Ultra-low power HW accelerator for the integration of Binary Neural Networks on inertial sensors

Antonio De Vita

UNIVERSITY OF SALERNO

DEPARTMENT OF INDUSTRIAL ENGINEERING

Ph.D. Course in Industrial Engineering
Curriculum in Electronic Engineering – XXXIII Cycle

ULTRA-LOW POWER HW ACCELERATOR FOR THE INTEGRATION OF BINARY NEURAL NETWORKS ON INERTIAL SENSORS

Supervisor
Prof. Gian Domenico Licciardo

Ph.D. student
Antonio De Vita

Scientific Referees
Prof. Nicola Petra
Prof. Maurizio Valle

Ph.D. Course Coordinator
Prof. Francesco Donsì

List of publications

Journal articles

De Vita, A., Russo, A., Pau, D., Di Benedetto, L., Rubino, A., Licciardo, G.D. (2020) A Partially Binarized Hybrid Neural Network System for Low-Power and Resource Constrained Human Activity Recognition. IEEE Transactions on Circuits and Systems I: Regular Papers (Early Access).
doi: 10.1109/TCSI.2020.3011984.

De Vita, A., Licciardo, G.D., Femia, A., Di Benedetto, L., Rubino, A., Pau, D. (2019) Embeddable Circuit for Orientation Independent Processing in Ultra Low-Power Tri-Axial Inertial Sensors. IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 6, pp. 1124-1128.
doi: 10.1109/TCSII.2019.2928476.

Licciardo, G.D., Di Benedetto, L., De Vita, A., Rubino, A., Femia, A. (2019) A Bit-Line Voltage Sensing Circuit With Fused Offset Compensation and Cancellation Scheme. IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 66, no. 10, pp. 1633-1637.
doi: 10.1109/TCSII.2019.2928456.

Conference proceedings

De Vita, Pau, D., A., Di Benedetto, L., Rubino, A., Pétrot, F., Licciardo, G. D. (2020) Low Power Tiny Binary Neural Network with improved accuracy in Human Recognition Systems. 2020 23rd Euromicro Conference on Digital System Design (DSD), (*early access*).
doi: 10.1109/DSD51259.2020.00057.

De Vita, A., Pau, D., Parrella, C., Di Benedetto, L., Rubino, A., Licciardo, G.D. (2020) Low-Power HW Accelerator for AI Edge-Computing in Human Activity Recognition Systems. 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp. 291-295.
doi: 10.1109/AICAS48895.2020.9073913.

De Vita, A., Licciardo, G.D., Femia, A., Di Benedetto, L., Pau, D. (2019) μW Pre-processing Unit for Virtual Sensors Based on Tri-axial Smart Accelerometers. 2019 17th IEEE International New Circuits and Systems Conference (NEWCAS), pp. 1-4.
doi: 10.1109/NEWCAS44328.2019.8961264.

De Vita, A., Licciardo, G.D., Femia, A., Di Benedetto, L., Rubino, A., Pau, D. (2019) Low-Power Integrated Circuit for Orientation Independent Acquisitions from Smart Accelerometers. AISEM Annual Conference on Sensors and Microsystems.
doi: https://doi.org/10.1007/978-3-030-37558-4_6

De Vita, A., Licciardo, G.D., Di Benedetto, L., Pau, D., Plebani, E., Bosco, A., (2018) Low-power Design of a Gravity Rotation Module for HAR Systems Based on Inertial Sensors. 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 1-4.
doi: 10.1109/ASAP.2018.8445130.

Table of Contents

List of Figures

List of Tables

Abstract

The research activity described in this thesis aims to demonstrate the possibility to embed Artificial Intelligence (AI) capabilities in wearable and portable devices by deploying and executing Neural Network (NN) models close to the sensing element. Among AI models, Deep Learning (DL) and Deep Neural Networks can achieve high performance in many tasks, e.g. image classification, activity recognition, and so on. However, DL models usually require a huge amount of memory resources and high-performance digital architecture to be executed. These specifications are hardly met by wearable and portable devices, which have to be as small as possible and guarantee a satisfactory battery lifetime. For this reason, the cloud computing strategy is often used. However, higher latencies occur in this case, which can be unacceptable in many latency-sensitive applications, such as autonomous vehicles or assisted microsurgery. Moreover, the data transfer consumes network bandwidth and energy. In this context, moving the computation close to the device is highly demanded, and it is named edge-computing. However, deploying DL models on edge devices is still a challenge. General-purpose platforms (i.e. CPUs, GPUs) are not the best solution in terms of energy efficiency, especially for wearable and battery-powered devices, where the device lifetime is a major concern. Thus, a lot of research is being made about the design of custom HW accelerators for DL and to move the circuitry needed to implement the computation closer to the sensing element, thus obtaining a smart sensor. In this thesis, a novel Hybrid Binary Neural Network (HBN) model is proposed, which exploits the advantages of Binarized Neural Networks (BNNs). Human Activity Recognition (HAR) based on inertial sensors has been selected as a case study. Also, a pre-processing algorithm has been developed to solve the device-orientation problem for 3-axis accelerometers. The pre-processing operations can improve the accuracy of the proposed system in some conditions when it is used in conjunction with the HBN model. The results show an accuracy of up to 99% in recognizing 5 human activities. After having developed the model, a custom ultra-low power HW accelerator has been designed and implemented with both FPGA and CMOS standard cells. Due to the very low operating frequency required by HAR applications, power consumption has been reduced by reducing the number of resources. The design can implement both the pre-processing operation and the HBN model. The results show that the HW accelerator has a power consumption of 6.3 µW and an area occupation of 0.20 mm2 when synthesized with CMOS 65 nm Low-Power (LP) High Voltage Threshold (HVT) standard cells. The proposed design has at least 7.3 times lower power consumption than the state-of-the-art solution. Also, a FPGA-based demo board has been developed to demonstrate the real-time operation of the system.

Introduction

The research activity presented in this thesis aims to design an ultra-low power Hardware (HW) accelerator for Neural Networks (NNs) that can be embedded in wearable and portable devices. The objective is to demonstrate the possibility to integrate the HW accelerator into the sensor circuitry, in order to realize an ultra-low power Artificial Intelligence (AI)-based smart sensor.

Among AI techniques, Deep Learning (DL) is currently widely used thanks to its ability to reach very high performance in terms of accuracy, and it can outperform human performance in many tasks. Unfortunately, DL requires a huge amount of computations, which is far beyond the standard capabilities of modern portable or wearable devices. For this reason, cloud computing is often used because it offers scalable storage and processing services. In addition to scalability, the cloud provides easier maintainability than distributed Internet of Things (IoT) based solutions can offer. However, in many latency-sensitive applications, for example in autonomous vehicles or assisted microsurgery, the delay waiting for a result from a cloud-based AI is unacceptable. Moreover, sending a huge amount of data to the cloud for storage and processing might consume all network bandwidth making it a non-scalable and energy-hungry solution. Thus, moving the computation close to the device is highly demanded in many applications, and it is named edge-computing. However, deploying DL models on edge devices is still a challenge. General-purpose platforms (i.e. CPUs, GPUs) are not the best solution in terms of energy efficiency, especially for wearable and battery-powered devices, where the device lifetime is a major concern. Currently, a lot of research is being made about the design of custom energy-efficient HW accelerators for Deep Learning. Ultimately, great advantages can be gained in terms of reduced power consumption and area occupation by moving the circuitry needed to implement the computation closer to the sensing element, either on a single chip or on a System on Chip (SoC), thus obtaining a smart sensor.

In this regard, this research activity aims to design an ultra-low power smart sensor targeting μW power consumption. This is made possible by the use of a binarized version of a Convolutional Neural Network (CNN), which is a CNN in which weights and the activations are binarized, i.e. constrained to +1 or −1. Binarization enables the implementation of NN models on resource-constrained devices with lower power consumption and footprint. However, a major drawback of Binarized Neural Networks (BNN) is a significant decrease in accuracy compared to standard CNN models. As a solution, a new Hybrid Binary Neural Network (HBN) is proposed, where binarization has been carefully implemented to obtain a tradeoff between accuracy and required resources. Human Activity Recognition (HAR) is taken as a case study to test the proposed HBN model. More in detail, HAR based on inertial sensors, such as accelerometers and gyroscopes, has been considered. Also, a preprocessing algorithm has been developed, which allows extracting useful features from the raw data coming from a 3-axis accelerometer. Pre-processing operations allow compensating the accuracy loss in some conditions. The proposed HAR system is made up of a sensor, the pre-processing module, and the HNN. The sensor is a low-power digital 3-axis MEMS accelerometer, which can be used in conjunction with a 3-axis gyroscope. During pre-processing, raw data from the accelerometer are filtered to separate the high-frequency component from the low frequency/DC component, which roughly corresponds to the gravity acceleration. Then, a reference frame transformation is performed to represent the high-frequency component compared to a common reference system, to eliminate the dependence of the acquired data on the sensor orientation. Finally, classification is achieved through the HBN. The model has been constructed and tested in Lasagne, by implementing the binary layers with custom functions. The results show that the system reaches an accuracy of up to 99% in recognizing 5 different human activities.

After having developed the model, a custom ultra-low power HW accelerator has been designed and implemented with both FPGA and CMOS standard cells. Due to the very low operating frequency required by HAR applications, power consumption has been reduced by reducing the number of resources. The design can implement both the pre-processing operation and the HBN model. The results show that the HW accelerator has a power consumption of 6.3 μW and an area occupation of 0.20 mm^2 when synthesized with CMOS 65 nm Low-Power (LP) High Voltage Threshold (HVT) standard cells. A FPGA-based demo board has been also realized to prove the real-time operation of the proposed HAR system.

The structure of the thesis is structured as follows.

Chapter I briefly introduces the theory behind NNs, describing how these models are built and trained. A focus is devoted to BNNs, which are the basis to understand the proposed HBN model. Also, state-of-the-art about HW implementations is provided.

Chapter II introduces HAR, which is the application field that has been used as a case study for the proposed system. An overview of algorithms and

HW solutions for HAR is provided, along with the description of two public datasets that have been used to test the proposed system.

Chapter III introduces the device-orientation problem in inertial sensors. State-of-the-art solutions are described based on the types of sensors that are involved. Then, a new solution is proposed to solve the device-orientation problem thanks to an HW-friendly algorithm.

Chapter IV introduces the proposed HBN model and describes how it is used to build the HAR system. The results from many tests on both two public datasets and a custom dataset are reported in terms of accuracy.

Chapter V describes the proposed HW accelerator. The results are reported for both the FPGA and CMOS standard cell implementations. Also, the FPGA-based demo board is briefly described.

Chapter I
Introduction to Neural Networks

I.1 Advancements in Deep Learning

Over the last few years, we have witnessed the huge spread of Deep Learning (DL) for a great variety of applications (LeCun, 2015), especially in *image recognition* (Krizhevsky, 2012), (Tompson, 2014), *speech recognition* (Hinton, 2012), (Nassif, 2019), *autonomous driving* (Li, 2019a) (Grigorescu, 2019), etc. As depicted in Figure I.1, DL is a subset of Machine Learning (ML), which in turn is a subset of Artificial Intelligence (AI). The term AI was first coined in 1956 by John McCarthy, who defines it as "the science and engineering of making intelligent machines" (McCharthy, 2007). This basically means the automated reproduction of human cognitive activities. Among the various AI techniques, ML allows the automated extraction of abstract knowledge from data and experience, which means ML algorithms can make predictions or decisions learning from data with which they are *trained*, without being explicitly programmed to do so. DL is a special case of ML, in which multi-layered representations are used to extract knowledge. The reason why DL has been so successful is that it allows overcoming a major issue of conventional ML techniques. Indeed, the latter are not able to extract useful information directly from raw data. Consequently, considerable domain-specific expertise is required to identify a suitable internal representation or a feature vector from which the knowledge can be effectively extracted. This process is usually called *feature extraction*. The great advantage of DL is the ability to avoid the feature extraction phase, thanks to its multi-layered structure, thus allowing extracting useful information from the raw data directly. However, there is a price to pay for this advantage. In fact, DL models have a huge computational complexity and require a large amount of memory to store all their parameters: as an example, the AlexNet (Krizhevsky, 2012) architecture requires approximately 837M FLOPs to perform a single inference step, and 60M parameters need to be stored. Thus, the execution of a DL model is usually unfeasible on CPUs or MCUs, and larger parallel-processing platforms, such as GPUs, must be used. Alternatively, DL can be delegated to the cloud, implementing the so-called *cloud computing* paradigm (Bianchi, 2019).

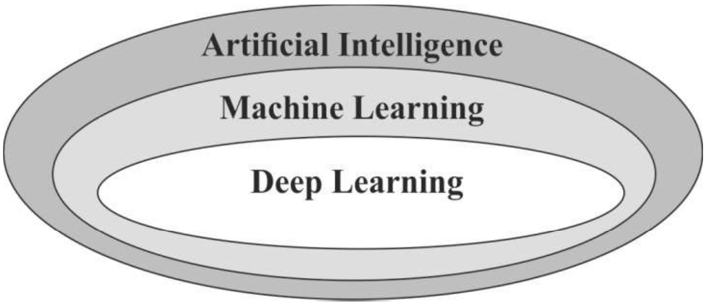


Figure I.1 *Graphical representation of the relation between Artificial Intelligence, Machine Learning, and Deep Learning.*

I.2 The Classification Problem

One of the main tasks of DL models is classification, which is the task of assigning a label to input data from a fixed set of categories. A typical example is image classification: for example, an image classification model can take a single image and assigns a probability to 4 labels (cat, dog, hat, mug). This is a trivial task for a human to perform, but at the same time is a very complex problem to solve with conventional computer vision algorithms. In fact, it is not easy to come up with an algorithm for identifying cats in images, because each category should be carefully specified and described in the code. In doing so, so many aspects should be considered, such as viewpoint variation, scale variation, deformation, illumination conditions, etc. Therefore, a different approach must be used, which is referred to as the *data-driven* approach: many examples of each class are provided to the machine, and then, based on these examples, learning algorithms are developed that allows learning the features of all classes. The set of examples is almost always called *training dataset*. An example is shown in Figure I.2. Thus, the complete pipeline is the following:

1) *Input construction*: the input consists of a set of images (or a different kind of data for different tasks), each one is labeled with one of many different classes. This is the training dataset.
2) *Learning*: at this point, the objective is to use the training dataset to learn what every one of the classes looks like. This process is referred to as “training the classifier” or “learning a model”.
3) *Evaluation*: in the end, the quality of the classifier is evaluated by asking it to predict labels for a new set of images, that is different

from the training dataset. The predicted labels are then compared with the true labels (*ground truth)*.

The same concepts apply to other kinds of input data as well for example signals sampled by microphones or inertial sensors.

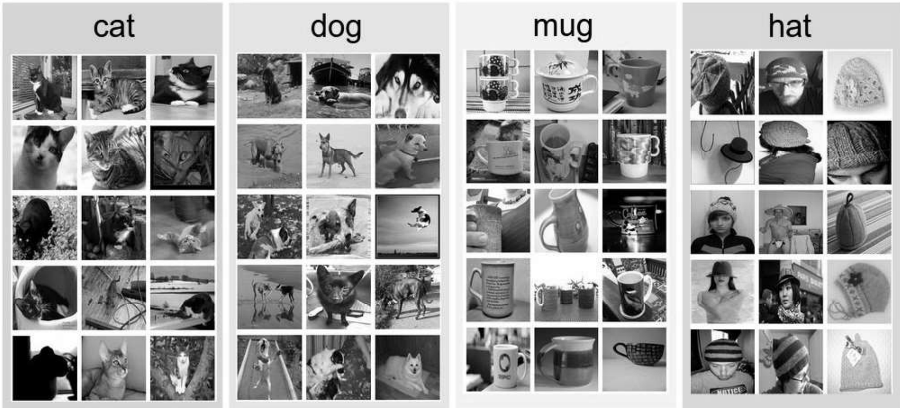


Figure I.2 *Example of a training dataset. In this dataset, 4 classes are considered: cat, dog, mug, hat. Each image in the dataset is labeled with one of the 4 classes.*

I.2.2 ***Score Function: the Linear classification example***

To understand the meaning of the computational model of a neuron, linear classification must be introduced. To this aim, image classification is again used as an example. Let us assume a training dataset of images $x_i \in R^D$, each one is associated with a label y_i. Here $i \in 1 \ldots N$ and $y_i \in 1 \ldots K$. That is, there are N examples (each one with dimensionality D) and K distinct categories. What we need is an approximation of the function that maps the raw image pixels to class scores. This is named *score function*, and it is defined as in (1).

$$f : R^D \mapsto R^K \tag{1}$$

The simplest possible function is the linear mapping:

$$f\left(x_i, W, b\right) = Wx_i + b \tag{2}$$

In (2), we are assuming that the image x_i has all its pixels flattened out to a single column vector of shape $[D\times1]$. The matrix W and the vector b are the parameters of the score function, and they have size respectively $[K\times D]$ and $[K\times1]$. The parameters in W are usually called *weights*, and b is called the *bias vector* because it influences the outputs scores, but without interacting with the actual data x_i. Note that the single matrix multiplication

Wx_i is effectively evaluating K separate classifiers in parallel, where each classifier is a row of W. The goal is to set the parameters, W and B, in such a way that the computed scores match the ground truth labels across the whole training set. Intuitively, we wish that the correct class has a score that is higher than the scores of the incorrect classes.

Since the images are stretched into high-dimensional column vectors, we can interpret each image as a single point in the space, i.e. each point is a point in D-dimensional space. If D is very large, we cannot visualize the D-dimensional space. But if we imagine squashing all the dimensions into only two dimensions, then we can try to visualize what the classifier is doing. This is represented in Figure I.3. Each row of W is a classifier for one of the classes. The geometric interpretation of these numbers is that as we change one of the rows of W, the corresponding line in the image-space will rotate in different directions. The biases b, on the other hand, allow our classifier to translate the lines. In fact, without the bias term, plugging $x_i = 0$ would always give a score of zero regardless of the weights, so all lines would be forced to cross the origin.

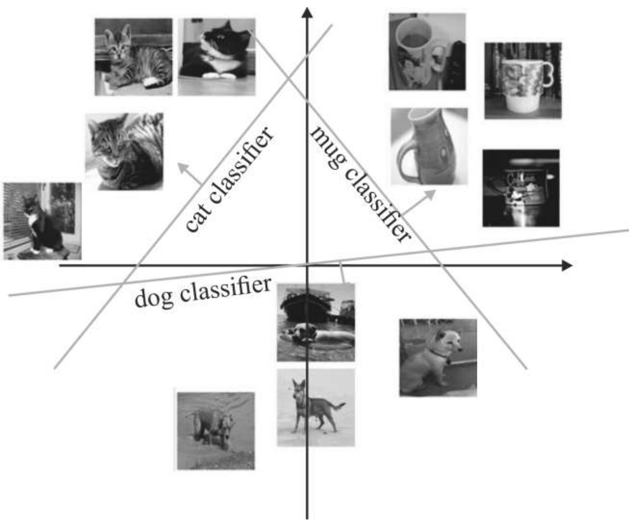


Figure I.3 *Representation of the image space, where each image is a single point, and three classifiers are visualized. The cat classifier line shows all points in the space that get a score of zero for the cat class. The arrow shows the direction of increase, so all points to the left of the cat classifier line have positive (and linearly increasing) scores, and all points to the right have negative (and linearly decreasing) scores.*

I.2.3 ***Loss Function***

Another important aspect of classification methods is the *loss function*. The loss function, sometimes also referred to as the *cost function* or the *objective*, measures how compatible a given set of parameters is, compared to the ground truth table in the training dataset. The loss will be high if we are doing a poor job of classifying the training data, and it will be low if we are doing well. Thus, the classification becomes an *optimization problem* in which we will minimize the loss function compared to parameters of the score function.

There are several ways to define the details of the loss function. A commonly used loss is the *Multiclass Support Vector Machine (SVM)* loss:

$$L_i = \sum_{i \neq y_i} \max\left(0, s_j - s_{y_i} + \Delta\right) \tag{3}$$

where s_j is the score for the j-th class, s_{yi} is the score for the correct class, and Δ is a fixed margin. The function in (3) is often called *hinge loss*. Thus, the SVM loss is set up so that the correct class for each image has a score higher than the incorrect classed by some fixed margin Δ. However, there is one problem that should be considered with the hinge loss: if a set of parameters W allows classifying correctly all the examples (so the loss is zero for each example), then any multiple of these parameters λW, where $\lambda > 1$, will also give zero loss. In order to remove this ambiguity, a preference should be specified for a certain set of weights. A typical way to do so is to modify the loss by adding a *regularization penalty* $R(W)$. An example is the $L2$ norm regularization, which gives preference to smaller weights by introducing an elementwise quadratic penalty over all the parameters:

$$R(W) = \sum_k \sum_l W_{k,l}^2 \tag{4}$$

Notice that the value of the regularization function in (4) does not depend on data, whereas it does depend on the weights. Thus, integrating the regularization term in (3), we obtain the complete SVM loss:

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W) \tag{4}$$

Two components can be identified: the *data loss* (which is the average loss over all examples) and the *regularization loss*. The parameter λ does not result from the optimization process, and it must be arbitrarily defined and then tuned to obtain the best classification performance. These parameters are called *hyperparameters*.

Besides SVM, another very used classifier is *Softmax*. Different from SVM, the Softmax classifier gives a more intuitive meaning to the score assigned to each class. Indeed, a probabilistic interpretation is assigned to the output: while the function mapping in (2) stays unchanged, the hinge loss is replaced by the *cross-entropy loss*:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \tag{5}$$

where f_j is the j-th element of the vector of class scores f. Even in this case, the complete loss can be computed by applying (4). The function:

$$S_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{6}$$

is the Softmax function. It takes a vector of real-valued scores and transforms it into a vector of values between 0 and 1 that sum to one. As a result, the expression in (6) can be interpreted as the probability that the j-th class is the correct one. Thus, ideally, we want the value of $S_j(z)$ to be close to 1 when j is the correct class, while we want it to be close to 0 when j is one of the wrong classes. In this way, the loss in (5) will be close to 0 if the probability associated with the correct class is high, while it will be higher otherwise.

I.3 Learning Parameters

After having defined the classifier and the loss function, an optimization method should be found to determine the set of weights W for the classifier that minimizes the loss function. During the optimization process, the classifier is said to *learn* its parameters, therefore we usually talk about the *learning process*. Typically, in ML models, parameters are learned by feeding them with many examples, that is we *train* a certain model on a specific dataset. For this reason, the optimization process is also called the *training process*.

I.3.1 ***Optimization: Gradient Descent***

Finding the best set of weights W might seem a very difficult or even impossible task, especially if we are trying to figure out the best configuration of weights for a whole DL model. However, the problem of identifying a set of weights W that is slightly better is significantly less difficult. This basically means that the approach is to start with a random W and then iteratively update it, making it better each time. On the basis of the above, making W better means updating it in order to get a lower value when evaluating the loss function. An initial strategy might be to generate random perturbations δW to W, and if the new loss is lower, then the update is performed. However, this is not the best solution. Indeed, the best direction along which W should be changed can be computed, as it is mathematically guaranteed to be the direction of the steepest descend. This direction will be related to the *gradient* of the loss function, which is a generalization of the slope for multiple variable functions. In practice, the gradient is a vector of slopes (more commonly referred to as *derivatives*) for each dimension in the input space. The mathematical expression for the derivative of a 1-D function compared to its input is:

$$\frac{d\,f(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{7}$$

When the function takes a vector of numbers instead of a single number, the derivatives are called *partial derivatives*, and the gradient is the vector of partial derivatives in each dimension. The procedure of repeatedly evaluating the gradient and then performing a parameter updating is called *Gradient Descent* (GD). A graphical representation of this process is shown in Figure I.4. At the beginning of the process, *W* is randomly set. Then the value of the cost function is decreased at each step by following the gradient.

