Proceedings of the 7th International Conference on Trends in Electronics and Informatics (ICOEI 2023)
IEEE Xplore Part Number: CFP23J32-ART; ISBN: 979-8-3503-9728-4

Smartphone based Human Activity Recognition using CNNs and Autoencoder Features

Sowmen Mitra
Department Of Computer Science And Technology
School Of Artificial Intelligence
Hebei University Of Technology
Beijing, China
sowmenmitra7@gmail.com

Proma Kanungoe
Department Of Information Technology
School Of Information Technology And Engineering
Vellore Institute Of Technology
Vellore, India
proma.kanungoe2018@vitalum.ac.in

2023 7th International Conference on Trends in Electronics and Informatics (ICOEI) | 979-8-3503-9728-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICOEI56765.2023.10126051

***Abstract*—Recognition of human activities is essential for many applications, and the widespread availability of low-cost sensors on smartphones and wearables has enabled the development of mobile apps capable of tracking user activities "in the wild." However, dealing with heterogeneous data from different devices and real-time scenarios presents significant challenges. In this study, a novel learning framework is proposed for Human Activity Recognition (HAR) that combines a Convolutional Neural Network (CNN) with an autoencoder for feature extraction. The study also investigates the importance of preprocessing techniques, including orientation-independent transformation, to mitigate heterogeneity when dealing with multiple types of smartphones. The results show that the proposed approach outperforms state-of-the-art methods in HAR, with an accuracy of 95.74% on the heterogeneous dataset used in this study. Furthermore, the study demonstrates that proposed framework can be effectively deployed on smartphones with limited computational resources, making it suitable for real-world applications.**

Keywords—Mobile Sensing, Human Activity Recognition, Convolutional Neural Networks, Autoencoders

I. INTRODUCTION

Recognition of daily human activity is essential for many applications: healthcare monitoring, security concerns, fitness tracking, and user-adaptive systems.

With the wide spread of low-cost sensors on smartphones and wearables, the development of mobile apps capable of tracking user activities *"in the wild"* leads to relevant challenges that need to be tackled. As stated in [1], many variables come from users and smartphones. Users are demographically different (age, stature, weight) and perform activities differently with their style. Devices instead share among the other operating systems, hardware, and sensing capabilities.

This study starts with the model proposed in [2], showing how the controlled environment influences the classification. In particular, no heterogeneity among devices (and sensors) leads to biased data and non-natural settings. For this reason, the dataset provided in [3] is adopted, which emphasizes device heterogeneity. This study further investigates the effectiveness of statistical and manual-engineered features, which are usually insufficient in natural settings due to the heterogeneous scenario. Moreover, the effects of orientation-independent transformation are examined as a preprocessing data block, which should make data agnostic regarding sensor position and orientation. With this work, heterogeneity impairments in actual use case scenarios are analyzed using previous state-of-the-art models and propose a novel learning framework to tackle HAR impairments in these situations. Proposed framework will be made by a CNN architecture augmented with features extracted from an autoencoder, with an eye on mobile portability [4]. Ultimately, the proposed results are compared with state-of-the-art works [5].

The main contributions can be summarized as follows:

- This research work proposes an architecture that combines CNN and automatic feature extraction to perform HAR. In particular, augmenting a traditional CNN model with an autoencoder rather than manual features can lead to better results.
- In this study, the importance of an excellent preprocessing pipeline to mitigate heterogeneity when dealing with multiple types of smartphones are examined. Also, orientation-independent transformation can give promising results in actual use case scenarios where smartphones can be in any position and orientation.
- This study shows that good results can also be obtained w.r.t. state-of-the-art works when few computational resources are available, i.e., smartphones.

This research work is structured as follows. Section II describes the state-of-the-art; the system and data models are respectively presented in Sections III and IV. The proposed signal processing technique is detailed in Section V, and its performance evaluation is carried out in Section VI. Concluding remarks are provided in Section VII.

II. RELATED WORKS

HAR using smartphone accelerometer and gyroscope data has been well studied in the literature. Many research work tackles this problem using machine learning or deep learning techniques. For example, in [6], K. Frank et al. proposed a set of handcrafted features combined with machine learning techniques such as SVMs, artificial neural networks, decision trees, and Bayesian approaches to perform the final classification. D. Anguita et al. [7] made the popular UCI dataset [8] and achieved the best results using 561 hand-designed features and various classifiers on top of them.

Other state-of-the-art models, instead, use automatic feature extraction methods to avoid manually designed features. In [2] A., Ignativ's use of neural networks has allowed them to extract precise and concise information from a time series. Also investigates the impact of time series length on recognition accuracy. The accuracy of the proposed approach is evaluated on two commonly used

datasets: the UCI [8] and WISDM [9] datasets. This model provides reliable and accurate results. It performs well and is quick to use. There is no need for tedious manual editing or tricky setup, so that it can be put to work immediately.

