Neural Architectures for Human Activity Recognition

David Grün

TECO, Karlsruhe Institute of Technology

Abstract. Human activities are made of complex sequences of motor movements. This makes designing neural architectures for Human Activity Recognition (HAR) a challenging task. As HAR is a time series classification problem, convolutional and recurrent units are needed to capture temporal context. Meanwhile, the input data oftentimes consists of data from multiple sensors. There might be information in the way different sensors relate to each other so it is important to extract this information aswell. This review gives an overview of the current state-of-the-art of using deep learning for HAR based on wearable on-body sensors. Since designing neural architectures for HAR tasks requires expert level knowledge, I'm going to propose a Neural Architecture Search (NAS) search space for HAR tasks.

Keywords: Human Activity Recognition, Wearable Sensors, Deep Learning, Neural Architecture Search

1 Introduction

1.1 Fundamentals

Human Activity Recognition is about classifying human activities based on observations using various sensors. Observations can made using external sensors like cameras or using on-body wearable sensors like accelerometers or gyroscopes[14]. Arguably, on-body sensors are favourable over cameras for multiple reasons[13]: On-body sensors do not require the setting to be stationary like cameras do. Sensors positioned in strategic locations on the body provide more accurate data. The data coming from wearable sensors is more target-specific as images/videos contain additional information such as the background. That's why this review will focus on HAR using wearable on-body sensors.

In sensor-based HAR, data is collected continuously from each sensor to form time series data. Data can come from a single sensor like contained in a smartphone, or from multiple sensors. A classification method then predicts an activity label based on the input time series data. Activities can be forms of locomotion like walking, running and standing. They can also be sporadic movements such grabbing an object or opening a door.

The state-of-the-art way of mapping the collected sensor data to class labels is deep learning. Deep learning refers to neural networks consisting of multiple

layers. Layers are arranged sequentially, where every layers output is the input of the next layer. When labeled data is available, neural networks can be trained in a supervised way using gradient descent.

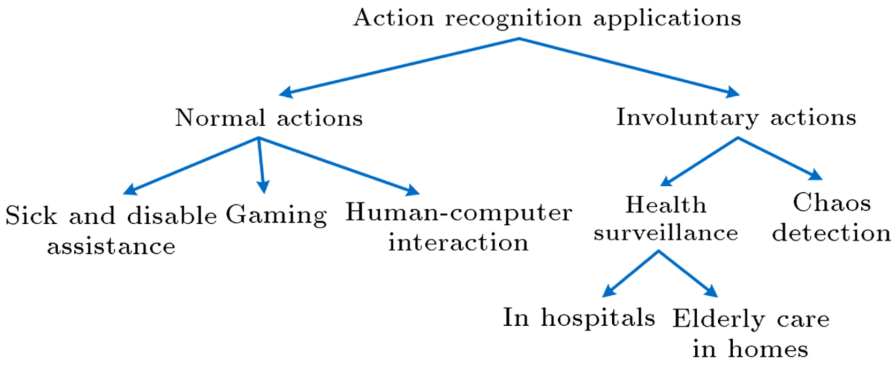

Fig. 1. Applications of Human Activity Recognition[1]

Human activity recognition has many real-world applications. In health care, Human Activity Recognition can be used to monitor the physical activity of patients to help in the treatment and prevention of obesity and cardiovascular diseases[2]. In the fields of Augmented Reality and Virtual Reality, Human Activity Recognition is an important component that allows the user to interact with its virtual environment[3]. An overview of applications of Human activity Recognition is given in Fig. 1.

1.2 Challenges

When designing a neural network architecture for a Human Activity Recognition task, one must take into account various challenges. As sensor-based Human Activity Recognition is a Time Series Classification task, it shares challenges with other Time Series Classification domains such as Speech Recognition and Natural Language Processing. The most important being temporal feature extraction. Since Human Activity Recognition oftentimes includes data from multiple sensor channels, another challenge is the detection of cross-channel relations.

