Human Activity Recognition: a comparison of different convolutional neural network architectures

Damnko[†]

Abstract—**Human body motion analysis based on wearable inertial measurement units (IMUs) has been receiving a lot of attention in recent years. This is due to the significant role in both the scientific and industrial communities, with uses that span from mobile health systems to sports and human computer interaction. Human Activity Recognition (HAR) system with high recognition accuracy using only a single sensor is still a technical challenge. In this paper I explore both supervised and partially supervised approaches using convolutional, deep convolutional and denoising autoencoders approaches. The goal of this paper is to enhance on the classification accuracy of previous related works and decrease reliance on human engineered features. Since the raw IMU data is composed of time series, it is split using a running window approach and segments of roughly one second of data are fed to the neural networks. The above approaches are tested with two different variations of the original dataset, obtained with data augmentation approaches, to compensate the disparity in the representation of the different classes. The dataset contains measurements of one single IMU sensor positioned in the belt of different users performing 7 actions:** ***running, jumping, walking, falling, sitting, standing, lying*****. The results show that the best CNN-based HAR system achieved an F1-Score of 0.972. The worst represented class, falling, with just 4 minutes of recorded data, has an F1-Score of 0.919.**

Index Terms—**Neural networks, machine learning, denoising autoencoders, convolutional neural network, human activity recognition, inertial sensors**

I. Introduction

In recent years, there has been substancial effort on employing wearable devices for HAR. Due to their small size, portability, and high processing power, IMUs are widely used for complex motion analysis: usage in sports training, videogames, as well as medical applications such as analysis of patients' health based on gait abnormalities and fall detection [1], smart assistive technologies, such as in smart homes [2], in rehabilitation [3], in health support [4], in skill assessment [5] or in industrial settings [6]. HAR has been studied by using computer vision approaches [7], [8], IMUs approaches [9]–[11] and hybrid approaches using feeds from both inertial sensors and cameras [12]. On-body sensors enjoy the merits of information privacy: the signals they acquire are target specific and do not reveal any personal information on the specific user nor other nontarget subjects in the scene. Moreover this allows activity recognition regardless of the location of the user, which wouldn't be possible in a fixed-camera setup. A typical HAR system is composed of two key elements: one or more smart sensor and a pattern recognition system. With IMUs, human movements are translated into time series information of acceleration via accelerometer and angles via gyroscope. Although multiple sensors can be used in order to improve the accuracy of the method, as in [13], this approach is less practical due to the use of multiple sensors. This study focuses classifying activities on data recorded by a single IMU sensor places on the belt of the subjects.

The goal of this paper is twofold: to improve the classification accuracy of previous related works and decrease reliance on human engineered features in order to address increasingly complex recognition problems. HAR approaches with manual feature extraction [14] as well as automatic [9]–[11] have already been proposed, however none of them succed in describing and comparing in a systematic and detailed way the training process for deep, convolutional, and stacked convolutional autoencoder models.

Implementing an effective HAR system using only a single sensor is still a technical challenge. For this reason I experimented with different machine learning approaches: convolutional neural networks (CNNs), 50 layers deep CNNs, stacked denoising autoencoders, for classification of 7 activities. A key aspect when using convolutions is whether to convolve only along the time dimension. This paper shows that best results are achieved when convolving first along the time dimension and then considering the cross correlation between multiple sensor data. I also experimented with some data augmentation techniques, which gave an increased accuracy on underrepresented classes.

This paper contributes to the current research material in numerous ways:

- by exploring in depth novel techniques such as stacked denoising autoencoders and deep CNNs which has been gaining a lot of traction is numerous fields
- by proposing effective task dependent and non-handcrafted features extraction techniques. These features also own more discriminative power, since the CNN can be trained under the supervision of output labels
- by exploring two data augmentation techniques which can improve the classification accuracy of specific classes
- by releasing the open source code, which can be used as starting point for future developments

The structure of this paper is organized as follows: Section II describes some related works. Section III my proposed pipeline. Section IV the input data and signal processing, Section V present an in depth description of the learning framework. Finally, in Section VI, the experimental results are presented.

[†]Name Surname: email-address

II. Related Work

In the past, HAR approaches were developed by following the standard pipeline in pattern recognition activities: segment extraction from an input time-series sequence, computing human engineered features and predicting class labels. Various modeling algorithms such as Support Vector Machine (SVM), Naive Bayes [14], Random Forest or sequence based approaches such as Dynamic Time Warping [15] or Hidden Markov Model [16], [17] were frequently used.

However, the performance of the aforementioned approaches is heavily dependant on the hand-crafted features. Extracting such features for machine learning systems is subjective, more error prone and can result in poor expressivity of the feature set. In [14] the authors manually chose 19 features, derived from various sensor measures, which were carefully selected because of their discriminative power between the various activities that were tracked. Their work applies and compares different Bayesian estimation techniques. In most cases, the last activity a person has performed influences their current activity, this knowledge can provide valuable input for the network. The work on this paper represents the baseline which I wanted to improve upon, by using more recent techniques and automatic feature extraction.

In [13] the authors use a CNN for human activity recognition activities for collecting and analyzing realistic data from manual processes in industrial scenarios. The experiment was carried out using multiple IMU sensors and the data collected was augmented using gaussian noise and random resampling. Data was then segmented and fed to a CNN using only temporal convolutions. The CNN performed consistently better compared to other techniques such as Bayes, Random Forest and SVM.

Approaches that are able to exploit the temporal dependencies in time-series data appear as the natural choice for modelling human movement captured with sensor data. Deep recurrent neural networks (RNNs), especially those that rely on Long Short-Term Memory cells (LSTMs), have recently achieved impressive performance across a variety of scenarios. Many recent papers explore this approach and compare with a baseline CNN. Each LSTM unit keeps track of an internal state that represents it's ”memory”. Over time the cells learn to output, overwrite, or reset their internal memory based on their current input and the history of past internal states, leading to a system capable of retaining information across hundreds of time-steps. In [10] the authors implement two flavours of LSTM recurrent networks: a deep forward LSTMs containing multiple layers of recurrent units that are connected ”forward” in time and bi-directional LSTMs which contain two parallel recurrent layers that stretch both into the ”future” and into the ”past”. Testing against 3 well known benchmark datasets in the HAR field, the recurrent nn achieves higher classification accuracy in all but one dataset.

