [11]. As well as describing the file structure of the data, the configuration files can also be used to specify which preprocessing steps should be applied to the raw data.

This system of using configuration files to specify data loading and preprocessing steps is intended to enable authors to share the exact setup of their experiments in such a way that it can be easily reproduced, simply by sharing a configuration file and specifying which version (or which commit) of the data loading module was used. Currently, preset configuration files are available for all 6 datasets. Where the datasets contain data from multiple users, a Leave-One-Subject-Out (LOSO) cross validation split is defined. In the case of the Skoda dataset, data from only one user is present, so the data is split into a training set containing 70% of the data, a validation set containing 10% of the data, and a testing set containing 20% of the data. For the Opportunity dataset, we also provide a preset file to mimic the train-valid-test split of the Opportunity challenge [8].

B. Model

The *model* submodule contains implementations of popular DL architectures used within the area of HAR, as well as necessary functionalities for training and evaluation. The key functionalities of this module are:

- Define model architectures used for training & prediction.
- Train defined models using processed datasets.
- Validate model performances using best-practice validation techniques within the area of HAR.
- Save prediction results as well as other relevant training and validation metrics for further analysis.

The repository currently supports three DNN architectures: DeepConvLSTM as proposed by Ordonez and Roggen [1], a shallow version of DeepConvLSTM proposed by Bock et al. [4] and the state of the art model Attend and Discriminate proposed by Abedin et al. [3]. Each model is defined in a separate python file within the folder *models*. In order to ensure that each model added to the repository supports a set of basic functionalities needed for logging and saving checkpoints, each model inherits from a *BaseModel* class, which itself inherits from the pytorch *Module* class. Adding a new model to the repository is as easy as creating a file *model.py* which contains your model, ensuring that it subclasses *BaseModel*, and adding it to the models directory.

The module supports the most commonly used optimizers and weight initialisation schemes, and provides additional functionality at train time including data augmentation via label smoothing [13] and mixup [14], and loss function enhancement via sample weighting and consideration of the center-loss function [15].

Trained models can be evaluated using fixed split or LOSO cross-validation. A fixed split splits the dataset into train, validation and test data. Splits can be defined via the YAML files within the dataloading submodule (see section II-A). During LOSO cross-validation s models are trained, where s is the amount of subjects in the dataset, making each subject the validation set exactly once. The final result is then the averaged validation results across all models. We further

provide functionality to have experiments run multiple times using a set of user-defined random seeds.

To allow for further analysis (even at a later point in time), users can decide whether calculated evaluation metrics (loss, accuracy and F1-score (macro and weighted)), raw predictions, console logs, and the best performing model should be saved to disk. We further added support for logging results using *weights and biases* (https://wandb.ai), a popular website for tracking experiments.

C. Analysis

The purpose of this module is to analyse the performance of a trained model in detail. Its main functionalities include:

- Output a concise evaluation report based on common evaluation metrics like accuracy and F1-score.
- Rerun analysis on saved train and test results.

As of submitting this paper, the analysis submodule prints out the average train (and test) results across runs. For the LOSO cross-validation it additionally prints out subject-wise results. We also allow users to run analysis of saved train and test results retrospectively with the *analysis.py* script. Within later iterations of this repository, we plan to extend the analysis submodule to automatically create and save plots of average, epoch-wise results.

III. Work in Progress

In order to demonstrate the capabilities of the repository and how it can be used for collaboration between organisations, we show results of our experiments which tested the applicability of grokking, which was introduced by Power et al. [16], to HAR datasets. Grokking describes the phenomenon that a neural network can be trained for long periods of time and still learn patterns in the dataset at late stages of the training process. Power et al. [16] show that though training loss converged to 0 and validation loss started increasing again, a second decrease in validation loss was witnessed during later stages of the training.

Within the DL community there is a common belief that perfect training results are a sure sign of overfitting. Evaluation metrics like the generalization gap suggest that the performance difference between training and validation set should be kept as small as possible, while heuristics like early stopping suggest to stop the training process early once a plateau is reached and validation performance stagnates/ decreases. Through the “Grokking" mechanism, Power et al. suggest that though a network may seem to be overfitting as evidenced by an increase in validation loss, it could still be learning patterns in the data, leading to an improvement in the validation metrics at later epochs.

To investigate whether grokking can be also exhibited within HAR we trained the DeepConvLSTM model as proposed by Ordonez and Roggen [1] for 300 epochs. We further tested whether regularization techniques like a learning rate scheduler (LRS) and a weighted cross entropy loss would lead to a better (grokking) performance. As input data we used the Opportunity challenge dataset [8] with the split suggested by the competition. We applied the same preprocessing techniques as Ordonez and Roggen [1], and additionally normalised each sensor-axis using z-scores. In total we tested four different training settings using:

1) LRS and weighted loss. (LRS + weighted)
2) Only LRS. (LRS)
3) Only weighted loss. (weighted)
4) None. (none)

Within all settings we kept all other hyperparameters consistent with those as reported by Ordonez and Roggen [1]. We employed a LRS decay factor of 0.9 and a step size of 10.