An important parameter in GD is the size steps, determined by the *learning rate* hyperparameter. The effect of the learning rate on the optimization process is depicted in Figure I.5. If the learning rate is too small, then the optimization process will have to go through many iterations to converge, which will require a long time. On the other hand, if the learning rate is too high, the algorithm may diverge failing to find a good solution. Finally, not all functions have a regular trend, and this makes it more difficult to reach the minimum. In Figure I.6, two main challenges of GD are shown: if the random initialization starts the process on the left, then it will converge to a *local minimum*, which is not as good ad the global minimum; if it starts on the right, the gradient will be very low, and it will take a very long time to cross the plateau. Each iteration that leads to a parameter update is called *epoch.*

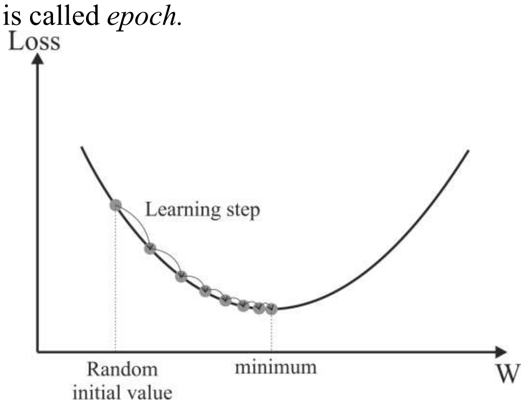


Figure I.4 *Graphical representation of the optimization process using Gradient Descent. The gradient of the loss function is computed at each step, and the parameters W are updated in the direction of the minimum.*

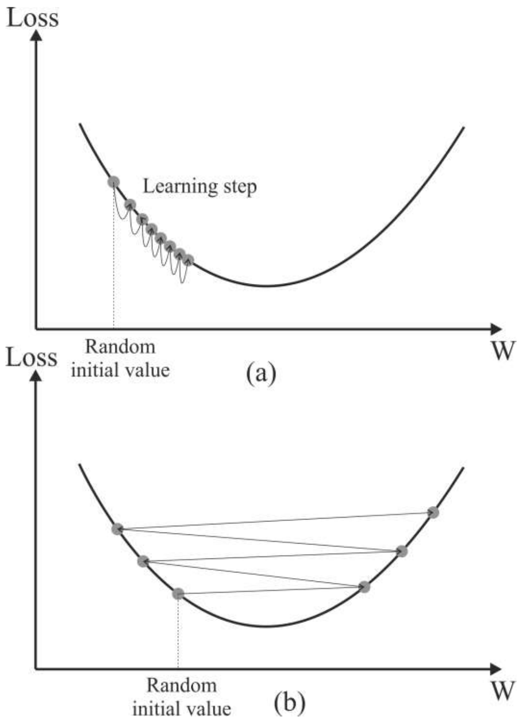


Figure I.5 *Impact of the learning rate on the convergence of the optimization process. In (a) the learning rate is too small, and the minimum is not reached. In (b) the learning rate is too high, and the process does not converge.*

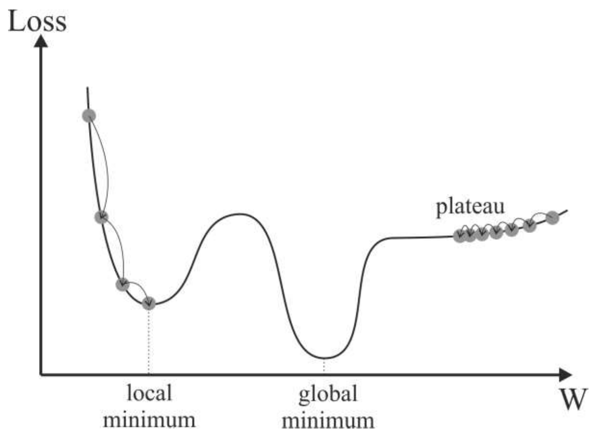


Figure I.6 *Example of a loss function with complex shape. Local minima and plateaus are the main issues: In the first case, the GD fails to reach the global minimum, as it gets trapped in a local minimum; in the second case, the gradient is very low and a large number of iterations are required to reach to effectively minimize the cost function.*

I.3.2 ***Mini-batch Gradient Descent and Stochastic Gradient Descent***

Notice that when using GD, the loss function must be evaluated on the whole training dataset before the parameters update happens. Nevertheless, in most applications, the training data can have millions of examples. Therefore, a huge amount of time would be required to compute the loss function for each update step. A very common approach is to compute the gradient over *batches* of the training data. This batch is then used to perform a parameter update. This method is called *Mini-batch Gradient Descent* (MGD). The reason this works well is that the examples in the training data are supposed to be correlated. Thus, the gradient from a mini-batch is a good approximation of the gradient of the full objective. In practice, much faster convergence can be achieved by evaluating the mini-batch gradients to perform more frequent parameter updates. The extreme case of this is *Stochastic Gradient Descent* (SGD), where the mini-batch contains only a single example. This is relatively less common because in practice it can be computationally much more efficient to evaluate the gradient for 100 examples, than the gradient for one example 100 times. Even though SGD technically refers to using a simple example at a time to evaluate the gradient, in most cases the term is used even when referring to MGD. The size of the mini-batch is a hyperparameter, and it is usually based on memory constraints. Typically, the size of the mini-batch is set to be a power of 2 because many vectorized operations work more efficiently when their inputs are sized in that way.

Due to their stochastic nature, MGD and SGD are much less regular than the standard GD algorithm. In particular, MGD is less regular than standard GD, and SGD is less regular than MGD. Thus, instead of gently decreasing until it reaches the minimum, the loss will experience some oscillations, decreasing only on average. Over time, it will end up being very close to the minimum, but once it gets there it will continue to oscillate around, never settling down. So, once the process stops, the final parameters are good, but not optimal. On the other hand, when the loss function is very irregular, this can help in jumping out of local minima, thus MGD and SGD have a better chance of finding the global minimum than standard GD does.

To summarize, the introduction of randomness helps in solving the local minima issue, but at the same, it avoids obtaining an optimal solution due to constant oscillations. One solution is to gradually reduce the learning rate: the steps are larger at the beginning of the optimization process (which helps make quick progress and escape local minima), then they get smaller and smaller, allowing the algorithm to settle at the global minimum. The function that determines the learning rate at each iteration is called the *learning schedule*. If the learning rate is reduced too quickly, the process may get stuck at a local minimum. If the learning rate is reduced too slowly, there may be oscillations around the global minimum for a long time, and the process may end up with a suboptimal solution if it is stopped too early.

I.3.3 ***Backpropagation***

Backpropagation is a way of computing gradients of expressions through the recursive application of chain rule. Thus, the problem consists of computing the gradient of a certain function $f(x)$, where x is a vector of inputs. In the specific case of DL models, the function f will correspond to the loss function, and the inputs x will consist of the training data and the weights. Please notice that training data is given and fixed, whereas the weights are variables that must be optimized. Hence, during the optimization process, we usually compute the gradient for the parameters (e.g. W, b) so that we can use it to perform a parameter update.

Interpretation of the gradient

To give an easy interpretation of the gradient, let us consider the elementary function:

$$f(x,y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x \tag{8}$$

It should be remembered that the derivatives indicate the rate of change of a function compared to a certain independent variable if this variable has an infinitesimal variation. The concept is expressed by (7). For example, if $x = 4$, $y = -3$ then $f(x, y) = -12$. According to (8), the derivative on x is equal to -3. Thus, if the variable x increased by a very tiny amount, the effect on f would be a decrease by three times that amount. This can be seen by rewriting the equation in (7) as:

$$f(x+h) = f(x) + h\frac{d f(x)}{dx} \tag{9}$$

As mentioned, the gradient of a function f is the vector of partial derivatives. So, for the specific example that we are considering

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] = [y, x] \tag{10}$$

For sake of completeness, the partial derivatives of other common basic inputs are reported below.

$$f(x,y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1 \tag{11}$$

$$f(x,y) = \max(x,y) \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1\,(x \geq y) \quad \frac{\partial f}{\partial y} = 1\,(y \geq x) \tag{12}$$

Compound expressions with the chain rule

Based on (8), (11), and (12), the gradient of more complicated expressions that involve multiple composed functions can be computed. Let us consider the example:

$$f(x,y,z) = (x+y)z \tag{13}$$

The function in (13) is simple to differentiate directly, but a particular approach will be taken in the following to provide the idea behind backpropagation. More in details, the expression in (13) can be decomposed into two expressions:

$$q = x + y \quad f = qz \tag{14}$$

The derivatives of both expressions in (14) can be computed by applying (8) and (11):

$$\frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q, \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \tag{15}$$

However, the gradient on the intermediate value q is not required. Instead, we are interested in the gradient of f compared to x, y, and z. The *chain rule* states that the correct way to combine the gradient expressions in (15) is through multiplication, i.e.:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} \tag{16}$$

In practice, this is a simple multiplication of the two numbers that hold the two gradients. To clarify that, let us consider a simple example:

- Set the inputs: $x = -2$; $y = 5$; $z = -4$;
- Perform the forward pass:
  $q = x + y = 3$
  $f = q * z = -12$
- Backpropagation in reverse order:
  - backpropagation through $f = q*z$:
    gradient on z = *dfdz* = $q = 3$
    gradient on q = *dfdq* = $z = -4$
  - backpropagation through $q = x + y$:
    dfdx = *dfdq* * 1 (the multiplication by 1 is the chain rule)
    dfdy = *dfdq* * 1 (the multiplication by 1 is the chain rule)

In the end, we obtained the gradients of f compared to x, y, and z, which are respectively *dfdx*, *dfdy*, *dfdz*. The process above can be also represented with a graph, as shown in Figure I.7. The graph can be swiped both in the forward direction and in the backward direction. In the forward pass, the values from inputs to outputs are computed, while in the backward pass the backpropagation is performed by applying the chain rule as explained above.

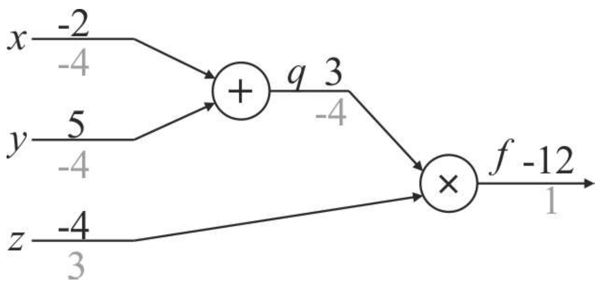


Figure I.7 *Graph of the computation for the function in (13) and of the backpropagation process. In the forward direction, the output value for the function is evaluated (values in black). In the backward direction, the backpropagation is performed, which starts at the end, and recursively applies the chain rule to compute the gradients (values in grey).*

I.4 A fundamental element: the neuron

One of the most used ML/DL models is the Artificial Neural Network (ANN), or simply Neural Network (NN). The idea behind ANNs is to mimic the behavior of the human brain.

I.4.1 ***The neuron***

The basic computational element of the brain is the *neuron*. Approximately 86 billion neurons can be found in the human nervous system and they are connected by approximately 10E+14 – 10E+15 synapses. In Figure I.8, it is shown that each neuron receives input signals from its *dendrites* and produces output signals along its (single) *axon*. The axon eventually branches out and connects via *synapses* to the dendrites of other neurons. In Figure I.9 the computational model of a neuron is depicted. The signals that travel along the axon (e.g. x_0) interact multiplicatively (e.g. w_0x_0) with the dendrites of other neurons based on the synaptic strength at that synapse. (e.g. w_0). The dendrites carry the signals to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along the axon. In the computational model, the frequency of the firing communicates information. Based on this rate-code interpretation, the firing rate of the neuron is modeled with an activation function, *f*, which represents the frequency of the spikes along the axon. Historically, a common choice of function is the sigmoid function σ, since it takes a real-valued input (the strength after the sum) and squashes it to the range between 0 and 1. In summary, as reported in Figure I.9, each neuron performs a dot product with the input and its weights, adds the bias, and applies the activation function. Thus, the key arithmetic operation in neurons is the Multiply-Accumulate (MAC) operation.

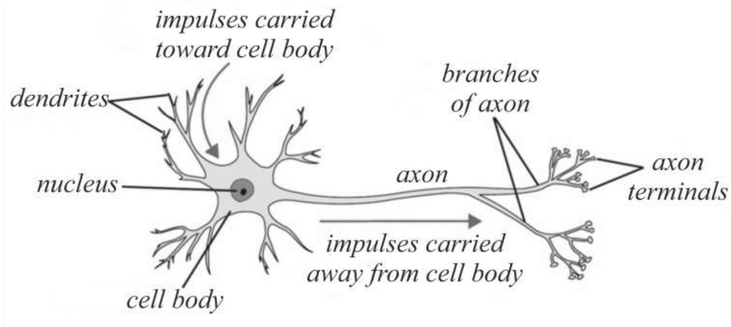


Figure I.8 *Basic structure of a human neuron and its components.*

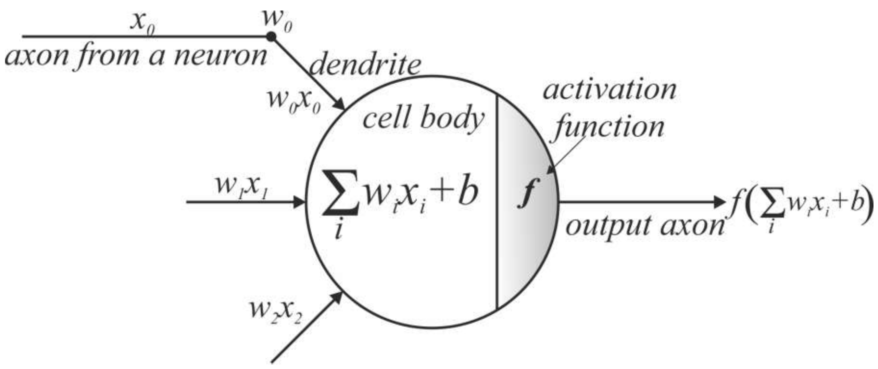


Figure I.9 *Computational model of the neuron. The input signals of the neuron are denoted by x_i, and each input is weighted by the synaptic strength w_i. All the weighted inputs are summed up in the cell body, and an activation function, f, is applied.*

I.4.2 ***Neuron as linear classifier***

The computational model of the neuron in Figure I.9 might look familiar, as it reminds the equation of the linear classifier in (2). Thus, similarly to a linear classifier, a neuron can "like" or "dislike" certain linear regions of its output space. Hence, with an appropriate loss function on the neuron's output, the single neuron can become a linear classifier. For example, considering the sigmoid function as an activation function, the output of the neuron can be interpreted as a binary classifier. In particular, it can be seen as the probability:

$$P(y_i = 1 \mid x_i; w) \tag{17}$$

Since the probabilities of each class must sum to one, the probability of the other class will be:

$$P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w) \tag{18}$$

With this interpretation, the cross-entropy loss in (5) can be used during the optimization process. This is called *binary Softmax classifier* (also known as *logistic regression*). Since the sigmoid function is restricted to be in (0, 1), the predictions of this classifier are based on whether the output of the neuron is greater than 0.5.

I.4.3 ***Commonly used activation functions***

An activation function (or *non-linearity*) takes a single value and executes a specific mathematical operation on it. In practice, several activation functions are used in DL models. Here below, the most common ones are shown.

Sigmoid

The mathematical expression for the sigmoid is reported in (19) while the function is plotted in Figure I.10:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{19}$$

The sigmoid function allows shrinking the range of representation for a certain number between 0 and 1. More precisely, large negative values become almost 0, while large positive values become almost 1. Although the sigmoid function is a good representation of the firing rate of a neuron (0 corresponding to a neuron that is not firing, and 1 corresponding to a neuron that is firing at the maximum frequency), it is almost ever used due to two major drawbacks:

- The sigmoid function tends to saturate, thus *killing* the gradient. In fact, when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. During backpropagation, the chain rule is applied, and this gradient must be multiplied by the gradient of the neuron's output. As a consequence, when the local gradient is close to zero, no signal will be backpropagated through that neuron.

- The sigmoid outputs are not zero-centered. Again, this can be an issue during the backpropagation process. Indeed, if the input to a neuron is always positive (the sigmoid output is the input for other neurons), then the gradient on the weights w in (2) will become either all positive or negative. This may introduce some undesirable zig-zagging effects in the gradient updates for the weights.

Tanh

The mathematical expression for the tanh is reported in (20), where it is expressed in terms of the sigmoid function, while the function is plotted in Figure I.11.

$$\tanh(x) = 2\sigma(2x) - 1 \tag{20}$$

The tanh non-linearity is very similar to the sigmoid, but it shrinks the range of representation of a number between −1 and +1. It shares with the sigmoid the same issue about output saturation, but it has the advantage of being zero-centered. Therefore, in practice, the tanh non-linearity is always preferred to the sigmoid non-linearity.

ReLU

The *Rectified Linear Unit* (ReLU) is currently one of the most used output activations. The function is plotted in Figure I.12, and it computes the function:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{21}$$

The function is basically a threshold at zero. Thanks to its linear form, the ReLU function has been found to greatly accelerate the convergence of SGD (or MGD) compared to sigmoid or tanh (Krizhevsky, 2012). Also, the ReLU can be implemented very easily, whereas both sigmoid and tanh require complex operations to be computed. However, there is an issue with the ReLU activation function: the gradient of ReLU is 0 when its input is lower than 0. Thus, certain units can become "dead" because the backpropagated error is canceled whenever there is a negative input into the neuron.

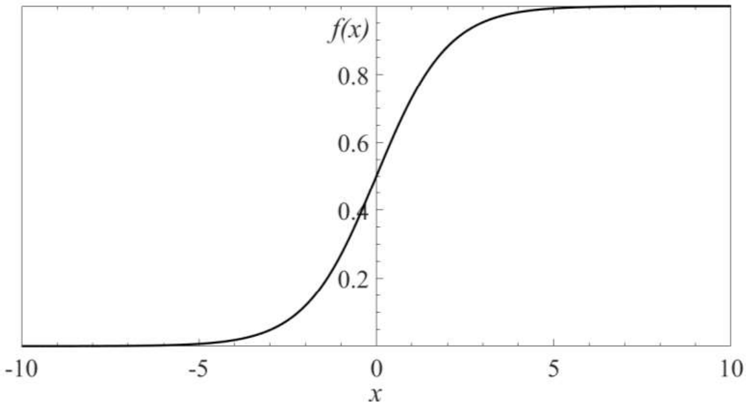


Figure I.10 *Sigmoid function.*

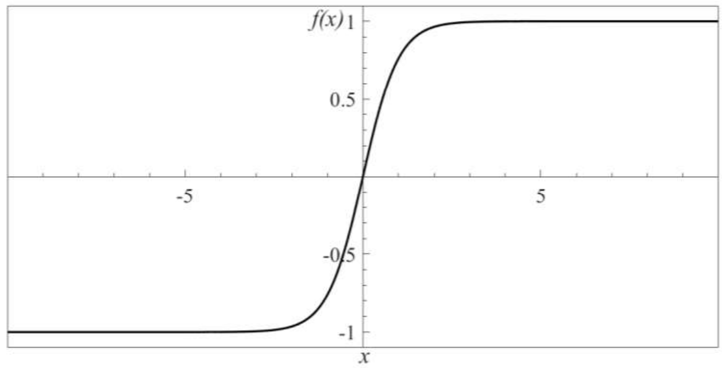


Figure I.11 *Tanh function.*

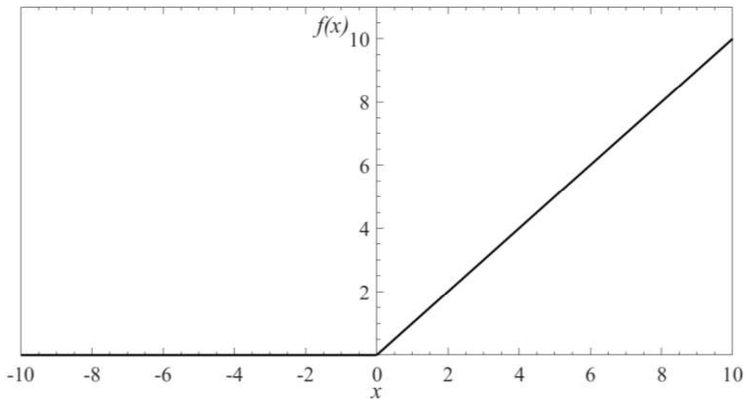


Figure I.12 *ReLU function.*

I.5 Artificial Neural Networks

An ANN is built by connecting many neurons to each other. In particular, the output of some neurons is inputted to other neurons. In standard ANN, cycles are not allowed because that would result in an infinite loop during computation, especially during the forward pass.

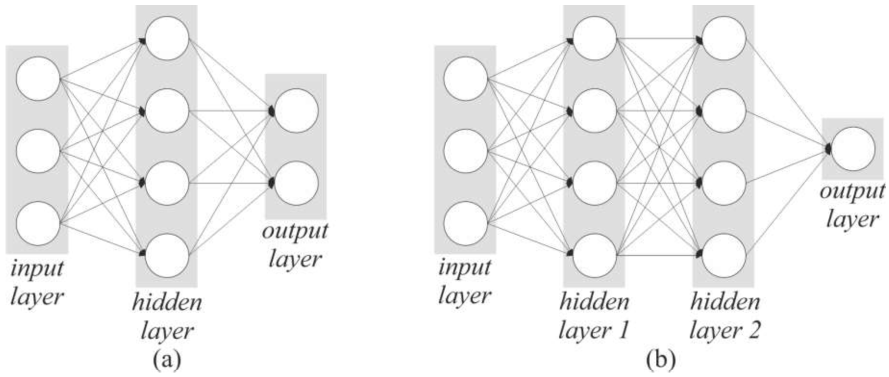


Figure I.13 *Example of ANNs that use a stack of FC layers. (a) 2-layer NN with 3 inputs, and with one hidden layer of 4 neurons (or units) and one output layer with 2 neurons. (b) 3-layer NN with 3 inputs, and with two hidden layers of 4 neurons (or units) each and one output layer.*

I.5.1 ***Layer organization in ANNs***

An important thing to notice is that neurons are not chaotically connected to each other. On the contrary, ANNs are organized structures where neurons are collected in *layers*. Different types of layers exist, however, let us take the example of *Fully Connected* (FC) layers to clarify the concept. FC layer is by far one of the most used layers. As its name suggests, all neurons in a FC layer are connected to every output from the previous layer, while neurons within the layer share no connections. Two examples of ANNs that use a stack of FC layers are shown in Figure I.13. We must distinguish between 3 types of layers:

- *Input layer*: it is the first layer of a NN, and corresponds to the input.
- *Output layer*: it is the last layer of a NN. Unlike all other layers, here it is common that neurons do not have an activation function. This is because the output layer is usually taken to represent the class scores in classification or a real-valued target in regression.
- *Hidden layers*: they are all the layers between the input layer and the output layer.

Notice that an N-layer NN is a NN with N layers, but in N the input layer is not counted. Thus, in Figure I.13a a 2-layer NN is represented, while in Figure I.13b a 3-layer NN is represented. A particular case is the single-layer NN, in which no hidden layers exist, and the input is directly mapped to the output. ANN like the ones in Figure I.13 (all layers are FC layers) are sometimes called *Multi-Layer Perceptrons* (MLPs).

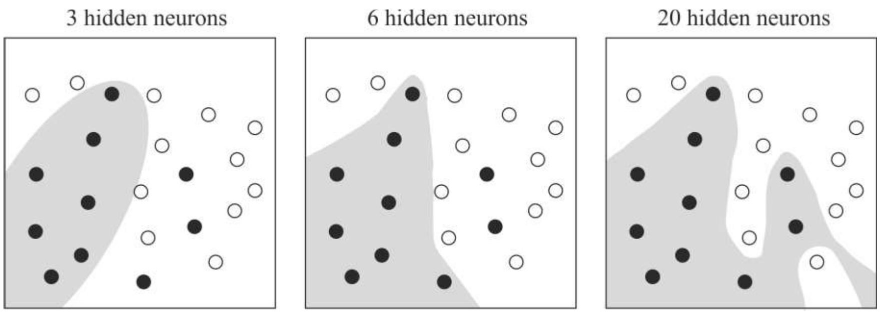


Figure I.14 *Example of a binary classification problem. The black balls represent the first class, while the white balls represent the second class. The gray region is the decision region for the first class, otherwise, the second class is chosen. Considering a NN with one hidden layer, a better decision region can be obtained by increasing the number of neurons.*

I.5.2 ***Sizing ANNs***

Mainly two metrics can be used to measure the size of a NN: the number of neurons, or more commonly the number of parameters. The latter allows getting an insight into the amount of memory required to store all parameters. Considering Figure I.13, the two metrics can be computed as follow:

- In Figure I.13a, the NN has 4+2 = 6 neurons (inputs must not be counted); the number of parameters is (3×4)+(4×2) = 20 weights and 4+2 = 6 biases, for a total of 26 learnable parameters.
- In Figure I.13b, the NN has 4+4+1 = 9 neurons; the number of parameters is (3×4)+ (4×4)+(4×1) = 32 and 4+4+1 = 9 biases, for a total of 41 learnable parameters.

