Going deep into Human Activity Recognition

Elia Bonetto and Filippo Rigotto
Department of Information Engineering, University of Padova – Via Gradenigo, 6/b, 35131 Padova, Italy
{eliabntt94,rigotto.filippo}@gmail.com

***Abstract*—In latest years, thanks to the increased number of smartphones and wearable devices integrating IMUs, Human Activity Recognition (HAR) has become a key research topic in monitored and assisted living either for medical or tracking reasons. First attempts provided manual feature crafting, followed by analysis done either with deep neural networks or other approaches like Hidden Markov models. More recently instead, direct analysis on raw signals has been attempted. Here we continue this trend by exploring some possible approaches with convolutional and recurrent neural networks and look over automatic feature extraction techniques, such as autoencoders. Most of the datasets in this field are highly imbalanced and some classes lack of enough data. To face this, we propose two augmentation techniques for rebalancing. Finally, we introduce new ways and metrics to select the best learning epoch to address overfitting and get the best learning results overall. Our tests confirm that augmenting the initial dataset is worth the effort, and we achieve performance that surpass what is declared for it. Moreover, we discovered that working with raw signals in the sensor reference frame is better than working with their transformation to the body frame. As for encoded data by means of autoencoders, we could not find any performance improvement: in some cases, worse results are obtained.**

***Index Terms*—Activity recognition, inertial sensors, machine learning, neural networks, autoencoders, deep classification**

I. Introduction

PHYSICAL activities recognition, commonly referred as Human Activity Recognition (HAR), has gained momentum as a key research area nowadays. Tracking and detecting human activities is a relevant task for both trained personnel, e.g. first responders and medical or military services, and for common people: for fitness, for prevention or quick notification of falls, for example in case of elderly people assistance.

HAR can be performed visually with the aid of many cameras pointed to track people motion but this approach has privacy issues and only works in restricted or indoor areas. Instead, the use of inertial sensors (IMUs) worn by the users has less drawbacks, it is more robust (for example, it does not rely on the field of view of the camera), cheap and more ubiquitously deployable thanks to the huge spreading of sensor-equipped smartphones and watches [1]. IMUs are typically composed of accelerometers, gyroscopes and magnetometers that sense and periodically sample linear acceleration, angular velocity and magnetic field variations in the three spatial directions, all within the size of an open hand. To analyze and classify the data stream offered by these sensors, *features* extraction is needed and can be performed both manually, as in the traditional approach by means for example of the Fourier transform (FFT) [2] or statistical analysis, or by automatic learning through neural network (NN) architectures, a rising trend in the literature [1].

In this work, we focus on a specific dataset to evaluate the current state of the art, comparing deep networks composed of convolutional and recurrent layers, and we propose some new architectures made as a combination of the previous models or built around the autoencoder (AE) concept to enable automatic feature extraction. Furthermore, we introduce some balancing techniques for the dataset, trying to achieve better results overall. Our proposals perform better with respect to the state of the art on the considered dataset. We also manage to use a lighter version than the original proposal for some of the considered networks. Implementation is modular, to allow for the use with other datasets, and open-source on GitHub, to allow external contributions and future developments.

This work's contributions can be summarized in:

- probing usefulness of sensor's frame signal processing
- exploring data augmentation techniques that push forward prediction accuracy
- defining and combining (new) models based on established networks that can classify activities from inertial sensor's raw measurements
- exploring new metrics to select the best training epoch for models, different from standard setups
- making use of the autoencoder architecture to extract features to be fed to deep networks, support vector machines and logistic regression

This report is structured as follows. In section II we review the current state of the art. Our system's pipeline is defined in section III, preprocessing steps on the original signals' data of the reference dataset are detailed in section IV. Analyzed architectures and learning parameters are presented in section V, while in section VI we outline our results.

II. Related work

Activity recognition is a prolific research field and many techniques and algorithms have been used to tackle the subject. Delving into sensor-based HAR, the trend in the literature until a few years ago, as per [3], was to manually craft features and then process them by means of Hidden Markov Models (HMM) [4], Principal Component Analysis (PCA), Support Vector Machines (SVM), Bayesian Networks (BN) [5] or Random Forest (RF) ensembles [6]. Features are extracted using filters, Fourier transforms, moments or other statistical properties (like mean and variance) of the signals [2]. In this case a heavy preprocessing phase is needed, and these manual features, apart from being most often poorly generalizable, may exclude some important information that can only be extracted by using automatic methods [1]. Moreover, these methods focus mainly on single-sequence classification, often lacking the study of the temporal correlation between signals since they rely on per-sequence handcrafted features [7]. Recent advances in machine learning paved the way to raw signal analysis, automatic feature extraction and classification. In this regard, Wang *et al.* [1] very recently surveyed the literature collecting both model architectures and popular datasets.

Deep convolutional neural networks (CNN) could learn much more high-level and meaningful features while achieving unparalleled performance: their key advantages are the ability to capture local dependencies and resilience to scale changes [8] thanks to their ability to extract hidden information from data. Temporal 1D convolution is successfully used by Chen and Xue, [9] who employ a deep CNN with small kernels on data that come only from a single accelerometer. Even if it is a promising approach, not considering also gyroscope data may lead to underestimated results: accuracy is expected to be lower when dealing with stationary activities. The authors in [10] do the same, but they consider accelerometer vectors' magnitude instead of raw 3D data to reduce rotational interference. Moya Rueda *et al.* [11] extend the computation considering multiple sensors and organizing data in sliding windows among the set of sensors in parallel. The authors in [12] collect and merge data from 5 sensors fixed to different parts of the body, and compare a CNN, a MLP and a SVM: the deep network is both faster and more accurate. 2D convolutions are used by Bevilacqua *et al.* [13] to account for spatial and temporal dependencies among signals. Accelerometer and gyroscope data is stacked and organized in overlapping windows before being fed into a 3-layers convolutional network with small kernels and pooling layers, dropout [14] and a final 3-layers fully-connected network. This novel approach leads to even higher accuracy values. A further improvement by Ha *et al.* [15], [16] consists in separating each sensor's data with padding and adjusting the filter size to not simultaneously perform convolution over different sensors.