[11] combines effective preprocessing techniques i.e. downsampling of the input raw data and RNNs to prove how this technique achieves 9% better accuracy compared to a 1-D CNN on a benchmark dataset.

Extending on the process of HAR, some researchers were successful in performing activity recognition tasks using depth cameras [18], [19] which are nowadays widely available and affordable.

[12] evolved on this concept by merging input data coming from a single rgb camera and sensor values coming from a wrist-worn IMU sensor. Their proposed feature extractor for the IMU sensor is based on a convolutional autoencoder while for feature extraction in the rgb image a CNN is used, leveraging residual modules for a better propagation of the gradients during training. The usage of both camera signal and IMU signals improved the classification accuracy and prevented performance degeneration caused by the failure of joint estimation due to eg. image occlusion and noise.

III. Processing Pipeline

Analysis of raw data coming from IMU sensors generally follows a pipeline-based approach composed of different blocks. Fig. 1 shows a schematic representation of the preprocessing and classification phases.

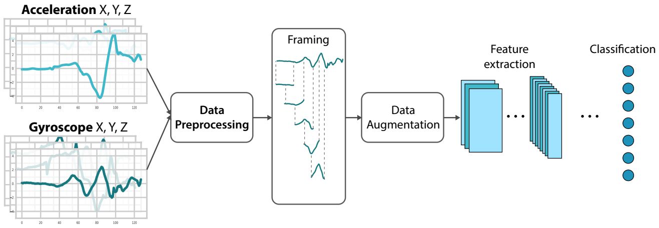

Fig. 1: Schematic representation of the processing pipeline.

Time series data from the IMU sensors is converted into contiguous segments through a sliding-window approach, the output of this process are feature vectors which represent the input of the machine learning models. Once the dataset is created, it is augmented and divided in training and test set and finally normalized with respect to the mean and variance of the training set. The basic blocks of all the approaches mentioned in this paper are the ones which typically constitute a CNN. Each CNN contains at least one convolution layer along the temporal domain, one pooling layer and at least one fully connected layer prior to a softmax layer. In the convolutional layer, a mathematical operation (i.e. convolution) applies a set of local filters (or kernels) to obtain the most representative features. After the convolution operation, a bias is added to the result. Subsequent max-pooling operations look for the maximum within a region of specific width and height, this corresponds to a subsampling which introduces translational invariance to the system and improves its robustness. The fully connected layer is applied at the end, combining all features' maps obtained by the previous convolutional steps and using it as input for a 7 neurons softmax layer which computes the probability of each activity class. Regularization techniques such as dropout, early stopping and batch normalization are used at different stages of the neural network in order to avoid overfitting to the training data.

Stacked denoising convolutional autoencoders (SDAs) are built on the same basic blocks mentioned above but follow a slightly different processing pipeline. Fig. 4 represents the structure of a stacked autoencoder. The main building block is the autoencoder which can be separated into two parts, an encoder and a decoder. The encoder part consists of convolution layers and fully-connected layers and the decoder part consists of fully-connected layers and deconvolution layers. An ordinary autoencoder where the output Y is of the same dimensionality as input X can achieve perfect reconstruction simply by learning an identity mapping. This criterion alone is unlikely to lead to the discovery of a more useful representation ofthe input. The traditional approach to autoencoders uses a bottleneck to produce an under-complete representation, the resulting Y can thus be seen as a lossy compressed representation of X. Since the reconstruction criterion alone is unable to guarantee the extraction of useful features, a more challenging and more interesting objective would be to clean a partially corrupted input (denoising). It's important to emphasize that the goal is not the task of denoising per se, rather denoising is investigated as a training criterion for learning to extract useful features that will constitute better higher level representation of the input [20]. A key function of SDAs is unsupervised pre-training. Once each layer is pre-trained to conduct feature selection and extraction on the input from the preceding layer, a second stage of supervised fine-tuning can follow. The unsupervised pre-training of such architecture is done one layer at a time. Each layer is trained as a denoising autoencoder by minimizing the error in reconstructing its input. Once all layers are pre-trained, the network goes through a second stage of training known as supervised fine-tuning whose goal it to minimize prediction error on a supervised task. During this latter supervised phase only the dense layers are trained, while the rest is kept frozen.

IV. Signals and Features

The dataset used in this paper is taken from [14] [1]. The IMU used provides the measurements in the sensor frame (SF) and the necessary attitude information in order to rotate them to the global frame. However, acceleration and angular velocity relative to the human body seem to be the most relevant information (and not relative to an earth-fixed reference frame or the sensor frame as the sensor can be placed on the body in any orientation and position) [14]. The three axes of the body frame are defined to intersect at the sensor location, the z axis is directed towards the head, while the other axis (x and y) form the plane orthogonal to this vertical axis. The sensor was placed on the belt of the test candidates either on the right or the left part of the body. As shown in Tab. 1 the final dataset contains over 5 hours of activity data. Class unbalance represents a major obstacle in the classification task.

	Standing	**Walking**	**Sitting**	**Lying**	**Running**	**Jumping**	**Falling**
Minutes	121	72	59	28	15	8	2

TABLE 1: Total activity data times for each recorded activity.

The original dataset contains the recordings of one IMU sensor:

- acceleration values along x-y-z axis
- gyroscope values along x-y-z axis
- attitude matrix to convert previous values from sensor to global frame
- magnetometer values along x-y-z axis

Each entry is either labeled with one of the activities among *running, jumping, walking, falling, sitting, standing, lying* or as a transition state between these base labels.

A. Data preprocessing

The current classification task goal is to predict one of the base labels. Detecting transients was out of the scope of this paper and thus they were initially removed from the dataset. One mislabeling error was found and corrected, which was influencing just 5 frames in the original dataset.

B. Framing

After the initial preprocessing, next step is to segment the time series data into contiguous segments through a sliding-window approach (Fig. 2).