However, the works presented up to this point, as well as the majority of works available in the literature, evaluate

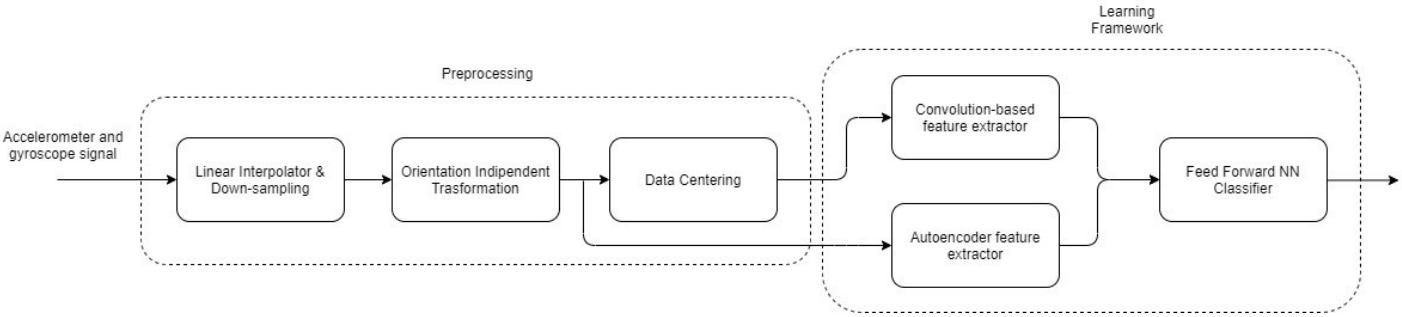

Fig. 1. Our proposed signal processing pipeline & learning strategy

model performances in controlled environments with acquired datasets, such as UCI and WISDM, without considering the HAR model impairments when used in real-world scenarios, like mobile apps. As stated in a recent work by K. Chen et al. [10], the problem is severe. The authors analyze the main HAR challenges, such as recognizing concurrent and composite activities, dealing with sparsely annotated data, class imbalance, sparsely user-distributed datasets, privacy, computational cost, etc. A. Stisen et al. in [3] studied the technical details and problems due to heterogeneity among users, sensors, and environments. They acquired a specific dataset that shows these heterogeneities and studied the impact on models. Also, they proposed various clustering and classification algorithms to perform HAR in these conditions. In [11], Y. Vaizman et al. acquired a huge in-the-wild dataset to promote practical applications that work in real-life settings. They collected the so-called Extrasensory dataset [12], composed of over 300,000 minutes of sensor data with context labels from 60 subjects who used their smartphones conveniently. They demonstrate the importance of fusion multi-modal sensors in resolving the in-the-wild HAR problem. However, they used more sensor signals, like magnetometer, compass, GPS, audio, and phone state, in which we were not interested.

Furthermore, most collected data has incorrect context labels because users aren't always precise when labeling data, which could lower the proposed model's performance metrics.

III. Processing Pipeline

As reported in [3], three major types of heterogeneities yield impairment in HAR:

- **Sensor Biases (SB)**: To keep the overall cost of a smartphone low, cheap accelerometer and gyroscope sensors are used, yielding poor-calibrated, inaccurate, and range/granularity-limited acquired signals. We could observe low precision, resolution, range, and biases.
- **Sampling Rate Heterogeneity (SRH)**: Often, popular smartphones vary in default and supported sampling frequencies for accelerometer and gyroscope sensors. For example, in the dataset proposed in [3], the sampling frequency varies from 50Hz to 200Hz.
- **Sampling Rate Instability (SRI)**: This phenomenon is specific to a single device and regards the regularity of the time span between successive measurements. Different factors could accentuate this problem, including heavy multitasking or high I/O load in the mobile device. In particular,

multitasking operations affect the sensor sampling rate because smartphones prioritize various running tasks. For example, in our collected dataset with a 100 Hz sampling rate (which means a time span between consecutive samples of 10*ms*), we observe a period ranging mainly be- tween 3*ms* and 15*ms* with an average of 7*ms*, even if the smartphone was in an *airplane mode* to reduce at a minimum this effect. Fig. 2 shows the amount of different time- spans between consecutive measurements in ourdataset.

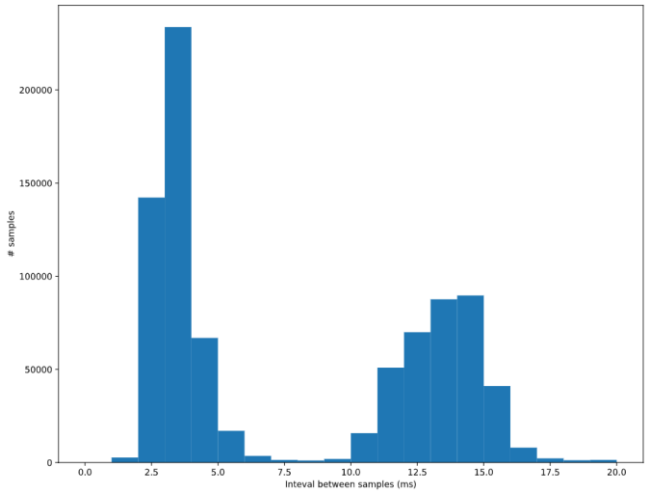

Fig. 2. Histogram of the different time span between consecutive samples included in our collected dataset

Furthermore, in a real use-case scenario, we must also consider that smartphones can be positioned and oriented differently in the human body. For example, a smartphone could lay in trouser pockets (back and front) or maybe inside a pouch or a bag with different orientations. These different problem settings have huge effects on the prediction accuracy of a HAR predictor, especially if the model has been trained on a dataset consisting of activity measurement from one fixed position and orientation, as is usually the case in publicly available datasets.

We adopted three main preprocessing blocks in our pipeline to tackle all these problems, as shown in Fig.1

The first block, Linear Interpolator, mitigates the problems regarding the previously discussed SRH and SRI heterogeneity. Its primary purpose is to down-sample the input data to a fixed sampling rate, in our case, 50 samples/second equally spaced in time.

The second block, Orientation Independent Transformation (OIT), represents data from different smartphone orientations in a new space whose orientation is independent of the smartphone and aligned with gravity and the direction of motion. In this way, a user can place his smartphone in whatever position they want, mitigating the problem of different position and rotations of the smartphone, as we will see in Sec VI.