Modern NNs have more than 100 million parameters and are usually made up of 10-20 layers, from which the name DL. In fact, a famous theorem formulated by Cybenco in 1989 (Cybenco, 1989) states that a NN with at least one hidden layer is a *universal approximator*, that is the NN can approximate any continuous function. The problem of this theorem is that nothing is said about the number of neurons required in the hidden layer to get a good approximation. In practice, it turns out that deeper networks (with more hidden layers) can perform much better than NN with a single hidden layer. At this point, it is crucial to understand how to know the number of hidden layers to use and, also, how large each layer should be. To this aim, it is useful to introduce the concept of *capacity* of a NN. The capacity is related to the space of representable functions. In general, the capacity of NN increases as the size and the number of layers increases. The idea is that more neurons can better cooperate to express much more different functions. This is illustrated in Figure I.14. However, attention must be paid because it is easier to overfit the training data when the capacity of the NN is high. *Overfitting* occurs when a NN with high capacity learns noise in the data instead of the abstract relationship. Fortunately, there are many ways to prevent overfitting, such as L2 regularization, dropout (Srivastava, 2014), etc. In practice, it is always better to use these methods to control overfitting instead of reducing the size of the NN.

I.5.3 ***Data pre-processing***

In most cases, ANNs are not fed with raw data, and some kind of data pre-processing is performed. Here below, 3 common forms of data-preprocessing are briefly described.

Mean subtraction

Mean subtraction consists of subtracting the mean computed across every feature in the data and has the geometric interpretation of centering the data around the origin along each dimension.

Normalization

Normalization is a pre-processing technique used to make all samples from input data approximately the same order of magnitude. This can be achieved in two ways. A solution is to divide each dimension by its standard deviation, whereas the other one is to normalize all values in the range $[-1, 1]$, where -1 represents the minimum value, and +1 represents the maximum value.

PCA and Whitening

In this kind of pre-processing, first, a mean subtraction is performed as described above. Then, the covariance matrix is computed which gives information about the correlation between dimensions of the data. The two operations are described in (22) and (23) respectively, where X_i is the i-th input tensor with N elements, and C is the covariance matrix.

$$\overline{X} = \sum_{i=1}^{D} X_i \tag{22}$$

$$C = \left(\overline{X}^T \cdot X \right) \Big/ N \tag{23}$$

At this point, the Single Value Decomposition (SVD) factorization of the covariance matrix can be performed, which gives the eigenvectors and the singular values of C. It should be noted that the eigenvectors constitute a set of orthonormal vectors, thus they can be considered as basis vectors. Thus, the zero-centered data obtained with (22) is projected on the eigenvectors to decorrelate the data. This pre-processing is named *Principal Component Analysis* (PCA) *reduction*. It allows reducing the number of dimensions D of input data. While zero-centering and normalization are commonly used also in DL models, PCA is mostly used in single layer ANN or standard Pattern Recognition (PR) methods (Abdi, 2010).

I.6 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are historically related to image recognition because they introduce many advantages when elaborating images. Despite this, they are successfully used for other purposes as well, especially times series classification (Zhao, 2017). CNNs, are very similar to ANNs: they are made up of neurons, and they have learnable weights and biases. The neuron retains the same structure as explained in paragraph I.4, and the whole network still expresses a differentiable score function. What actually changes is the way neurons are connected between consecutive layers.

As explained in paragraph I.5, ANNs receive an input in the form of a single vector and elaborate it through a series of hidden layers. Each neuron in each hidden layer of the network is fully connected to all neurons of the previous layer. To understand the limitations of this type of organization, let us take the CIFAR-10 dataset (Krizhevsky, 2009) as an example. The CIFAR-10 dataset consists of 60000 32×32 color images in 10 classes, with 6000 images per class. Thus, the size of each input image is 32×32×3, and each neuron in the first hidden layer of a given ANN would have $32 \cdot 32 \cdot 3 = 3072$ weights. If we imagine that each weight is represented using a 32-bit Floating-Point (FP) (IEEE, 2019) coding, 12 KB would be required just to store the weights of each neuron in the first hidden layer. Moreover, many hidden layers are usually needed, and hundreds or also thousands of neurons are present in practice in each hidden layer. In this context, the number of total parameters becomes incredibly high, and with it the memory requirement. In addition to that, a so high number of parameters quite often leads to overfitting. Thus, the full connectivity is wasteful and must be replaced by a different kind of organization.

I.6.1 ***Architecture of a CNN***

CNNs allow overcoming the issues introduced by full connectivity. As said before, CNNs are historically related to image processing. In fact, the core idea in CNNs is to replace the set of weights in a certain layer with a set of filters. Thus, the output of the layer is computed by filtering its input. However, differently from standard signal-processing techniques, the filter parameters are not hand-crafted, whereas they are the result of the learning process. Another difference with regular ANNs is that CNNs have neurons arranged in 3 dimensions: *width*, *height*, *depth*. This is represented in Figure I.15. For example, the input images in CIFAR-10 are an input volume of dimensions 32×32×3 (width, height, depth, respectively). Every layer in CNNs transforms a 3D input volume to a 3D output volume of neuron activations. A simple CNN is a sequence of layers.

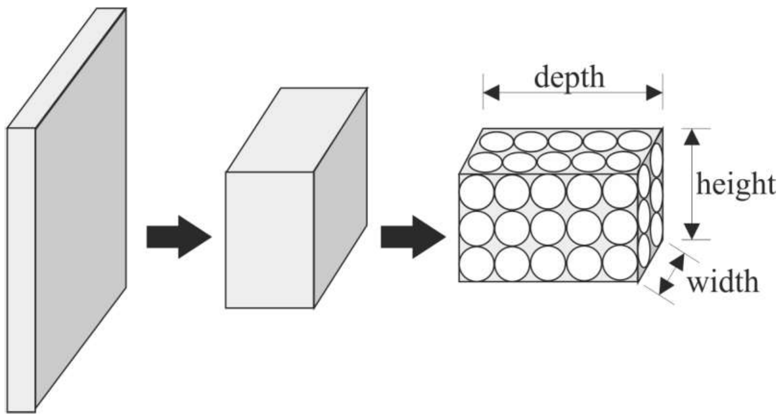


Figure I.15 *Neurons in layers are arranged in three dimensions: depth, height, and width. Neurons are graphically represented by white circles, while each box represents the set of input activations for a layer. These correspond either to the output activations of the previous layer or to the input image for the first layer.*

A CNN is a sequence of layers. The three main types of layers used to build these networks are *Convolutional layers* (CONV), *Pooling layers*, and *Fully-Connected layers* (FC). The latter type of layer is exactly the same as seen in regular ANN.

Convolutional layer

The CONV layer is the main component of a CNN. The parameters of a CONV layer consist of a set of learnable filters. Every filter has a small size along width and height but extends to the full depth of the input volume. If we keep considering the image processing as an example, a typical filter on the first layer of a CNN might have size 5×5×3, i.e. 5 pixels width and height, and 3 because images have depth 3 (the color channels). During the forward pass, each filter slides across the width and height of the input activations and compute dot products between the weights of the filter and the input. At the end of this process, a 2-dimensional *activation map* (or *feature map*) is obtained, which is the result of applying that filter to the full input volume. In a CONV layer, many filters are applied to the input, and each of them will produce a different feature map. All feature maps are concatenated along the depth dimension, thus constituting the output of the layer.

This way of computing a CONV layer may be expressed in terms of neurons as well. To do that, we must imagine that each neuron of a layer is connected to only a local region of the input volume. The size of this region is a hyperparameter called the *receptive field* of the neuron, which corresponds to the filter size. It should be noticed that the size of the receptive field along the depth dimension is always equal to the depth of the input volume. However, the receptive field alone is not enough to define the size of the output volume. To this aim, three hyperparameters must be defined: *depth*, *stride*, and *zero-padding*:

- The *depth* of the output volume corresponds to the number of filters that are associated with the given layer.
- The *stride* defines the step with which the filter slides across the input. When the stride is 1, then the filters shift one position at a time. When the stride is 2 then the filters shift of 2 positions at a time, thus skipping one activation. This will produce an output activation map smaller than the input one.
- In some cases, it may be useful to pad the input activation map with zeros around the border. The size of *zero-padding* is a hyperparameter, which can be used to control the size of the output activation map. In most cases, zero-padding is used to preserve the size of input along the width and height dimensions.

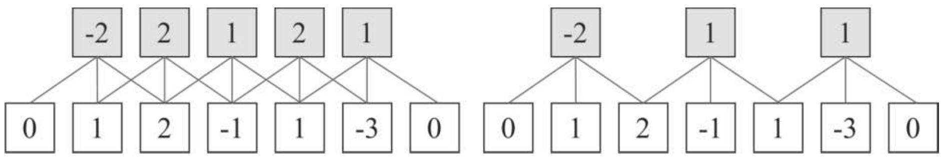


Figure I.16 *In the examples above, the white boxes represent the input activations, while the grey ones are the outputs. Thus, the input size $W = 5$, the receptive field $F = 3$, and the zero-padding $P = 1$. Two different cases are considered: on the left, the input stride $S = 1$, thus the output size is equal to* $(5 + 3 + 2) / 1 + 1 = 5$*; on the right, the input stride $S = 2$, thus the output size is equal to* $(5 + 3 + 2) / 2 + 1 = 3$.

Based on what said above, the spatial size of the output feature map (i.e. width×height) can be computed as follow:

$$O = (W - F + 2P)/S + 1 \tag{24}$$

where O is the size of the output feature map, W is the size of the input feature map, F is the receptive field of the CONV layer, S is the stride with which filters are applied, and P is the amount of zero-padding. The equation in (24) holds both for 2D square filters and 1D filters. An example is shown in Figure I.16. Looking at the example on the left, it can be noticed that the input size and the output size are equal: also 5. In general, when the stride is 1, setting zero-padding to:

$$P = (F - 1)/2 \tag{25}$$

ensures that the input activation map and the output activation map will have the same size.

Local connectivity is not the only feature of neurons in CONV layers. Indeed, *parameter sharing* is also used to control the number of weights. This is based on the concept that if some weights are useful to compute the output activation from some position (x_1, y_1), then it should also be useful to compute the output activation at a different position (x_2, y_2). Thus, each neuron in a single 2-dimensional slice of depth, namely a *depth slice*, is constrained to use the same weights. For example, if the input volume has a size 32×32×3, all neurons in each depth slice of size 32×32 will use the same weights. Therefore, only 3 unique sets of weights (one for each depth slice) are required. Notice that if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can be computed as a *convolution* of the weights with the input volume. Hence the name Convolutional Layer. For this reason, the set of weights are generally referred to as *filters* or *kernels*.

Pooling layer

Another commonly used layer in CNN is the Pooling layer, which is used to progressively reduce the spatial size of the feature maps. This layer is used to reduce the amount of memory and computational power required by the model. Consequently, it also helps in preventing overfitting. The Pooling layer operates independently on every depth slice in the input and resizes it spatially. The most common forms of pooling are max-pooling (MaxPool) and average-pooling (AveragePool), which perform the max and average operations respectively with a particular filter size. The most common form is MaxPool with filter size 2 (or 2×2) with a stride 2, which corresponds to down-sampling the input volume by 2. An example is given in Figure I.17.

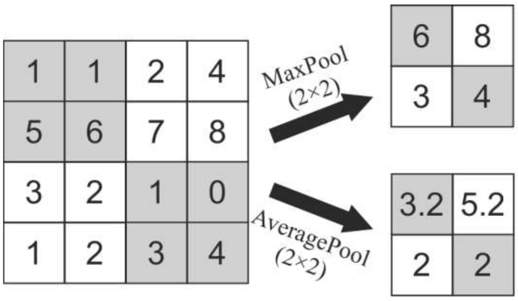


Figure I.17 *Example of MaxPool and AveragePool. In both cases the size of pooling is 2×2 and the stride is 2. The size of the input volume (4×4) is scaled down by a factor of 2, resulting in an output volume of size 2×2.*

Normalization layer

Many types of normalization layers exist for use in CNNs. One of the most useful ones is the *Batch Normalization* (BatchNorm) layer (Ioffe, 2015). This normalization layer aims to standardize the inputs to a layer for each mini-batch. In fact, the distribution of the inputs to layers may change after each mini-batch, thus making it hard to properly accomplish the training process. In particular, BatchNorm performs a rescaling of data to have a mean of zero and a standard deviation of one, i.e. a standard Gaussian. This has the effect of stabilizing and speeding up the training process.

Fully-Connected layer

In a FC layer, all neurons of a certain layer are connected to the neurons of the previous layer. This is the layer described for regular ANNs in paragraph I.5. Differently from CONV layers, neurons in FC layers are not arranged along 3 dimensions. As a consequence, when a FC layer receives input from a CONV layer, the 3-dimensional feature map needs to be *flattened* into a vector of input activations.

I.7 Binarized Neural Networks

The main issue of NNs, especially Deep Neural Networks (DNNs), is that they require a huge amount of memory to store parameters and computational power. For example, AlexNet (Krizhevsky, 2012) and ResNet (He, 2016) require 200 MB of memory, VGG-Net (Simonyan, 2014) requires 500 MB of memory. It is clear that it is hard to deploy those models on portable and wearable devices, which are constrained both in terms of resources and power consumption. Currently, quantization is the most appealing solution to this problem. The core idea in Quantized Neural

Networks (QNNs) is to reduce the number of bits used to represent values in the model. In fact, 32-bit Floating-Point (FP) values are generally used to compute neural network models. Quantization techniques aim to execute the model by using a lower number of bits and representing values with a Fixed-Point (FI) coding. The lower number of bits allows reducing the amount of memory required to store parameters (i.e., a 4× memory saving is obtained by switching from 32-bit FP to 8-bit FI), whereas using FI rather than FP is beneficial in terms of hardware complexity. Indeed, FP arithmetic circuits are generally larger and consume more power than the FI counterpart. The advantage in terms of area and energy consumption is clear if considering the values in Table I.1. The energy consumption decreases with the number of bits of the arithmetic operators, and FP circuits are more consuming than the FI counterpart. But more importantly, memory accesses typically consume more than arithmetic operations, and the cost of memory accesses increases with memory size.

The price to pay for the reduction of the memory requirements and the computational power is generally a decrease in the accuracy of the model. Thus, QNNs need to be careful designed in order to preserve as much as possible the original accuracy.

Table I.1 *Energy consumption and Area occupation for different arithmetic operations and memory accesses. Energy values are from Horowitz (2014). Area values come from synthesis with TSMC 45 nm standard cells.*

Operation	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8 KB)	5	N.A.
32b DRAM Read (Off-chip)	640	N.A.

The extreme case of quantization is *binarization*. Binarization is a 1-bit quantization, that is values can only have two possible values. Generally, −1 and +1 are used. NNs that exploit binarization are referred to as *Binarized Neural Networks* (BNNs). One critical aspect is that backpropagation cannot be directly applied to BNNs, since it is not possible to update weights in small increments. The simplest workaround is to train the NN model with 32-bit FP values and then quantize the resulting weights and biases. This approach is usually named *post-training quantization*. Unfortunately, the highest degradation of accuracy is obtained with this method (Finkelstein, 2019).

I.7.1 ***Binarization of weights***

Courbariaux *et al.* (2015), for the first time, provided a way to train NNs using binary weights, which is named BinaryConnect. It should be noticed that many MAC operations are replaced by simple Additions/Subtractions (ADD/SUB) operations when using binary weights. This is a huge advantage, as FI adders are much less expensive both in terms of area and energy than FI MAC circuits (David, 2007). Using binary values during training provides a more representative loss to train against post-training quantization. The binarization operation transforms the real-valued weights into two possible values. A very straightforward binarization operation is based on the sign function:

$$w_b = \begin{cases} +1 & \text{if } w \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{26}$$

where w_b is the binarized and w the real-valued weight. Although this is a deterministic operation, averaging this discretization over the many input weights of a hidden unit could compensate for the loss of information. An alternative that allows having a finer and more correct averaging process is to binarize stochastically:

$$w_b = \begin{cases} +1 & \text{with probability } p = \sigma(w) \\ -1 & \text{with probability } 1-p \end{cases} \tag{27}$$

where σ is the *hard sigmoid* function:

$$\sigma(x) = \text{clip}\left(\frac{x+1}{2}, 0, 1\right) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right) \tag{28}$$

The reason why hard sigmoid is used is that it is far less computationally expensive than the original version in (19).

The key point in BinaryConnect (Courbariaux, 2015) is that weights are binarized during the forward and backward propagations but not during parameter updates. In fact, keeping good precision during updates is necessary for SGD to work, because a large number of almost infinitesimal changes in the direction that most minimize the loss function must be performed. The core idea is that what matters most at the end of training is the sign of the weights, but before that, a lot of small changes to a continuous-valued quantity are performed, and only at the end its sign is considered. In more detail, at training time, BinaryConnect randomly picks a binary value (+1 or −1) for each weight, for each mini-batch, for both the forward and the backward steps. However, the update is accumulated in a real-valued variable storing the parameter. Since the binarization is not influenced by variations of the real-valued weights when its magnitude is above the range [−1, 1], the real-valued weights are clipped within the [−1, 1] interval right after the weight updates. The real-valued weights would otherwise grow very large without any impact on the binary weights.

One thing to notice is that the derivative of the sign function in (26) is zero almost everywhere, which seems to be incompatible with the backpropagation algorithm. This is true even if stochastic quantization in (27) is used. A solution to this problem was proposed in the paper by Bengio *et al.* (2013), where the *straight-through estimator* (STE) is used to estimate the gradient through stochastic neurons. The idea is simply to backpropagate through the sign function as if it had been the identity function.

I.7.2 ***Binarization of activations***

Thanks to the binarization of weights, a large amount of memory can be saved compared to a FP-32 NN model. Also, binarizations allow replacing MAC operations with simpler ADD/SUB operations. However, a further advantage can be gained by binarizing activations as well. In doing so, an additional amount of memory is saved (i.e. buffers to store activations can be smaller) and bit-wise operations can further reduce the computational complexity, and hence the power consumption, of the model.

Binarization of the activations in BNNs was introduced for the first time by Courbariaux *et al.* (2016). In order to binarize the activations, they are passed through a sign function using a STE in the backward step, similar to what happens in the binarization of weights. The sign function is used as the activation function in the NN model. A version of the STE that considers the clipping effect is applied to the deterministic sign function in (26). In particular, it was observed that to obtain good results, the gradient must be canceled out if the input to the activation function is too large. Thus, considering the activation function, i.e. the sign function:

$$q = \mathrm{sign}\left(r\right) \tag{29}$$

the STE of the gradient of the cost function C compared to r is computed as:

$$\frac{\partial C}{\partial r} = g_r = g_q 1_{|r|\leq 1} \tag{30}$$

where:

$$1_{|r|\le 1} = \begin{cases} 1 & when\,|r| \le 1 \\ 0 & otherwise \end{cases} \tag{31}$$

I.7.3 ***Bitwise operations***

When using binary values, the dot product between weights and activations can be reduced to bitwise operations. The binary values can be either +1 or −1, which are encoded with 1 and 0, respectively. Using an XNOR logical operation at bit level is equivalent to perform multiplication on the binary values. This is shown in Table I.2.

Table I.2 *Equivalence between XNOR logical operation between Bit1 and Bit2 and multiply operation between Value1 and Value2.*

Bit1 (Value1)	Bit2 (Value2)	XNOR (Multiply)
0 (−1)	0 (−1)	1 (+1)
0 (−1)	1 (+1)	0 (−1)
1 (+1)	0 (−1)	0 (−1)
1 (+1)	1 (+1)	1 (+1)

Thus, the XNOR function can be used to perform multiply operations between binary values. However, dot product operation also requires the accumulation of the results from the element-wise multiplication between elements of the input vectors. It turns out that the accumulation operation can be performed through the *popcount* operation, which is the process of counting the number of 1s in a binary value. More in detail, the accumulation can be performed by counting the number of 1s in a group of XNOR products, multiplying this value by 2, and subtracting the total number of bits producing an integer value. The reason why this holds is given below. Let us consider the accumulation of N binary values. The relation between the bit representation x_b (i.e., 0 and 1) and the integer values x (i.e., −1 and +1) is the following:

$$x = 2x_b - 1 \tag{32}$$

When the accumulation of N binary values is performed, the result is:

$$\sum_{i=1}^{N} x_i = 2\sum_{i=1}^{N} \left(x_b\right)_i - N \tag{33}$$

I.7.4 ***Energy consumption in Binarized Neural Networks***

BNNs can drastically reduce memory size and accesses, and replace most arithmetic operations with bitwise operations. In comparison with 32-bit FP NNs, BNNs require 32 times smaller memory size and 32 times fewer memory accesses, with relevant advantages in energy consumption according to Table I.1. Moreover, considering that the key arithmetic operation in NNs is the MAC operation, 32-bit FP MAC operations are replaced by 1-bit XNOR-popcount operations. This can lead to faster execution times and fewer hardware resources required in digital design architectures, especially for specialized ones (Simons, 2019). Despite this, it should be pointed out that training a BNN takes longer than traditional NNs due to the STE heuristic needed to approximate the gradient of the real-valued weights.

I.7.5 ***Accuracy of Binarized Neural Networks***

While BNNs are compact and efficient compared to their full precision counterparts, they suffer from degradation of accuracy. A summary of the state of the art in terms of accuracy is reported in Table I.3. Results refer to classification performance on the ImageNet dataset (Deng, 2009). Looking at the values in the table, it turns out that a minimum accuracy loss of about 3 percentage points must be accepted when using binary weights. The accuracy loss is even higher when binarizing both weights and activation, where the minimum accuracy loss is about 20 percentage points.

Table I.3 *Comparison of accuracy on the ImageNet dataset (Deng, 2009) between 32-bit FP model and BNNs. For each topology, the bit-width is expressed as W/A, that is weights/activations. Where binarization occurs, the accuracy loss compared to the FP model is reported.*

Topology	Bit-width (W/A)	Top-1 accuracy	Accuracy Loss	Source
AlexNet	32/32	57.1%	-	(Zhang, 2018)
AlexNet	1/32	35.4%	−21.7%	(Courbariaux, 2015)
AlexNet	1/1	27.9%	−29.2%	(Hubara, 2016)]
ResNet-18	32/32	69.6%	-	(Zhang, 2018)
ResNet-18	1/32	60.8%	−8.8%	(Rastegari, 2016)
ResNet-18	1/1	42.7%	−26.9%	(Lin, 2017)
ResNet-34	32/32	73.3%	-	(Zhang, 2018)
ResNet-34	1/32	70.4%	−2.9%	(Qin, 2020)
ResNet-34	1/1	52.4%	−20.9%	(Lin, 2017)
ResNet-50	32/32	76.0%	-	(Zhang, 2018)
ResNet-50	1/32	72.8%	−3.2%	(Yang, 2019)

I.7.6 ***Hardware Implementation of Binarized Neural Networks***

FPGA implementation

FPGAs are very used to accelerate BNNs when performing inference. Indeed, most CPUs and GPUs are optimized for executing integer and FP operations, but they may not be the best solution to execute bitwise operations efficiently. FPGAs allow for custom data paths, so that custom operations, such as XNOR and popcount operations, can be highly optimized. It should be noticed that FPGAs also contain DSPs, which can be fundamental when accelerating FP NNs models, but they are not used as extensively for BNNs, since bitwise operations are required. Compared with CPUs and GPUs, FPGA can accelerate a BNN with lower power consumption, even though the power stays in the range of tens of W. A list of FPGA implementations is reported in Table I.4, where all accuracy results refer to training on the CIFAR-10 dataset (Krizhevsky, 2009).