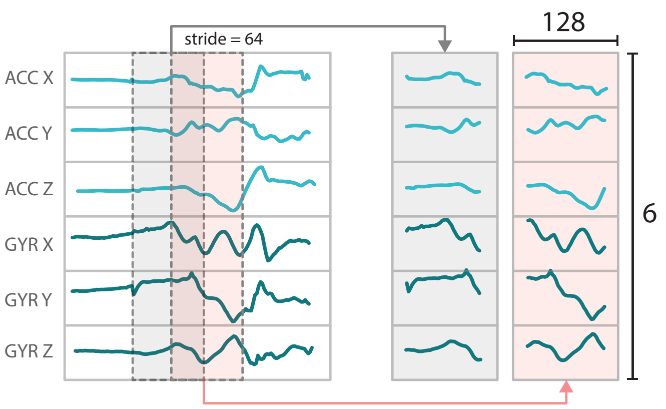

Fig. 2: Framing procedure.

Each frame or window will be then fed to the machine learning algorithm. Since 1sec is the minimal duration of an activity [14], I used this as a reference for the window length. The sensor used has a capture rate of 100Hz, which means 100 samples/sec, so using a window length of 128 frames is still within the minimal duration of an activity, moreover it's the same value employed by [14]. The window is moved along the temporal domain progressively extracting samples that will generate the training and testing dataset. The step size of the sliding window is of 64 frames, so each window has 50% overlapping with the previous one. Although the information in these is highly redundant, this allows to generate a large number of samples, which is important for training a CNN. The label associated to each frame corresponds to the most

[1] The dataset is available at http://www.kn-s.dlr.de/activity/

frequent label of the 124 frames composing the window. After windowing, the dataset is divided into training and test set following a 80/20 division, maintaining the class proportions in both test and training, as shown in Tab. 2. Training set is composed of 22836 samples, test set of 5709 samples.

	Full	Test	Train
Standing	0.396532	0.396567	0.396523
Walking	0.236889	0.236819	0.236907
Sitting	0.193309	0.193204	0.193335
Lying	0.092626	0.092661	0.092617
Running	0.049361	0.049396	0.049352
Jumping	0.024838	0.024873	0.024829
Falling	0.006446	0.006481	0.006437

TABLE 2: Class proportions in full dataset compared to test and training set.

Finally, the input data is normalized before it is fed into the feature extractor. I computed mean and standard deviation for each axis of the considered sensor values (acceleration and gyroscope) and normalized the input data by subtracting mean and dividing by the standard deviation. Both training and test set were normalized using the values computed on the training set, by doing this, no bias or additional information in introduced in the test set.

C. Data augmentation

Data augmentation is the practice of augmenting real IMU training data with simulated data for improving the recognition accuracy. Appropriate augmentation can substancially improve classification performance, especially in situations with small dataset size, noisy labels, and large intra-class variability [21]. The latter being an especially known problem in HAR tasks [16]. During the data augmentation process, the imbalance issue is tackled by creating a larger number of augmented samples for the under represented classes. The samples are re-balanced such that each class has at least 35% percent of the largest number of samples per class in the training set.

In the field of computer vision, synthetic data (computer-generated data that mimics real data) has been used used to augment or create training data for increasing classification performance [22].

As there exist many kinds of data augmentation techniques, the most promising from [21] are implemented in this paper. Permutation is a simple way to randomly perturb the temporal location of within-window events. To perturb the location of the data in a single window, I first slice the data into 5 samelength segments, and randomly permute the segments to create a new window. Another factor that can introduce label-invariant variability of wearable sensor data are differences in sensor placement. For example, an upside-down placement of the sensor can invert the sign of the sensor readings without changing the labels. Therefore, augmentation by applying arbitrary rotations to the existing data can be used as a way of simulating different sensor placements. To achieve the intra-class balance mentioned before, the input vectors quantities summarised in Tab. 3 were generated by combining the two mentioned data augmentation approaches.

	Original size	Increment	Augmented size
Standing	9055	0	9055
Walking	5410	0	5410
Sitting	4415	0	4415
Lying	2115	1054	3169
Running	1127	2042	3169
Jumping	567	2602	3169
Falling	147	3022	3169

TABLE 3: Number of samples generated to augment the original dataset and representation of the new class balance in the augmented dataset.

Considering the conspicuous amount of data that was discarded when removing all the transients from the original dataset, specifically 1 hour of data, I tried an alternative data augmentation approach that leveraged this wealth of data. Transients are unlabeled frames that sit between two consecutive labeled activities. The idea of this approach is to assign specific labels to this transients according to the previous and following activity, specifically half of the frames of each transient was labeled as the preceding activity and the remaining half was labeled as the following activity.

As a result of this data augmentation phase, I obtained 3 datasets: the original non-augmented, the augmented through rotation and permutation (`aug`) and another one with transitions converted to labels (`trans`).

V. Learning Framework

The neural networks here described are implemented in Keras, a lightweight library to build and train neural networks. The model training and classification are run on a *p2.xlarge* EC2 AWS instance which features Intel Xeon E5-2686 v4 (Broadwell) processor, 61GiB of ram and one NVIDIA K80 GPU.

The following sections describe in detail the topology and the parameters of the three basic machine learning models used to solve the HAR task tackled in this paper. Sensor data are re-organized to be of shape `?x1x128x6` where 128 is the length of a sensor window, 6 is the number of sensor signals (acceleration x,y,z - gyroscope x,y,z), and 1 is the depth. This format (known as ”channel first” in Keras) was used for compatibility reasons with the GPU backend of Keras. All models are using mini-batch gradient descent (64 samples per batch) with RMSProp update rule with $learningrate = 0.001$ and $\rho = 0.9$.