Our last block consists of a data-centering operation that centers the signal among the y-axis and is used only as input for the Convolutional Layer. The reason for this choice is apparent in Sec. VI. As reported in [2], time series centering standardizes the input data, making the task for CNN easier. Data normalization must be avoided because it does not help since it significantly distorts time series shape, removing magnitude information critical for activities differentiation.

After the preprocessing pipeline, we adopt a novel learning framework. It is composed of CNN augmented with features coming from the code of an autoencoder. As discussed in [2] CNNs learns filters that are applied to small sub-regions of the data, and therefore they can capture local data pattern and their variations. Additionally, due to a small number of connections and high parallelism, the amount of computations and running time of CNNs is significantly lower compared to another deep learning algorithm. Such a model is perfect for real-time HAR applications, even in a constrained environment like smartphones. The only drawback of CCNs is that they fall behind in capturing the global properties of the signal. In [2], authors resolve this problem by augmenting CNNs with basic statistical features. But instead of what was done in this latter work, where they used a few hand-made parts, we decided to opt for automatically extracted features. These are created by an autoencoder which should provide a more robust representation of information. As an implication, we can train the autoencoder separately and then use the trained model in combination with CNN.

IV. Signals and Features

A. Dataset & Measurement Setup

Many datasets are available for this type of application in literature: [8], [9], [12], [13]. However, there are nontrivial problems in real-world applications: above all, the environment is not controlled. So we should be capable of dealing with technical details such as smartphone orientation or sensor accuracy, as reported in Sec. III.

For our application, we decided to use the *Heterogeneity* Dataset (*HD*) [13] described in [3], where data are collected directly from smartphone sensors. The dataset is composed of nine users (aged 25-30). All of these users followed a scripting set of activities. They did sit, stand, walk, bike, and stairs up/down while carrying eight smartphones. All eight phones were placed in a pouch and worn around the waist. For this study, we had ten people undergo two sessions—one in a lab with all four devices and one at home with only the phone. In our case, we were interested only in smartphone-acquired signals, leaving out from our experiments the smartwatch evaluations.

We also collected a novel dataset called *Oriented Dataset (OD)*. This new dataset aims to see whether the model can generalize well on new unseen users and test how OIT preprocessing could increase performance in this new context. This dataset is collected with a single smartphone, at a sampling rate of 100 Hz in seven different positions: top, bottom, left, and right are contained in the same way as in work [3], but instead of a fixed orientation, we rotated the smartphone inside the pouch to each possible direction. Then we also perform the same activities with the smartphone held in hand and in the trouser's pocket to see to what extent the model can generalize activity recognition in new unseen positions. In this case, we condensed sit and stand activities into no activity for two reasons: first, we are not interested in distinguishing between them, and second, they are made indistinguishable by OIT.

B. Signal preprocessing

Notation: With $x \in \mathbb{R}n$, we define a column vector as $x=(x_1, x_2, x_3, \ldots, x_n)^T$ Where with superscript ^{T} we mean the transpose operator. With $\|x\|$, we mean the L2-norm operator $\|x\| = (\sum_{i=1}^{n} x_i^2)^{1/2}$, and with $\bar{x}=\sum_{i=1}^{n} x_i/n$ the mean of a vector. With $x \cdot y = x^T y$, we mean the inner product of two vectors. With $\vec{x}$ We mean a 3D vector $\vec{x}=(x_1, x_2, x_3)^T$ and with $\hat{x}$ the corresponding 3D versor $\hat{x} = \vec{x}/||\vec{x}||$. For any two 3D vectors, $\vec{x}$ and $\vec{y}$. We indicate their cross-product as $\vec{x} \times \vec{y}$. We represent with a_x, a_y and a_z the x, y, and z components of the accelerometer signal, and the gyroscope signal is represented with g_x, g_y, *and* g_z. We also define matrices with uppercase and bold letters. For example, to define a $3 \times n$ matrix composed of 3 vectors $x, y, z \in \mathbb{R}^n$, we use the notation $M = [x, y, z]^T$

Linear Interpolation & Down-sampling. As described in Sec. IV-A, our dataset is composed of signals collected with different sampling rates from many mobile phones, and they are usually out of sync due to the SRI. For this reason, we apply to all signals a simple preprocessing pipeline block composed of two steps: linear interpolation and down-sampling. With linear interpolation, we project signals with different sampling frequencies, i.e., 50, 100, and 200 Hz, to a 50 Hz signal, which means we are also applying at the same time down-sampling. Linear/nearest interpolations are usually preferred w.r.t. cubic spline or smooth cubic spline, as investigated in [3], because they typically mitigate the introduction of noise or artifacts. In linear interpolation, the value at time t is the piecewise-linear interpolation, i.e., the linear interpolation between the input samples adjacent to t in the sequence of input sample timestamps. Let (x_0, y_0) and (x_1, y_1) be two known points; the linear interpolant is the straight line between these points. For a value x in the interval (x_0, x_1), the value y along the straight line is given from the equation of slopes

$$\frac{y-y_0}{x-x_0} = \frac{y_1-y_0}{x_1-x_0} \quad (1)$$

The linear interpolation on a set of data points (x_0, y_0), (x_1, y_1), . . . , (x_n, y_n) is defined as the concatenation of linear interpolants between each pair of data points.

Orientation Independent Transformation. To project the signals acquired during an activity in a new space independent of the rotation of the smartphone, three orthogonal versors need to be found. We decided to adopt the technique proposed in [14], although many other works offered a similar solution as in [15], [16]. In summary, we must find these three orthogonal versors: vertical $\hat{v}$, horizontal $\hat{h}$, and lateral $\hat{l}$. As the names suggest, the vertical versor is aligned with the user's torso, pointing up; the horizontal versor is aligned with the direction of motion, pointing forward; and the last versor tracks lateral movements, which is orthogonal to the other two.