In addition, different humans tend to perform certain activities in different ways. For instance people have different walking styles. Even the same person might perform the same activity in different ways. This leads to big variability in data samples depicting the same activity which makes classification more difficult. one must take into account that a neural network trained on observations taken from a small subset of people will perform significantly worse, when classifying activities from strangers.

Data sets are rather small because the collection of sensor data is time consuming. Designed architectures must be able to learn from a small amount of

OPPORTUNITY						Skoda		
Gestures			Modes of Locomotion					
Name	# of Repetitions	# of Instances	Name	# of Repetitions	# of Instances	Name	# of Repetitions	# of Instances
Open Door 1	94	1583	Stand	1267	38,429	Write on Notepad	58	20,874
Open Door 2	92	1685	Walk	1291	22,522	Open Hood	68	24,444
Close Door 1	89	1497	Sit	124	16,162	Close Hood	66	23,530
Close Door 2	90	1588	Lie	30	2866	Check Gaps Door	67	16,961
Open Fridge	157	196	*Null*	283	16,688	Open Door	69	10,410
Close Fridge	159	1728				Check Steering Wheel	69	12,994
Open Dishwasher	102	1314				Open and Close Trunk	63	23,061
Close Dishwasher	99	1214				Close both Doors	69	18,039
Open Drawer 1	96	897				Close Door	70	9783
Close Drawer 1	95	781				Check Trunk	64	19,757
Open Drawer 2	91	861						
Close Drawer 2	90	754						
Open Drawer 3	102	1082						
Close Drawer 3	103	1070						
Clean Table	79	1717						
Drink from Cup	213	6115						
Toggle Switch	156	1257						
Null	1605	69,558						

Fig. 2. Class imbalance of popular HAR datasets[5]

data and then apply their knowledge on a lot of data. There is certain activities that happen very rarely and are hard to simulate. For example accidents. Not being able to collect more data of a certain activity leads to an imbalance in the amount of training samples per class. Fig. 2 shows the class imbalance in popular HAR datasets. For example, the class "Open Fridge" has only 196 instances while the "Null" class has 69558. During training, classifiers must be encouraged to perform equally well on samples from both big and small classes.

Human Activity Recognition is commonly performed by embedded devices such as smart watches. Embedded devices have limited resource capabilities. That's why it's important to design classifiers that are not only accurate, but also are computationally efficient.

2 State-of-the-Art Human Activity Recognition

2.1 Deep Learning for human activity recognition

Due to their depth, Deep Neural Networks are able to perform well on increasingly complex classification problems. They became the state-of-the art method to solve classification problems of various domains. Their ability to capture temporal features through convolution operations and recurrent units makes them suitable for Time Series Classification. As Human Activity Recognition is a subdomain of Time Series Classification, Deep Neural Networks have become the state-of-the-art in Human Activity Recognition as well. Human Activity Recognition comes with specific challenges, that are not part of Time Series Classification in general. Custom architectures are required to address those challenges.

2.2 State-of-the-Art Neural Architecture

State-Of-The-Art HAR using neural networks evolves around 3 tasks. The first task is to extract local temporal features from the input time series data. In the next step, interactions between different sensor-channels have to be captured, based on the temporal features. In the last step, the local temporal features and the interactions between sensor-channels have to be contextualized on a

global temporal level. The general structure of such an architecture is shown in Fig. 3. When evaluating the performance of models designed for HAR tasks, it's important to account for class imbalance.