Figure 3 shows the average training and validation results obtained from all four settings across 5 runs using a set of 5 random seeds. On the one hand, one can see that the network obtains (near) perfect training performance after around 50 epochs. On the other hand, within all settings, the validation loss only decreases for 5 epochs, before monotonically increasing over time. Nevertheless, we are still exhibiting an increase in validation metrics over time. Comparing the different settings, one can see that setting 1, 3 and 4 perform the most stable. Only setting 2 shows a slight decrease in validation performance over time. It is also the only setting which is not able to obtain perfect training results.

TABLE I
Test F1-score (macro) after every 50^{th} epoch.[a]

	Setting			
Epoch	1	2	3	4
50	53.63%	54.23%	52.05%	55.44%
100	56.58%	55.86%	53.31%	55.29%
150	58.13%	56.86%	54.21%	55.63%
200	57.66%	56.44%	52.37%	55.95%
250	58.62%	56.83%	51.70%	56.43%
300	58.49%	58.04%	52.64%	56.03%

[a]Averaged across 5 runs using set of 5 random seeds.

Table I holds the average test F1-score (macro) after every 25^{th} epoch. The table shows that both setting 1 and 2, which employ a LRS, produce better testing results, suggesting that a scheduled learning rate is necessary in order to produce more generalised results. Our work so far suggests that some models can benefit from longer training with a reduced learning rate and higher weighting given to minority classes in the loss function. We hypothesize that this improves the generalization of the model by encouraging the parameters to settle into a local minimum.

To conclude, our results question the applicability of a generalization gap as well as early stopping for HAR. Though we did not exhibit a second decrease in validation loss like Power et al. [16], our results still show that long training times do not necessarily hurt the generalization of trained models. Our future research will include evaluating this approach on a different dataset (e.g. RWHAR [7]), employing different types of LRS and investigating whether this trend can also be exhibited when employing LOSO cross-validation. Especially the latter will show whether this form of “grokking" can still be witnessed if the validation data attributes to a previously

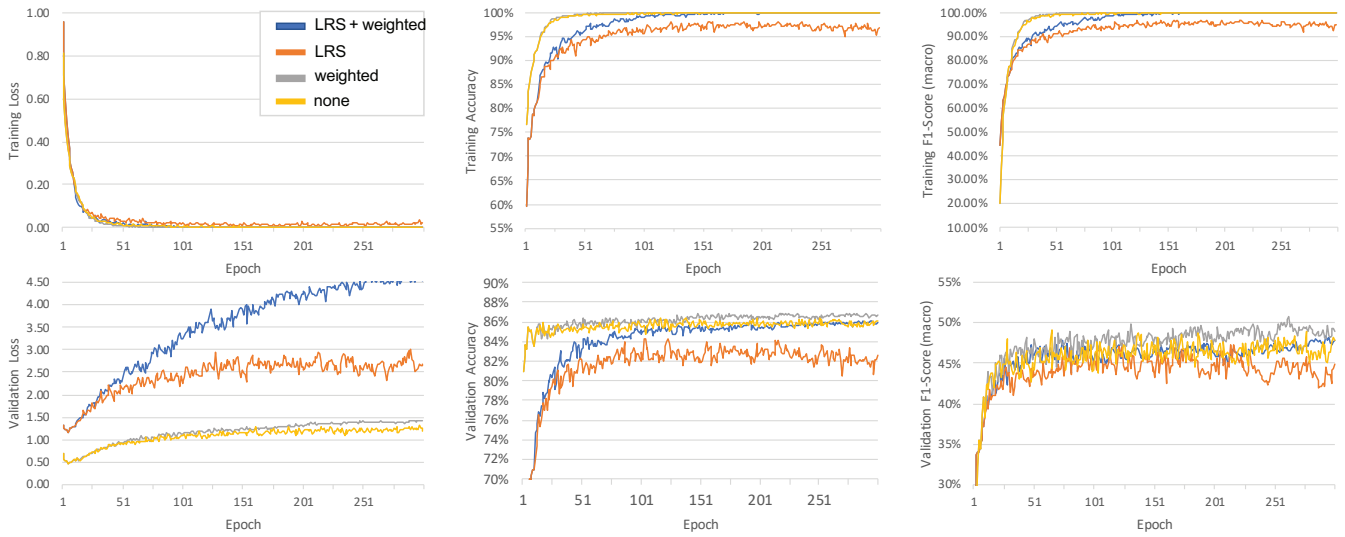

Fig. 3. Average train and validation results on the opportunity challenge dataset [8] using either a learning rate scheduler (LRS) and/ or weighted cross-entropy loss (weighted) across 5 runs using a set of 5 random seeds. Though training performance reaches near perfection and validation loss increases, validation metrics steadily increase across time. Using only a LRS has validation metrics slightly decrease at later stages of the training.

unseen subject, since the validation and test data of the Opportunity challenge contain data of subjects which are also part of the train dataset.