Starting with the vertical versor, we need to find where the gravity vector $\vec{p}$ lie in the original space. Although the gravity vector is constant in stationary conditions, during a user activity, it continuously changes in the original coordinate system of the smartphone. Therefore, we can only consider the gravity's mean direction within the current user activity. So, to estimate it, we must consider only the accelerometer signal: $\vec{p} = (\overline{a_x}, \overline{a_y}, \overline{a_z})^T$, and we now could find the vertical versor with $\hat{v} = \vec{p}/\|\vec{p}\|$.

This is the first axis of our new space, and we can project dates onto this new versor to obtain our first component axis. To do that, we define the acceleration matrix $A = [a_x, a_y, a_z]^T$ and the gyroscope matrix $G = [g_x, g_y, g_z]^T$. Now we could project the data onto $\hat{v}$ by:

$$a_v = A\hat{v} \ , \ g_v = G \cdot \hat{v} \tag{2}$$

Now we have to find a horizontal plane parallel to the floor, where the activity motion mainly occurs. We must remove the av component from the original data to do so. We represent the accelerometer data on this new plane as M representing the motion plane. To find M, we have to:

$$M = A\text{-}\hat{v}a_v^T$$

In this new plane, we could see that the direction with the most significant variance of projected data represents the main direction of motion, i.e., In that direction, the user is currently performing the activity w.r.t the current smartphone orientation. By applying PCA [17], we can find the path along which the variance of measurements is maximized. This is the horizontal vector h. We now could compute the horizontal versor $\hat{h} = \vec{h}/||\vec{h}||$, and we are now able to project our data onto this second new axis:

$$a_h = A \cdot \hat{h} \ , \ g_h = G \cdot \hat{h} \tag{3}$$

Applying a cross-product between the two last obtained versors is sufficient to find the last axis, so $\hat{l} = \hat{v} \times \hat{h}$. We are now able to project the data among this new axis:

$$a_l = A \cdot \hat{l} \ , \ g_l = G \cdot \hat{l} \tag{4}$$

Combining Eq. (2), Eq. (3), and Eq. (4) leads us to the final accelerometer and gyroscope transformed signals represented in this new orientation-independent space, which are $[a_v, a_h, a_l]^T$ and $[g_v, g_h, g_l]^T$

Centering. Our last signal preprocessing block involves data centering. As proved in [2], this could slightly improve performance within a CNN-based learning model because centering the time series makes the task easier for the CNN. We denote with xc the centered vector of x, i.e

$$x^c = x - \overline{x} = (x_1 - \overline{x}, \ x_2 - \overline{x}, \ ..., \ x_n - \overline{x})^T \tag{5}$$

Centering is applied for each time window only on accelerometer data. Thus the new centered acceleration matrix is:

$$A^c = [a_x^c, a_y^c, a_z^c]^T \tag{6}$$

C. Feature vector

Time windows. When dealing with time-based signals (or time series in general), we must remember that correlation between samples occurs and handle this helpful information. However, in the domain of HAR, even if samples can be correlated in time, the correlation does not persist over a long period. In literature, many time window intervals were experimented with [2], and they discovered that the best time window interval is from 1s to 2.5s. For this reason, we adopt a sliding window approach with a fixed window length of 2.5s and a 50% overlap between two successive windows.

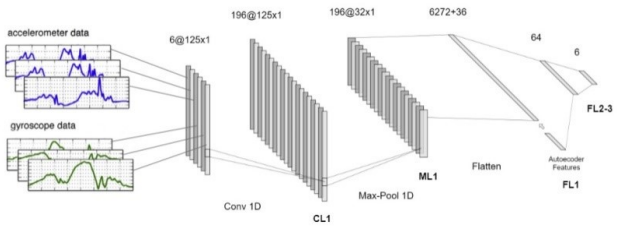

Fig. 3. In our proposed learning framework, CL1 is a convolutional-1D layer, ML1 is a max-pooling layer, FL1 is the fully connected layer which represents previously extracted features that are concatenated with the autoencoder features, FL2 and FL3 are thoroughly combined layers

Features. Features are the essential elements every machine learning algorithm needs to learn something. Everything fed into a machine learning algorithm can be considered a feature, but here we use three types of features targeting different kinds of information.

- *Raw features*. These enable the model to learn directly from data and hopefully from its shape to generalize and classify activities. Raw features are represented by a 6xl25 matrix where rows are accelerometer and gyroscope x, y, and z dimensions while columns are samples over time.
- *Manual features*. Manual features take into account the statistical mode of the signal and are used in [2] and[7] with excellent results. We exploit manual parts like mean, standard deviation, a sum of the absolute values, and the histogram of each input data channel computed on local time windows applied only for an accelerometer signal a. As defined in [2], we produce a primary feature vector: $f_b = (\overline{a}_x, \overline{a}_y, \overline{a}_z, \sigma_{a_x}, \sigma_{a_y}, \sigma_{a_z}, \widetilde{a}_x, \widetilde{a}_y, \widetilde{a}_z \ , \ \psi_{a_x a_y a_z} \ , \ h_{a_x}^T, h_{a_y}^T, h_{a_z}^T)$ Where for a generic column vector $x = (x_1, x_2, x_3, \ldots\ldots\ldots, x_n)^T$ with $x^c = (x_1^c, x_2^c, x_3^c, \ldots\ldots, x_n^c)^T$:

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{n}(|x_i^c|)}{n}} \tag{7}$$

$$\tilde{x} = \frac{\sum_{i=1}^{n}(|x_i^c|)}{n} \tag{8}$$

$$\psi_{xyz} = \frac{\sum_{i=1}^{n} \sqrt{x_i^2 + y_i^2 + z_i^2}}{n} \quad (9)$$

And h_x is a column vector that sorts the values of x into ten equally spaced bins along the x-axis between the minimum and maximum values of x^1 . This led to a manual features vector of 40 manually extracted features.