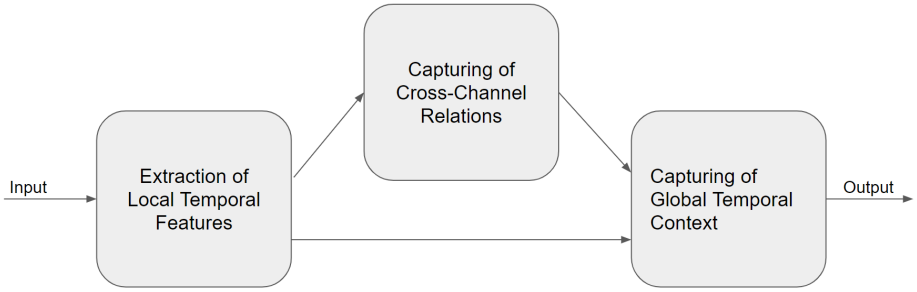

Fig. 3. General state-of-the-art architecture for sensor-based HAR

2.3 Local Temporal Feature Extraction

A single data point, as part of a time series, holds no valuable information on it's own. The data point needs to be set into relation with adjacent sensor values to extract temporal features. Those features then hold actual valuable information. Convolutional Neural Networks were first introduced into HAR in [4]. A single convolutional layer was used to extract temporal features.

A convolutional layer for time series classification takes an input sequence $x \in \mathbb{R}^{C \times D \times T}$, where C denotes the amount of input channels, D the amount of sensors and T the amount of time steps. The layer then produces an output $y \in \mathbb{R}^{C' \times D \times T'}$. The sensor dimension remains untouched, while C' is the amount of output sensor channel features. The feature maps are computed by sliding a 1-dimensional filter of length k along the temporal dimension of the input sequence. The filter computes a weighted sum over k data points adjacent on the time axis, where the weights are learnable via gradient descent. If the input sequence consists of more than one channel a different filter is applied to each channel to produce the weighted sum. The weighted sums produced by a filter are stored in a feature map. To produce C' feature maps, C' filters are needed.

Multiple convolutional layers can be arranged in sequence to progressively learn more complex features with increasing depth. DeepConvLSTM[5] as well as the HAR framework proposed in [6] feature 4 convolutional layers. Fig. 4 illustrates the sliding window operation using multiple layers. Layer $(l-1)$ provides the input sensor data two filters then compute the 2 feature maps of layer l. Layer $(l+1)$ is composed of one feature map, which is computed from the two feature maps of Layer l using 2 filters.

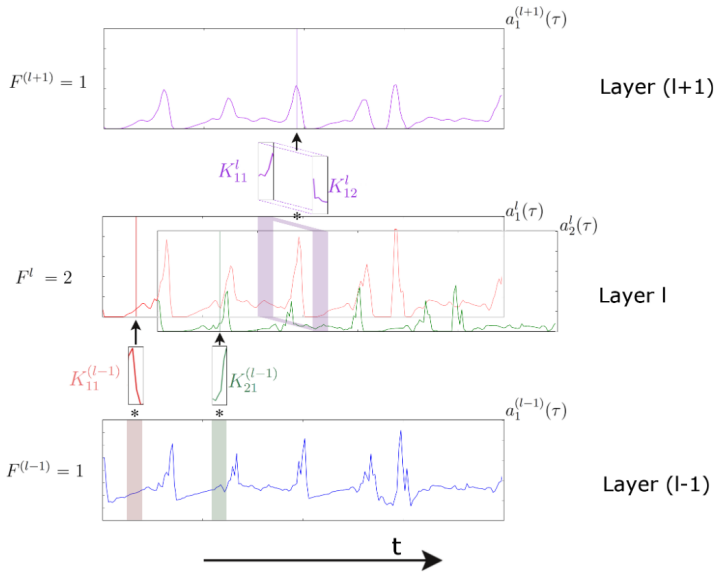

Fig. 4. Representation of a temporal convolution over a single sensor channel in a three-layer convolutional neural network[5]. For layer l, F^l denotes the amount of feature maps. K_{jf}^l is the filter applied to channel channel f in layer l to create the feature map j of layer $(l+1)$.

2.4 Extraction of Cross-Channel Relations

In HAR the input oftentimes consists of data from multiple sensors. In the previous step temporal features were extracted from every sensor in isolation. But there might lie information in the way different sensor channels behave in relation to each other.