A. CNN with only 1D temporal convolutions (`m_1d`)

This is the simplest of the architectures explored in this paper and involves only temporal convolutions, i.e. convolutions along the time domain. The topology of the model is described in detail in Tab. 4

Each conv section is constituted by (i) a convolution layer that convolves the input or the previous layer's output along the temporal domain with a set of kernels to be learned; (ii) a batch normalization along the time domain, (iii) a rectified

Input	Conv	D.M.	Conv	D.M.	Flat	Dense	D	Softmax
1@(6x128)	30@(1x5)		40@(1x5)			150		7

TABLE 4: `m_1d` model representation. Each `Conv` layer is made of (i) a convolution layer, (ii) a batch normalization and (iii) a ReLU activation. `D.M.` stands for dropout and max pooling, `D` stands for dropout. All dropout layers were set to 0.3 except the last one, set to 0.2.

linear unit (ReLU) layer that maps the output of the previous layer by the function $relu(v) = max(v; 0)$. Dropout layers set the activation of randomly-selected units during training to zero with probability 0.3 for the first two occurennces and 0.2 in the last dropout layer and this is added as a form of regularization. Max pooling is used to further reduce the dimensionality and increase the spatial invariance of features. After the convolutions, all the feature maps values of the previous layer are concatenated using a fully-connected layer (indicated as *Flat* in Tab. 4). Following is a fully-connected layer with 150 neurons and lastly another fully-connected layer with $C = 7$ neurons, where C is the number of output classes. The output of this layer is governed by the softmax function

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{\kappa} e^{z_k}}$$

Where κ is the total number of classes (7 in this case) and z_j represents the j-th entry of the score vector z. This softmax function provides the posterior probability of the classification results. Then, an entropy cost function can be constituted based on the true labels of training instances and probabilistic outputs of softmax function

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} log(p_{ij})$$

Where i indexes samples, j indexes classes and y is the sample label (one-hot vector in multiclass classification), $p_{ij} \varepsilon (0, 1)$ and $\sum_j p_{ij} = 1$ is the prediction for a sample. The network parameters are optimized by minimizing the cross-entropy loss function using mini-batch gradient descent with the RMSProp update rule.

B. CNN with 1D and 2D convolutions (`m_1d2d_01`)

This model extends the previous one by testing also the effect of cross correlating the signals of different sensors by using both 1D and 2D convolutions. The topology of this model is summarized in Tab. 5

Keeping Tab. 5 as a reference, the first 3 convolutional blocks operate only temporal convolutions, starting from the 4th block, cross-correlation among input vectors is considered too. In these 2D convolutional layers, zero-padding is also applied in order to keep the $(Y \times Z)$ feature map dimension invariant during convolution operations, this was necessary in order to achieve a deeper network.

I tested two other different variations of this model, specifically:

- `m_1d2d_01_reg` which features the same topology, but L2 regularization was applied to the last 2 dense layers, in an attempt to prevent overfitting. L2 regularization adds a penalty parameter to the loss function, the value of the multiplier was set to $\lambda = 0.001$, setting it to 0 reverts to the case of no normalization.
- `m_1d2d` which instead of having a single batch normalization at the beginning, it has a batch normalization layer after each convolutional layer

C. CNN with skip connections (`m_resnet`)

The main benefit of a very deep network is that it can represent very complex functions. However, a huge barrier to training them is vanishing gradients: while backpropagating from the final layer back to the first layer, the weight matrix is multiplied on each step, and thus the gradient can decrease exponentially quickly to zero. Residual networks (ResNets) were first proposed by [23] and managed to build very deep CNNs by using skip connections (or shortcuts) to help the backpropagation of the gradient. The basic building blocks of a ResNet are identity and convolutional shortcuts (Fig. 3): identity shortcuts can be directly used when the input and output are of the same dimensions (Fig. 3 left). When the dimensions increase, projection shortcuts (Fig. 3 right) are used to match dimensions (done by 1×1 convolutions).

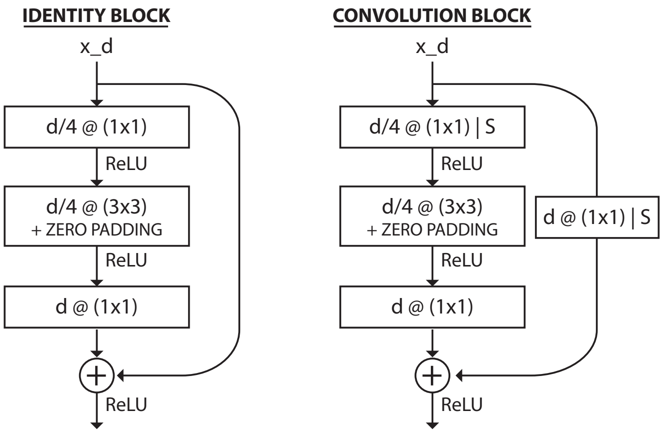

Fig. 3: Building blocks of a ResNet. Identity block is used when input and output dimension are the same, otherwise the convolution block is used. `x_d` indicates the input `x` of depth `d`. `| S` suffix indicates that a custom stride, different from 1×1 is applied to that convolution.

In this paper I implement *ResNet-50* and I suggest [23] for full details on the implementation.
This model is achieved by stacking groups of identity blocks and convolutional blocks to form a 50 layer deep network, the topology is described in Tab. 6.

D. Stacked denoising convolutional autoencoders (`m_ae`)

The main building block of this model is the autoencoder, an architecture composed of an encoder and a decoder. In this setup, the autoencoders are composed only of convolutional layers. The topology of this setup is represented in Fig. 4. Two autoencoders are stacked together, followed by a fully-connected layer for the final classification.

Input	BN	Conv	Conv	D	Conv	Conv	D	MP	Conv	MP	Conv	D	Flat	Dense	D	Softmax
1@(6x128)		16@(1x4)	32@(1x4)	0.3	64@(1x3)	64@(3x3)	0.3		64@(3x2)		64@(3x2)	0.3		150	0.2	7

TABLE 5: `m_1d2d_01` model representation. Each `Conv` layer is made of (i) a convolution layer and (ii) a ReLU activation, stride is set to (1x1). `BN` stands for batch normalization. `D` represents a dropout layer and the number beneath is the percentage of elements that will be set to zero. `MP` stands for max pooling. `Flat` concatenates all the feature map values of the previous layer. Dense is a fully connected layer. `X@`$(Y \times Z)$ indicates the number of kernels, X, and the size of the kernel matrix, $Y \times Z$.

Input	Conv	BN	ReLU	MP	S1	S2	S3	S4	AP	Flat	Softmax
1@(6x128)	32@(1x4)				$\begin{bmatrix}32@(1\times1)\\32@(3\times3)\\128@(1\times1)\end{bmatrix}\times 3$	$\begin{bmatrix}64@(1\times1)\\64@(3\times3)\\256@(1\times1)\end{bmatrix}\times 4$	$\begin{bmatrix}128@(1\times1)\\128@(3\times3)\\512@(1\times1)\end{bmatrix}\times 6$	$\begin{bmatrix}256@(1\times1)\\256@(3\times3)\\1024@(1\times1)\end{bmatrix}\times 3$	1x2		7

TABLE 6: `m_resnet` model representation. `BN` stands for batch normalization, `MP` for max pooling, `S1-S4` represent the 4 stages of the identity-convolutional blocks. `AP` stands for average pooling, pooling is done only along the temporal dimension.