- **Autoencoder features.** Autoencoder features are automatically extracted from the signal—the autoencoder, as described in Sec. V-A can compress data and learn a good representation of features keeping only exciting data. The code size we use is made up of 36 features.

V. Learning Framework

A. Autoencoder

In this section, we describe the autoencoder model based on [18], [19], and [20]. An autoencoder is an artificial neural network used to learn efficient data coding in an unsupervised manner. An autoencoder aims to learn a representation (encoding) for a data set by training the network to ignore signal noise. Given $X=[a_v, a_h, a_l, g_v, g_h, g_l]^T$ As a 6x125 matrix

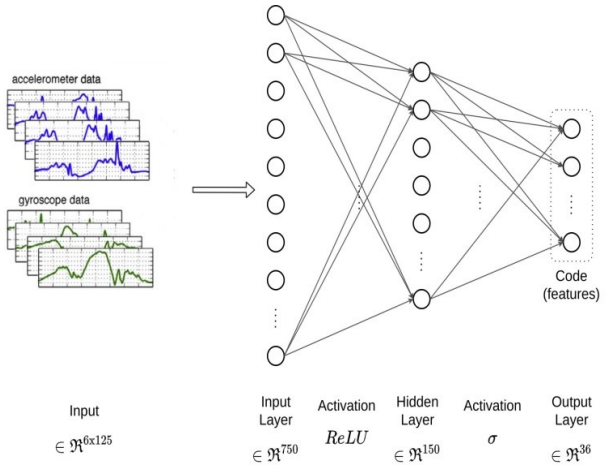

Fig. 4. *(Auto)-encoder structure*

and x as the column vector 750x1 obtained by flattening X, we can define the autoencoder in two blocks. The first block is the encoder which is a function of the input:

$$f_\theta(x) = \sigma(W_2 f'_\theta(x) + b_2) \quad (10)$$

$$f'_\theta(x) = \mathrm{ReLU}(W_1 x + b_1) \quad (11)$$

Where θ_i is a vector of W_i and b_i Which are the weight matrix and the bias vector for layer i. We can derive $y = f_\theta(x)$ as the output of this block. The second block is the decoder, where the input is reconstructed back starting from the code obtained by the encoder

$$g_\theta(y) = \sigma(W_4 g'_\theta(y) + b_4 \quad (12)$$

$$g'_\theta(y) = \mathrm{ReLU}(W_3 y + b_3) \quad (13)$$

Thus, the reconstructed input is $z = g\theta(y)$. We want to minimize the distance between x and z w.r.t, a distance measure with a loss function like MSE.

The main goal of the autoencoder for this application is to extract valuable representations of the input signal in a few features, and this is done by looking at the autoencoder's code, i.e., the output of the encoder block:

$$\Phi e = y \quad (14)$$

The autoencoder follows the structure reported in Fig. 4 and is based on a Neural Network architecture. The encoder is built with an input layer of 6x125 neurons flattened into a vector 750x1, one hidden layer with 150 neurons and *ReLU* activation function, and an output layer, i.e., the code, of 36 neurons with a sigmoid activation function. The decoder replicates the exact structure of the encoder but is reversed, where the activation function on the output layers is the linear activation, i.e., the identity.

B. Convolutional Neural Network

The final CNN architecture proposed in this work is shown in Fig. 3. We decided to start with the architecture presented in a position [2]. We tried to improve it by augmenting the CNN with the encoder-extracted features. CNN architecture is compelling when dealing with images, but they were proven to be good feature extractors for motion data. A CNN is composed of essentially two parts: the first one is in charge of extracting features performing a dimensionality reduction over the input data through a series of convolution and max pooling layers, and the second part of the CNN is responsible for giving the final classification with usually a single fully-connected layer. Original data collected from smartphone sensors are preprocessed accordingly to what was previously described in Sec. III. Our input matrix presented to our CNN is the matrix $X=[a_v^c, a_h^c, a_l^c, g_v^c, g_h^c, g_l^c]^T$ Formed by accelerometer and gyroscopes signals, linearly interpolated at 50 samples/second per channel, with a fixed time window of 2.5s, with OIT and data centering. Note that this matrix has a dimension n × 6 because TensorFlow Convolutional 1D Layers work with input dimension of (*batch_size, samples, channel*).

In detail we have the following stacked layers(CL=Convolutional Layer, ML=Max-pooling Layer, FL= Fully-connected Layer):

- **CL1**: The first convolutional layer performs a Convolution 1D over the input signal. It is composed of 196 filters of size (1×16). Since convolution is defined over all the input channels, we also capture the relation between accelerometer and gyroscope signals in this layer. The stride is set to 1, and the padding is configured as "same" to have the same output dimension as the original one. After the convolution operation, we apply a $ReLU(x) = \max(0, x)$ activation function to learn the non-linear correlation between signals and extract richer features. Furthermore, even if we are dealing with a small architecture, the *ReLU* activation function could prevent vanishing gradient problems, is less prone to overfitting since it induces the sparsity in the hidden units, and is extremely fast to compute, making it perfect when dealing with low computational resources [2].
- **ML1**: After the convolutional layer, a max-pooling or average-pooling layer is usually applied to reduce and summarize the obtained representation. We decided to use a max-pooling layer of size (1 × 4), decreasing by four times the original input shape. We call this new features representation with Φc.