The Cross-Channel Interaction Encoder (CIE) proposed in [6] is designed to capture the interaction between any two sensor channels. As input, it takes the feature maps computed by the convolution operation in the previous step. The correlation $a_t^{d,d'}$ between sensor channel features x_t^d and $x_t^{d'} \in \mathbb{R}^C$ at time step t is computed via a dot-product. The correlations are then normalized using the embedded gaussian function:

$$a^{d,d'} = \frac{\exp\left(f\left(x_t^d\right) \cdot g\left(x_t^{d'}\right)\right)}{\sum_{d'=1}^{D} \exp\left(f\left(x_t^d\right) \cdot g\left(x_t^{d'}\right)\right)},$$

where f and g are linear functions providing learnable parameters. The output $o_t^d \in \mathbb{R}^C$ for sensor channel d at time step t is then computed as

$$o_t^d = v\left(\sum_{d'=1}^{D} a^{d,d'} \cdot h\left(x_t^{d'}\right)\right),$$

where h and v are more linear funcitons providing learnable parameters. The output of the CIE as a whole then has the same dimensions as its input.

2.5 Capturing Global Temporal Context

After extracting local features and correlations between sensors, the data is contextualized on a global level to compute class probabilities. In the process of extraction cross-channel relations, valuable information lying in the features computed by the convolution operation is lost. That's why it's important to form a residual connection to include both temporal feature maps and cross-channel relations in the contextualization process as done in [6].

Long-Short-Term-Memory recurrent cells (LSTMs) perform better at capturing temporal context than plain linear layers[5]. LSTMs are invoked for every time step of their input sequence. Meanwhile they have a hidden state to keep track of temporal context. LSTMs can be unidirectional or bidirectional. Unidirectional LSTMs process the input time series in chronological order only while bidirectional LSTMs process the input time series in reversed order aswell. It has been shown, that unidirectional LSTMs perform better on some HAR datasets, while bidirectional LSTMs performed better on other HAR datasets[7]. In some cases it was best to use a combination of unidirectional and bidirectional LSTM cells. So the LSTM variation has to be chosen based on the HAR task. Gated Recurrent Units (GRU) is another kind of recurrent layer. It replaced LSTMs in [6]. Compared to LSTMs, GRUs have fewer parameters and thus are easier to train[6].

In time series data, not every time step is equally important. When processing the last time step, the recurrent layer has collected the most information from all previous steps. That's why a plausible strategy is to discard the output from every previous time step and only regard the output from the last step. This is the strategy used by DeepConvLSTM[5]. A more general approach is to regard the output of every time step and compose the output of the recurrent layer of a weighted sum over the output from all time steps. The weights are parameters trainable via gradient descend. Recurrent layers implemented like this are called attentional. DeepConvLSTM-Attn[9] extended DeepConvLSTM[5] by adding attention to each LSTM unit, which led to better performance. If the output from the last time step is infact the most important one, the attentional recurrent layer can learn to ignore the output from all previous time steps. So it's logical to assume that attentional recurrent layers are superior to the former approach.

2.6 Model evaluation

A core challenge in HAR is class imbalance. Assume we would evaluate a model on the OPPORTUNITY dataset and the model would predict the class to be "Null" for every input sample. Using the percentage of correctly classified samples would give the model an accuracy of 75% (see Fig. 2) although the model got the samples of all classes except the "Null" case wrong. The F_1 score, as

suggested in [5], accounts for that by considering the accuracy for every class equally. The F_1 score is computed as follows:

$$F_1 = \sum_i 2 \cdot w_i \cdot \frac{precision_i \cdot recall_i}{precision_i + recall_i},$$

where i is the class index, $w_i = \frac{n_i}{N}$, with n_i is the size of class i and N is the size of the dataset. $precision_i$ is defined as $\frac{TP}{TP+FP}$ and $recall_i$ is defined as $\frac{TP}{TP+FN}$, where TP, FP and FN represent the amount of true positive, false positive and false negativ classifications regarding class i.