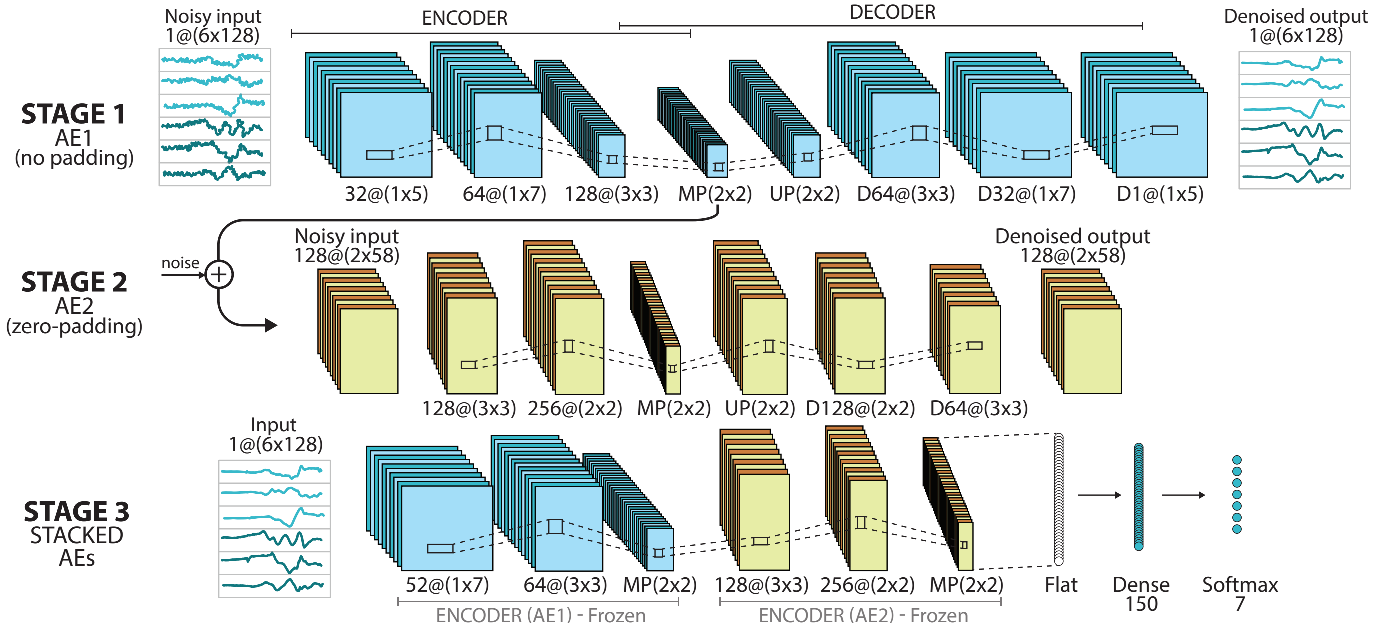

Fig. 4: Processing pipeline of a stacked denoising autoencoder. `MP` stands for max pooling, `UP` stands for up scaling, the opposite of a max pooling operation. Operations with `D` prefix stand for de-convolution operations, which revert the effect of a convolution. Zero-padding indicates that padding was used in order to keep the same feature map size during the various convolution operations.

Training this kind of models follow a two-step procedure: (i) a corrupted input is used for the initial denoising-training of each individual layer, so that it may learn useful feature extractors. Once the mapping has been learnt, the output of the encoder part (with noisy data given as input) is used as input of the following autoencoder. Noise is always applied to the input of all the autoencoders during training. I tested with two different types of noise, as suggested by [20]:

- Additive isotropic gaussian noise
- Masking noise, where a fraction of the elements of the input (chosen at random for each example) is forced to 0. This drastically corrupt a fraction of the elements while leaving the others untouched

I also experimented with two different variations of the first autoencoder: this first one being represented in Fig. 4 (`ae-long`) and another one that featured a shallower first autoencoder (`ae-short`), without an intermediate convolutional layer.

The network parameters of the autoencoders are optimized by minimizing the mean squared error loss function:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

Once a stack of encoders has thus been trained, its output representation can be used as input to a stand-alone supervised learning algorithm. (ii) An output layer is added on top of the stack and the parameters of the whole system are fine-tuned to minimize the error in predicting the supervised target by performing gradient descent. It's important to note that during the fine-tuning phase the uncorrupted inputs are used and the layers of the stacked autoencoders are frozen i.e. no training is performed on these layers, only the weights on the output layers are updated.

VI. Results

The evaluation of the models followed a progressive approach and the goal was twofold:

- find the best dataset i.e. augmented vs non-augmented
- find the model with the best classification accuracy

In this paper I did not take into consideration the complexity of the model, so there was no constraints on the training or prediction time that would indeed have to be considered in the context of real-time prediction.

To evaluate the classification accuracy of the models I used the F1-Score parameter computed on a validation set, comprising 20% of the total observations of the dataset. F1-Score is generally more appropriate then accuracy especially in cases of uneven class distribution, it is defined as the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Where precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

$$Precision = \frac{TP}{TP + FP}$$

Recall is the ratio of correctly predicted positive observations to the all observations in true positive class.

$$Recall = \frac{TP}{TP + FN}$$

A. *Dataset results*

To evaluate the dataset which offered the best classification accuracy, I tested the three datasets: (i) original, (ii) augmented with permutation and noise (`aug`) and (iii) augmented with replacement of the transition states (`trans`) with the following models: `m_1d`, `m_1d2d`, `m_1d2d_01`, `m_2d`. Refer to Section V for details on the model specifications and topology.