The introduction of recurrent networks (RNN), in particular of Long Short-Term Memory (LSTM) [17] and Gated Recurrent Unit (GRU) [18] cells, allowed to learn temporal sequences dependencies more flawlessly by holding memory of past values. Thanks to this, recurrent modules are good learners for IMU sensor's data that clearly depends not only from a single value, but from a sequence [19], [20]. Both LSTM and GRU have been developed to avoid the vanishing/exploding gradient problem of RNNs, and the difference between them is the number of available *update* gates.

In this field most works use LSTM networks [21]. Simple vanilla approaches like stacking LSTM gates one after another showed an improvement over previous CNNs methods thanks to their relation to time sequences, even if working on plain raw signals. An alternative is to use a *bidirectional* network, that is, two LSTMs layers working one on the original sequence (learning from future) and the other one on the reversed version (learning from past), trying to be more robust and essentially doing data augmentation inside the network [7]. This approach is the less adopted because it is highly dependent on the number of units of the LSTM cell and so to the input size and the different datasets. Last, experiments mixing convolutional layers before LSTM(s) classification layers took place [22]. Taking advance from both representation brought by CNNs and temporal dependency by RNNs, results show an improvement with respect to previous methods: a proof that feature extraction techniques before RNNs could be a way to obtain major improvements on HAR and other time-related tasks. GRU layers have been less used for this task, probably because they showed lower overall performance [23], [24].

III. Processing pipeline

As mentioned in the introduction, this work uses sensor data collected by the Institute of Communications and Navigation, German Aerospace Center (DLR)[1], base of the work done in [2]. Nonetheless, source code is modular, to enable the possibility to use other datasets, like the well known from UCI[2]. The processing phase of raw signals data, needed to build an usable dataset to feed the networks, is kept at a minimum without using for example filters or noise reduction techniques to avoid distorting too much the dataset, to further prove the usability of non-elaborated features. In the original version, the dataset is not balanced w.r.t. the classes frequency. To contrast this, after the generation of training and test sets, augmentation is experimented and used to create two further dataset versions: using a hand-crafted method consisting of random data rotation and shuffling, and an algorithm suited for this task such as ADASYN [25].

In the first part of this work, classification of data is performed using common architectures inspired from the literature: 1D and 2D CNNs, LSTM and GRU-based RNNs.

A second part of the study involves the generation of features from data thanks both to a stack of convolutional layers and to autoencoders, made of convolutional, LSTM or both kind of layers. Classification is then performed by means of the previous defined networks, generating a mixture of possible configurations, and of other classification algorithms like SVM and logistic regression (LR), using a stochastic gradient descent (SGD) optimization method.

To select the best training epoch we have introduced some brand-new metrics to counteract eventual overfitting problems, variability of the training progress and to take into account not only accuracy but also precision and recall scores. Validation split was not carried out due to the small dataset size and the fact that augmentation only occurs on the training set: validating over an augmented dataset could have led to overfitting the data, resulting in poor performance.

A visual representation of the full process is in Fig. 1.

IV. Signals preprocessing

The dataset contains IMU sensor measurements of several scheduled movements and activities taken and labelled by different people. The XSENS MTX IMU device is positioned on the belt of the user, and collects data at a rate of 100Hz and

[1]https://www.dlr.de/kn/desktopdefault.aspx/tabid-12705/22182_read-50785
[2]https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

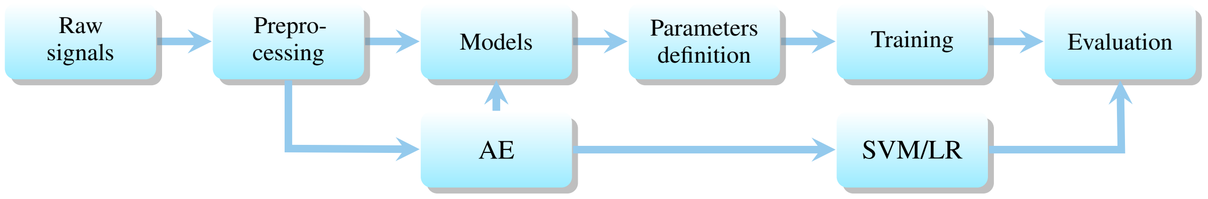

Fig. 1: Full processing pipeline from raw signals' dataset to networks evaluation.

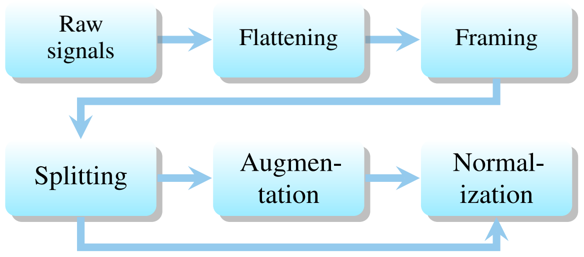

Fig. 2: Data preprocessing steps.

each measure consists of the snapshot time plus raw readings of an accelerometer, a gyroscope and a magnetometer.

Each sensor outputs 3D data, according to the device reference frame, and also computes the necessary transformation matrix $\boldsymbol{T}$ of shape 3×3 to change the reference frame into a body-aligned frame. Mathematically, for each sensor data $\boldsymbol{v}_s$:

$$\begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix} = \boldsymbol{v}_b = \boldsymbol{T}\boldsymbol{v}_s = \boldsymbol{T}\begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix}$$

The original 17 tracked activities are grouped and reduced to 8 classes, mislabeling in classification are fixed. The composition of samples in the dataset after relabeling is reported in the second column of Tab. 1.