IV. Future Work

We would like to invite the members of the community to join our effort and contribute to include more datasets, models, preprocessing and analysis functions and data augmentation techniques. As changing the model architecture is just one way to obtain better results, we want to give users the option to use their own custom loss functions, learning rate schedules and optimizers, which in turn opens up the opportunity to also evaluate semi- and unsupervised learning problems. We further plan to add support for models implemented in TensorFlow and package and index the code for distribution on PyPI, so that it can be installed with a package manager, once the basic set of features is complete. The repository and source code for all experiments mentioned within section III is publicly available via https://github.com/STRCSussex-UbiCompSiegen/dl_har_public.

References

[1] F. J. Ordóñez and D. Roggen, "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition," *Sensors*, vol. 16, no. 1, 2016.

[2] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep Learning for Sensor-Based Activity Recognition: A Survey," *Pattern Recognition Letters*, vol. 119, pp. 3 – 11, 2019.

[3] A. Abedin, M. Ehsanpour, Q. Shi, H. Rezatofighi, and D. C. Ranasinghe, "Attend and Discriminate: Beyond the State-of-the-Art for Human Activity Recognition Using Wearable Sensors," *Proc. of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 1, 2021.

[4] M. Bock, A. Hölzemann, M. Moeller, and K. Van Laerhoven, "Improving Deep Learning for HAR with Shallow LSTMs," in *Int. Symp. on Wearable Computers*, 2021, pp. 7–12.

[5] L. Pellatt and D. Roggen, "Fast Deep Neural Architecture Search for Wearable Activity Recognition by Early Prediction of Converged Performance," in *Int. Symp. on Wearable Computers*. ACM, 2021, pp. 1–6.

[6] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen, "Smart Devices are Different: Assessing and MitigatingMobile Sensing Heterogeneities for Activity Recognition," in *13th Conf. on Embedded Networked Sensor Systems*. ACM, 2015, pp. 127–140.

[7] T. Sztyler and H. Stuckenschmidt, "On-Body Localization of Wearable Devices: An Investigation of Position-Aware Activity Recognition," in *Int. Conf. on Pervasive Computing and Communications*. IEEE, 2016, pp. 1–9.

[8] D. Roggen, A. Calatroni, M. Rossi, T. Holleczek, K. Förster, G. Tröster, P. Lukowicz, D. Bannach, G. Pirkl, A. Ferscha, J. Doppler, C. Holzmann, M. Kurz, G. Holl, R. Chavarriaga, H. Sagha, H. Bayati, M. Creatura, and J. d. R. Millàn, "Collecting Complex Activity Datasets in Highly Rich Networked Sensor Environments," in *7th Int. Conf. on Networked Sensing Systems*. IEEE, 2010, pp. 233–240.

[9] H. Gjoreski, M. Ciliberto, L. Wang, F. J. Ordóñez, S. Mekki, S. Valentin, and D. Roggen, "The University of Sussex-Huawei Locomotion and Transportation Dataset for Multimodal Analytics with Mobile Devices," *IEEE Access*, vol. 6, pp. 42 592–42 604, 2018.

[10] P. Zappi, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Troster, "Activity Recognition From On-Body Sensors by Classifier Fusion: Sensor Scalability and Robustness," in *3rd Int. Conf. on Intelligent Sensors, Sensor Networks and Information*. IEEE, 2007, pp. 281–286.

[11] A. Reiss and D. Stricker, "Introducing a New Benchmarked Dataset for Activity Monitoring," in *Int. Symp. on Wearable Computers*. IEEE, 2012, pp. 108–109.

[12] O. Ben-Kiki, C. Evans, and B. Ingerson, "YAML Ain't Markup Language (YAML (TM)) Version 1.2," YAML.org, Tech. Rep., 2021.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *Conf. on Computer Vision and Pattern Recognition*. IEEE, 2016, pp. 2818–2826.

[14] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond Empirical Risk Minimization," 2017.

[15] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A Discriminative Feature Learning Approach for Deep Face Recognition," in *European Conf. on Computer Vision*. Springer, 2016, pp. 499–515.

[16] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra, "Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets," in *Workshop on the Role of Mathematical Reasoning in General Artificial Intelligence*, 2021.