- **FL1:** After the convolutional layer and the max-pooling layer, the output of this layer is flattened into 6272 neurons. At this stage, we decided to concatenate the encoder extracted features Φc with

  the Convolutional ones named Φc, creating our exclusive features vector $\Phi = (\Phi_c, \Phi_e)^T$ that is presented to the final fully- connected layers to perform the classification.

- **FL2-3:** These final layers comprises 64 and 6 dense neurons, respectively. These two last layers are used to perform the classification of the activity. For the FC2, we use the *ReLU* activation function, and for the final category, we apply the soft-max activation function, which computes probability distribution over the predicted classes.

As for optimization techniques, we decided to use dropout and l_2-regularization. These latter techniques are commonly used in these architectures, yielding better performance in the test set and commonly in generalization capabilities, preventing overfitting. We apply a dropout rate of 0.05 and an l_2-regularization of $5e^{-5}$ to the FL2 layer. We have tried different hyperparameter values for these two techniques without substantial changes in classification performance on the test set. Finally, the network parameters are optimized using Adam: a modification of stochastic gradient descent incorporating Momentum. Finally, the network is trained to minimize the Categorical cross-entropy loss function.

VI. Result

A. Heterogeneity Dataset (HD)

In this section, we evaluate performances between previous works and different model architectures and the influences of the preprocessing techniques applied here. Similar to what was done in [2], we carried out from the HD dataset some representative users to test the model and then used the remaining ones to train the model. In this case, we selected users *a* and *b* since we found that they are very representative of all others. The proposed models tend to be less precise with user *a* and more accurate with user *b*. This mainly depends on the user's style in walking, doing stairs, and so on. This way, we can compare results with other works that evaluate unseen users' performances.

In this training and test set settings, we evaluate the best hyper-parameters and results for the models, excluding OIT preprocessing block, since this dataset was collected with a fixed orientation. *sit* and *stand* activities are included. The rest of preprocessing blocks are enabled unless otherwise specified.

TABLE I. HD Classification Results With Different Features Of CNN Augmentation And Data Preprocessing. The Tests Are Made With Our Proposed Architecture, Linear Interpolation And Data Centering, 196 (1x16) Filters, Max-Pool (1x4), 64 Fully Connected Neurons, A Dropout Rate Of 0.05, L2-Regularization Of 5e−5 And Adam Learning Rate Of 2e−5.

Method	Accuracy	Precision	Recall	F1-Score
CNN+No centering	77.0	78.0	75.8	76.8
CNN+No centering+ Manual F.	75.6	76.8	73.9	75.3
CNN+centering+ Manual F.	86.2	86.9	70.2	77.6
CNN+No centering+ Encoder F.	**89.2**	**88.9**	**86.1**	**86.1**

Autoencoder. We performed a grid search to search for the best autoencoder model hyper-parameters. Results are reported in Tab. II, where values that perform well on the validation dataset are written in bold. The best hyper-parameters model used to be the one with a relatively small code size: from 24 to 36 features which is also a good thing as we do not want the autoencoder to learn the identity but only to keep useful information. In these settings, we get an MSE of 0.87.

TABLE II. Grid-Search For Best Hyper-Parameters On Autoencoder

Hyper-Parameter	Values
Code size	{2,3,4,5,6,12,18,24,30,**36**,42,48,54,60, 72}
Batch size	{32,**128**)}
epochs	{**150**,200}

Even if the primary goal for this autoencoder is to automatically extract features from the signal for the direct CNN model, we also implement two simple classifiers to use the autoencoder's feature and check their effectiveness directly.

The first is a K-Nearest Neighbors (KNN) clustering algorithm, and we perform clustering on the encoder's code. The idea is that the autoencoder should extract relevant features that may be similar class by class. We fine-tuned KNN parameters with a grid search, looking for the best values of distance measure and several neighbors. Tab. III shows the values. We select the Euclidean distance measure and five as the number of neighbors. In this case, we obtained an accuracy of 76.2%. From the grid search on KNN, we surprisingly noticed that performances do not vary significantly among different hyper-parameters: most results are less than 5%, far from the best. This may indicate the maximum capability of this autoencoder model, so to obtain better performances, we have to add complexity to the model.

TABLE III. *Grid-Search For KNN Classifier*

Hyper-parameter	Values
Distance measure	{**euclidean**, manhattan, chevy-shev, Minkowski, standardized euclidean, mahalanobis}
Number of Neighbors	{4,5,6,7,8}

The second classifier instead is based on a Feed Forward Neural Network (FFNN). The simple architecture consists of two dense layers of 100 neurons, each with 0.1 dropouts, and a ReLU activation function. At the same time, the last is a thick layer with several neurons equal to the number of classes and soft-max activation function. The network is trained with Adam optimizer to minimize the categorical cross-entropy loss function. In this case, we obtained an accuracy of 81.8%.

CNN Network. We first test how several convolutional filters and dense neurons in CL1 and FL2 will influence classification performances with data centering and manually extracted features. The results are presented in Tab. IV We chose 196 convolutional filters and 64 dense neurons thanks to its balance between accuracy and F1-Score performances, obtaining 86.2% and 77.6%, respectively.

To better appreciate how our preprocessing blocks affect overall model performances, we also try to disable or enable some of them. In Tab. I, we reported our obtained results from experiments. We see that augmenting the CNN with manually extracted feature when data are not centered leads to no significant change in performances; instead, the model obtained nearly 10% more accuracy and precision metrics when also enabling data centering preprocessing with manual features augmented CNN. This proves the benefits of data centering stated previously.