Tab. 7 presents the F1-Scores for the aforementioned models and datasets.

	m_1d			**m_1d2d**			**m_1d2d_01**			**m_2d**		
	std	**trans**	**aug**	**std**	**trans**	**aug**	**std**	**trans**	**aug**	**std**	**trans**	**aug**
Falling	0.757	0.233	0.773	0.794	0.238	0.882	0.8	0.519	0.879	0.794	0.258	0.838
Jumping	0.92	0.584	0.934	0.931	0.587	0.94	0.908	0.538	0.952	0.926	0.598	0.926
Lying	0.984	0.919	0.989	0.992	0.935	0.976	0.992	0.927	0.992	0.992	0.919	0.991
Running	0.972	0.901	0.981	0.989	0.903	0.995	0.986	0.895	0.998	0.988	0.902	0.991
Sitting	0.885	0.785	0.836	0.897	0.839	0.874	0.917	0.874	0.929	0.904	0.825	0.903
Standing	0.945	0.845	0.928	0.951	0.86	0.942	0.962	0.888	0.964	0.951	0.858	0.952
Walking	0.983	0.915	0.987	0.99	0.906	0.989	0.988	0.933	0.992	0.986	0.915	0.989
	0.945	0.836	0.932	0.954	0.851	0.945	0.961	0.878	**0.967**	0.954	0.849	0.955

TABLE 7: F1-Scores of selected models with respect to three different datasets: (i) original `std`, (ii) augmented with permutation and noise (`aug`) and (iii) augmented with replacement of the transition states `trans`. Red cells indicate a degradation of the classification accuracy with respect to the standard dataset, green cells indicate an improvement. Best overall result is highlighted in bold.

Before evaluating the impact of the different datasets, it's interesting to note that, as [24] suggested, the model `m_1d2d_01` with a single batch normalization layer performs better compared to its similar version, with multiple batch normalization layers (`m_1d2d`). Overall model with the best F1-Score among the ones in Tab. 7 is `m_1d2d_01`, featuring both 1D and 2D convolutions, which achieves a F1-Score value of 0.967. Lets now consider the impact of different datasets: `trans` variation offers no improvement over the standard dataset, while the `augm` version consistently improve the classification accuracy with respect to the standard dataset. The major improvements are proportional to the augmentation entity of the underrepresented classes, specifically as can be seen from Tab. 8 *falling* benefits from the biggest average increase accuracy and corresponds to the least represented class. Classes which had no augmentation received negligible variations with respect to the standard dataset.

	Augmentation entity	**Mean improvement**
Falling	3022	0.046
Jumping	2602	0.019
Running	2042	0.008
Lying	1054	0.001
Sitting	0	-0.01
Standing	0	0
Walking	0	0.004

TABLE 8: Mean F1-Score improvement for all models of Tab. 7 referred to the dataset augmented with permutation and noise `aug` with respect to the standard dataset. Augmentation entity represents the number of observations added to specific classes.

It is thus reasonable to assume that the `aug` dataset improves the overall classification accuracy and will be the default dataset considered for tests on the subsequent models.

B. *Autoencoder results*

Autoencoders were trained using the `aug` dataset. As mentioned in Section V-D, four different variations are considered:

- `ae-long-gaus` represented in Fig. 4 and whose inputs are corrupted using gaussian noise
- `ae-long-zero` represented in Fig. 4 and whose inputs are corrupted using zero masking
- `ae-short-gaus` which is the shallower version of model in Fig. 4 and whose inputs are corrupted using gaussian noise
- `ae-short-zero` which is the shallower version of model in Fig. 4 and whose inputs are corrupted using zero masking

A qualitative assessment of the reconstruction accuracy of the autoencoder can be seen in Fig. 5. The model seems to be able to remove noise pretty effectively and the rebuilt signal overlaps most of the times with the original uncorrupted input.

Tab. 9 Compares the different stacked autoencoders F1-Scores.

	ae-long-gaus	**ae-long-zero**	**ae-short-gaus**	**ae-short-zero**
Falling	0.667	0.571	**0.720**	0.656
Jumping	0.838	0.805	**0.849**	0.766
Lying	**0.985**	0.985	0.985	0.985
Running	**0.956**	0.945	0.947	0.949
Sitting	**0.867**	0.823	0.850	0.847
Standing	**0.935**	0.912	0.926	0.920
Walking	**0.979**	0.965	0.972	0.975
Mean	**0.934**	0.911	0.925	0.921

TABLE 9: F1-Scores for various stacked autoencoders, `long` prefix indicates the deeper version of the first autoencoder, as presented in Section V-D, `short` prefix indicates the shallower version. `gaus` suffix indicates the model trained with inputs corrupted by gaussian noise, `zero` instead indicates input corruption with zero-masking.

While the best F1-Score is achieved by `long-gaus` model with a value of 0.934, just 0.009 improve with respect to the shallower version, the shallow version was able to achieve a

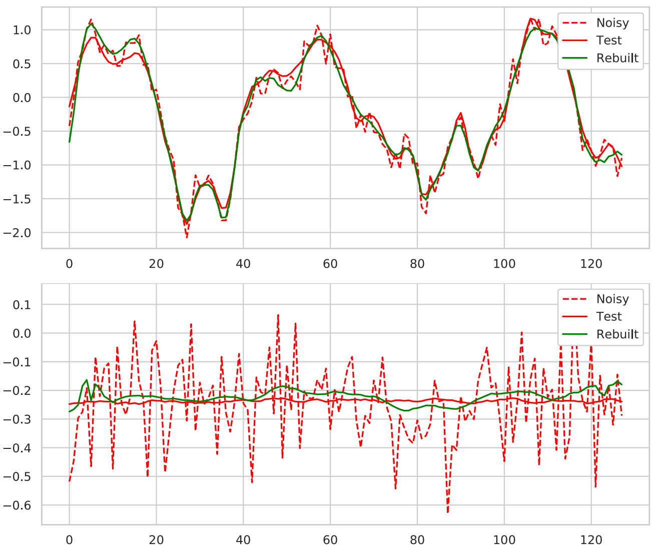

Fig. 5: Reconstruction of some sample signals done by the first autoencoder of the `ae-long` model. *Noisy* is the noisy input used for training, *Test* is the orinal uncorrupted signal, *Rebuilt* is the reconstructed signal by the trained autoencoder.