Tab. 1: Minutes for each activity in several versions of the dataset: the original, after framing and after augmentation.

		Time		
	Activity	orig.	framing	augm.
🟦	running	15	30	68
🟧	walking	72	144	144
🟩	jumping	8	15	64
🟪	standing	121	241	241
🟫	sitting	59	117	117
⬜	lying	28	56	76
🟨	falling	2	4	60
🟦	transition	60	—	—
	total	365	609	772

Transitions are pruned from the dataset, and the rest of the data is sorted by activity and then framed in windows of length 128 and an overlapping rate of 50%. This also implies padding sensor data with zeros to the nearest subsequent multiple of 64, to have entirely filled windows. Updated times for each activity are in the third column of Tab. 1.

The 70-30% split of the dataset to obtain training and test sub-sets is done before the eventual augmentation of data, which is performed only on the training set: it is easy to notice from Tab. 1 that classes in the original dataset are not balanced. Two different techniques have been applied to balance them:

- using ADASYN [25] from *imblearn*[3]: an adaptive algorithm that generate samples according to a density distribution, computed for every minority class. It forces the learning algorithm to focus on difficult examples
- hand-made rotation of some random samples (axis and angle are both randomized) and shuffling inside a window

In both cases the dataset is rebalanced (only the training set) by bringing the three less-represented classes to have around 30% of the number of samples of the most represented class, which is *standing*. There are some caveats: activities *lying* and *sitting* are not randomly rotated as this may cause labeling errors due to confusion with similar activities like *standing*, and activities *jumping* and *falling* are not subject to permutation because there may be temporal correlation inside the window (a jump read reversed in time may be confused with a fall). Adjusted times for both processes are in the fourth column of Tab. 1.

Eventually, data is normalized using training set's mean and standard deviation: this is a common procedure in machine learning that might improve accuracy and training.

V. Learning framework

We developed networks that works directly processing raw framed signals, and networks that combine automatic feature extraction prior to classification. For "plain" models we have ultimately chosen four alternatives, based on, but not equal to, previous works in the literature, as they differ in combination and number of layers, kernels size and other parameters. For the second part, we explored the combination of CNN and LSTM layers, and we also implemented three alternatives through autoencoders, to be used with SVM, LR and the previously defined networks as classifiers, except one for reasons that will be clear in section V-B2.

Combining all these numbers to the six prepared datasets — with and without normalization, manually augmented, augmented using ADASYN — for each of the two reference frames, the total number of configurations is $[5+3\times4]\times6\times2 = 204$[4], excluding SVM and LR tests.

Practically, all the networks are implemented as Keras [26] models (using TensorFlow [27] backend). Training parameters like the optimizer (SGD[5], RMSprop [29] or Adam [30]), the loss type, the learning rate, its decay and momentum, the

[3]https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.ADASYN.html
[4]5 networks, 3 AEs over all but one networks, 6 datasets, 2 reference frames
[5]For more information, we refer the reader to [28].

number of epochs and batch size are specified and saved separately from models: this modularity allows to perform tests on the same model with different configurations.

A. *Plain models*

Here we review the best models we selected. These are only a small part of all the structures we investigated: much more models can be found in the companion Jupyter notebooks, especially regarding fully connected networks (not included here) and CNNs, since experiments with LSTM and GRU units are mostly related to the number of nodes in the cell and to dropout rate values. All these models are learned with the Adam optimizer and *categorical crossentropy* loss. If not differently specified, default values are employed for learning rate (0.001), decay rate (0) and other parameters. Investigating hyperparameters in this kind of networks is still a major problem, as pointed out in [7]. Our values agree with the guidelines defined in the paper. Dropout modules are used for regularization and to avoid overfitting data.

1) 1D CNN: We developed networks based on one or two convolutional layers, plus one or two dense (fully connected) layer followed by *softmax* activation to have class probabilities, with optional *L2* regularization on convolution kernels and dropout after every layer. As per standard practices, each convolutional layer is followed by batch normalization [31], ReLU activation and max-pooling. We additionally included models from surveyed papers [9], [11], whose drawbacks have been evidenced in section II. A single convolutional layer is not capable to learn from data, overfitting occurs and validation loss always grows. Adding more than one dense layer at the end leads to longer learning time with no substantial gain in accuracy. Regularization is not much helpful in CNNs, what keeps the loss stable is dropout, for which the rate is set by trial and error. The selected model, which has the highest overall and class-wise accuracies, uses two layers, dropout and a final dense layer, and is reported below.

```
def Conv1D_2C1D_model(input_shape, num_classes=7):
  return Sequential([
    Conv1D(filters=64, kernel_size=5,
         input_shape=input_shape),
    BatchNormalization(axis=1),
    Activation('relu'),
    Dropout(0.3), # rate
    MaxPooling1D(pool_size=2),
    Conv1D(32, 5),
    BatchNormalization(axis=1),
    Activation('relu'),
    Dropout(0.3),
    MaxPooling1D(2),
    Flatten(),
    Dense(num_classes, activation='softmax')
  ], name='Conv1D-2C1D-do0.3')
```

2) 2D CNN: We investigated networks from papers [13] and [15]. 2D convolution can be performed by reshaping data to have the three sensors' signals stacked. Input data has shape `(?,9,128,1)` where 128 is due to the framing operation and the last number is the number of channels, only 1 in this case. While the first network is not brilliant and in our preliminary tests achieves only 86% accuracy, the second instead is a promising setup: as mentioned in section II, 95% accuracy is reached by operating the trick to interpose zero-padding to avoid 2D convolution kernels to sweep more than one signal at a time. Due to padding, input shape becomes `(?,18,128,1)`. As the selected model, it is reported below.

```
def Conv2D_Ha_model(input_shape, num_classes=7):
  return Sequential([
    Conv2D(filters=32, kernel_size=(4,4),
         activation='relu', input_shape=input_shape),
    MaxPooling2D(pool_size=(3,3), strides=(1,1)),
    Conv2D(64, (5,5), activation='relu'),
    MaxPooling2D((3,3),(1,1)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
  ], name='Conv2D-Ha')
```

3) LSTM: Many tests brought us to set on the model with two LSTM layers, due to improved performance w.r.t. the versions with only one layer. The *bidirectional* configuration is highly dataset-related and very difficult to tune, especially in the number of cells [7]. Models with three layers are too slow and do not improve on the selected model, for this dataset.