Table I.4 *Comparison of FPGA implementations of BNN accelerators. All accuracy results refer to training on the CIFAR-10 dataset (Krizhevsky, 2009).*

FPGA	LUTs	BRAMs	Clock Freq. [MHz]	Power [W]	Acc. (%)	Source
Zynq7 045	20264	-	-	-	66.63	(Zhou, 2017)
Virtex7 690T	20352	372	450	15.44	78	(Nakahara, 2016)
KintexUltra 115	35818	144	125	-	79.1	(Fraser, 2017)
ZynqUltra 3EG	41733	283	300	10.7	80.10	(Blott, 2018)
Zynq7 020	25700	242	100	2.25	80.10	(Blott, 2018)
Zynq7 045	46253	186	-.	11.7	80.1	(Umuroglu, 2017)
Zynq7 020	14509	32	143	2.3	81.8	(Nakahara, 2017)
KintexUltra 115	93755	386	125	-	85.2	(Fraser, 2017)
Zynq7 020	23426	135	143	2.4	85.9	(Yang, 2018)
Virtex7 980T	556920	-	340	-	86.06	(Zhou, 2017)
Zynq7 020	53200	280	200	-	86.98	(Ghasemzadeh, 2018)
Zynq7 020	46900	140	143	4.7	87.73	(Zhao, 2017)
KintexUltra 115	392947	1814	125	-	88.3	(Fraser, 2017)
Zynq7 020	29600	103	-	3.3	88.61	(Guo, 2018)

ASIC Implementation

As with FPGA, Application-Specific Integrated Circuits (ASICs) provide the possibility to implement a custom data path with custom arithmetic operations. ASICs provide by far the best performance both in terms of power consumption and frequency. On the other hand, a great effort is required to design integrated circuits, and the cost can be amortized only if the volume of production is huge. In Table I.5 a list of ASIC implementations of BNN accelerators is reported. It can be seen that the power consumption is orders of magnitude lower than the one with FPGAs.

Table I.5 *Comparison of ASIC implementations of BNN accelerators.*

Technology	Area [mm^2]	Freq. [MHz]	Energy efficiency [TOPS/W]	Power [mW]	Latency [ms]	Memory [KB]	Source
28 nm	1.29	50	90	2.85	25	52	(Yin, 2018)
40 nm	12.7	30	0.698	1900	40	2144	(Li, 2019b)
65 nm	17.6	100	658	N.A.	N.A.	295	(Valavi, 2018)
28 nm	5.76	N.A.	532	0.899	N.A.	328	(Bankman, 2018)

Chapter II
Human Activity Recognition

II.1 Definition and Applications

Human Activity Recognition (HAR) is the ability to recognize human activities using sensors and it is a promising technology for many application fields. From an analytical point of view, let us imagine that a certain person is performing an activity belonging to a predefined set of activities A:

$$A = \left\{ A_i \right\}_{i=1}^{N} \tag{34}$$

where N denotes the number of activities. The activity information is captured by a sequence of sensor readings:

$$s = \left\{ d_1, d_2, ..., d_n \right\} \tag{35}$$

where d_t denotes the sensor reading at time t. HAR aims to predict the activity sequence A^p based on sensor reading s. To do that, a model f must be built:

$$A^p = \left\{ A_j^p \right\}_{j=1}^{n} = f(s), \quad A_j^p \in A \tag{36}$$

where n denotes the length of the sequence. The true activity sequence (ground truth) is denoted as A^t:

$$A^t = \left\{ A_j^t \right\}_{j=1}^{n}, \quad A_j^t \in A \tag{37}$$

In HAR, model f must be learned by minimizing the discrepancy between the predicted activity A^p and the ground truth activity A^t.

Based on the set of activities to be recognized, different applications can be found for HAR. In healthcare, it can be used for Parkinson's disease monitoring or rehabilitation purposes (Eskofier, 2016), (Bisio, 2016) (Abobakr, 2018). Also, HAR is widely used in assisted living, especially for elderly care (De, 2018). Other applications are video surveillance (Xian, 2017), gesture recognition (Normani, 2018), gait analysis (Cola, 2017), fitness (Chinimilli, 2017).

II.2 Sensors in Human Activity Recognition Systems

HAR can be classified based on the type of sensor used to acquire data. In Table II.1 a list of the most used sensors in HAR systems is provided. In particular, data can be acquired using either image sensors, such as cameras, or inertial sensors, such as accelerometers, gyroscopes, or other Inertial Measurement Units (IMUs). Physical health sensors are sometimes used with motion sensors to detect the activities of patients for rehabilitation purposes or capturing their vital signals for health condition evaluation (Chen, 2014). A key problem in HAR is where to place the sensors since different body parts provide different sensitivity to different activities, and hence different performance in terms of accuracy (Yu, 2016). As an example, placing an accelerometer at the ankle is expected to detect the motion information caused by legs or feet on the arm, thus allowing identifying activities like walking or running. On the other hand, an accelerometer placed at the chest might not measure the movements of arms.

Table II.1 *Most used sensors in HAR systems.*

Category	Sensor
Inertial Sensors	Accelerometer Gyroscope Magnetometer
Physical Health Sensors	Electrocardiogram (ECG) Hearth Rate (HR) Electroencephalograph (EEG) Electromyogram (EMG)
Environmental Sensors	Temperature Humidity Light sensor Barometer
Others	Camera Microphone GPS

Another classification among HAR systems is based on the number of sensor positions and the number of types of sensors used. More precisely, four configurations can be identified:

- One to One: this is the basic modality in HAR systems, where one single sensor is placed at one single body part.

- One to Multi: in this configuration, one single type of sensor is placed at multiple body parts, thus obtaining complementary signals from different positions.
- Multi to One: in this configuration, a sensor device with more than one type of sensors built-in on one body part, aiming to capture different kinds of information.
- Multi to Multi: in this last scenario, multiple devices, each embedded with one or more types of sensors, are placed at multiple body parts, thus combining the advantages of all the above configurations.

It should be noticed that better accuracies are to be expected by using more sensors. However, the higher the number of sensors, the higher is the complexity of the HAR system, and hence the power consumption. The most potential body positions that have been explored to deploy one or more sensors are hands, arms, wrists, chest, pockets, head, feet, shank, thighs, trunk, vest, waist, ankles, belt, pelvic, hip, legs, abdomen, back, knees, ears, neck.

To summarize, two major categories of HAR techniques can be identified: the first is usually named video-based HAR, while the second one is named Inertial-Sensor-based HAR (IS-HAR), or just sensor-based HAR. The continuous improvements of sensor technology, mainly in terms of reduced area and power consumption, as well as the increment of the processing power of portable devices in the era of pervasive computing, have been favored the development and the diffusion of IS-HAR systems. This thesis focus on the design of an ultra-low power smart sensor, thus IS-HAR is a perfect candidate as a case study.

II.3 Classification Techniques

IS-HAR systems can be further classified based on the model used to achieve the classification. Classification can be achieved either through conventional PR methods (Bulling, 2014), such as Decision Trees, SVM, naïve Bayes, and hidden Markov, or through DL models (Yu, 2016), such as CNNs, autoencoder (AE), etc.

II.3.1 ***Pattern Recognition methods***

In a typical PR-based classification method, two main signal processing stages can be identified: first, a set of features are extracted from raw samples from the sensors, then a ML model is used to achieve classification based on the extracted features.

Feature Extraction

During *feature extraction*, a set of features are extracted from sensor data. More in detail, features are extracted as feature vectors X_i from an input data window w_i. The total number of features extracted forms the *feature space*. The core idea is that activities that belong to the same class will be clustered in the same region of the feature space. In HAR systems, features need to be robust against *user variability*, that is features must not depend on the specific user who is performing the activity, as well as *intraclass variability*, that is the same activity must be clustered in the same region even though it can be performed in different ways. In the following, the most used features in HAR are listed below:

- *signal-based features*: these are mostly statistic features, such as mean, variance, kurtosis, and the like. These features are popular due to their simplicity as well as their high performance across a variety of HAR problems (Ravi, 2007). Also, frequency-domain features are sometimes used (Kang, 1995).
- *body model features*: these are calculated from a 3D skeleton using multiple on-body sensors (Zinnen, 2009). Polynomial features that describe signal trends such as mean, slope, and curvature are used for trajectories of limbs (Blanke, 2010)
- *event-based features*: these are extracted when a certain event occurs. An example is features extracted from repetitive eye movement sequences (Bulling, 2011).
- *multilevel features*: in this case, data is first clustered, for example using *k*-means. Then statistics like duration, frequency, and occurrence of data are encoded to provide expressive features.

Classification

Many PR methods can be used to achieve classification in HAR systems. Among the most used ones, there are Hidden Markov Models (HMMs), Decision Trees, k-Nearest Neighbor (kNN), and Naïve Bayes. A list of references where PR methods are used for HAR purposes is provided in Table II.2.

II.3.2 ***Deep Learning methods***

Despite PR methods allow achieving satisfactory accuracies in many cases, some drawbacks must be considered. Firstly, the features need to be extracted in a heuristic and hand-crafted way, which heavily relies on human expertise and domain knowledge. Even though human knowledge may help in certain task-specific settings, in general, this will result in a lower chance and a longer time to build a successful HAR system. Also, PR methods allow achieving good accuracies only in classifying a limited number of activities, especially when more complex activities need to be recognized.

DL models tend to overcome those limitations, as they do not require feature extraction. Indeed, features can be automatically learned by the DL model itself, thanks to its deep multi-layered internal representation. Among the most used DL models in HAR, there are DNNs, CNNs, AEs, and Recurrent Neural Networks (RNNs). A list of references where PR methods are used for HAR purposes is provided in Table II.3.

Table II.2 *Examples of Pattern Recognition methods for Human Activity Recognition. The activities which are classified are specified for each case, as well as the accuracy and the sensors used to sample data.*

Method	Activities	Accuracy	Sensors	Ref
HMM	Leaving Toileting Showering Sleeping Breakfast Dinner Drink	94.5%	Wireless sensor network	(VanKasteren, 2008)
Decision Tree	Walking Sitting Standing still Watching TV Running Stretching Scrubbing Folding laundry Brushing teeth Riding elevator	84.3%	Accelerometers	(Bao, 2004)
HMM	Sitting Standing Walking Jogging Walking upstairs Walking downstairs Riding a bicycle Driving a car Elevator down Brushing teeth	91%	Accelerometer Audio IR/visible light High-frequency light Pressure Humidity Temperature Compass	(Lester, 2005)
kNN	3 Tai Chi movements	85%	Accelerometers Gyroscopes	(Kunze, 2006)

SVM	Walking Upstairs Downstairs Standing Sitting Laying	89.3%	Accelerometer Gyroscope	(Anguita, 2012)
Naïve Bayes	Take medication Prepare breakfast Prepare lunch Prepare dinner Breakfast Lunch Dinner Eat a snack Watch TV Enter the lab Play a videogame Relax on the sofa Leave the lab Visit in lab Put waste in the bin Wash hands Brush teeth Use the toilet Wash dishes Wash clothes Work at the table Dressing Go to the bed Wake up	68%	Binary sensors fixed to everyday objects Proximity tags Location-aware smart floor sensing Accelerometer	(Jiménez, 2018)

Table II.3 *Examples of Deep Learning methods for Human Activity Recognition. The activities which are classified are specified for each case, as well as the accuracy and the sensors used to sample data.*

Method	Activities	Accuracy	Sensors	Ref
CNN	Walking Walking upstairs Walking downstairs Sitting Standing Laying	95.18%	Accelerometer Gyroscope	(Jiang, 2015)

RNN	Write notes Open engine hood Close engine hood Check door gaps Open door Close door Open/close two doors Check trunk gap Open/close trunk Check steering wheels	95.80%	Accelerometers	(Ordóñez, 2016)
DNN	11 low-level activities	83.30%	Accelerometer	(Zhang, 2015)
AE	Walking Walking upstairs Walking downstairs Sitting Standing Laying	98.22%	Accelerometer Gyroscope	(Gao, 2019)

II.4 Time-Latency Requirements in Human Activity Recognition

In addition to the accuracy level, latency is another factor that has often to be taken into account in HAR systems. Latency can be defined as the time that has elapsed from the beginning of an activity to its detection by the system (Dinarević, 2019). However, the particular HAR tasks must be distinguished to understand whether latency has to be actually minimized. In particular, low latency is an important requirement in all those applications where immediate feedback may be required, such as fall detection and epilepsy seizure detection. In those cases, high accuracy is not sufficient and an optimal trade-off should be investigated to guarantee an actual effective solution. Nevertheless, few studies report their achieved latency in the literature (Rault, 2017). For some other applications of HAR, such as the distance walked in a day and general daily activities, a higher latency can be accepted without compromising the validity of the solution.

The main model parameter that can be tuned to control the latency of the system is the window size. In fact, regardless of the classification technique, the data stream must be segmented in data windows for processing. The most widely used segmentation method in HAR is the sliding window approach (Banos, 2014). The signals are split into windows of a fixed size, with the possibility to have some overlap. Banos *et al.* (2014) proved that the best trade-off between recognition speed and accuracy is obtained within the window size interval 1−2 s.

II.5 Public Datasets for Human Activity Recognition

The importance of HAR among the current research topics is proven by the existence of several public datasets. Those datasets are fundamental to test a new model and fairly compare it with state-of-the-art. Below, some of the most used public datasets for HAR are briefly described.

II.5.1 ***PAMAP2 dataset***

The PAMAP2 Physical Activity Monitoring dataset (Reiss, 2012) contains data of 18 different physical activities, performed by 9 subjects wearing 3 IMUs and a heart rate monitor. Each IMU was placed in a different position. In particular, *hand*, *chest*, and *ankle* have been chosen. Each IMU was made up of a 3D accelerometer, 3D gyroscope, 3D magnetometer, and temperature. Two different scales are available for the 3D accelerometer, which are ±16g and ±6g. It should be noticed that due to high impacts caused by certain movements (especially running), the accelerometer with ±6g range gets sometimes saturated. The sampling frequency was set to 100 Hz. The heart rate sensor monitored the bpm of each subject, with a sampling frequency of approximately 9 Hz.

Each of the subjects had to follow a protocol, containing 12 different activities, which are: *lying, sitting, standing, walking, running, cycling, Nordic walking, ascending stairs, descending stairs, vacuum cleaning, ironing, rope jumping*. Furthermore, a list of optional activities to perform was also suggested to the subjects. From the list, in total 6 different activities were performed by some of the subjects in addition to the protocol. Namely, they are *watching TV, computer work, car driving, folding laundry, house cleaning, playing soccer*.

The dataset is not balanced, that is a different number of samples is associated with each activity. This is detailed in Table II.4, where the time in seconds for each activity is specified. Also, details about the PAMAP2 dataset are summarized in Table II.6.

II.5.2 ***SHL dataset***

The University of Sussex-Huawei Locomotion (SHL) dataset (Ciliberto, 2017) was collected at the University of Sussex. It was recorded over 7 months in 2017 by 3 users engaging in 8 different modes of transportation in a real-life setting in the United Kingdom. More in detail, the SHL dataset, where labels are: *Car, Bus, Train, Subway, Walk, Run, Bike,* and *Still*. The dataset contains multi-modal data from a body-worn camera and 4 smartphones, carried simultaneously at typical body locations, that are *Bag, Hand, Hips,* and *Torso*. All data is sampled at 100 Hz by using the following sensors: 3D accelerometer, 3D gyroscope, 3D magnetometer, pressure sensor, and temperature sensors. As for the PAMAP2 dataset, the SHL dataset is not balanced. Details about the amount of data per class are reported in Table II.5, whereas generic details about the SHL dataset are summarized in Table II.6.

Table II.4 *Available time window in seconds for each activity in the PAMAP2 dataset. An ID is associated with each activity. Also, a check sign is used to identify the 6 optional activities.*

ID	Activity	Optional	Available time window [s]
1	Lying		1925.16
2	Sitting		1851.80
3	Standing		1899.23
4	Walking		2387.53
5	Running		981.92
6	Cycling		1645.93
7	Nordic walking		1881.00
9	Watching TV	✓	836.45
10	Computer work	✓	3099.31
11	Car driving	✓	545.18
12	Ascending stairs		1172.00
13	Descending stairs		1049.27
16	Vacuum cleaning		1753.45
17	Ironing		2386.82
18	Folding laundry	✓	998.74
19	House cleaning	✓	1871.83
20	Playing soccer	✓	469.12
24	Rope jumping		493.54
		Total:	27248.27

Table II.5 *Available time window in hours for each activity in the SHL dataset. An ID is associated with each activity.*

ID	Activity	Available time window [h]
1	Still	127
2	Walking	127
3	Run	21
4	Bike	79
5	Car	88
6	Bus	107
7	Train	115
9	Subway	89
	Total:	753

Table II.6 *Summary of SHL and PAMAP2 public datasets. For each dataset, the following features are specified: number of classes, sensors used to sample data, sampling frequency for each sensor, possible carry positions.*

Dataset	#classes	Sensors	Sampling Frequency	Positions
PAMAP2	12+6	3D accelerometer (±6g scale)	100 Hz	Ankle Hand Chest
		3D accelerometer (±16g scale)	100 Hz	
		3D gyroscope	100 Hz	
		3D magnetometer	100 Hz	
		Heart-rate monitor	≃9 Hz	
SHL	8	3D accelerometer	100 Hz	Bag Hand Hips Torso
		3D gyroscope	100 HZ	
		3D magnetometer	100 Hz	
		Pressure sensor	100 Hz	
		Temperature sensor	100 Hz	

II.6 HW Solutions for Human Activity Recognition

Even though a lot of research is being carried out on HAR, in most of the literature, the development of a HAR system is associated with the development of algorithms or models, and most of the efforts are made to achieve high accuracy. Unfortunately, such high accuracies are obtained using MCUs, CPUs, or GPUs, and those systems are not embeddable on ultra-low power wearable devices or smart sensors due to their high power consumption.

Nicosia *et al.* (2018) addressed this problem by exploiting a light harvester to improve the lifetime of a sensor-rich wearable node. The module is based on a 32-bit MCU, which features a Floating-Point Unit (FPU) single-precision (32-bit), 128 KB of RAM, and 1 MB of Flash ROM. In addition, the module features many Micro Electrical Mechanical Systems (MEMS) sensors, such as accelerometers, gyroscope, and magnetometer, and environmental sensors, such as pressure, humidity, and temperature sensors. The MCU is able to run a 5-layer CNN, which achieves an average accuracy of 96.13% when classifying 5 different activities, i.e. *stationary*, *walking*, *running*, *biking*, and *driving*. The samples are acquired using a 3D accelerometer with a sampling frequency of 26 Hz. Measurements show that the current absorbed by the board in dark conditions is 1.75 mA.

However, to obtain more energy-efficient systems, a dedicated HW circuitry should be designed, which is the objective of the research activity in this thesis. In the following, the only works found in the literature that propose a dedicated HW accelerator for HAR applications are briefly described.

Hanai *et al.* (2009) presented a versatile recognition processor, which is able to perform detection and recognition of image, video, and acceleration signals. Concerning HAR, the processor can recognize human activities such as walking, reading, and typing from short and low-quality 3D acceleration signals, with a time window from 2s to 10s. Data is sampled at 50 Hz with a resolution of 8-bit. The classification is achieved using Haar-like features and cascaded filters. The processor is fabricated in 90 nm CMOS technology, occupies 0.89 mm^2, and runs at 54 MHz with a 0.9 V supply. For activity recognition from 8-bit 50 Hz acceleration signals, the power consumption per frame rate is 0.15 µW/fps with an accuracy of 93%.

In the paper by Kodali *et al.* (2017), seven different IoT applications were implemented using a FC DNN accelerator designed in 28 nm CMOS technology. Among the various applications, HAR is performed. The accuracy has been estimated for 5 different public HAR datasets. The best accuracy has been obtained for the Smartphone-based Human Activity Recognition Feature-Extracted dataset and is equal to 93.6%. The HW accelerator can operate with a supply voltage as low as 0.56 V. With this voltage level, and with a frequency of 443 MHz, the power consumption is equal to 1.12 mW. Based on this operating point, the authors have derived an estimate of the power consumption for each application. The results are summarized in Table II.7. It must be noticed that the authors did not take into account the power dissipation due to leakages, which can be significant when scaling down the frequency in a 28 nm CMOS technology.

In the work of Jafari *et al.* (2019), a scalable and low-power embedded deep CNN named SensorNet is presented, which allows classifying multimodal time-series signals. A custom low-power hardware architecture has been designed to allow the deployment of the CNN model in an embedded real-time system. SensorNet performance is evaluated using 3 different case studies, including HAR. When performing HAR, the model achieves 98.0% accuracy. The architecture is implemented with CMOS 65 nm technology, and results show a total power consumption of 18.5 mW when the throughput is 67 label/s. This throughput is obtained with a clock frequency of 100 MHz.

Table II.7 *Application power requirements for the HW accelerator proposed in the work of Kodali et al. (2017)*

Dataset	Frequency [kHz]	Power [nW]	Energy per inference [nJ]
OPP	258.6	722.0	129.9
PAMAP2	83.3	210.6	130.0
DG	29.6	74.8	17.9
Smartphone (Raw)	59.2	149.7	96.0
Smartphone (FE)	39.5	99.9	128.1

Chapter III
Orientation Estimation in Inertial Sensors

III.1 Device-Orientation problem

Inertial sensors measure motion-related physical quantities, such as acceleration, angular rate, etc. For this kind of measurement, it is very important to define which is the reference frame. As an example, a 3-axis accelerometer provides the components of the measured acceleration on the three axes defined by its own orientation in the space. Looking at Figure III.1, the components of the measured accelerations are provided on the axis x, y, and z. Therefore, it is easy to realize that the physical meaning associated with the acceleration measured on each axis is strongly dependent on the orientation of the sensors. This is not a trivial issue, especially when the sensor is embedded in portable or wearable devices, such as smartphones or smartwatches. Indeed, it is not possible to know in advance which will be the orientation of the device, thus making it very difficult to extract useful features from the raw measurement. This problem is known as the *device-orientation* problem, and it is especially present in IS-HAR systems, where the sensor cannot keep a fixed orientation in the space due to human movements, and it can often be placed in different positions as well. In this context, 3 different reference frames are defined in the paper by Jahn *et al.* (2017): the *Device Coordinate System* (DCS), which is the reference frame defined by the orientation of the sensor, the *World Coordinate System* (WCS), that is the reference frame relative to the world's gravity force and the magnetic north; the *User Coordinate System* (UCS), that is the reference frame defined by the user's heading. The latter can be useful to track the movements of a person or an object. More generally, many solutions have been proposed in the literature to mitigate the effects of the unknown orientation in inertial sensors by transforming the measurements from the DCS to the WCS. A graphical representation of DCS and WCS is provided in Figure III.1. In all cases, some signal processing operations must be performed on the raw measurement data, such as filtering, 3D rotations, and the like. Unfortunately, the computational complexity of such operations does not fit with the capabilities of wearable devices, where the HW resources are limited and the energy consumption is constrained to be as low as possible. During the first part of my research activity, I focused on the definition of an HW-friendly algorithm to solve the device-orientation problem in 3-axis accelerometers.

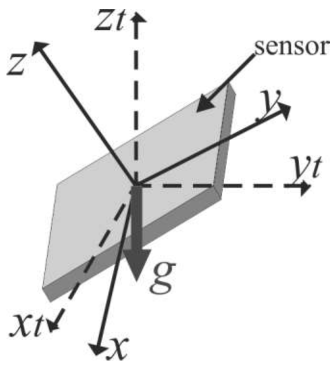


Figure III.1 *Graphical representation of the 2 possible reference frames for an inertial sensor. The Device Coordinate System is the reference frame defined by the device (solid line in the figure). The World Coordinate System is the reference frame defined by the world's gravity force (dotted line in the figure). In this figure, the World Coordinate System is defined as the reference frame whose z-axis is opposite to the gravity vector, g.*

III.2 State-of-the-art solutions to the device-orientation problem

In this paragraph, many solutions to the device-orientation problem will be presented. Three different cases can be detected based on the sensors which are used in the system: accelerometer + magnetometer, accelerometer + gyroscope, only accelerometer.