TABLE IV. HYPER-PARAMETERS SELECTION USING HD RESULTS WITH DATA CENTERING AND MANUAL FEATURES AUGMENTED CNN

CNN Filters	FC2 Neurons	Accuracy	F1-Score
196	1024	84.6	75.3
196	512	86.1	75.7
196	**64**	**86.2**	**77.6**
96	1024	82.8	69.9
96	512	84.4	73.7
96	64	89.0	73.0
48	1024	80.0	79.4
48	512	83.7	74.9
48	64	84.0	72.3

However, We could not reach the same performances presented in [2], where the authors obtained an accuracy of 97.6% in the same settings. These empirically confirm that the performances of state-of-the-art models trained with one type of sensor are worse when dealing with the heterogeneity of smartphone sensors. Moving on, augmenting the CNN with the encoder feature increments the model performances, meaning that encoder features are more robust than manual features, as explained previously. To compare our best results in this setting with the ones presented in [3], we decided to perform their *Leave-one-user-out cross-validation* evaluation, consisting of testing the model with data from one user and training with data from all the others in a cross-validation fashion and then averaging the obtained results.

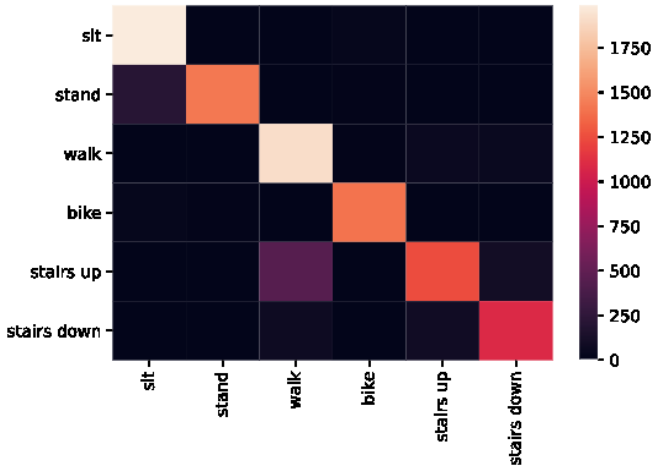

Fig. 5. CNN confusion matrix

We obtained an average F1- score of 85.8% in this evaluation set, beating their best model result of nearly 10% more in the F1-Score metric. This also proves that using users a and b to do our evaluation is a good compromise of the real *Leave-one-user-out cross-validation* evaluation performances, Since we obtain nearly the same results (90.2% instead of 89.2% in accuracy and 85.8% instead of 86.1%), a confusion matrix in this latter setting is reported in Fig. 5. We could see that the model performs nicely overall in all the considered d activities, with some difficulties distinguishing between stand and *sit* and *walk* with *stairs* activities.

B. Oriented Dataset (OD)

With this newly collected dataset, we want to test our models' performance in an actual use case scenario, where smartphones could be placed in different positions and orientations. In this case, we trained the models with the entire HD training set and then used the OD as a test set. It is important to remember that in this case, we apply OIT, so we condensed HD's sit and stand activities into a one class *no activity* category.

TABLE V. AUTOENCODER LOSS IN DIFFERENT SCENARIOS

Scenario	Loss(MSE)
HD + OIT + OD validation	0.75
HD + OIT + OD validation(allpos)	0.82
HD + OD validation	10.42

Autoencoder. As Tab. V confirms, an exciting result is that OIT is necessary when dealing with different orientations. For example, without OIT, we can see that the autoencoder trained and tested on the same data goes from 0.75 to 10.42 MSE, which is more than ten times worse. Furthermore, hand or pocket-up/down data do not inflate the loss too much. This is good because it indicates that the autoencoder is producing robust features.

Tab. VI and Tab. VII show KNN and FFNN evaluation for the best autoencoder with 36 features and the two classifiers with hyper-parameters selected in Sec. VI-A. These results indicate that KNN is more stable and not influenced by the sensor's position/orientation w.r.t. FFNN. Also, the two models preserve evaluation order: on *pouch*, the position gets the best results on both models, while the worse position *hand+pocket*, as we expect.

TABLE VI. KNN EVALUATION ONTO OD BETWEEN SMARTPHONE POSITIONS (POUCH LEFT/RIGHT/TOP/BACK, HAND AND POCKET-UP/DOWN, ALL POSITIONS)

Positions	Accuracy	Precision	Recall	F1-score
Pouch	80.1	86.4	80.7	79.0
Hand+Pocket	79.5	83.7	79.5	79.6
All	80.0	83.8	79.1	78.2

TABLE VII. FFNN EVALUATION ONTO OD BETWEEN SMARTPHONE POSITIONS (POUCH LEFT/RIGHT/TOP/BACK, HAND AND POCKET-UP/DOWN, ALL POSITIONS)

Positions	Accuracy	Precision	Recall	F1-score
Pouch	74.4	84.7	74.4	74.8
Hand+Pocket	67.8	78.0	67.8	70.0
All	69.9	81.3	69.9	72.4

CNN Network. Combining autoencoder features into the CNN, we were able to reach better performances w.r.t the simpler KNN and FFNN previously presented classifiers. For the *hand+pocket* new positions, we noticed a performance drop of nearly 15%, but given the complexity of this new unseen context, we are pretty satisfied with the obtained results. As Tab. 8 confirms, the final proposed learning strategy with smart data preprocessing led to good results even with new problem settings that nearly match real use case scenarios. We also tried to disable OIT in the CNN model, obtaining a drop in performances by almost 45% for all metrics considered. This demonstrates that OIT is an extremely useful preprocessing technique for autoencoder and CNN models.