Different to what is done in [20], we dropped one LSTM layer and added dropout instead of manually inserting an L2 regularization term in the structure of the LSTM.

```
def TwoLSTM_model(input_shape, num_classes=7):
  return Sequential([
    LSTM(512, return_sequences=True,
         batch_input_shape=input_shape),
    Dropout(0.2),
    LSTM(512, return_sequences=False),
    Dense(num_classes, activation='softmax')
  ], name='TwoLSTM')
```

4) GRU: In this case the one-layer model achieves the same performance of the two-layers one, but in a much longer time (circa 3×). The model with three GRUs instead overfit the data, reaching an accuracy peak slightly lower from the two-layer model, and then performance degrades and loss grows. In this last case a study on the learning rate decay would be worth, but nonetheless taking into account the LSTM network's results, the fact that the final model already reach nice overall performance, plus the time needed to train such a network leads in this case to drop further experiments on this architecture.

```
def TwoGRU_model(input_shape, num_classes=7):
  return Sequential([
    GRU(512, return_sequences=True,
         input_shape=input_shape),
    Dropout(0.2),
    GRU(512, input_shape=input_shape),
    Dense(num_classes, activation='softmax')
  ], name='TwoGRU')
```

B. *Combined models*

We explored the possibility of automatically learn features from raw signals by combining a CNN with a LSTM cell and by means of several autoencoder architectures.

1) CNN-LSTM stack: One "mixed" model is obtained by stacking together convolutional and recurrent layers. The scope is to get the best from both, so to get sort of a "preprocessing" step thanks to convolutions, with the goal of learning features that are then handled and classified by LSTMs, thanks to their time-related capabilities. In this case the input has been reshaped in folds of size `(?,4,32,9)` to use

`TimeDistributed`, that apply the same `Dense` operation to every timestep of a 3D tensor[6]. Moreover, this resulted in an overall speed-up of LSTM layer's training time.

In [22] the defined network is bigger, featuring two more convolutional layers and one more LSTM, and lack the presence of either dropout and pooling layers. Moreover, the settings about learning and decay rate are different: we use respectively 1e–4 and 0 instead of 1e–2 and 0.9.

```
def CNN_LSTM_model(input_shape, num_classes=7):
  return Sequential([
    TimeDistributed(Conv1D(256, 1,
         activation='relu'), input_shape=input_shape),
    TimeDistributed(Conv1D(256, 3,
         activation='relu')),
    TimeDistributed(Dropout(0.1)),
    TimeDistributed(MaxPooling1D(2)),
    TimeDistributed(Flatten()),
    LSTM(128),
    Dense(num_classes, activation='softmax')
  ], name='CNN-LSTM')
```

2) Autoencoders: They should provide an improved representation of the input data by using learned encoded features instead of raw data. They have been constructed by employing only convolutional layers, only LSTM cells and both of them. As a first step, data has to be de-framed by a flattening-like operation obtaining sets of shape `(?`[7]`,1,9)` from `(?,128,9)`, that is, we process each single measure separately. We used the defined networks to perform learning also over the original framed dataset and by expanding the last dimension instead of the first one (obtaining a final shape of `(?,1,128×9)`), but without obtaining good results.

```
def CNN_AE_model(input_shape, num_features):
  return Sequential([
    # encoder
    Conv1D(128, 1, activation='relu', padding='same',
         input_shape=input_shape),
    Conv1D(64,  1, activation='relu', padding='same'),
    # decoder
    Conv1D(64,  1, activation='relu', padding='same'),
    Conv1D(128, 1, activation='relu', padding='same'),
    Conv1D(num_features, 1, activation='softmax')
  ], name='CNN_AE')
```

```
def LSTM_AE_model(input_shape, num_features):
  return Sequential([
    # encoder
    LSTM(128, activation='relu',
         return_sequences=True,
         input_shape=input_shape),
    LSTM(64,  activation='relu',
         return_sequences=True),
    # decoder
    LSTM(128, activation='relu',
         return_sequences=True),
    TimeDistributed(Dense(num_features,
         activation='softmax'))
  ], name='LSTM-AE')
```

```
def CNN_LSTM_AE_model(input_shape, num_features):
  return Sequential([
    # encoder
    Conv1D(128, 1, activation='relu',
         input_shape=input_shape),
    Conv1D(64,  1, activation='relu'),
    # decoder
    LSTM(128, activation='relu',
         return_sequences=True),
    TimeDistributed(Dense(num_features,
         activation='softmax'))
  ], name='CNN-LSTM-AE')
```

After autoencoders have been trained, the best model with respect to accuracy is selected, the decoder side is discarded, the encoded sequences are reshaped back to the original shape and then fed into the CNN-LSTM stack and each of the classification networks defined in V-A, except the 2D CNN: its particular configuration and padding as a fundamental and logical preprocessing step make it senseless to use this network coupled with autoencoder features.