III.2.1 ***Accelerometer + Magnetometer***

As explained above, the raw measurements need to be transformed from the DCS to the WCS to cancel the dependence from the sensor orientation. A solution can be to exploit the measurement of the north direction provided by a magnetometer. In the paper by Ustev *et al.* (2013), the orientation angle is computed by using both the accelerometer and the magnetometer embedded in a smartphone. When the device is not subject to an external acceleration, it measures the gravity acceleration pointing towards the center of the Earth. This allows computing the tilt angles. At the same time, the magnetometer provides the magnetic vector in the three axes of the DCS. By fusing data from the accelerometer and magnetometer, it is possible to detect the orientation of the device. The authors collected data from 20 participants, who were asked to perform locomotive activities (i.e., running, standing, biking, sitting, walking). All activities were performed for 3 minutes and a total of 15 minutes of movement data was collected from every participant. Different kinds of tests were performed using a k-NN classifier to measure accuracy. During the orientation tests, the device was carried in different orientations to investigate the orientation dependency. Three different orientation tests were performed based on the input data:

1. in the first test, the input data was the raw measurement from the accelerometer;
2. in the second test, data from the accelerometer was fused with the magnetometer data to detect the orientation of the device. Then, the gravity component on each axis can be isolated to obtain the linear acceleration;
3. in the third test, the linear acceleration was converted from the DCS to the WCS.

The accuracy results are summarized in Table III.1. The highest accuracy is obtained when the conversion from the DCS to the WCS is performed.

Table III.1 *Results from the orientation test for different input data (Ustev, 2013).*

Test	Accuracy
1	83%
2	93%
3	97%

III.2.2 ***Accelerometer + Gyroscope***

Another solution to perform the transformation from the DCS to the WCS can be to use both an accelerometer and a gyroscope. Florentino-Liaño *et al.* (2012) computed the inclination of the sensor in terms of the angles of roll, θ_x, and pitch, θ_y, by using the following formulas:

$$\theta_x = \arctan\left(\frac{\overline{a_y^{DCS}}}{\overline{a_z^{DCS}}}\right) \tag{38}$$

$$\theta_y = \arcsin\left(\frac{-\overline{a_x^{DCS}}}{\sqrt{\left(\overline{a_x^{DCS}}\right)^2 + \left(\overline{a_y^{DCS}}\right)^2 + \left(\overline{a_z^{DCS}}\right)^2}}\right) \tag{39}$$

where $\overline{a_x^{DCS}}$, $\overline{a_y^{DCS}}$, and $\overline{a_z^{DCS}}$ are the means of the measured acceleration in the DCS obtained during periods of little or no linear acceleration. It should be noticed that it is not possible to determine the angle of yaw, θ_z, by using only an accelerometer and gyroscope. However, this is not an issue in HAR, because it is not important if the user is facing towards North or in any other direction. Using (38) and (39), and assuming a yaw angle of zero, a rotation matrix is defined to partially transform the measurements from the DCS to the WC as in (40).

$$\begin{pmatrix} \cos(\theta_y) & \sin(\theta_x)\sin(\theta_y) & \cos(\theta_x)\sin(\theta_y) \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ -\sin(\theta_y) & \sin(\theta_x)\cos(\theta_y) & \cos(\theta_x)\cos(\theta_y) \end{pmatrix} \tag{40}$$

Once the initial orientation has been computed in this manner, the angular velocity can be integrated over time to update the rotation matrix. Florentino-Liaño *et al.* (2012) point out that always transforming the data from the DCS to the UCS does not help to distinguish between low-motion activities, such as standing, sitting, and lying. Indeed, the only acceleration that is measured in those cases is the gravity acceleration, which will always be contained in the z_t direction (z_t is defined in Figure III.1). To solve this problem, the transformation can be only performed once, while the user is standing still.

III.2.3 ***Only Accelerometer***

The problem of the device orientation can also be solved by using only an accelerometer. This can be the best choice when there are power and area constraints. In the work of Mizell (2003), a solution was proposed to estimate the accelerometer orientation using a single tri-axial accelerometer only. For a chosen sampling interval, typically a few seconds, an estimate of the gravity component on each axis is obtained by averaging all the measurements in the sampling interval on the respective axis. This corresponds to estimate the gravity vector $g = (g_x, g_y, g_z)$, where g_x, g_y, and g_z are the averages of all the measurements in the sampling interval on the x, y, and z axis respectively. Let $a = (a_x, a_y, a_z)$ be the vector made up of the three acceleration measurements taken at a given point in the sampling interval. Thus, the linear acceleration can be computed as:

$$v = \left(a_x - g_x,\, a_y - g_y,\, a_z - g_z\right) \tag{41}$$

Then, the linear acceleration in (41) is projected on the gravity vector g using the dot product, thus obtaining p:

$$p = \left(\frac{v \cdot g}{g \cdot g} \right) g \tag{42}$$

In other words, p is the component of the linear acceleration in the direction of the gravity vector, i.e. the "vertical component" of the linear acceleration. Since a 3D vector is the sum of its vertical component and horizontal component, the latter can be computed simply by vector subtraction.

III.3 Proposed Solution

A solution to the device-orientation problem is proposed in this thesis. Considering that the aim is to develop an ultra-low power smart sensor, a signal processing technique has been defined which takes in input data from a single 3D accelerometer. The application that has been considered is HAR. In the proposed solution, the measured acceleration is transformed from the DCS to the WCS in two stages: the *filtering* stage and the *vector rotation* stage. In the first one, the gravity acceleration is isolated from the measured acceleration, and it is used as a reference in the vector rotation stage to define the WCS. In particular, the WCS has been defined as the reference frame in which the z-axis is opposite to the gravity vector.

III.3.1 ***Filtering Stage***

In the filtering stage, the components of the acceleration acquired by the tri-axial accelerometer are filtered to separate the high-frequency component from the low frequency/DC component. The latter roughly corresponds to the gravity acceleration, while the former is related to human motion. The filter is a 4th order high-pass IIR Butterworth filter, with a cutoff frequency of 0.4 Hz. The main characteristics of the filter are summarized in Table III.2, where f_{-3dB} is the −3 dB cutoff frequency, and f_c is the sampling frequency. The sampling frequency has been selected considering the typical ones in HAR (Bulling, 2014).

Table III.2 *Initial filter specifications*

Filter Response	Frequency Behavior	f_{-3dB}	Order	f_c
IIR Butterworth	High-pass	0.4 Hz	4	25 Hz

IIR Filters

A causal IIR filter (Mitra, 2000) is described in (43):

$$\sum_{k=0}^{N} d_k y[n-k] = \sum_{k=0}^{M} p_k x[n-k] \tag{43}$$

where $x[n]$ is the input sequence, $y[n]$ is the output sequence, $h[n]$ is the impulsive response, $\{d_k\}$ and $\{p_k\}$ are real coefficients. The equation in (43) can be rewritten as:

$$y[n] = -\sum_{k=1}^{N} \frac{d_k}{d_0} y[n-k] + \sum_{k=0}^{M} \frac{p_k}{d_0} x[n-k], \quad d_0 \neq 0 \tag{44}$$

From (44), it is clear that the output of the filter can be computed using a recursive method. Applying the Z-transform to (43), the transfer function of an IIR filter is obtained:

$$H(z) = \frac{p_0 + p_1 z^{-1} + \ldots + p_{M-1} z^{-(M-1)} + p_M z^{-M}}{d_0 + d_1 z^{-1} + \ldots + d_{N-1} z^{-(N-1)} + d_N z^{-N}} = \frac{p_0}{d_0} \frac{\prod_{l=1}^{M} \left(1 - \xi_l z^{-l}\right)}{\prod_{l=1}^{N} \left(1 - \lambda_l z^{-l}\right)} \tag{45}$$

Using (45), the filter can be described using ξ_l and λ_l, which are zeros and poles of the transfer function H respectively. For values of z in the region $|z| = 1$, known as the unit circle, the transfer function can be expressed as a function of a single real variable, ω, by defining $z = e^{j\omega}$. This corresponds to the Discrete-Time Fourier Transform (DTFT).

Structure Identification

The structure of the filter has been identified to minimize the required HW resources. In particular, a Coupled-All Pass structure has been adopted, in which the number of the required multipliers and registers is equal to the order of the filter. A comparison with other well-known filter structures is shown in Table III.3, in which N is the order of the filter. The comparison shows that the Couple All-Pass structure requires about N multipliers less, and about N adders more than the other two structures. Since a multiplier requires much more resources than an adder, the Coupled All-Pass form is the most convenient structure in terms of required resources. The scheme of a Coupled All-Pass filter is represented in Figure III.2.

Table III.3 *Required HW resources for different filter structures.*

Filter Structure	#Mult	#*Reg*	#Adders
II Direct Form Transposed	$2N+1$	$2N+1$	$2N$
Cascaded Form	$2N+1$	$2N+1$	$2N$
Coupled All-Pass	N	N	$3N+2$

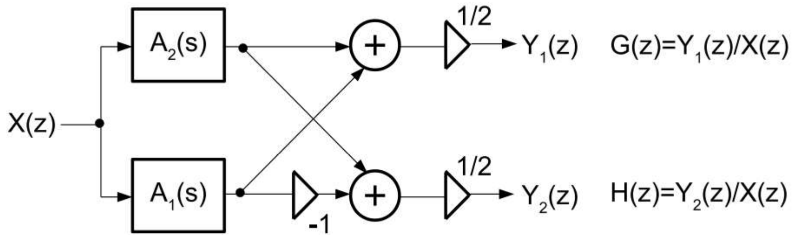


Figure III.2 *Coupled All-Pass realization of G(z) and its power complementary function H(z).*

Coupled All-Pass filters

A Coupled All-Pass filter is based on the concept of *power complementary* property of the transfer functions (Vaidyanathan, 1986). Consider the following transfer function:

$$G(z) = \frac{P(z)}{D(z)} = \frac{p_0 + p_1 z^{-1} + \ldots + p_{N-1} z^{-(N-1)} + p_N z^{-N}}{d_0 + d_1 z^{-1} + \ldots + d_{N-1} z^{-(N-1)} + d_N z^{-N}} \tag{46}$$

In (46), assume that $P(z)$ is symmetric, i.e. $p_k = p_{N-k}$. If $G(z)$ is either a high-pass or low-pass Butterworth, Chebyshev, or elliptic filter, and if the has odd order, it can be expressed as the sum of two all-pass transfer function $A_1(z)$ and $A_2(z)$:

$$G(z) = \frac{1}{2}\left[A_2(z) + A_1(z)\right] \tag{47}$$

Also, we can define the transfer function $H(z)$ as:

$$H(z) = \frac{1}{2}\left[A_2(z) - A_1(z)\right] \tag{48}$$

The two transfer functions $G(z)$ and $H(z)$ are *power complementary*, that is:

$$\left|G\left(e^{j\omega}\right)\right|^2 + \left|H\left(e^{j\omega}\right)\right|^2 = 1 \tag{49}$$

In the same way as $G(z)$, $H(z)$ can be written as:

$$H(z) = \frac{Q(z)}{D(z)} = \frac{q_0 + q_1 z^{-1} + \ldots + q_{N-1} z^{-(N-1)} + q_N z^{-N}}{d_0 + d_1 z^{-1} + \ldots + d_{N-1} z^{-(N-1)} + d_N z^{-N}} \tag{50}$$

In (50), $Q(z)$ is symmetric, i.e. $q_k = q_{N-k}$. The equations (47) and (48) are represented by the scheme in Figure III.2. The all-pass transfer functions can be easily identified using the following method: if λ_0, λ_1, …, λ_N, are the poles of $G(z)$, and if they are ordered so that $\angle(\lambda_k) < \angle(\lambda_{k+1})$, the odd-indexed poles belong to $A_2(z)$, while the even-indexed poles and the zero-indexed poles belong to $A_1(z)$. Also, thanks to the power complementary property, if $G(z)$ is a low-pass transfer function, then $H(z)$ is a high-pass filter function, and vice versa. Moreover, the power complementary property guarantees that the two transfer functions have the same cutoff frequency. Based on the above, the problem of identifying the structure for the desired filter is converted to the problem of identifying the structures of the all-pass filters. Whatever filter structure, in principle, could be used for this purpose. However, to minimize the required hardware resources and obtain the advantages shown in Table III.3, the *two pair extraction approach* (Vaidyanathan, 1986) has been used. This approach is described in the following. Consider an m^{th} all-pass transfer function:

$$A_m(z) = \frac{d_m + d_{m-1}z^{-1} + \ldots + d_1 z^{-(m-1)} + d_0 z^{-m}}{d_0 + d_1 z^{-1} + \ldots + d_{m-1} z^{-(m-1)} + d_m z^{-m}} \tag{51}$$

In an all-pass transfer function, the coefficients in the numerator are the same coefficients of the denominator but with the opposite order. From (51), it is possible to obtain an $(m-1)^{th}$ all-pass transfer function using the following relationship:

$$A_{m-1}(z) = z\frac{A_m(z) - k_m}{1 - k_m A_m(z)} = \frac{d_N' + d_{N-1}' z^{-1} + \ldots + d_1' z^{-(N-1)} + z^{-N}}{1 + d_1' z^{-1} + \ldots + d_{N-1}' z^{-(N-1)} + d_N' z^{-N}} \tag{52}$$

where:

$$\begin{aligned} k_m &= A_m(\infty) = d_m \\ d_i' &= \frac{d_i - d_m d_{m-i}}{1 - d_m^2} \end{aligned} \tag{53}$$

Thus, $A_m(z)$ can be obtained from (52):

$$A_m(z) = \frac{k_m + z^{-1} A_m(z)}{1 + k_m z^{-1} A_m(z)} \tag{54}$$

At this point, a two pair can be considered, i.e. a discrete-time Linear Time-Invariant (LTI) system with two inputs and two outputs. Say X_1, X_2 the inputs and Y_1, Y_2 the outputs, and assume that $X_2 = AY_2$, as represented in XXX. It can be shown that the behavior of such a system can be described by the equations:

$$\begin{aligned} Y_1 &= k_m X_1 + (1 - k_m) z^{-1} X_2 \\ Y_2 &= (1 + k_m) X_1 + k_m z^{-1} X_2 \end{aligned} \tag{55}$$

Defining $V_1 = k_m(X_1 - z^{-1}X_2)$, the equations in (55) can be rewritten as:

$$
\begin{aligned}
Y_1 &= V_1 + z^{-1}X_2 \\
Y_2 &= X_1 + V_1
\end{aligned}
\tag{56}
$$

In the end, the mth order all-pass transfer function, $A_m(z)$, has been represented as a function of the $(m-1)^{th}$ order all-pass transfer function, $A_{m-1}(z)$, as depicted in Figure III.4. In Figure III.5, the structure that implements the equations (55) is shown.

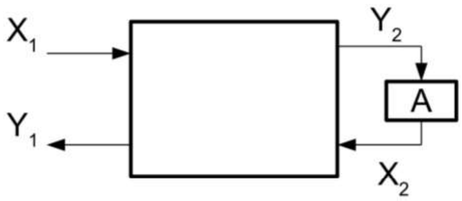


Figure III.3 *Schematic representation of a two-pair with a constraint on the second port.*

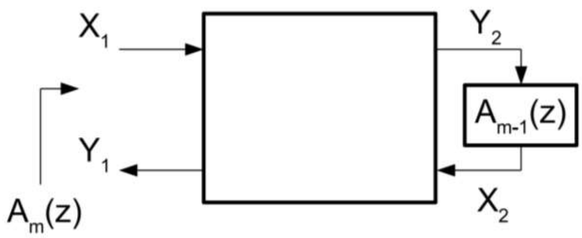


Figure III.4 *Two pair representation of $A_m(z)$.*

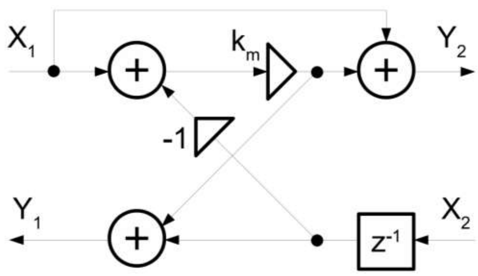


Figure III.5 *Realization of the two-pair using a single multiplier.*

The procedure described above can be repeated recursively until the all-pass function $A_0(z)$ is realized. As a result, $A_m(z)$ is realized using m all-pass cascaded cells as shown in Figure III.6. From Figure III.6, it is evident that using the two-pair extraction approach, the realization of an m^{th} order all-pass filter requires m registers and m multipliers. Since the realization of an N^{th} order coupled all-pass filter requires two all-pass filters, whose orders sum up to N, the realization of the whole filter requires N registers and N multipliers, according to Table III.3.

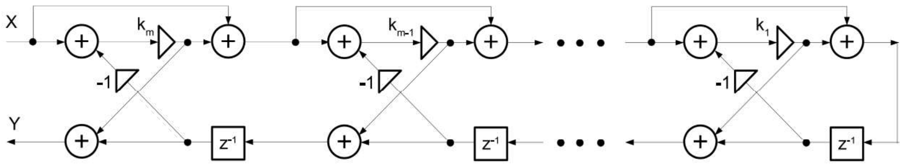


Figure III.6 *Realization of an m^{th} order all-pass filter using the two-pair extraction approach, in which the two-pair is realized using a single multiplier.*

Filter Design

Starting from the specifications in Table III.2, the filter has been implemented using a Coupled All-Pass structure. However, the implementation of a Coupled All-Pass filter requires the order to be odd, while the order specified in Table III.2 is even (and equal to 4). This problem has been addressed by changing the order to 5. The modified specifications are reported in Table III.4.

Table III.4 *Filter specifications for a coupled all-pass implementation.*

Filter Response	Frequency Behavior	f_{-3dB}	Order	f_c
IIR Butterworth	High-pass	0.4 Hz	5	25 Hz

The frequency response has been approximated using *fdatool*, a tool available in MATLAB for the design of filters. Using the specifications in Table III.4, the following transfer function has been obtained:

$$H(z) = \frac{0.922 - 4.609z^{-1} + 9.219z^{-2} + 9.219z^{-3} + 4.609z^{-4} - 0.922z^{-5}}{1 - 4.837z^{-1} + 9.362z^{-2} - 9.063z^{-3} + 4.387z^{-4} - 0.850z^{-5}} \quad (57)$$

In Figure III.7, the amplitude of the transfer function is represented as a function of the normalized frequency (with respect to the sampling frequency).

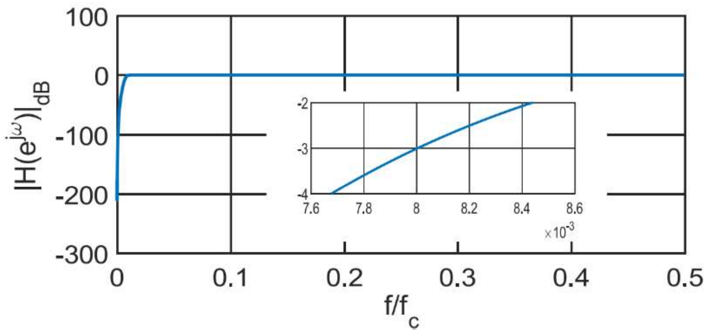


Figure III.7 *Ideal frequency response of the filter; the frequency response around the normalized cutoff frequency is shown in detail.*

Sizing the wordlength

The original SW model performed the filtering operation using a FP double-precision (64-bit) encoding but, in order to minimize the area occupation and the power consumption of the proposed HW solution, the filter has been implemented using a Fixed-Point encoding. However, the resolution of a FP encoding is much lower than the resolution of a FI one. Thus, using a FI encoding, an error is introduced in the representation of the filter coefficients. As a consequence, also poles and zeros of the filter experience variations with respect to their theoretical value, and in the worst-case scenario the filter could become unstable. In Figure III.8, the comparison between the ideal frequency response of the high-pass Coupled All-Pass filter and the quantized ones is shown. The frequency response obtained with a FP 64-bit encoding can be considered as the ideal one. Decreasing the number of bits used to represent the coefficients from 32 to 20, the difference becomes larger. By carrying out the same analysis for the low-pass frequency response, analogous results have been obtained (Figure III.9). The filter is stable for all the considered word-length. Based on this analysis, a 24-bit FI encoding has been chosen to represent the coefficients. In this case, for both the high-pass and the low-pass frequency responses, the cutoff frequency has an error of the 0.19%, which can be considered negligible.

Using (53), the coefficients k_1, …, k_5 for the Coupled All-Pass filter have been obtained, which scheme is represented in Figure III.10.