2.7 Example Architecture

The architecture presented in Fig. 5 achieves state-of-the art performance on common HAR datasets. As input, it takes time series data from multiple sensors. A convolutional backbone is used to extract local temporal features from the input time series. The Cross-Channel Interaction encoder then extracts cross-channel information from the feature maps produced by the convolutional backbone. The output of the Cross-Channel Interacion encoder and the feature maps then are added together to form the input to a Attentional GRU Encoder to capture global temporal context. A Softmax operation then is applied to the output of the Attentional GRU Encoder to obtain class probabilities.

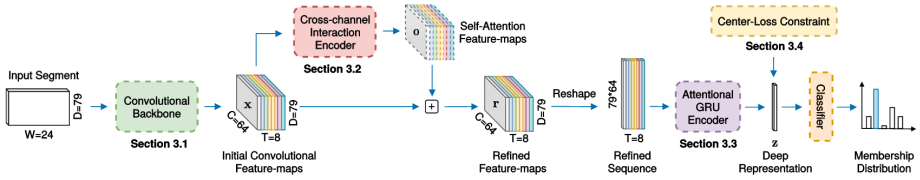

Fig. 5. State-Of-The-Art Architecture for sensor-based HAR proposed in [6]

During the training process, center loss is used in addition to cross entropy loss. Center loss encourages the model to minimize the euclidean distance between features obtained from samples of the same class. This helps to combat the problem of high intra-class variability, which is a key challenge in sensor-based HAR.

3 Neural Architecture Search

Specific HAR problems need specific neural architectures. Designing neural architectures for specific HAR tasks requires expert level knowledge. The basic idea of Neural Architecture Search (NAS) is to provide a search space instead of

a specific model. A search strategy then tries to find the best neural architecture contained in the search space, thus taking responsibility away from the model designer. Elsken et al. gave a good overview over NAS in their survey[10].

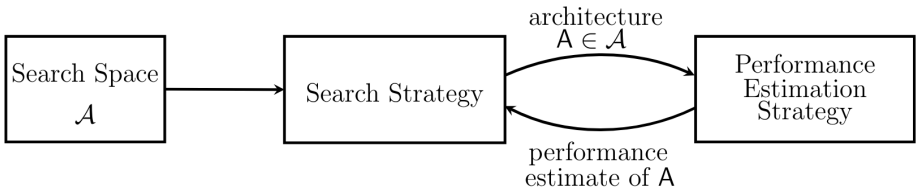

Fig. 6. Abstract illustration of Neural Architecture Search methods [10]

3.1 Search space

The search space defines all possible architectures the optimisation algorithm can choose from. For example, the search space can be defined to be n layers long. For each layer, the search space could contain linear, convolutional, pooling and recurrent layers. For a architecture of length n, this would make the search space contain 4^n possible combinations. For each possible layer, there can also be different options for hyperparameters in the search space increasing the amount of possible combinations even further. For example different filter sizes for convolutional layers, or the amount of nodes in a linear layer.

3.2 Search Strategy

Since the size of the search space increases exponentially, trying out every single combination of operations in the search space is oftentimes not an option. The search strategy defines how architectures are chosen from within the search space. Many different search strategies can be used to explore the space of neural architectures, including random search, Bayesian optimization, evolutionary methods, reinforcement learning, and gradient-based methods[10]. NAS using standard differential evolution as the search strategy was successfully applied to time series classification[11].

Discrete Search strategies such as evolutionary methods, reinforcement learning are computationally expensive. Differential Architecture Search (DARTS)[12] is an alternative approach that allows the training of an optimal neural architecture via gradient descend. Instead of picking one operation from the search space, all operations are computed simultaneously. Their output is then weighted by a weight vector α. The weight vector α can then be learned via gradient descend, which is much more efficient than discrete search strategies.