We further tried also to perform classification through SVMs and LR thanks to a SGD approximation, defined in *scikit-learn*[8]. In this case we find out that *alpha*, the multiplier of the regularization term that is also used to compute the initial learning rate, should be in the range $[1e{-}5, 1e{-}3]$, with most of the best performance obtained using 1e–4. We combined both SVM and LR with different kind of regularization: L1, L2 and Elastic Net[9] have been tested. Overall the possible combinations are $3 \times 2 \times 6 = 36$, to be added to the previous 204 tests.

```
sklearn.linear_model.SGDClassifier(loss='hinge',
     penalty='l2', alpha=0.0001)
# for loss: hinge -> SVM, log -> LR
# for penalty: 'l2', 'l1', 'elasticnet'
```

C. Metrics

As a multi-class classification problem, to select the best training point (i.e. epoch), beside accuracy, *precision* and *recall* metrics are equally meaningful. We very briefly recall their definitions, using Tab. 2: the ratio of true positives to the total predicted positives, and the ratio of true positives to all predictions classified in the same class. The *F1-score* is the harmonic average between the two. Mathematically,

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN} \qquad F1 = 2 \times \frac{P \times R}{P + R}$$

Tab. 2: Sample confusion matrix: actual vs predicted class.

	Pos	Neg
Pos	TP	FN
Neg	FP	TN

Because of the imbalanced dataset, overall accuracy may be misleading since less frequent classes have low impact on the final value, even if for example classification for these samples fails most of the times. This is enhanced by the fact that test

[6]This is implicit since Keras 2.0 but we prefer to specify it anyway.
[7]"?" is used as wildcard to indicate an unknown number of samples as it depends on the dataset.

[8]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[9]A linear combination of L1 and L2 regularization terms.

set cannot be augmented. Because of that, we will focus both on global and per-class metrics, hoping to find a model that maximize each of accuracy values.

We introduced these metrics in the training and evaluation phases for all our models, and set up to save a snapshot of the model during training when any of the following quantities (computed on the test set) reach a new maximum:

- accuracy: A
- accuracy over loss: $\mathrm{AoL} = A/L$
- sum of acc., precision and recall: $\mathrm{APR} = (A + P + R)$
- the previous sum, over loss: $\mathrm{APRoL} = (A + P + R) \;/\; L$

Additionally, the model is also saved when at the end of an epoch accuracy is not higher but near current maximum, and loss is lower than the current stored value.

We choose the sum strategy to globally account for each of the metrics. We understand that having a precise scope for the work, for example like fall detection[10], this reasoning may change a lot, due to possible allowance of false positives and absolute inadmissibility of false negatives. But this is a general work without a detailed scope, and it is indifferent for our objectives to have higher precision or recall.

We choose the division by the loss to account for the fact that it is also an important value to track during training: more often a model with slightly less accuracy but lower loss is more powerful than a model with slightly higher accuracy and high loss. This is strictly related to the concept of overfitting.

VI. Results

The 12 generated datasets, as described in section IV, are:

- w.r.t. two reference frames (sensor and body)
- with and without normalization
- optionally augmented manually or with ADASYN

The results are explained in a top-down fashion to progressively prune out combinations. Subsequent considerations still apply to previously excluded parts. The complete set of results can be generated using the notebooks. Note that for Keras models and scikit-learn's SGD implementations accuracy equals weighted recall, so we will use this as comparison metric.

A. Reference frame

Comparing the same networks, trained over the same type of dataset, we can note how performance are superior with sensor-referenced data, instead of using data referenced to the body frame, and it is easy to see how even plain global accuracy is lower on such datasets, an example is in Tab. 3. This is a surprising aspect because the coordinate transformation should act as a stabilizer to equalize measurements and generalize them as noted in [4].

B. Normalization

As common practice suggest we performed normalization on datasets using training set's mean and standard deviation. In general, by looking at global accuracy, there is no strict rule to say that using it is better than leaving data as is. The

[10]Fall detection may be re-thought as a binary classification problem.

Tab. 3: Best accuracy value (%) on some datasets, body (`B*`) and sensor (`S*`) reference frame. *Mixed* is CNN-LSTM, due to space constraints.

Dataset	Conv1D	Conv2D	2LSTM	2GRU	Mixed
BADA	98.062	95.528	98.085	98.085	98.447
SADA	**99.194**	**97.898**	**99.194**	**99.159**	**99.451**
BFRA	97.933	95.166	98.120	98.143	98.435
SFRA	**99.159**	**98.073**	**99.159**	**99.113**	**99.358**

results are so near that could even be linked to stochasticity of trainings and to highly imbalanced classes. From Tab. 4 we can note how there are cases like CNN-LSTM and SADA(n) where normalized dataset goes better, and cases like TwoGRU and SAHC(n) where it is the opposite. So we cannot come up with a definitive answer for this normalization question.

Tab. 4: Best accuracy value (%) for sensor-referenced datasets: ADASYN-augmented (`SADA`), manually augmented (`SAHC`), normalized but not augmented (`SFRA`). Not normalized versions of these three datasets are `SADAn`, `SAHCn` and `SFRAn`.

Dataset	Conv1D	Conv2D	2LSTM	2GRU	Mixed
SADA	**99.194**	97.898	99.194	99.159	**99.451**
SADAn	99.066	98.003	**99.241**	**99.288**	99.183
SAHC	97.957	**98.284**	99.206	97.840	99.416
SAHCn	99.043	98.260	99.229	99.253	99.183
SFRA	99.159	98.073	99.159	99.113	99.358
SFRAn	98.984	98.272	98.564	99.206	99.113

C. Augmentation

Evaluation is problematic also in this section: from Tab. 4 we can note how augmented datasets do not *always* perform better than non-augmented ones. What is noted instead is that *at least* one of the two augmented versions performs better than "plain" ones. This behavior was expected, because augmentation is performed to enhance less represented classes and as such they do not have great influence on a global metric. A proof can be seen in Fig. 3 and Fig. 4 where clearly *jumping* and *falling* reaches way better per-class accuracies. Moreover, ADASYN achieve better results w.r.t. manual augmentation in all networks but Conv2D, considering both normalized and non-normalized versions. This also proves augmentation is useful and worth. Paired with previous results, here are the best models overall: TwoGRU over SADAn, CNN-LSTM over SADA.