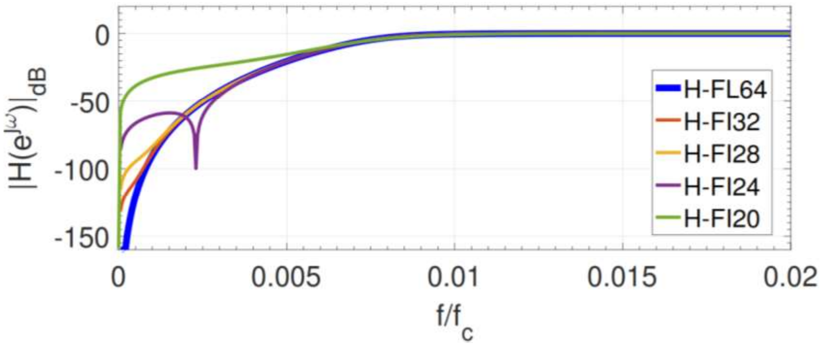


Figure III.8 *Comparison between the high-pass frequency response obtained using filter coefficients represented in FP 64-bit encoding (H-FL64), assumed as the ideal frequency response, and the high-pass frequency responses obtained using filter coefficients represented in FI 32-bit (H-FI32), FI 28-bit (H-FI28), FI 24-bit (H-FI24), FI 20-bit (H-FI20). The filter is realized using a Coupled All-Pass structure.*

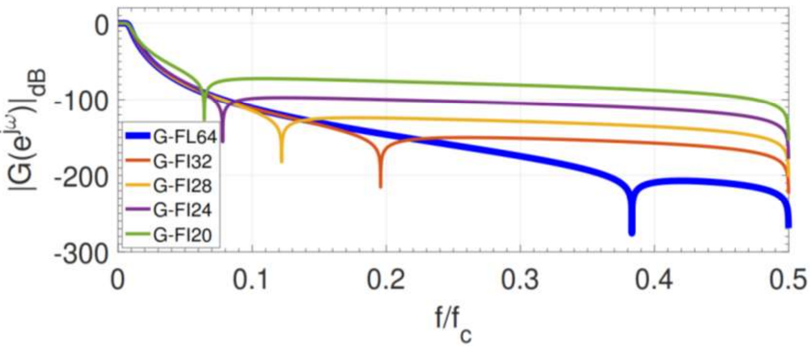
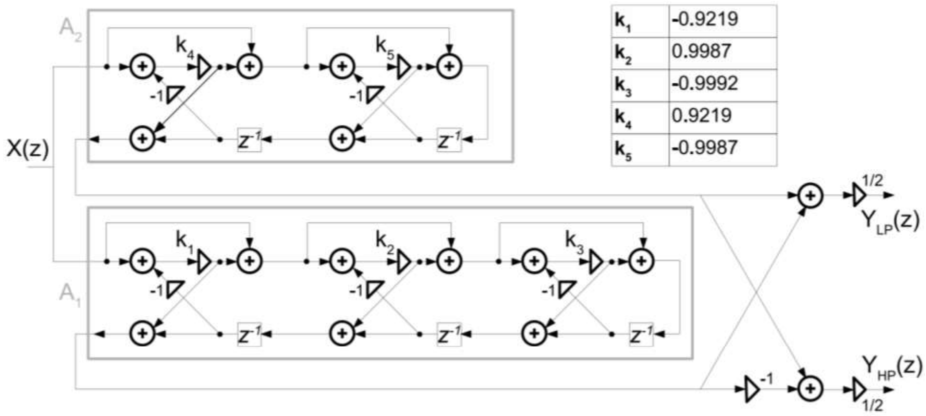


Figure III.9 *Comparison between the high-pass frequency response obtained using filter coefficients represented in FP 64-bit encoding (H-FL64), assumed as the ideal frequency response, and the high-pass frequency responses obtained using filter coefficients represented in FI 32-bit (H-FI32), FI 28-bit (H-FI28), FI 24-bit (H-FI24), FI 20-bit (H-FI20). The filter is realized using a Coupled All-Pass structure.*



k_1	-0.9219
k_2	0.9987
k_3	-0.9992
k_4	0.9219
k_5	-0.9987

Figure III.10 *Realization of the filter using a Coupled All-Pass structure.*

III.3.2 ***Vector Rotation Stage***

The operations performed during the vector rotation stage aim to obtain a vector representation of the measured acceleration in a coordinate system where the *z*-axis has the same direction as the gravity vector. As depicted in Figure III.1, the tri-axial accelerometer defines the DCS as $S = \{x, y, z\}$, while gravity defines the WCS as $T = \{x_t, y_t, z_t\}$. When the sensor lies in a plane parallel to the ground surface, the overlap between S and T occurs. In all the other cases, acceleration data from the sensor are expressed by three coordinates in S. The three associated coordinates in T are computed during the vector rotation stage. In geometric terms, T is obtained through a rotation of S around the rotation axis u by an angle θ, where u and θ are defined in terms of the gravity vector g as follows:

$$\hat{u} = \frac{\vec{g} \times \hat{z}}{\left\|\vec{g} \times \hat{z}\right\|} = \left(\frac{g_y}{g_{xy}}, -\frac{g_x}{g_{xy}}, 0 \right), \quad g_{xy} = \sqrt{g_x^2 + g_y^2} \tag{57}$$

$$\theta = \cos^{-1}\left(\frac{g_z}{\left\|\vec{g}\right\|} \right), \quad \left\|\vec{g}\right\| = \sqrt{g_x^2 + g_y^2 + g_z^2} \tag{58}$$

Rotation algorithms

The mathematical operation performed in the vector rotation stage is a 3D rotation. The first step in the signal processing definition has been to identify the most suitable rotation algorithm. Four approaches are mainly used in the literature for vector orientation recalculation (Kok, 2017), (Janota, 2015): the

Euler angles, which express the rotation of a vector in the space as three consecutive rotations around coordinate axes; the rotation matrix, which is calculated from the Euler angles but it is often preferred to them because of singularities, as happens in the work of Wu *et al.* (2016) for attitude estimation; the Rodrigues' rotation formula (or Euler's rotation vector) (Dai, 2015), which expresses the rotation between two coordinate frames in terms of an angle and a unit vector around which the rotation takes place, as in the work of Ginting *et al.* (2018) for the attitude control of a quadrotor; the unit quaternions, which use an alternative 4-dimensional representation of the orientation to avoid the gimbal lock issue of Euler angles (Kok, 2017), as in the paper by Emokpae *et al.* (2018) for the estimation of finger orientations in a smart glove for rehabilitation therapy. It is worth noting that all the above approaches describe the same quantities and, hence, can be used interchangeably. Regardless of the specific issues, the differences between them are essentially in the computational effort to process raw data from the sensors.

Proposed rotation algorithm

All these techniques require a large number of arithmetical operations and they also involve the computation of trigonometric functions, divisions, and square roots. Thus, a new hardware-friendly algorithm has been developed which aims to define a modular calculation scheme. Modularity allows identifying a minimum set of operations per cycle to carry out the entire GR operation. Therefore, a very reduced set of operations can be identified, and, taking advantage of the very low sample rate (25 Hz) of the LIS2DW12 accelerometer, many cycles can be executed before a new sample is acquired.

The proposed algorithm assumes that $\hat{z}_t$ is defined by $\vec{g}$, namely:

$$\hat{z}_t = -\frac{\vec{g}}{\|\vec{g}\|} = -\left(-\frac{g_x}{\|\vec{g}\|}, -\frac{g_y}{\|\vec{g}\|}, -\frac{g_z}{\|\vec{g}\|}\right) \tag{59}$$

To obtain $\hat{x}_t$, $\hat{x}$ must rotate in the same way as $\hat{z}$. To this end, $\hat{x}_t$ can be expressed as:

$$\begin{gathered}\hat{x}_t = \vec{x}_u + \vec{x}_n = \left(-\frac{|g_x| g_x g_z}{\|\vec{g}\| g_{xy}^2} + \frac{g_y^2}{g_{xy}^2}, -\frac{|g_x| g_y g_z}{\|\vec{g}\| g_{xy}^2} - \frac{g_x g_y}{g_{xy}^2}, \frac{|g_x|}{\|\vec{g}\|}\right) \\ \vec{x}_u = (\hat{x}\cdot\hat{u})\hat{u} \qquad \vec{x}_n = \|\hat{x} - (\hat{x}\cdot\hat{u})\hat{u}\|(\hat{z}_t \times \hat{u})\end{gathered} \tag{60}$$

where $\vec{x}_u$ is the component of $\hat{x}_t$ parallel to $\hat{u}$, and $\vec{x}_n$ is the component of $\hat{x}_t$ perpendicular to $\hat{u}$. Finally:

$$\hat{y}_t = \hat{z}_t \times \hat{x}_t = \left(\frac{a_y g_z - a_z g_y}{\|\vec{g}\|}, \frac{a_z g_x - a_x g_z}{\|\vec{g}\|}, \frac{a_x g_y - a_y g_x}{\|\vec{g}\|} \right) \tag{61}$$

where (59) has been redefined as $\hat{x}_t = \left(a_x, a_y, a_z\right)$ for convenience. The WCS, T, is completely defined by (59)-(61). So, the components of the acceleration vector in T, (v_{xr}, v_{yr}, v_{zr}), can be obtained by projecting the acceleration vector in S, $v = (v_x, v_y, v_z)$, onto $\{x_t, y_t, z_t\}$:

$$v_{xr} = \vec{v} \cdot \hat{x}_t, \quad v_{yr} = \vec{v} \cdot \hat{y}_t, \quad v_{zr} = \vec{v} \cdot \hat{z}_t \tag{62}$$

In Figure III.11 a block diagram of the derived algorithm is represented, where the divisions in (59)-(61) have been grouped as a unique final step. In this way, the scheme reveals a repetition pattern: 2 or 3 parallel multiplications are followed by either a sum or two cascaded sums. This pattern can be used as a basic building block to implement the entire scheme. To this purpose, (59)-(61) must be rewritten in terms of de-normalized vectors:

$$\begin{gathered} \vec{x}_{dn\_t} = g_{xy}^2 \|\vec{g}\| \hat{x}_t = \left(i_x, i_y, i_z\right) = \\ \left(-|g_x| g_x g_z + \|\vec{g}\| g_y^2, \ -|g_x| g_y g_z - \|\vec{g}\| g_x g_y, \ g_{xy}^2\right) \end{gathered} \tag{63}$$

$$\begin{gathered} \vec{y}_{dn\_t} = g_{xy}^2 \|\vec{g}\|^2 \hat{y}_t = \left(j_x, j_y, j_z\right) = \\ \left(i_y g_z - i_z g_y, \ i_z g_x - i_x g_z, \ i_x g_y - i_y g_x\right) \end{gathered} \tag{64}$$

$$\vec{y}_{dn\_t} = \|\vec{g}\| \hat{z}_t = \left(k_x, k_y, k_z\right) = \left(-g_x, -g_y, -g_z\right) \tag{65}$$

As a consequence, also (62) must be rewritten as:

$$v_{xr} = \frac{\vec{v} \cdot \vec{x}_{dn\_t}}{g_{xy}^2 \|\vec{g}\|}, \quad v_{yr} = \frac{\vec{v} \cdot \vec{y}_{dn\_t}}{g_{xy}^2 \|\vec{g}\|^2}, \quad v_{zr} = \frac{\vec{v} \cdot \vec{z}_{dn\_t}}{\|\vec{g}\|} \tag{66}$$

In Table III.5, the computational effort of many rotation techniques has been compared with the proposed one in terms of the number of additions, multiplications, and additional mathematical functions required to process a generic rotation, starting from raw data provided by a tri-axial accelerometer.

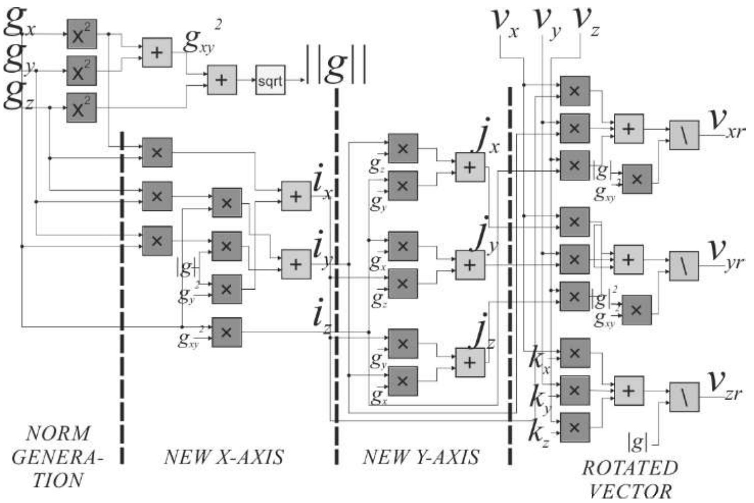


Figure III.11 *Proposed calculation scheme for the vector rotation stage.*

Table III.5 *Comparison of the number of operations and functions required to perform a reference frame transformation between the proposed algorithms and state-of-the-art methods.*

Method	Type of functions	#Add	#Mult	#Func
Euler angles	arctg, div, sqrt, sin, cos	12	27	13
Rodrigues' rotation formula	Arcsin, div, sqrt, sin, cos	18	27	7
Quaternions	Arcsin, div, sqrt, sin, cos	32	54	9
Proposed	sqrt, div	13	25	4

Square root algorithm

The circuit for the reference frame rotation has been implemented with both a FP and a Fixed-Point (FI) architecture. In each case, two different algorithms have been developed to perform the square root and division operations.

FP square root has been implemented by using the Taylor series expansion. Considering that the standard 32-bit floating-point coding (FP32) has been used (IEEE, 2019), the square root of a number n can be expressed as:

$$\sqrt{n} = \sqrt{2^{ex} \cdot significand} = 2^{ex/2} \cdot \sqrt{significand} \tag{67}$$

where *ex* is the exponent and *significand* is the significand for the FP number *n*. The significand can be approximated using the Taylor series. To keep the error below the precision of the significand ($2^{-23} = 1.19\times10^{-7}$), the square root function has been expanded around 11 different points using a third-order Taylor series. For the exponent, a division by two must be performed. If the exponent is an even number a right shift is performed, otherwise the exponent is decreased by 1, right-shifted by 1, and then multiplied by $\sqrt{2}$.

FI square root has been implemented by using the Taylor series expansion too. During GR, such operation is only used to calculate the norm of the gravity vector, $\|\vec{g}\|$, from the square of the norm of the gravity vector, $\|\vec{g}\|^2$. Since values are represented as a multiple of the acceleration due to gravity, the following assumption can be made:

$$\sqrt{\|\vec{g}\|^2} = \sqrt{1+\varepsilon} \tag{68}$$

where ε is the variation around the nominal value $\|\vec{g}_{nom}\|^2$. Based on (68), the following equality has been exploited:

$$\sqrt{r} = \sqrt{1+x} = \sqrt{1+x_0} + \frac{\left(x-x_0\right)}{2\sqrt{1+x_0}} + \frac{\left(x-x_0\right)^2}{8\sqrt{\left(1+x_0\right)^3}} + \frac{\left(x-x_0\right)^3}{16\sqrt{\left(1+x_0\right)^5}} \tag{69}$$

where r is the radicand, x is the variation around 1, and x_0 is the point around which the series is calculated. In particular, the square root function has been approximated using a third-order Taylor series. To keep the error below the precision of the adopted fixed-point encoding ($\sigma = 2^{-16}$), the function has been expanded around 10 different points over the range [0.8, 6]. In Figure III.12 the approximation error is represented. As one may notice, the range is not symmetrical about 1. In fact, for r values lower than 0.8, the curve becomes too steep, and too many points would be required to keep the error below σ. However, because of the low frequency associated with the human motion acceleration signal, the low-frequency component resulting from the filtering operation may be affected by significant fluctuations around the nominal value, based on the value of the cutoff frequency of the filter. For values $\|\vec{g}\|^2$ lower than 1, this can cause to go out of the range [0.8, 6]. To address this problem, the radicand is multiplied by 4 when its value is lower than 0.8, then the square root operation is performed, and the result is divided by 2. This allows expanding the range to [0.2, 6], and gives more flexibility for the use of this technique compared to the cutoff frequency of the filter. Moreover, the multiplication by 4 and the division by 2 can be easily performed through shifting operations.

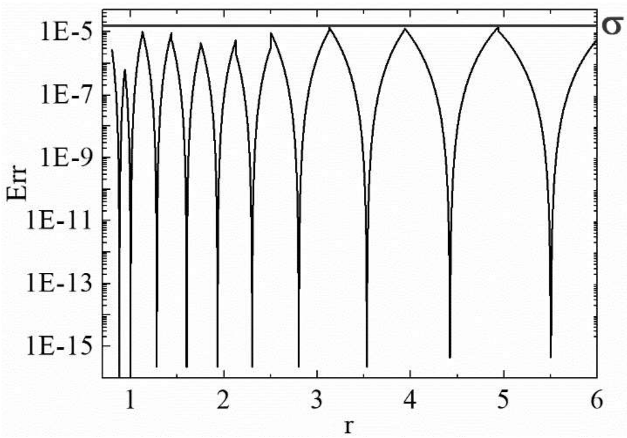


Figure III.12 *Approximation error in square root function computation using a third-order Taylor series expansion over the range* [0.8, 6]. *The function* $\sqrt{r} = \sqrt{1+x}$ *has been expanded around* 10 *points:* {−0.12, 0, 0.28, 0.60, 0.93, 1.30, 1.80, 2.53, 3.42, 4.50}.

Division algorithm

For both FP and FI encoding, the division algorithm is based on a conventional restoring algorithm (Richards, 1956). Nevertheless, some differences exist between the two cases. In the FP case, the restoring division algorithm needs to be applied to the significands only, and the division operation is completed by subtracting the exponents. In the FI case, the restoring algorithm can be applied to the two operands directly. However, to avoid the problem of starting the division process, the reciprocal value of the divisor is calculated through a restoring division algorithm, and the resulting value is then multiplied by the dividend.

Sizing the wordlength

In this section, the impact of reduced precision due to the use of fixed-point encoding is shown. The rotation algorithm has been tested on 70 datasets. Each dataset is composed of the acceleration data acquired by a LIS2DW12 tri-axial accelerometer and is associated with one of the following activities: *stationary*, *walking*, *running*, *biking*, and *driving*. Five different word-lengths have been investigated, which are 16-, 20-, 24-, 28-, and 32-bits. To represent the integer part 8 bits have been used to guarantee that no overflow occurs during the GR operation, while the remaining part is associated with the fractional part.

Using MATLAB, the input data have been converted to fixed-point using the *fi* function, while the behavior of the fixed-point arithmetic circuits has been emulated using the *setfimath* function. To evaluate the impact of the reduced precision, an ANN, composed of 5 layers, has been fed with fixed-point outputs from the vector rotation stage. The resulting predictions from the ANN have been compared with the predictions obtained using floating-point double-precision GR outputs (as required by the original model). In Figure III.13, the maximum error rate is shown for each activity and each word-length. The error rate experiences a very significant reduction when increasing the word-length from 16-bits to 20-bits and from 20-bits to 24-bits, but this reduction starts to slow down when further increasing the number of bits. Thus, a 24-bit fixed-point encoding has been chosen to implement the vector rotation stage. It should be noticed that this choice allows using the same number of bits both in the filtering and the vector rotation stages. This will be later exploited in the HW design, by using a sharing circuitry for the filtering and the vector rotation operations.

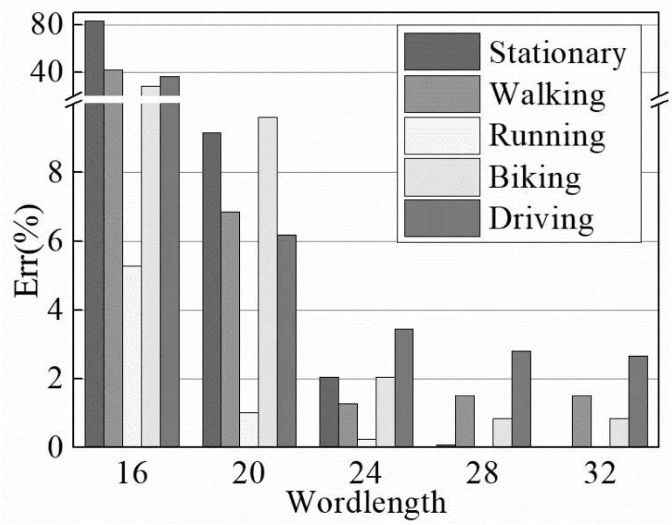


Figure III.13 *Maximum predictions error rate when an ANN is fed with fixed-point results from the vector rotation stage. Predictions obtained when the ANN is fed with floating-point double-precision outputs are taken as reference.*

Chapter IV
Hybrid Binary Neural Network

A new NN model has been studied to classify human activities. The model has been named Hybrid Binary Neural Network (HBN) because it exploits the low complexity of BNNs, but it uses non-binarized output activations for some layers in order to preserve accuracy compared to a 32-bit FP implementation. The HBN has been built and tested in Lasagne (Lasagne, 2018), which is a lightweight library to build and train neural networks. A custom dataset and 2 public datasets have been used to train and test the HBN. The custom dataset has been created by the System Research and Application group in STMicroelectronics, Agrate Brianza, Italy. Five activities were performed during the data collection, i.e. standing, walking, running, biking, and driving. The dataset is composed of 1,443,958 samples for training and 922,287 for testing. The public datasets that have been used are PAMAP2 and SHL, which have been described in paragraph II.5.

IV.1 Proposed HAR systems

The HBN has been used to perform the HAR task. Considering that inertial sensors usually embed both an accelerometer and a gyroscope, three possible configurations for the HAR system have been proposed and tested:

1. In the first configuration, the input comes from a 3-axis accelerometer only. Data is pre-processed using the algorithm proposed in paragraph III.3 before being classified by the HBN. The system is represented in Figure IV.1. Pre-processing operations aim to remove the uncertainties due to the unknown orientation of the sensor.
2. In the second configuration, the input comes from a 3-axis accelerometer only, and raw data is classified by the HBN without any pre-processing operations. The system is represented in Figure IV.2. This is the solution that requires the lowest number of resources among the proposed ones.

3. In the third configuration, the input comes from a 3-axis accelerometer and a 3-axis gyroscope. No data pre-processing is performed. The system is represented in Figure IV.3. In this case, higher accuracy can be expected because more information is provided as input. However, it should be noticed that higher resources and power consumption has to be expected as well. Indeed, 6 input channels are used as input for the HBN, against the 3 input channels used in configurations 1 and 2. Also, gyroscopes usually have higher power consumption than accelerometers. As an example, the current absorption of the accelerometer in the iNEMO inertial module (STMicroelectronics, 2018), produced by STMicroelectronics, is 170 µA in *high-performance* mode. When using both the accelerometer and the gyroscope in *high-performance* mode, the current absorption increases to 550 µA.

The details of each configuration are summarized in Table IV.1.

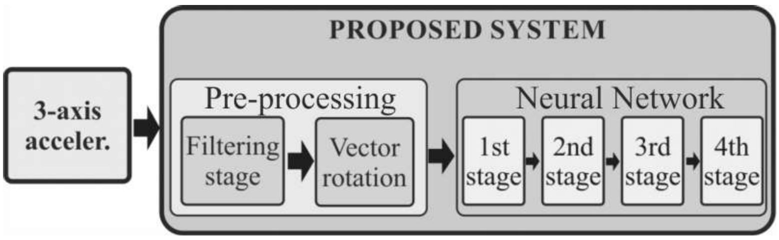


Figure IV.1 *Configuration* 1 *for the proposed HAR system. The input comes from a 3-axis accelerometer only. Data is pre-processed to remove the uncertainties due to the unknown orientation of the sensor. The classification is achieved by the HBN model.*

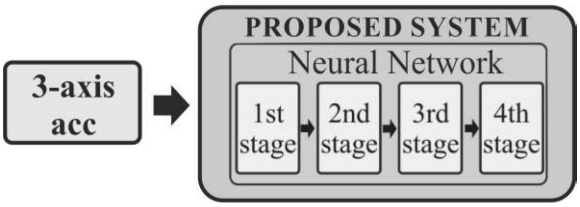


Figure IV.2 *Configuration 2 for the proposed HAR system. The input comes from a 3-axis accelerometer only. No pre-processing operations are performed. The classification is achieved by the HBN model.*

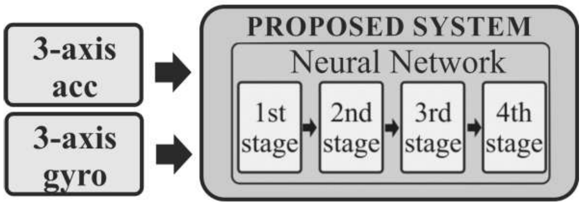


Figure IV.3 *Configuration 3 for the proposed HAR system. The input comes from a 3-axis accelerometer and a 3-axis gyroscope. No pre-processing operations are performed. The classification is achieved by the HBN model.*

Table IV.1 *Summary of the 3 configurations for the proposed HAR systems.*

Configuration	Input	Pre-Processing
1	3-axis acc	Yes
2	3-axis acc	No
3	3-axis acc + 3-axis gyro	No

IV.2 Hybrid Binary Neural Network architecture

The HBN is a CNN composed of two CONV layers and two FC layers, having all the weights binarized, namely they can only be equal to +1 or −1. For the layers whose output activations are binarized, the activation function is the sign function in (26). For the remaining layers, the activation function is the ReLU function in (21). The block diagram of the HBN model is shown in Figure IV.4, where 3 input channels and 5 classes are assumed. The inputs of the HBN are the components of the acceleration vector, pre-processed as shown in the following. An input window of 24 samples has been considered. Four sequential stages can be identified. The first stage is made up of a CONV layer and a normalization layer. The CONV layer applies a set of 8 filters (each one represents a channel) with length 5 on the signal, thus producing 8 different outputs per axis. In the normalization layer, each sample is scaled by a factor *p* and a mean value *m* is subtracted. Values for *p* and *m* are learned during the training phase. In the first stage, the output activations are binarized. The second stage is made up of a CONV layer and a Max-Pool layer. In this case, the input activations are composed of 8 channels that have been binarized considering that most of the operations are performed in this stage (Table IV.2 and Table IV.3). As for the first stage, each axis is processed separately. The CONV layer applies a set of 8 filters of size 8×5, while the Max-Pooling has size 4×1. Successively the ReLU activation function is applied. The structure of the third stage is similar to the first one, but in place of the CONV layer, there is a FC layer made up of 64 neurons. Even though weights are binarized, parameters needed for this stage require most of the memory size (Table IV.2 and Table IV.3). The last stage of the HBN is made up of a FC layer and a SoftMax classifier. The output of this last stage represents the probability of belonging to each class, therefore the number of units of the fully connected corresponds to the number of classes considered. Thus, the number of neurons in the last layer has been modified according to the dataset used to train and test the HBN. The complexity of the HBN is summarized in Table IV.2 and Table IV.3 in terms of memory required to store parameters and the number of operations to be performed. In particular, data in Table IV.2 reports the complexity of the HBN in configurations 1 and 2, when a single 3-axis accelerometer is used as input. Instead, data in Table IV.3 reports the complexity of the HBN in configurations 3, when a 3-axis accelerometer and a 3-axis gyroscope are used as input. In the latter case, the memory required to store parameters and the number of operations per sample are is 1.75 and 1.94 times higher, respectively.