4 Proposed NAS Search Space for sensor-based HAR

Since designing architectures for specific sensor-based HAR tasks is a complex task and requires expert level knowledge, I am going to propose a NAS search space for HAR based on current state-of-the-art architectures. The general structure of the search space is taken from the current structure shown in Fig. 3. The search space is split into 3 blocks:

- The first block is responsible for local temporal feature extraction. It is composed of Inception modules instead of plain convolutional layers.
- The second block is responsible for extracting inter-channel relations.
- The third block is responsible for global contextualisation. It will be composed of various recurrent units.

4.1 Block 1 (Local Temporal Feature Extraction)

Convolutional layers perform convolution using filters of the same length only. However, it might be beneficial to extract features of different lengths. That is not possible with conventional convolutional layers. In [8], the Inception module was first introduced into time series classification. The basic idea is to use multiple filters of different lengths simultaneously to extract features of different resolutions along to temporal dimension of the input.

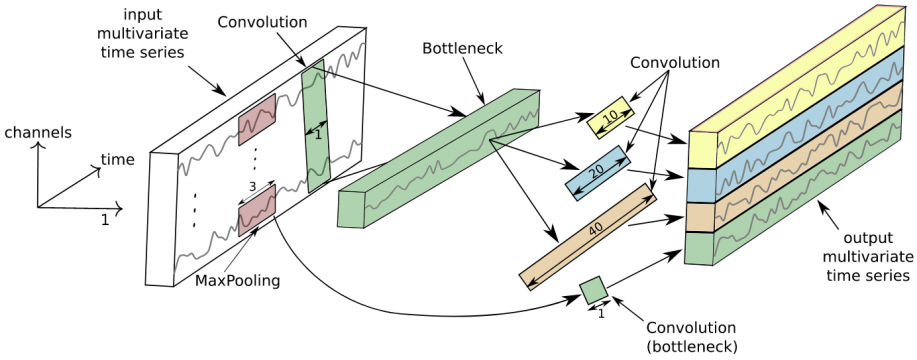

Fig. 7. Inception module for time series classification performing convolution with filters of length 10, 20 and 40 as well as a max pooling operation[8]

The first component of the inception module is the "Bottleneck". It performs a 1×1 convolution on the input time series to reduce the channel dimension and thereby the complexity of the module. The amount of output channels of the bottleneck is an important hyperparameter. Convolution operations with filters of variable length are then performed simultaneously on the output of the bottleneck. The reduction of the input complexity through the bottleneck

allows for relatively long filters. For example, in the module presented in Fig. 7, 3 filters with filter sizes 10, 20 and 40 are used. In parallel, a pooling operation is performed on the input time series to make the model as a whole invariant to small perturbations in the input data. The output of the pooling operation is reduced by its own bottleneck. The output all performed convolutions and the pooling operation is concatenated in the channel dimension to form the output of the inception module.

DeepConvLSTM[5] as well as the HAR framework proposed in [6] feature 4 convolutional layers. The search space for Block 1 will be composed of 5 sequential layers, increasing the dept by 1 compared to previous approaches to allow for the extraction of more complex features. The search space will include inception modules as well as an identity operation in case 5 sequential inception modules are too much. A inception module will consist of 3 filters. Current state-of-the-art architectures use a filter size of 5[6] and longer filters have been shown to provide better results[8]. That's why the search space will start at a small filter size around 5 and will include increasingly longer filters. Available options for the filter lengths are $\{\{3,5,7\},\{5,7,9\},\{8,12,16\},\{10,16,22\},\{14,20,26\}\}$. If the input time series is shorter than a filter length in the search space, the filter length is removed from the search space. The available number of produced feature maps are $\{4,8,16,32,64,128\}$. For the bottleneck size, parameters $\{8,16,32,64\}$ are available. Ranges for hyperparameters are chosen to be °°relatively wide to keep engineering bias low.

4.2 Block 2 (Extraction of Cross-Channel Relations)

The second block will take the feature map computed by the inception modules in the first block as input.