D. Models and metrics

By looking at Tab. 5, we notice how:

- defined metrics may couple together: this happens when the model is saved at the same epoch according to two different metrics. Most of the times A couples with APR while AoL couples with APRoL, but there may be exceptions like CNN-LSTM on SAHCn, where A differs from APR.
- generally, accuracy A (weighted recall) gives best results.
- the best model is CNN-LSTM.

Tab. 5: Saving epoch, per-class accuracy, global accuracy, weighted and standard precision and recall (%), for the two most prominent models. Each value is reported according to the four metrics presented in section V-C (top to bottom, same order).

Model	E	🟦	🟧	🟩	🟪	🟫	⬛	🟨	Acc.	W Prec.	Prec.	Recall
2GRU	101	100.000	99.014	93.897	99.588	99.637	99.874	87.273	99.288	99.284	99.293	99.282
	84	100.000	99.113	90.141	99.706	99.577	99.748	90.909	99.264	99.262	99.293	99.247
(SADAn)	101	100.000	99.014	93.897	99.588	99.637	99.874	87.273	99.288	99.284	99.293	99.282
	84	100.000	99.113	90.141	99.706	99.577	99.748	90.909	99.264	99.262	99.293	99.247
Mixed	27	100.000	99.211	92.488	99.794	99.758	99.874	94.545	99.451	99.454	99.464	99.452
	21	100.000	99.113	92.488	99.676	99.819	99.874	92.727	99.381	99.382	99.394	99.382
(SADA)	27	100.000	99.211	92.488	99.794	99.758	99.874	94.545	99.451	99.454	99.464	99.452
	21	100.000	99.113	92.488	99.676	99.819	99.874	92.727	99.381	99.382	99.394	99.382

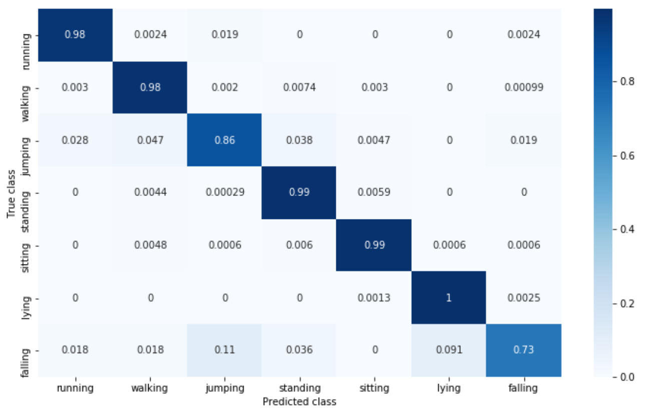

Fig. 3: Confusion matrix of Conv2D model, SAHC dataset.

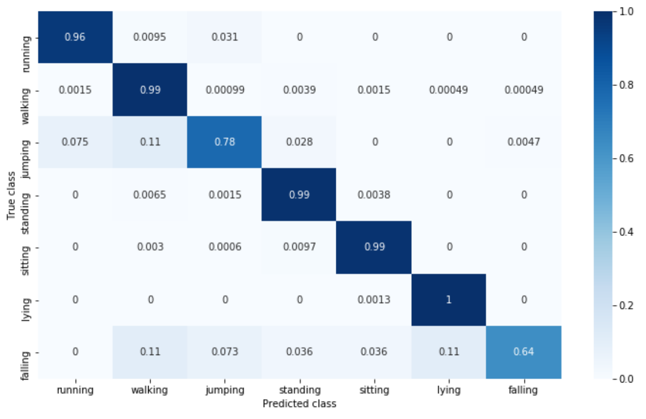

Fig. 4: Confusion matrix of Conv2D model, SFRA dataset.

In general, our models achieve high accuracy within a small number of epochs ranging from a few to a hundred or so. 2D CNN and CNN-LSTM exhibit slight overfitting (see Fig. 5), that could be tackled with additional regularization or learning rate's decay. Anyway, models are saved before this happens.

Only comparing the two best models in Tab. 4, CNN-LSTM performs slightly better, even if the 0.163% gained (from 99.288% to 99.451%) is a stunning 22.9% of the remaining available accuracy. Taking the maximum and minimum values, we span from 97.840% to 99.451%, covering about 74.6% of the remaining achievable accuracy.

As per Tab. 6, comparing results with the work done in [2],

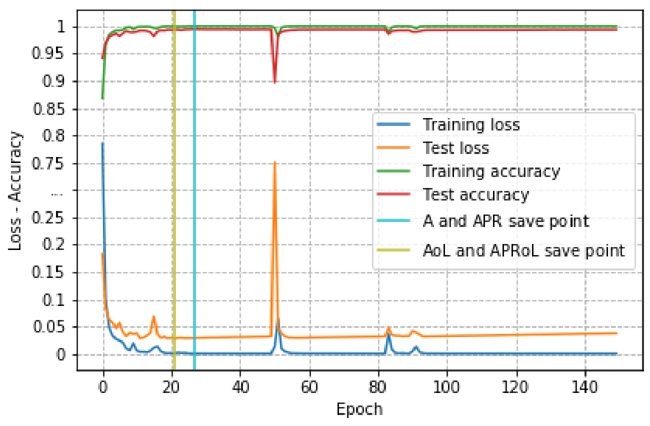

Fig. 5: CNN-LSTM, SADA dataset: training accuracy and loss.

which is based on Dynamic unrestricted Bayesian Networks, we achieve better precision and recall in most of the classes, especially in the augmented ones, with almost all our best configurations.

Finally, in general TwoLSTM, TwoGRU and CNN-LSTM are the models that perform, regardless of normalization and augmentation, *always* better than Conv2D-Ha and *almost* constantly better than Conv1D-2C1D.

As for timings, we notice in Tab. 7 that recurrent networks are much slower with respect to convolutional models, yielding only a low overall performance increase. Notably, thanks to pooling and convolutional layers, the combined model actually achieves the best training speed overall, letting us conclude along with previous considerations that it is the best in our pool of analyzed networks.