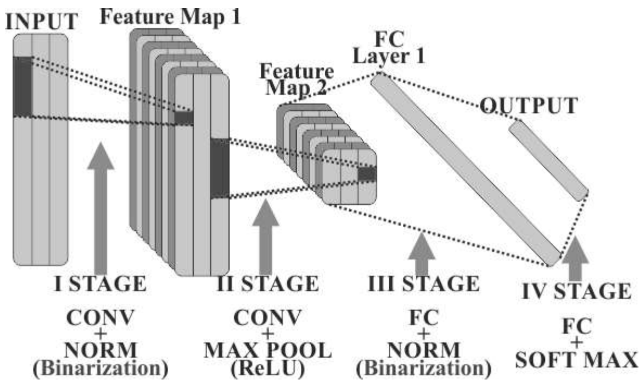


Figure IV.4 *Architecture of the exploited HBN. The "*(Binarization)*" label indicates where binarization occurs for the output activations. A 16-bits fixed-point format is assumed as input.*

Table IV.2 *Complexity of the proposed HBN model. Data refer to configuration* 1 *and configuration* 2*, i.e. when data from a single 3-axis accelerometer are provided as input.* 5 *output classes are assumed.*

Layer	Output shape	Parameters [bytes]	Number of parameters	Op. per sample	Type of op.
Conv1	20×3×8	5	40	2400	16b Add
Norm1	20×3×8	16	16	480	16b Add
Conv2	16×3×8	40	320	15360	1b Add
Max Pool	4×3×8	0	0	576	16 Add + Comp
FC1	64	768	6144	6144	16b Add
Norm2	64	128	128	64	16b Add
FC2	5	40	320	320	1b Add
Total:		1021	6968	25736	

Table IV.3 *Complexity of the proposed HBN model. Data refer to configuration* 3*, i.e. when data from a single 3-axis accelerometer and a 3-axis gyroscope are provided as input.* 5 *output classes are assumed.*

Layer	Output shape	Parameters [bytes]	Number of parameters	Op. per sample	Type of op.
Conv1	20×6×8	5	40	2400	16b Add
Norm1	20×6×8	16	16	480	16b Add
Conv2	16×6×8	40	320	15360	1b Add
Max Pool	4×6×8	0	0	576	16 Add + Comp
FC1	64	1536	12288	6144	16b Add
Norm2	64	128	128	64	16b Add
FC2	5	40	320	320	1b Add
Total:		1789	13112	49920	

IV.3 Accuracy performance of the proposed HAR systems

IV.3.1 ***Training settings***

The 3 configurations for the proposed HAR systems have been tested to measure the accuracy in the classification of human activities. To do so, the HBN has been trained and the following training settings and hyperparameters have been used:

- Optimization method: Adam (Kingma, 2015)
- Loss function: squared hinge loss
- Number of epochs: 30
- Batch size: 100
- Learning rate: from 3×10^{-3} to 3×10^{-7}

The mathematical expression of the squared hinge loss is the following:

$$L(y,\hat{y}) = \sum_{i=0}^{N}\left(\max\left(0,\, 1-y_i\cdot\hat{y}_i\right)^2\right) \tag{70}$$

where $\hat{y}$ is the predicted value and y is either 1 or −1. Also, the learning rate is not constant, but it decreases with a constant decay factor that is obtained by the following formula:

$$lr\_decay = \left(\frac{lr\_end}{lr\_start}\right)^{\frac{1}{num\_epochs}} \tag{71}$$

where *lr_end* is the learning rate at the end of the training process, i.e. during the last epoch, *lr_start* is the learning rate at the beginning of the training process, i.e. during the first epoch, and *num_epochs* is the number of epochs specified for the training process. Considering the above hyperparameters, the learning rate decay is equal to 0.736.

IV.3.2 ***Accuracy on PAMAP2 dataset***

The public dataset PAMAP2 (Reiss, 2012) has been used to test the proposed HAR system. The dataset is described in paragraph II.5.1 It provides data from 9 users performing 12 standard human activities. The accuracy performance has been evaluated in 2 different conditions:

1. In the first case, the accuracy of the HAR system has been tested in classifying 5 human activities (standing, walking, running, cycling, rope jumping) among the 12 activities provided in PAMAP2.
2. In the second case, the accuracy of the HAR system has been evaluated in classifying all 12 activities provided by the PAMAP2 dataset.

In both cases, k-fold cross-validation with $k = 5$ has been performed to measure accuracy. Also, 3 different sensor positions can be chosen in the PAMAP2 dataset, and 2 different accelerometer ranges can be selected. In total 6 different combinations of sensor positions and accelerometer ranges have been considered for both case 1 and case 2. These are specified in Table IV.4.

Table IV.4 *Possible combinations between sensor position and accelerometer range in the PAMAP2 dataset.*

ID	Sensor Position	Acc range
ankle16g	ankle	±16g
ankle6g	ankle	±6g
hand16g	hand	±16g
hand6g	hand	±6g
chest16g	chest	±16g
chest6g	chest	±6g

Accuracy Performance on 5 classes

In Figure IV.5, the accuracy performance of the proposed HAR system is graphed for each combination listed in Table IV.4 and for each configuration of the proposed HAR system. The numerical values are provided in Table IV.5. Results show that the best accuracy is always obtained with configuration 3 when both accelerometer and gyroscope are used. The highest accuracy, 99.94%, is obtained for the *chest6g* combination. By comparing configurations 1 and 2, it turns out that pre-processing operations allow increasing the accuracy at the ankle and hand positions, whereas no improvement is obtained at the chest position. This behavior can be explained by considering that no rotational movements are experienced by the sensor when it is located at the chest of the user, whereas they are present at the hand and ankle locations.

Accuracy Performance on 12 classes

The same evaluation performed on 5 classes has been repeated on all 12 classes provided in the PAMAP2 dataset. The accuracy performance is graphed in Figure IV.6, whereas the numerical values of the accuracy are provided in Table IV.6. In this case, only configurations 2 and 3 have been considered because the preprocessing operations in configuration 1 would have made indistinguishable some activities. In particular, when transforming the acceleration measurements from the DCS to the WCS, activities such as lying, sitting, and standing would have been indistinguishable. Even in this case, the best accuracy is always obtained with configuration 3, and the highest accuracy, 70.99%, is obtained for the *chest6g* combination.

IV.3.3 ***Accuracy Performance on the SHL dataset***

The public dataset SHL (Ciliberto, 2017) has been also used to test the proposed HAR system. The dataset is described in paragraph II.5.2. It was by 3 users engaging in 8 different modes of transportation. The accuracy performance has been evaluated in 2 different conditions:

1. In the first case, the accuracy of the HAR system has been tested in classifying 5 modes (car, walk, run, bike, still) among the 8 ones provided in SHL.
2. In the second case, the accuracy of the HAR system has been evaluated in classifying all 8 transportation modes provided by the SHL dataset.

In both cases, k-fold cross-validation with $k = 5$ has been performed to measure accuracy. Also, 4 different sensor positions can be chosen in the SHL dataset, i.e. Bag, Hand, Hips, and Torso.

Accuracy Performance on 5 classes

In Figure IV.7, the accuracy performance of the proposed HAR system is graphed for each sensor position and each configuration of the proposed HAR system. The numerical values of accuracy are provided in Table IV.7. As for the PAMAP2 dataset, results show that the best accuracy is always obtained with configuration 3, except for the Torso position where configuration 2 gives slightly higher accuracy. However, the difference is only 0.21 percentage points, which can be attributed to random variations. To prove that, the capacity of the proposed HAR system has been measured.

For this experiment, the dataset has not been split between the training and testing dataset, whereas the same data has been used both during training and testing. This allows getting the maximum achievable accuracy because the system is tested on the same data used for training.

Results are graphed in Figure IV.8 and they show again that the best configuration in terms of accuracy is always configuration 3. The numerical values of capacity are provided in Table IV.8. The highest values of accuracy and capacity are 98.73% and 98.82%, respectively, and they are both obtained with configuration 3 at the Bag position. By comparing configurations 1 and 2, it turns out that pre-processing operations allow increasing the accuracy only at the Hand position. This result reinforces the argument that pre-processing operations are not useful when no rotational movements are experienced by the sensor, as it happens for the Bag, Hips, and Torso positions.

Accuracy Performance on 8 classes

The same evaluation performed on 5 classes has been repeated on all 8 classes provided in the SHL dataset. The accuracy performance is graphed in Figure IV.9, whereas the numerical values of the accuracy are provided in Table IV.9. Even in this case, the capacity of the proposed HAR system has been measured as well. The capacity is graphed in Figure IV.10, whereas the numerical values are provided in Table IV.10. The highest values of accuracy and capacity are 97.33% and 97.50%, respectively, and they are both obtained with configuration 3 at the Bag position, as for the case of 5 classes.

IV.3.4 ***Accuracy on custom dataset***

The proposed HAR system has been also tested on a custom dataset. Data were collected with a 3-axis accelerometer. Five activities were performed during the data collection, i.e. standing, walking, running, biking, and driving. The dataset is composed of 1,443,958 samples for training and 922,287 for testing. In this case, only configurations 1 and 2 have been tested, due to the absence of data from a 3-axis gyroscope. Also, *k*-fold cross validation has not been performed because data were already packed in .pkl files. The results show an accuracy of 97.46% for configuration 1, while the accuracy of 93.60% is achieved with configuration 2.

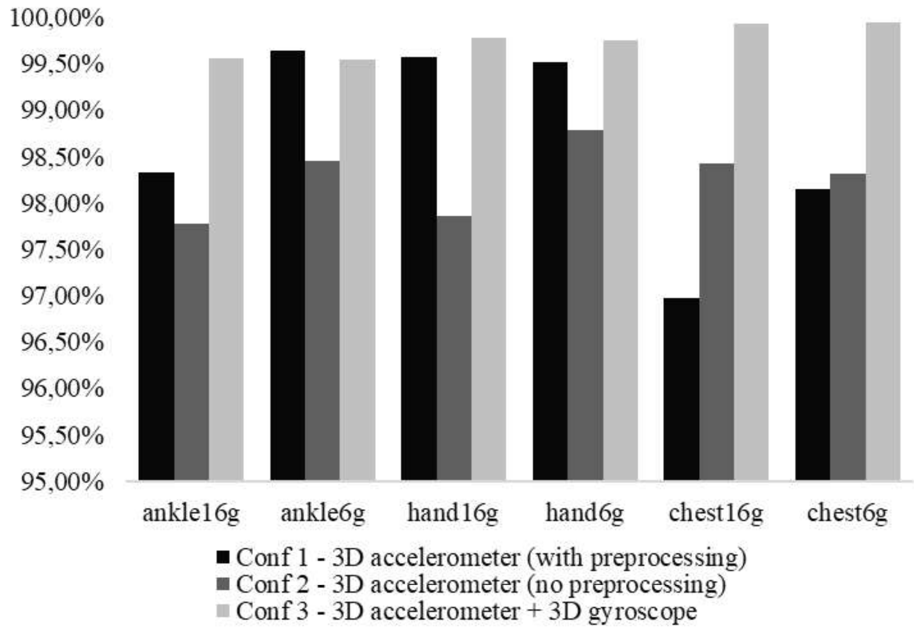


Figure IV.5 *Graph of the accuracy of the 3 configurations for the proposed HAR system on 5 classes from the PAMAP2 dataset. All combinations of sensor position and accelerometer range are considered.*

Table IV.5 *Numerical values of the accuracy of the 3 configurations for the proposed HAR system on 5 classes from the PAMAP2 dataset. All combinations of sensor position and accelerometer range are considered.*

Combination	Conf 1	Conf 2	Conf 3
ankle16g	98.33%	97.78%	99.56%
ankle6g	99.64%	98.46%	99.55%
hand16g	99.57%	97.87%	99.78%
hand6g	99.52%	98.78%	99.76%
chest16g	96.97%	98.42%	99.93%
chest6g	98.15%	98.32%	99.94%

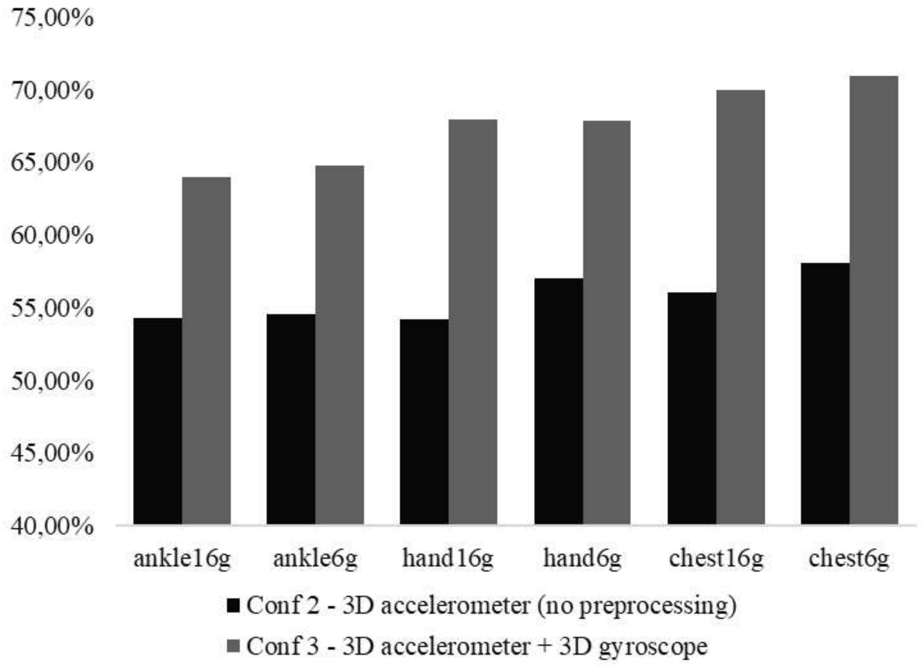


Figure IV.6 *Graph of the accuracy of the proposed HAR system (configurations 2 and 3 are considered) on 12 classes from the PAMAP2 dataset. All combinations of sensor position and accelerometer range are considered.*

Table IV.6 *Numerical values of the accuracy of the 3 configurations for the proposed HAR system on 12 classes from the PAMAP2 dataset. All combinations of sensor position and accelerometer range are considered.*

Combination	Conf 1	Conf 2	Conf 3
ankle16g	-	54.31%	63.95%
ankle6g	-	54.57%	64.77%
hand16g	-	54.25%	67.92%
hand6g	-	57.01%	67.90%
chest16g	-	56.08%	70.01%
chest6g	-	58.08%	70.99%

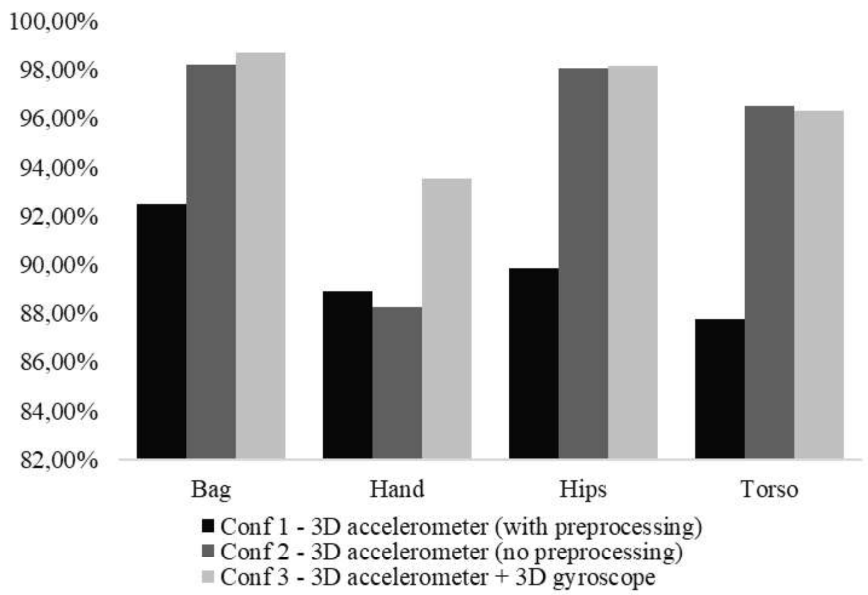


Figure IV.7 *Graph of the accuracy of the 3 configurations for the proposed HAR system on 5 classes from the SHL dataset. All sensor positions are considered.*

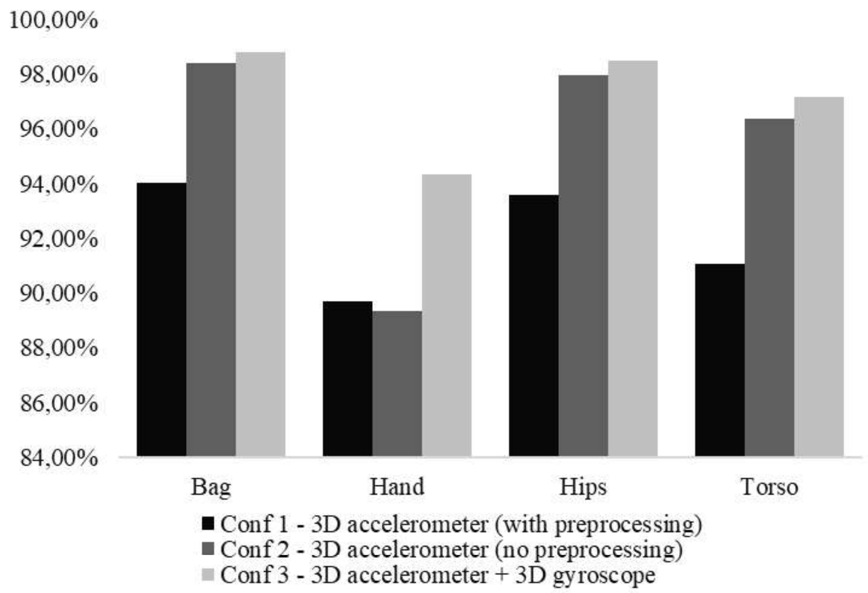


Figure IV.8 *Graph of the capacity of the 3 configurations for the proposed HAR system on 5 classes from the SHL dataset. All sensor positions are considered.*

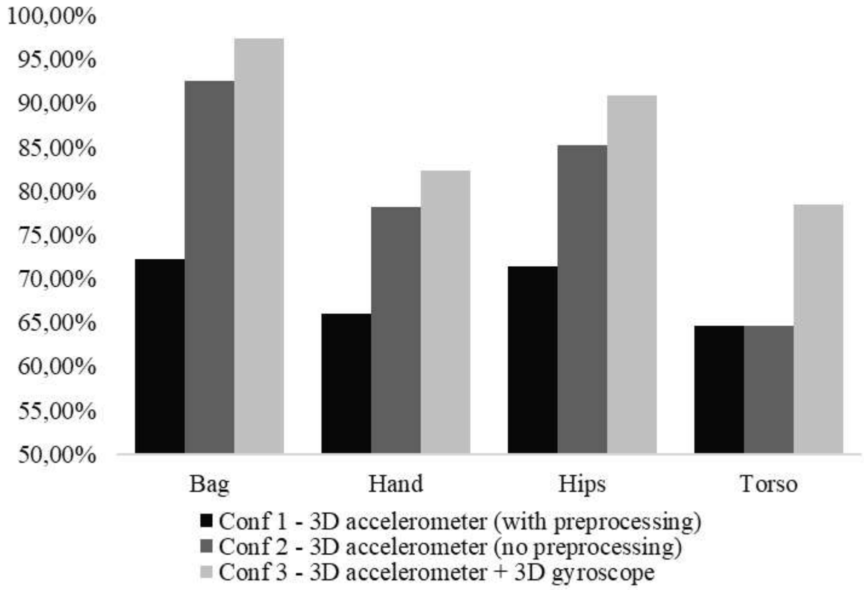


Figure IV.9 *Graph of the accuracy of the 3 configurations for the proposed HAR system on all 8 classes from the SHL dataset. All sensor positions are considered.*

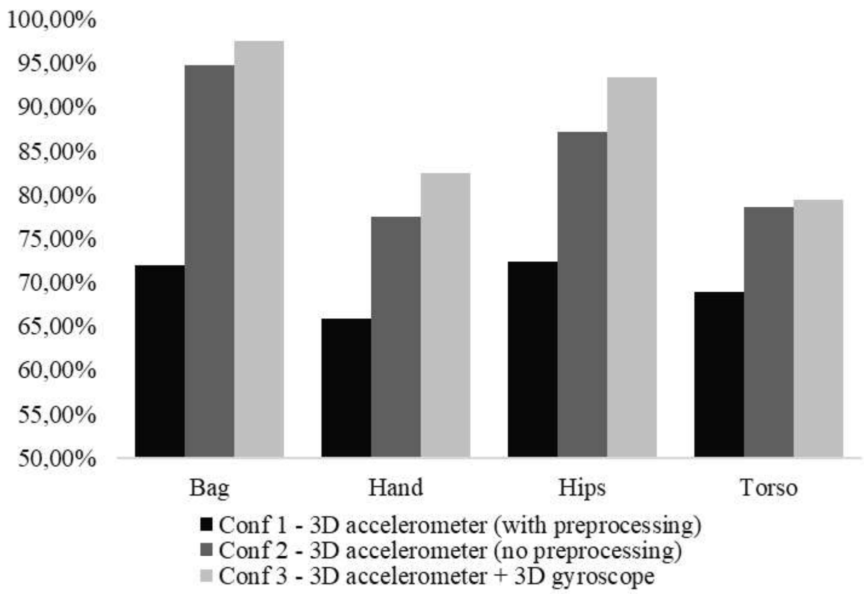


Figure IV.10 *Graph of the capacity of the 3 configurations for the proposed HAR system on all 8 classes from the SHL dataset. All sensor positions are considered.*

Table IV.7 *Numerical values of the accuracy of the 3 configurations for the proposed HAR system on 5 classes from the SHL dataset. All sensor positions are considered.*

Position	Conf 1	Conf 2	Conf 3
Bag	92.47%	98.22%	98.73%
Hand	88.89%	88.25%	93.54%
Hips	89.84%	98.07%	98.15%
Torso	87.77%	96.51%	96.30%

Table IV.8 *Numerical values of the capacity of the 3 configurations for the proposed HAR system on 5 classes from the SHL dataset. All sensor positions are considered.*

Position	Conf 1	Conf 2	Conf 3
Bag	94.02%	98.42%	98.82%
Hand	89.70%	89.34%	94.35%
Hips	93.57%	97.98%	98.49%
Torso	91.08%	96.39%	97.18%

Table IV.9 *Numerical values of the accuracy of the 3 configurations for the proposed HAR system on all 8 classes from the SHL dataset. All sensor positions are considered.*

Position	Conf 1	Conf 2	Conf 3
Bag	72.28%	92.49%	97.33%
Hand	66.06%	78.23%	82.28%
Hips	71.39%	85.16%	90.86%
Torso	64.63%	64.63%	78.43%

Table IV.10 *Numerical values of the capacity of the 3 configurations for the proposed HAR system on all 8 classes from the SHL dataset. All sensor positions are considered.*

Position	Conf 1	Conf 2	Conf 3
Bag	71.90%	94.78%	97.50%
Hand	65.92%	77.46%	82.49%
Hips	72.41%	87.21%	93.41%
Torso	68.98%	78.60%	79.48%

IV.3.5 ***Summary of the accuracy performance results***

Summing up all the results presented in the previous paragraphs, it turns out that the best configuration in terms of accuracy performance is always

Conf3. Also, pre-processing operations are convenient only for sensor positions that are affected by significant rotations during human activities, such as the ankle or the hand.

In Table IV.11 and Table IV.12 a summary of all the presented results is reported.