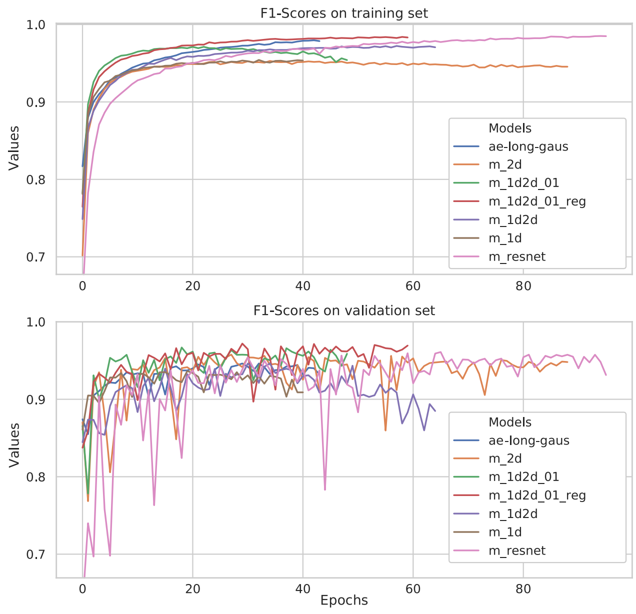

Fig. 6: F1-Scores for training and validation set.

better classification accuracy on underrepresented classes. The deeper version was probably able to build a more complex representation of the input data for classes with a higher number of samples, thus offering an improved F1-Score on specific classes i.e. *running, sitting, standing, walking*. Lastly, the autoencoder with gaussian noise performed slightly better compared to the zero-masking one.

C. Overall model comparison

The previous two sections were aimed to find the best version of the analysed models in order to perform additional fine tunings and thus do a final comparison with the remaining model, which is presented in this section.
In this section two more models are introduced:

- `m_1d2d_01_reg` which has the same topology as `m_1d2d_01` but adds L2 regularization in the last 2 dense layers
- `m_resnet` which is *ResNet-50*

The models were compared using the `aug` dataset and the F1-scores are summarized in Tab. 10.

	ae-long-gaus	**m_3d**	**m_1d2d_01**	**m_1d2d_01_reg**	**m_1d2d**	**m_1d**	**m_resnet**
Falling	0.667	0.838	0.879	**0.919**	0.882	0.773	0.872
Jumping	0.838	0.926	0.952	**0.964**	0.94	0.934	0.936
Lying	0.985	0.991	0.992	**0.993**	0.976	0.989	0.992
Running	0.956	0.991	0.998	**0.998**	0.995	0.981	0.989
Sitting	0.867	0.903	0.929	**0.94**	0.874	0.836	0.92
Standing	0.935	0.952	0.964	**0.969**	0.942	0.928	0.956
Walking	0.979	0.989	0.992	**0.992**	0.989	0.987	0.99
Mean	0.934	0.955	0.967	**0.972**	0.945	0.932	0.961

TABLE 10: F1-Scores of selected models.

Training curves of the experiments are depicted in Fig. 6

The regularized version of model `m_1d2d_01` was able to remove the overfitting problem which is very visibile at approximately 20 epochs (Fig. 6) and was thus be able to improve its classification accuracy. Batch size and the learning rate may be influencing the fluctuations in the validation accuracy plot of Fig. 6.
Overall the best model achieves an F1-Score of 0.972 and the confusion matrix related to this model is presented in Tab. 11

		Predicted							**Recall**
		Falling	Jumping	Lying	Running	Sitting	Standing	Walking	
True	Falling	**0.92**	0	0.027	0	0.054	0	0	0.919
	Jumping	0	**0.93**	0	0.007	0	0.021	0.042	0.930
	Lying	0.0057	0	**0.99**	0	0.0057	0	0	0.989
	Running	0	0	0	**1**	0	0	0	1
	Sitting	0	0	0	0	**0.92**	0.083	0	0.917
	Standing	0	0	0	0	0.015	**0.98**	0.0013	0.984
	Walking	0	0	0	0	0	0.0096	**0.99**	0.990
Precision		0.919	1	0.998	0.996	0.964	0.954	0.993	**0.972**

TABLE 11: Confusion matrix of model `m_1d2d_01_reg` referred to the `aug` dataset

As a comparison with the reference paper [14], Tab. 12 reports the F1-Scores for the best model of this paper and the best model of [14]

	Reference	**m_1d2d_01_aug**	**Diff**
Falling	0.889	0.919	0.03
Jumping	0.930	0.964	0.034
Lying	0.989	0.993	0.004
Running	0.963	0.998	0.035
Sitting	0.984	0.940	-0.044
Standing	0.989	0.969	-0.02
Walking	0.989	0.992	0.003

TABLE 12: F1-Scores of best model of reference paper [14] vs the best model of this paper along with the difference between the two approaches.

VII. Concluding Remarks

In this paper I explored the performance of various deep learning approaches based on CNNs for HAR using a single a wearable sensor. The goal was to improve on the result of [14] and prove that automatic feature extration would result in improved classification accuracy. This task was successfully tackled using a 8-layer CNN and the proposed data augmentation methods. The combination of rotational and permutational data augmentation methods improves the baseline F1-Score of 0.961 to 0.967, overall improvement is not substancial but specific underrepresented classes received most benefit.

By comparing CNNs with 1D (temporal-only) convolutions, CNNs with both 1D and 2D convolutions, ResNets and stacked denoising autoencoders, the best model was a 8-layer CNN with both 1D and 2D convolutions capturing also the cross-correlation between different sensor values. I managed to achieve an F1-Score of 0.972 on the best performing model. Many recent research papers as [10], [11], [18] show that recurrent neural networks consistenely outperform deep neural networks and CNNs in HAR tasks and this would definitely be an interesting architecture to test against the ones developed in this paper. I did not experiment with different sampling rates, which is a crucial topic when dealing with wearable devices: reduced sampling rates imply more efficient resource use in real-world deployments. Minimal changes in the model architechture may result in noticeable improvement in the classification accuracy, as I showed in Section VI-C. Random exploration of the parameter space for each model would be an effective strategy to iteratively fine-tune selected model as shown by [10]. Moreover, aside from the technical challenges I had to overcome to develop this project, I realised the importance of reading and undestanding research material on the topic of interest, specifically machine learning applied to HAR. Coming up with effective models requires both a deep understanding of machine learning and a background knowledge of HAR processes. Recent research papers are a great starting point to be aware of the state of the art architectures for the topic one's working on. This awareness gave me the motivation to do my best to write a clear, concise and understandable research paper and to open source my work at https://github.com/damnko/har-convolutional-neural-networks so that hopefully my effort can help the community. As a last thing, this was my first opportunity to realise the importance of being confident in working with datasets and coming up with creative solutions to rearrange and elaborate data. The data preprocessing pipeline is as important as developing and testing the model and may require a substancial investment of time.