E. Autoencoders

Training of autoencoders is good and fast: as reported in Tab. 8, they reach 99% accuracy on test set with a loss of 0.25 in less than five epochs.

Instead, application of encoded data to the networks have not brought the expected improvements in any datasets or methods we used. In general we noted comparable, and in some cases slightly lower, performance than the same networks trained on the original dataset. A few examples can be seen in Tab. 9.

Apart from being time costly to permute all of them, applications of SVM and LR in combination with L1, L2 and

Tab. 6: Comparison between our best networks and datasets combination and original results from [2] based on Dynamic Bayesian Networks: precision (first) and recall (second) values for each class. Best results in bold (virtually rounded).

Model	🟦 running		🟧 walking		🟩 jumping		🟪 standing		🟫 sitting		⬛ lying		🟨 falling	
Conv1D-2C1D	97.91	99.76	**98.87**	99.11	**98.99**	92.02	99.33	**99.65**	**99.88**	99.70	99.37	**99.37**	**89.80**	80.00
Conv2D-Ha	96.95	97.64	**98.28**	98.37	90.20	86.39	98.97	**98.94**	**98.32**	98.73	99.25	**99.62**	80.00	72.73
TwoLSTM	98.60	**100**	**99.41**	99.16	**98.03**	**93.43**	99.15	**99.50**	**99.70**	99.52	**99.87**	**99.37**	**86.44**	92.73
TwoGRU	99.06	**100**	**99.26**	99.01	**95.69**	**93.90**	99.44	**99.59**	**99.52**	99.64	**99.50**	**99.87**	**96.00**	87.27
CNN-LSTM	97.47	**100**	**99.41**	99.21	**98.99**	92.49	**99.56**	**99.79**	**99.82**	99.76	**100**	**99.87**	**92.86**	94.55
BN [2]	**100**	93	98	**100**	93	93	**100**	98	97	100	100	98	80	100

Tab. 7: Mean sample processing time and epoch duration, `SAHC` dataset, 27611 samples. Benchmark on dual-core 2.30GHz CPU and NVIDIA® T4.

	Conv1D	Conv2D	2LSTM	2GRU	Mixed
Sample	235us	219us	3ms	2ms	**175us**
Epoch	6.6s	6.0s	75s	53.8s	**4.8s**

Tab. 8: Accuracy and loss of trained AEs, for some datasets.

Dataset	CNN		LSTM		CNN-LSTM	
BAHC	0.996	0.250	0.994	0.250	0.996	0.250
BFRA	0.994	0.812	0.993	0.812	0.993	0.812
SADA	0.996	0.581	0.955	0.586	0.997	0.581
SAHC	0.996	0.385	0.995	0.385	0.995	0.385
SFRA	0.997	0.793	0.994	0.793	0.996	0.793
SFRAn	0.986	12.101	0.986	12.101	0.998	12.099

Elastic Net regularization terms do not provide good results: models tend to well learn long and stable activities, while being worse with *running*, *jumping* or *falling* (see Fig. 6). We reached a top value of 90% global test accuracy on a specific combination of AE and LR model. Furthermore, there is no clear linking between performance and LR or SVM or with respect to regularization terms, which makes essential to loop all combinations to see the best one.

The pattern of results obtained over each dataset and network is in accordance with our previous findings: these methods

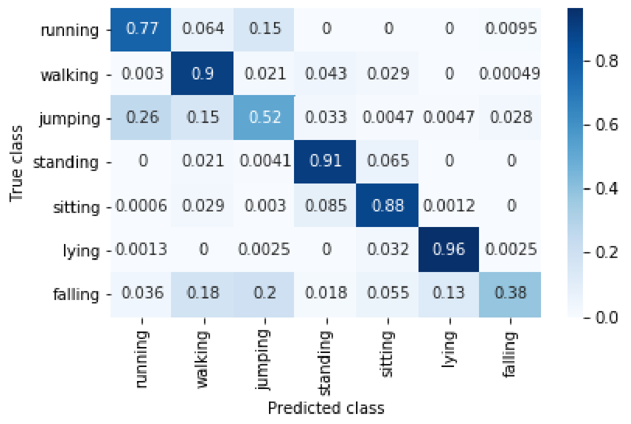

Fig. 6: SVM with L2 regularization, CNN-LSTM encoding, SAHC dataset.

Tab. 9: Comparison between models trained on encoded data and on the original data.

Model	Dataset	AE	DS Acc.	AE Acc.
Conv1D	SAHC	LSTM	99.416	98.751
TwoGRU	BFRA	CNN	98.085	96.298
Mixed	SADA	CNN-LSTM	99.451	98.459

cannot compete with the previously defined ones, and in our implementation it is better to process raw signals than to build methods on encoded features.

VII. Concluding remarks

We delved into HAR testing combinations of learning methods and datasets. We found both the applied preprocessing steps to be useful, especially the augmentation techniques. Many more of them can be applied, like adding noise to existing data, several ADASYN variations, more effective manual operations or with different quantities and class relative percentages. Moreover, it is not clear enough whether normalization is necessary and useful in this particular case. Our study lack an investigation about learning and decay rate and newly introduced metrics have not brought significant improvements in our experiments: probably this may change introducing a validation subset.

Transformations between sensor and body frames are probably much more effective when facing a multi-sensor system and less when dealing with only a single device. If this trend is confirmed, it would be interesting, in a multi-sensor scenario, to separately classify activities for each sensor and then merge together the results.

Evaluations evidenced the mixed convolutional–recurrent architecture to be the best network to tackle the problem. This confirms that mixing architectures is a promising way to proceed both in terms of performance and time efficiency, which is greatly reduced despite having recurrent cells. Our architecture is smaller than the one defined in [22], pointing out a possible reduction also on their use case.