The search space for the second block will contain the Cross-Channel Interaction Encoder[6] since it has been designed for the extraction of cross-channel relations for sensor-based HAR specifically and proved competent in doing so. For a specific HAR dataset, there might be no benefit in extracting cross-channel information. Therefore, a null-operation is added to the search space of the second block. The search space for the second block is relatively small at two options only.

4.3 Block 3 (Capturing Global Temporal Context

The third block will take as input, the cross-channel relations computed by the second block combined with a residual connection from the feature maps output from the inception modules of the first block.

Existing deep learning frameworks for HAR feature 2 recurrent layers for contextualization[5][6]. The search space for Block 3 will be composed of 3 sequential layers, again increasing the dept by 1 compared to previous approaches. Unidirectional as well as bidirectional LSTMs are included in the search space, since both variations show different performance on different datasets[7]. A GRU is included in the search space aswell. Additionally, a identity option is added

to the search space to prevent overfitting due to high depth. All recurrent layers will feature attention to learn the importance of time steps. The last recurrent layer will have as many cells as there are classes in the dataset. The amount of recurrent cells in the previous layers will be chosen from $\{8, 16, 32, 64, 128, 256\}$.

5 Conclusion

This review gave an overview of the state-of-the-art neural architectures for sensor-based HAR. Current state-of-the-art HAR evolves around extracting temporal features as well as cross-channel relations from the input time series and then contextualizing those features on a global temporal level. I proposed a NAS search space for HAR based on current state-of-the-art models. The proposed search space allows models to be slightly deeper than existing frameworks to be able to detect more complex features. It contains inception modules instead of conventional convolutional layers to allow for the extraction of features of different resolutions. For the extraction of cross-channel relations it uses the Cross-Channel Interaction Encoder, which is optional since cross-channel relations might not be important. For global contextualisation, the search space contains unidirectional and bidirectional LSTMs as well as GRUs.

References

1. Mokari et al.: Recognizing Involuntary Actions from 3D Skeleton Data Using Body States. Scientia Iranica (2020)
2. Labs et al.: Human Activity Recognition for Healthcare using Smartphones. the 2018 10th International Conference (2018)
3. Fangbemi et al..: Efficient Human Action Recognition Interface for Augmented and Virtual Reality Applications Based on Binary Descriptor. Springer International Publishing (2018)
4. Zeng, Nguyen, Yu, Mengshoel, Zhu, Wu, and Zhang: Convolutional Neural Networks for Human Activity Recognition using Mobile Sensors. Sixth International Conference on Mobile Computing, Applications and Services (2014)
5. Ordóñez, F. J., Roggen, D.: Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. Sensors (2016)
6. Alireza Abedin, Mahsa Ehsanpour, Qinfeng Shi, Hamid Rezatofighi, and Damith C. Ranasinghe. 2021. Attend and Discriminate: Beyond the State-of-the-Art for Human Activity Recognition Using Wearable Sensors. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 5, 1, Article 1 (March 2021)
7. Murad, A., Pyun, J.: Deep Recurrent Neural Networks for Human Activity Recognition. Sensors (2017)
8. Fawaz, H. I., et al.: InceptionTime: Finding AlexNet for time series classification. Springer Nature (September 2020)
9. Vishvak S Murahari and Thomas Plötz. On attention models for human activity recognition. In Proceedings of the 2018 ACM International Symposium on Wearable Computers.
10. Elsken, T., Metzen, J. H., Hutter, F.: Neural Architecture Search: A Survey. Journal of Machine Learning Research 20. (2019)

11. Rakhshani, H. et al.: Neural Architecture Search for Time Series Classification. IEEE (2020)
12. Liu, H. et al.: DARTS: Differentiable Architecture Search. ICLR 2019 Conference Blind Submission (2019)
13. Yang, J. B. et al.: Deep Convolutional Neural Networks On Multichannel Time Series For Human Activity Recognition. Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
14. Bulling, A. et al.: A Tutorial on Human Activity Recognition Using Body-Worn Inertial Sensors. ACM Computing Surveys, Vol. 46 (2014)