TABLE VIII. CNN CLASSIFICATION COMPARISONS ONTO OD BETWEEN SMARTPHONE POSITIONS (POUCH LEFT/RIGHT/TOP/BACK, HAND AND POCKET-UP/DOWN, AND ALL POSITIONS).

Positions	Accuracy	Precision	Recall	F1-Score
Pouch	85.3	92.0	73.8	81.9
Hand+Pocket	70.5	79.5	63.0	70.2
All	78.0	84.0	69.0	75.7

VII. CONCLUSION REMARKS

This study presents "in-the-wild" HAR solution for mobile fitness apps. Dealing with considerable heterogeneity and real-world circumstances where a smartphone can be positioned and angled in any way is difficult. Linear interpolation, orientation-independent transformation, and data centering in a preprocessing pipeline can reduce HAR impairments caused by these difficulties. CNN was one of the most promising methods for HAR, using automatic feature extraction and classification. The proposed work outperforms the original Heterogeneity Dataset work by enhancing CNN with robust characteristics like the encoder part of an autoencoder. It is also shown also showed that an orientation-independent transform is necessary to deploy models trained with controlled datasets in real-world contexts with promising outcomes.

In future studies, this model may be tested with a more demanding dataset that includes persons of varying weights, heights, nationalities, and activity paths. It might also be tested with people wearing different shoes and apparel. This work can also integrate with user locations like cars, buses, trains, planes, etc. This study has no opportunity to test open-set categorization approaches, which reject unknown or unobserved behaviors. The proposed algorithm sought to predict learned behaviors even when the user was doing other things with his smartphone using a smartphone app.

This research work has many issues, including poorly optimized NVIDIA Ubuntu OS drivers. Bad GPU memory management crashed the testing computer. When training the model with the same hyper-parameters numerous times, the pre-fetching TensorFlow dataset feature gives us extremely varied model metrics outcomes. After deactivating the option, the system delivered comparable results with the same hyperparameters. Replicating the existing findings from previous research works was tough because the outcomes didn't specify model hyper-parameters and training parameters.

VIII. REFERENCE

[1] H. Blunck, N. O. Bouvin, T. Franke, K. Grønbæk, M. B. Kjaergaard, P. Lukowicz, and M. Wu¨stenberg, "On heterogeneity in mobile sensing applications aiming at representative data collection," in Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication, pp. 1087–1098, 2013.

[2] A. Ignatov, "Real-time human activity recognition from accelerometer data using convolutional neural networks," Applied Soft Computing, vol. 62, pp. 915–922, 2018.

[3] 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 Proceedings of the 13th ACM conference on embedded networked sensor systems, pp. 127–140, 2015.

[4] J. Guo, K. Han, H. Wu, Y. Tang, X. Chen, Y. Wang, and C. Xu, "Cmt: Convolutional neural networks meet vision transformers," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12175–12185, 2022.

[5] J. Yun, D. Jiang, Y. Liu, Y. Sun, B. Tao, J. Kong, J. Tian, X. Tong,M. Xu, and Z. Fang, "Real-time target detection method based on lightweight convolutional neural network," Frontiers in Bioengineering and Biotechnology, vol. 10, 2022.

[6] K. Frank, M. J. Vera-Nadales, P. Robertson, and M. Angermann, "Reliable real-time recognition of motion related human activities using mems inertial sensors," in Proceedings of the 23rd International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS 2010), pp. 2919–2932, 2010.

[7] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones.," in Esann, vol. 3, p. 3, 2013.

[8] "Human activity recognition using smartphones data set." https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones.

[9] "Wisdm's activity recognition using smartphones data set." https://www.cis.fordham.edu/wisdm/dataset.php.

[10] K. Chen, D. Zhang, L. Yao, B. Guo, Z. Yu, and Y. Liu, "Deep learning for sensor-based human activity recognition: overview, challenges and opportunities," arXiv preprint arXiv:2001.07416, 2020.

[11] Y. Vaizman, K. Ellis, G. Lanckriet, and N. Weibel, "Extrasensory app: Data collection in-the-wild with rich user interface to self-report behavior," in Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1–12, 2018.

[12] "The extrasensory dataset: A dataset for behavioral context recognition in-the-wild from mobile sensors." http://extrasensory.ucsd.edu/.

[13] "Heterogeneity activity recognition dataset." https://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition.

[14] M. Gadaleta and M. Rossi, "Idnet: Smartphone-based gait recognition with convolutional neural networks," Pattern Recognition, vol. 74, pp. 25–37, 2018.

[15] K. Kunze, P. Lukowicz, K. Partridge, and B. Begole, "Which way am i facing: Inferring horizontal device orientation from an accelerometer signal," in 2009 International Symposium on Wearable Computers, pp. 149–150, 2009.

[16] A. Henpraserttae, S. Thiemjarus, and S. Marukatat, "Accurate activity recognition using a mobile phone regardless of device orientation and location," in 2011 International Conference on Body Sensor Networks, pp. 41–46, 2011.

[17] C. R. Rao, "The use and interpretation of principal component analysis in applied research," Sankhya¯: The Indian Journal of Statistics, Series A, pp. 329–358, 1964.

[18] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.," Journal of machine learning research, vol. 11, no. 12, 2010.

[19] F. Gu, K. Khoshelham, S. Valaee, J. Shang, and R. Zhang, "Locomotion activity recognition using stacked denoising autoencoders," IEEE Internet of Things Journal, vol. 5, no. 3, pp. 2085–2093, 2018.

[20] X. Gao, H. Luo, Q. Wang, F. Zhao, L. Ye, and Y. Zhang, "A human activity recognition algorithm based on stacking denoising autoencoder and lightgbm," Sensors, vol. 19, no. 4, p. 947, 2019