We also used autoencoders made of several kind of layers to extract features but results did not improve upon previous findings and we obtained worse results using SVMs and LR. We were not able to use the initial framed dataset, or to project along the third dimension instead of the first, and there is space to understand what will be the best way to treat encoded data, i.e. by applying specific networks, Conditional Random Fields

(CRF) or RF classifiers, instead of SVM and LR, leaving space for possible improvements.

While fulfilling this report, we learned the power of RNNs on time sequences, along with common guidelines in dataset preparation (framing, normalization and augmentation) and in defining and adjusting trainings' and networks' parameters. Notably, even simple networks can achieve stunning results and outperform even complex traditional approaches.

References

[1] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016786551830045X

[2] K. Frank, M. J. V. Nadales, P. Robertson, and M. Angermann, "Reliable Real-Time Recognition of motion related human activities using MEMS inertial sensors," in *ION GNSS*, Sep. 2010.

[3] O. D. Lara and M. A. Labrador, "A Survey on Human Activity Recognition using Wearable Sensors," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1192–1209, 2013.

[4] B. Florentino-Liaño, N. O'Mahony, and A. Artés-Rodríguez, "Human activity recognition using inertial sensors with invariance to sensor orientation," in *3rd International Workshop on Cognitive Information Processing (CIP)*, May 2012, pp. 1–6.

[5] K. Altun and B. Barshan, "Human Activity Recognition Using Inertial/Magnetic Sensor Units," in *Proceedings of the First International Conference on Human Behavior Understanding (HBU)*, 2010, pp. 38–51. [Online]. Available: http://dl.acm.org/citation.cfm?id=1881331.1881338

[6] Z. Feng, L. Mo, and M. Li, "A Random Forest-based ensemble method for activity recognition," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Aug. 2015, pp. 5074–5077.

[7] N. Y. Hammerla, S. Halloran, and T. Plötz, "Deep, Convolutional, and Recurrent Models for Human Activity Recognition Using Wearables," in *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 2016, pp. 1533–1540. [Online]. Available: http://dl.acm.org/citation.cfm?id=3060832.3060835

[8] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang, "Convolutional Neural Networks for human activity recognition using mobile sensors," in *6th International Conference on Mobile Computing, Applications and Services (MobiCASE)*, Nov. 2014, pp. 197–205.

[9] Y. Chen and Y. Xue, "A Deep Learning Approach to Human Activity Recognition Based on Single Accelerometer," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct. 2015, pp. 1488–1492.

[10] Song-Mi Lee, Sang Min Yoon, and Heeryon Cho, "Human activity recognition from accelerometer data using Convolutional Neural Network," in *IEEE International Conference on Big Data and Smart Computing (BigComp)*, Feb. 2017, pp. 131–134.

[11] F. Moya Rueda, R. Grzeszick, G. Fink, S. Feldhorst, and M. ten Hompel, "Convolutional Neural Networks for Human Activity Recognition Using Body-Worn Sensors," *Informatics*, vol. 5, no. 2, p. 26, May 2018. [Online]. Available: http://dx.doi.org/10.3390/informatics5020026

[12] T. Zebin, P. J. Scully, and K. B. Ozanyan, "Human activity recognition with inertial sensors using a deep learning approach," in *IEEE SENSORS*, Oct. 2016, pp. 1–3.

[16] S. Ha and S. Choi, "Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors," in *International Joint Conference on Neural Networks (IJCNN)*, Jul. 2016, pp. 381–388.

[13] A. Bevilacqua, K. MacDonald, A. Rangarej, V. Widjaya, B. Caulfield, and T. Kechadi, "Human Activity Recognition with Convolutional Neural Networks," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, Sep. 2018. [Online]. Available: arxiv.org/abs/1906.01935

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[15] S. Ha, J. Yun, and S. Choi, "Multi-modal Convolutional Neural Networks for Activity Recognition," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct. 2015, pp. 3017–3022.

[17] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997.

[18] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[19] D. Singh, E. Merdivan, I. Psychoula, J. Kropf, S. Hanke, M. Geist, and A. Holzinger, "Human Activity Recognition using Recurrent Neural Networks," in *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, 2018. [Online]. Available: http://arxiv.org/abs/1804.07144

[20] S. W. Pienaar and R. Malekian, "Human Activity Recognition Using LSTM-RNN Deep Neural Network Architecture," 2019. [Online]. Available: http://arxiv.org/abs/1905.00599

[21] Y. Guan and T. Plötz, "Ensembles of Deep LSTM Learners for Activity Recognition using Wearables," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, vol. 1, no. 2, p. 1–28, Jun. 2017.

[22] F. J. Ordóñez and D. Roggen, "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition," *Sensors*, vol. 16, no. 1, 2016. [Online]. Available: https://www.mdpi.com/1424-8220/16/1/115

[23] J. Park, K. Jang, and S. Yang, "Deep neural networks for activity recognition with multi-sensor data in a smart home," in *IEEE 4th World Forum on Internet of Things (WF-IoT)*, Feb. 2018, pp. 155–160.

[24] D. Arifoglu and H. Bouchachia, "Activity Recognition and Abnormal Behaviour Detection with Recurrent Neural Networks," *Procedia Computer Science*, vol. 110, pp. 86–93, Dec. 2017.

[25] Haibo He, Yang Bai, E. A. Garcia, and Shutao Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, Jun. 2008, pp. 1322–1328.

[26] F. Chollet *et al.*, "Keras," 2015. [Online]. Available: https://keras.io

[27] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Online]. Available: http://tensorflow.org

[28] S. Ruder, "An overview of gradient descent optimization algorithms," 2016. [Online]. Available: http://arxiv.org/abs/1609.04747

[29] A. Graves, "Generating Sequences With Recurrent Neural Networks," 2013. [Online]. Available: http://arxiv.org/abs/1308.0850

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations (ICLR)*, May 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[31] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, vol. 37, 2015, pp. 448–456. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045118.3045167