References

[1] M. Yamada, T. Aoyama, and S. M. et al., "Objective assessment of abnormal gait in patients with rheumatoid arthritis using a smartphone," *Rheumatol Int*, vol. 32, pp. 3869–3874, Dec. 2012.

[2] P. Rashidi and D. J. Cook, "Keeping the resident in the loop: Adapting the smart home to the user," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 39, pp. 949–959, Sept. 2009.

[3] S. Patel, H. Park, P. Bonato, L. Chan, and M. Rodgers, "A review of wearable sensors and systems with application in rehabilitation," *Journal of NeuroEngineering and Rehabilitation*, vol. 21, Apr. 2012.

[4] A. Avci, S. Bosch, M. Marin-Perianu, R. Marin-Perianu, and P. Havinga, "Activity Recognition Using Inertial Sensing for Healthcare, Wellbeing and Sports Applications: A Survey," in *23th International Conference on Architecture of Computing Systems*, (Hannover, Germany), Feb. 2010.

[5] M. Kranz, A. Moller, N. Hammerla, S. Diewald, T. Plotz, P. Olivier, and L. Roalter, "The mobile fitness coach: Towards individualized skill assessment using personalized mobile devices," *Pervasive and Mobile Computing*, vol. 9, pp. 203–215, Apr. 2013.

[6] T. Stiefmeier, D. Roggen, G. Ogris, P. Lukowicz, and G. Troster, "Wearable activity tracking in car manufacturing," *IEEE Pervasive Computing*, vol. 7, pp. 42–50, Apr. 2008.

[7] H. Jhuang, J. Gall, S. Zuffi, C. Schmid, and M. J. Black, "Towards Understanding Action Recognition," in *IEEE International Conference on Computer Vision*, (Sydney, NSW, Australia), Dec. 2013.

[8] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy, "Berkeley MHAD: A comprehensive Multimodal Human Action Database," in *IEEE Workshop on Applications of Computer Vision*, (Clearwater Beach, FL, United States), Jan. 2013.

[9] J. B. Yang, M. N. Nguyen, P. P. San, X. L. Li, and S. Krishnaswamy, "Deep convolutional neural networks on multichannel time series for human activity recognition," in *IJCAI'15 Proceedings of the 24th International Conference on Artificial Intelligence*, (Buenos Aires, Argentina), July 2015.

[10] N. Y. Hammerla, S. Halloran, and T. Plotz, "Deep, Convolutional, and Recurrent Models for Human Activity Recognition Using Wearables," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, (New York, New York, USA), Apr. 2016.

[11] E. Valarezo, R. Patricio, M. J. Park, G. Geon, Y. T. Kim, P. M, A. antari Mugahed, A. masni Mohammed, and T. Kim, "Human activity recognition using a single wrist imu sensor via deep learning convolutional and recurrent neural nets," *Journal of ICT, Design, Engineering and Technological Science*, vol. 1, pp. 1–5, Jan. 2017.

[12] I. Hwang, G. Cha, , and S. Oh, "Multi-modal human action recognition using deep neural networks fusing image and inertial sensor data," *Expert Systems with Applications*, vol. 105, pp. 42–50, Sept. 2018.

[13] G. Rene, L. J. Marius, R. F. Moya, F. G. A, F. Sascha, and T. H. Michael, "Deep neural network based human activity recognition for the order picking process," in *International Workshop on Sensor-based Activity Recognition (iWOAR)*, (Rostock, Germany), Sept. 2017.

[14] K. Frankz, M. J. V. Nadalesz, P. Robertsonz, and M. Angermannz, "Reliable Real-Time Recognition of Motion Related Human Activities Using MEMS Inertial Sensors," in *ION GNSS*, (Portland, Oregon, USA), Sept. 2010.

[15] D. R. P. C. P. A.-V. Nguyen-Dinh, "Limited-Memory Warping LCSS for Real-Time Low-Power Pattern Recognition in Wireless Nodes," in *European Conference on Wireless Sensor Networks*, (Porto, Portugal), Feb. 2015.

[16] A. Bulling, A. Bulling, and B. Schiele, "A tutorial on human activity recognition using body-worn inertial sensors," *ACM Computing Surveys (CSUR)*, vol. 46, pp. 1–33, Jan. 2014.

[17] F. J. Ordonez, G. Englebienne, P. de Toledo, T. van Kasteren, A. Sanchis, and B. Krose, "In-home activity recognition: Bayesian inference for hidden markov models," *IEEE Pervasive Computing*, vol. 13, pp. 67–75, Sept. 2014.

[18] S. Park, J. Park, M. Al-masni, M. Al-antari, M. Uddin, and T.-S. Kim, "A depth camera-based human activity recognition via deep learning recurrent neural network for health and social care services," *Procedia Computer Science*, vol. 100, pp. 78–84, 2016.

[19] A. Manzi, F. Cavallo, and P. Dario, "A 3D Human Posture Approach for Activity Recognition Based on Depth Camera," in *European Conference on Computer Vision*, (Amsterdam, The Netherlands), Oct. 2016.

[20] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, Jan. 2010.

[21] T. T. Um, F. M. J. Pfister, D. Pichler, S. Endo, M. Lang, S. Hirche, and D. K. Urban Fietzek, "Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring using Convolutional Neural Networks," in *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, (Glasgow, United Kingdom), Nov. 2017.

[22] X. Zhang, Y. Fu, S. Jiang, L. Sigal, and G. Agam, "Learning from Synthetic Data Using a Stacked Multichannel Autoencoder," in *IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, (Miami, FL, USA), Dec. 2015.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," tech. rep., Microsoft, Dec. 2015.

[24] F. Li, K. Shirahama, M. A. Nisar, L. Koping, and M. Grzegorzek, "Comparison of feature learning methods for human activity recognition using wearable sensors," *Sensors*, vol. 18, no. 2, p. Article No. 679, 2018.