Table IV.11 *Summary of the accuracy performance for the PAMAP2 and the SHL dataset. Both the best configuration and the best sensor position are reported for each dataset.*

Dataset	#classes	Best Configuration	Best Position	Accuracy
PAMAP2	5	Conf 3	Chest	99.94%
PAMAP2	12	Conf 3	Chest	70.99%
SHL	5	Conf 3	Bag	98.73%
SHL	8	Conf 3	Bag	97.33%

Table IV.12 *Summary of the accuracy performance for the PAMAP2 and the SHL dataset. Both the worst configuration and the worst sensor position are reported for each dataset.*

Dataset	#classes	Worst Configuration	Worst Position	Accuracy
PAMAP2	5	Conf 1	Chest	96.97%
PAMAP2	12	Conf 2	Hand	54.25%
SHL	5	Conf 1	Torso	87.77%
SHL	8	Conf 1	Torso	64.63%

Chapter V
HW accelerator design

During the last part of this research activity, a custom HW accelerator for the HAR system proposed in paragraph IV.1 has been designed. The architecture has been designed to be compliant with the specifications of an ultra-low power smart sensor. Thus, ultra-low power consumption and a small footprint are mandatory. A custom HW architecture has been designed to implement the reference frame transformation from the DCS to the WCS, i.e. the pre-processing operations required in the system represented in Figure IV.1. This custom HW architecture is the pre-processing module in the overall HW accelerator. It should be noticed that the pre-processing module allows implementing both the filtering stage and the vector rotation stage. Then, a custom HW architecture has been also designed to execute the HBN model, which is the HBN accelerator.

All the HW architectures that have been designed have been both implemented with FPGA and synthesized with CMOS standard cells. A Xilinx Artix-7 (xc7a35tfgg484-1) FPGA (Xilinx, 2020) has been used for the FPGA implementation, and the Xilinx Vivado toolchain has been used to design and simulate the circuit. The Cadence toolchain has been used to design and simulate the CMOS standard cells implementation: Cadence Genus has been used to synthesize the design, Cadence NCSIM has been used to simulate the design, and Cadence Joules has been used to estimate the power consumption at the RTL level. To increase the accuracy of the power estimation, Value Change Dump (VCD) files have been extracted from post-synthesis simulations using Standard Delay Format (SDF) files.

Also, a FPGA-based demo board has been developed to prove the real-time operation of the proposed HAR system.

V.1 Pre-processing module

The pre-processing module is the custom HW architecture that executes the pre-processing operations. As explained in paragraph III.3, the pre-processing operations can be divided into two stages: the filtering stage and the vector rotation stage. Considering the very low sampling frequency associated with HAR systems, the main contribution to the power consumption is given by leakage power. Thus, the main criterion in the design of the HW circuitry has been to reduce the number of required resources, i.e. the area of the circuit. In the following, it will be shown that a shared reconfigurable architecture has been designed to execute both the filtering stage and the vector rotation stage.

V.1.1 ***Gravity Rotation Unit***

HW module description

The architecture designed to implement the reference frame transformation is schematized in Figure V.1. The core is the Gravity Rotation Unit (GRU), which implements the repetition pattern described in paragraph III.3.2. An iterative structure has been implemented, which allows reducing the mapped resources and, hence, the overall power density, at the cost of multicycle processing. This approach makes use of the reduced bandwidth of typical human activities, which allows lowering timing and keeping power consumption low. Furthermore, all the partial results are stored in distributed registers, avoiding the energy-consuming write/read operations associated with SRAMs or external DRAMs, typically required in general-purpose microcontrollers and processors. Inputs to the GRU, namely the vectors (g_x, g_y, g_z, v_x, v_y, v_z) or the outputs fed back through a bank of registers, are simply selected by a MUX. Each output feeds a shift register made up of 4 Flip-Flops (FFs), which allows storing partial results from previous cycles. Outputs from each register, in turn, are provided as inputs to the MUX. Each new input requires a certain number of cycles to be processed. Since an onerous pipeline structure has been avoided, a 6-bit counter is used to manage the MUX. Three divisors complete the normalization step (66), while the initial square root circuitry is embedded into the GRU module, schematized in Figure V.2. The GRU is composed of three multipliers operating in parallel, two cascaded adder/subtractors, and three multiplexers. The module operates on the inputs *i1*-*i6* and provides three outputs at each cycle, which can be alternatively taken from the outputs of multipliers and adders. During the processing, the six inputs assume the values in Table V.1, which refer to the equations presented in paragraph III.3. In the table, "*ox_y*" indicates the output *ox* (with $x = \{1, 2, 3\}$) taken *y* cycles before the current one (with $y = \{1, 2, 3, 4\}$). The operations of GRU require 18 iterations, 8 of which (3 to 10) are devoted to the square root for the norm calculation.

Table V.1 *Sequence of GRU operations.*

cnt	Inputs						Outputs		
	i1	*i2*	*i3*	*i4*	*i5*	*i6*	*o1*	*o2*	*o3*
1	g_x	g_x	g_y	g_y	g_z	g_z	a_2	m_2	a_1
2	g_x	g_x	g_y	g_y	g_z	g_z	a_2	m_2	a_1
3 … 10	*Square root calculation to compute the norm of the gravity vector*								
11	$-g_x^2$	g_z	g_z	g_y	$-g_yg_x$	*1*	m_1	m_2	m_3
12	*o2_1*	$-g_x$	*o3_1*	g	g	g_y^2	a_1	a_2	-
13	g_x	g_{xy}^2	*o1_1*	g_y	*o2_1*	$-g_x$	m_1	a_1	-
14	*o1_1*	g_x	*o1_2*	$-g_z$	-	-	a_1	-	-
15	*o1_3*	g_z	*o1_2*	$-g_z$	-	-	a_1	-	-
16	v_x	*o1_4*	v_y	*o2_4*	v_z	*o1_3*	a_2	-	-
17	v_x	*o1_2*	v_y	*o1_3*	v_z	*o2_4*	a_2	-	-
18	v_x	$-g_x$	v_y	$-g_y$	v_z	$-g_z$	a_2	-	-

Differences between FP and FI implementations

Despite Figure V.1 and Figure V.2 describe well the architecture for both the FP case and the FI case, some differences must be taken into account.

In the FP architecture, the multipliers are 32-bit Booth multipliers, while the adders are 32-bit carry-ripple adders. The final divisions require 27 additional cycles, leading to 45 overall cycles to complete the processing. Therefore, if the accelerometer frequency is set to *f*s = 25 Hz, the lower limit for the clock frequency is 45×*f*s = 45×25 Hz = 1125 Hz.

In the FI architecture, the multipliers are 24-bit Booth multipliers, while the adders are 24-bit carry ripple adders. This choice is justified by the word-length sizing described in paragraph III.3.2. In this case, to further reduce the area occupation, and, hence, the power dissipation due to leakage, the multipliers have been implemented in an iterative fashion. Each multiplier is made up of a Booth cell only, and 12 cycles are required to carry out the multiplication. To synchronize the multipliers with the operation of the whole architecture, a dedicated clock signal (*clk_mult*) has been used, whose frequency is 12 times higher than the frequency of the general clock signal (*clk*). The final divisions require 23 additional cycles, leading to 41 overall cycles to complete the processing. Therefore, if the accelerometer frequency is set to f_s = 25 Hz, the lower limit for the main clock frequency is 41×fs = 41×25 Hz = 1025 Hz. Then, the lower limit for the frequency of the *clk_mult* signal is 12.3 kHz.

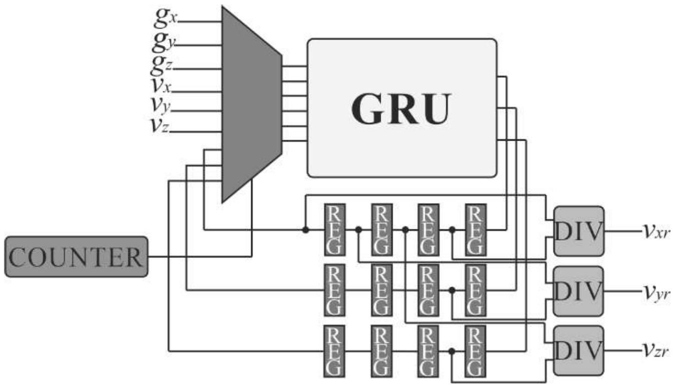


Figure V.1 *Block diagram of the HW module used to execute the reference frame transformation from DCS to WCS. The core of the HW module is the Gravity Rotation Unit (GRU).*

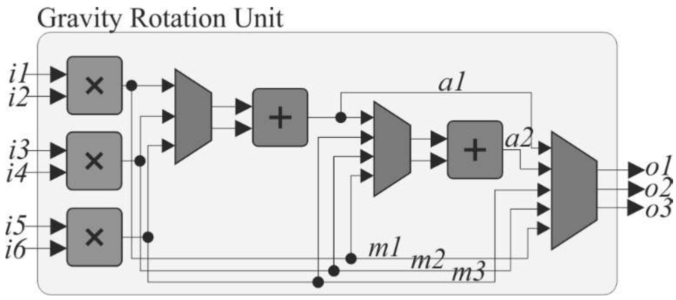


Figure V.2 *Block diagram of the Gravity Rotation Unit. The module is made up of 3 multipliers, 2 adders, and MUXs to properly manage the dataflow.*

Results

The proposed design has been implemented on the Xilinx xc7z020clg484-1 FPGA to test its functionalities and synthesized in TSMC 65 nm CMOS technology for both the FP case and the FI case. The main results are reported in Table V.2. The absence of comparisons with alternative designs is justified by the absence in the literature of HW designs with the same characteristics of low power consumption and reduced occupied area.

In the FP case, most of the resources (6460 LUTs = 75.2% of the overall resources) are required by the GRU. In turn, each multiplier occupies about 20% of the GRU. The remaining 2134 LUTs are mainly used by the divisors. Results from the synthesis in 65 nm CMOS technology report an area occupation of 0.05 mm^2 and power consumption of about 1.7 µW when the clock frequency is set to 1.125 kHz and clock gating is enabled. Thus, energy per cycle is 1.5 nJ, while energy per GR operation is 68 nJ. For low power aims, technology is the limiting factor since about 60% of the power consumption is due to leakages, although low-leakage libraries and devices with a High Voltage Threshold (HVT) have been used. Output values from the implemented design have been compared with the expected results from the test model. The comparison reveals a maximum error of 4×10^{-6} in the significand representation. This little discrepancy is justified by the propagation of the FP32 representation error during the 45 clock cycles needed to complete the elaboration of a sample. To verify the irrelevancy of this error, the neural network has been tested using the output values from the circuits. The test has shown that results perfectly match up with the predictions obtained in the test model.

Table V.2 *Comparison between FP and FI implementation of the HW architecture to execute the reference frame rotation operation. Results from both the FPGA implementation and the CMOS standard cell synthesis are reported.*

	Floating-Point (32-bit)		Fixed-Point (24-bit)	
	FPGA	CMOS 65 nm	FPGA	CMOS 65 nm
Dynamic power [mW]	< 1	$1.7\cdot10^{-3}$	<1	$0.89\cdot10^{-3}$
Static power [mW]	104	$0.68\cdot10^{-3}$	104	$0.26\cdot10^{-3}$
Total power [mW]	≅ 104	$1.02\cdot10^{-3}$	≅ 104	$0.63\cdot10^{-3}$
#LUTs	8594	-	2760	-
#FFs	1285	-	1305	-
Area [mm^2]	-	0.05	-	0.024

In the FI case, a significant amount of resources is required to implement the dividers, which use about 45% of the LUTs. Despite the GRU is the core unit of the proposed accelerator, it requires 28% of the LUTs only. This is achieved thanks to the iterative implementation of the multipliers. The remaining resources are needed to implement the glue logic, i.e. the input MUX. The FPGA implementation shows that the delay associated with the critical path is 15.417 ns for *clk_mult* intra-clock paths, while it is 28.380 ns for *clk* intra-clock paths. Thus, the maximum operating frequency for the *clk_mult* signal is 64.86 MHz. As a consequence, the maximum operating frequency for the *clk* signal is 5.40 MHz, which is much higher than the specified lower limit. Results from the synthesis in 65 nm CMOS technology report an area occupation of 0.024 mm^2 and a dissipated power of about 0.89 µW when the general clock (*clk*) frequency is set to 1.025 kHz. Thus, the energy per clock cycle is 0.87 nJ, while energy per GR operation is 35.6 nJ. For low power aims, technology is the limiting factor since more than 70% of the dissipated power is due to leakages, although low-leakage libraries and devices with a HVT have been used. The results show a 2× reduction compared to the FP implementation both in area occupation and power consumption, with minimum impact on the overall accuracy of the system.

V.1.2 ***Filter stage circuitry***

Coupled-All pass filter realization

As shown in Figure III.6, a highly regular structure is obtained when all-pass filters are realized through the two-pair extraction approach. This allows obtaining an iterative implementation of the filter, in which the same fundamental cell (see Figure III.5) is re-used a number of times equal to the order of the filter. Based on this method, a Coupled All-Pass filter also can be implemented in an iterative fashion. Considering what has been explained in paragraph III.3.1, the same fundamental cell can be used to emulate $A_1(z)$ first and then $A_2(z)$. Considering Figure V.3, first, the cells I, II, and III of the filter $A_1(z)$, and then the cells IV and V of the filter $A_2(z)$ are emulated using the fundamental all-pass cell. Lastly, cell VI is emulated using the same fundamental cell in which only the adder is used. How the all-pass fundamental cell must be used is described by the following equations:

$$
\begin{aligned}
Y_1 &= V_1 + z^{-1} X_2 \\
Y_2 &= X_1 + V_1 \\
V_1 &= k_m \left(X_1 - z^{-1} X_2 \right)
\end{aligned}
\tag{72}
$$

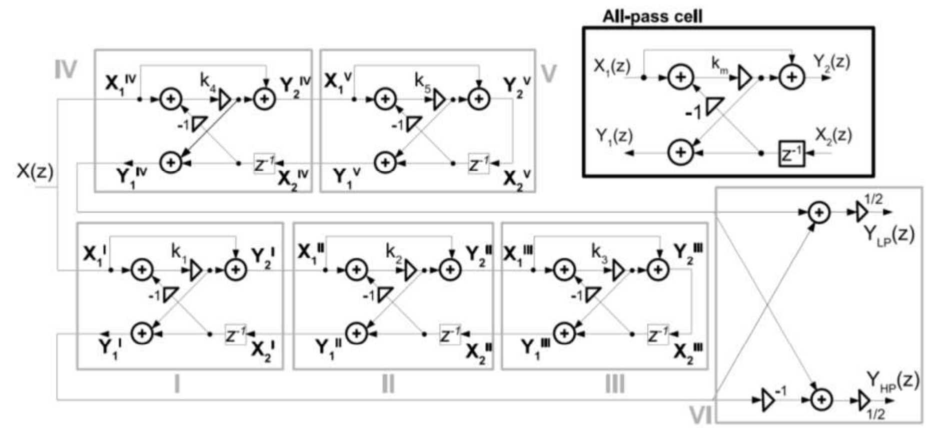


Figure V.3 *Realization of the filter using a Coupled All-Pass structure and iterating on an All-pass fundamental cell. The latter is detailed in the dark black box in the upper right corner of the figure. Each used cell is identified with a Roman numeral.*

Re-using the Gravity Rotation Unit resources

Noting that V_1 is required for the computation of both Y_1 and Y_2, the circuitry for its calculation is implemented once but its result is used to calculate Y_1 and Y_2 in two consecutive cycles. The scheme representing the computation of V_1 is represented in Figure V.4, while the scheme for the computation of Y_1 and Y_2 is represented in Figure V.5. In both cases, the scheme can be realized using part of the GRU described earlier. In the end, the whole fundamental cell can be realized using part of the circuitry of the GRU, as shown in Figure V.6.

Having used an iterative implementation, 12 cycles are needed to process a single sample from the accelerometer. However, the accelerometer provides three outputs in parallel, each related to a different axis. As a consequence, in order to process the three samples, 36 total cycles are needed. If the sampling frequency of the 3-axis accelerometer is 25 Hz, the minimum allowable clock frequency for the filter is 36×25 Hz, which is 900 Hz.

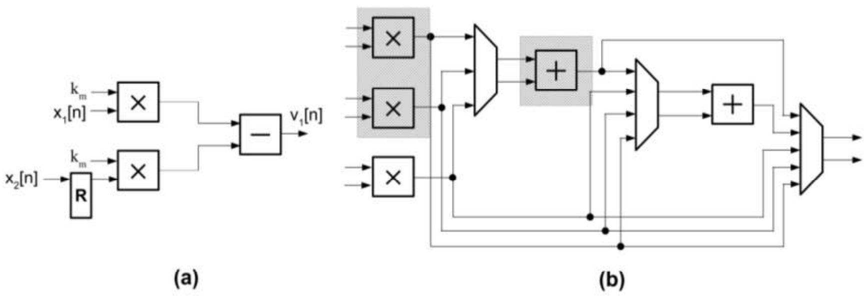


Figure V.4 *(a) Scheme for the calculation of* V_1 *and (b) part of the GRU needed to implement the scheme.*

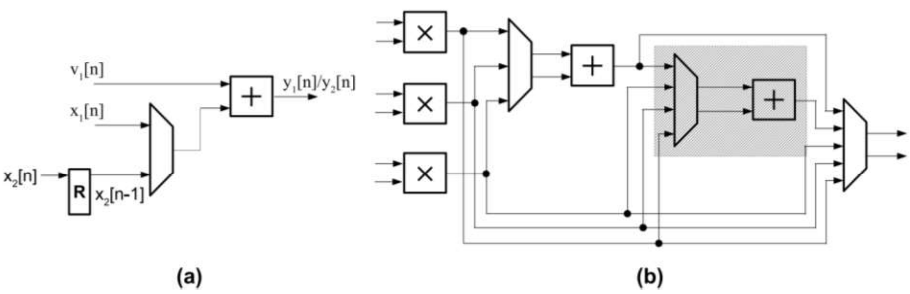


Figure V.5 *(a) Scheme for the calculation of* Y_1*,* Y_2 *and (b) part of the GRU needed to implement the scheme.*

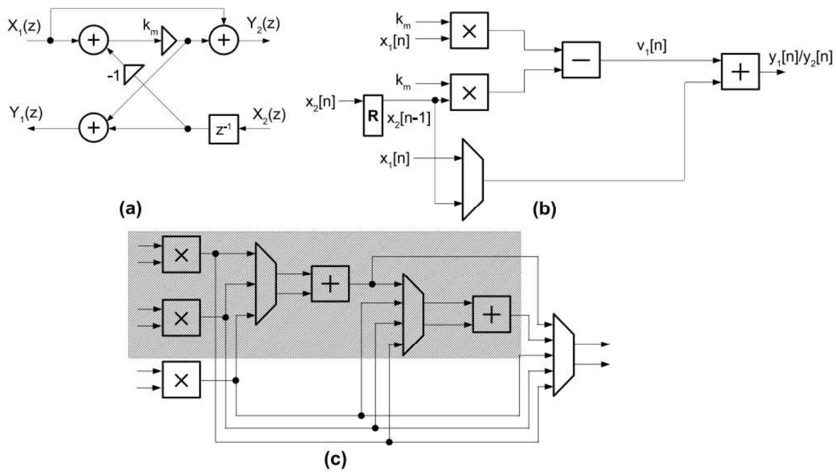


Figure V.6 *(a) Fundamental all-pass cell, (b) HW implementation for its realization, and (c) the corresponding part of the GRU needed to implement the scheme.*

V.1.3 ***Pre-processing module architecture***

As shown in the previous paragraph, the filtering stage and the vector rotation stage can be implemented using a shared circuitry. This allows obtaining a reconfigurable architecture, which can operate both in "*filtering mode*" and in "*GR mode*". The reconfigurability is obtained through multiplexers (MUXs) that allow proper routing of the signals. To minimize the power consumption of the architecture, FI 24-bit coding has been used. The scheme of the designed reconfigurable architecture is shown in Figure V.7, where the MUXs in black are the ones that allow switching from one mode to another. Using this scheme, 36 cycles are required to filter the input samples from the ADC of the sensor ("from sensor" in the figure). Then, the GR operation requires 34 cycles, where the last 18 cycles are needed to perform the normalizations in (66). Considering that the divisions are implemented through a dedicated circuitry, only the first 16 cycles of the GR operation need to be serialized with the 36 cycles of the filtering operation, while the remaining 18 cycles can be run in parallel. Thus, the total number of cycles required to perform the whole pre-processing pipe is 52. If the sampling frequency of the accelerometer is 25 Hz, the minimum allowable clock frequency for the design in Figure V.7 is 25×52 Hz = 1.3 kHz (12×1.3 kHz = 15.6 kHz for the *clk_mult* signal).

The design in Figure V.7 has been synthesized using Cadence in TSMC 65 nm CMOS technology. The results are summarized in Table V.3. The power consumption has been estimated at 15.6 kHz.

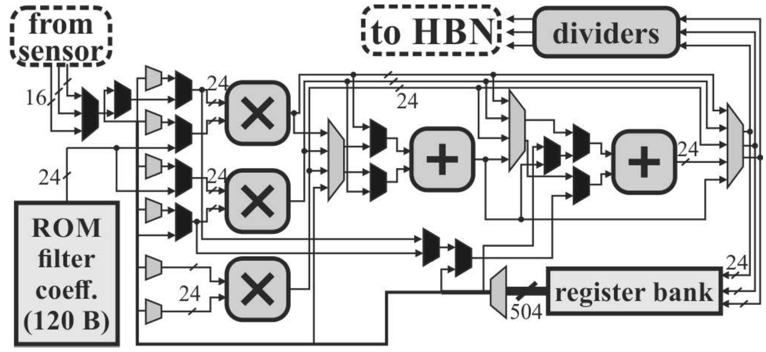


Figure V.7 *Block diagram of the HW architecture which implements the overall preprocessing module.*

Table V.3 *Synthesis results of the pre-processing module.*

Technology	Dynamic power [uW]	Leakage power [uW]	Total power [uW]	Area [mm^2]
65 nm HVT	0.45	0.81	1.26	0.030

V.2 HBN accelerator

A custom HW accelerator to execute the HBN model proposed in paragraph IV.2 has been designed as well. Generally, Two aspects are critical in the HW implementation of ANNs: the large number of arithmetic operators and the allocation of a large amount of memory for storing weights and partial results, as well as the power dissipation related to the numerous memory accesses (Sze, 2017). In the proposed implementation, weight binarization has reduced the MAC operations to simple ADD/SUB operations, namely each CONV layer calculates the following quantities:

$$\sum_{i=1}^{N} w_i x_i + b = \pm x_1 \pm x_2 \pm ... \pm x_N + b \tag{72}$$

where w_i and x_i are the weights and the inputs to a certain neuron, respectively, and b is the bias. Since the energy cost of data read/write operations from off-chip memories can be up to 200× higher than on-chip data transfer (Sze, 2017), an effort has been done to use only on-chip memories.

V.2.1 ***Architecture of the HBN accelerator***

Two designs of the HNN accelerator are proposed. The first one is the FIFO-based design, where memories have been implemented by using distributed FIFOs and RAM has been completely avoided. A second version is the RAM-based design, which uses RAM to store weights and biases, while FIFOs continue to be used to store the output activations. The choice to present both solutions derive from the need to find different optimal area/power trade-off in different utilization scenarios. Auxiliary circuitry of sensors, indeed, are equipped with a very limited amount of RAM, which could be insufficient for the HAR operations, and compel to use FIFOs. In turn, FIFOs permit higher operation frequencies than RAM, and the associated dynamic power scale up with a lower slope than RAM. This makes FIFOs convenient for higher frequencies applications. On the contrary, RAM could be a convenient choice for target platforms such as FPGA, which could advantage of distributed memories and the lower power dissipation for data transfers, due to the locality of data. The block diagram of both HBN designs is shown in Figure V.8. The architecture exploits 3 cores since this is the minimum number of cores that can process in parallel the 3 components of the pre-processed acceleration. In Figure V.9 and Figure V.11, the architectures of the cores of the FIFO-based and the RAM-based HNN accelerator are detailed. Thanks to weight binarization, the processing element (PE) is a 3-levels adder tree that uses 16-bits FI arithmetic in both cases. The first level of the adder tree is made up of 3 adders so that a dot product between vectors of length 5 can be performed in one cycle, and a bias or a result from the previous cycle can be summed up as well.

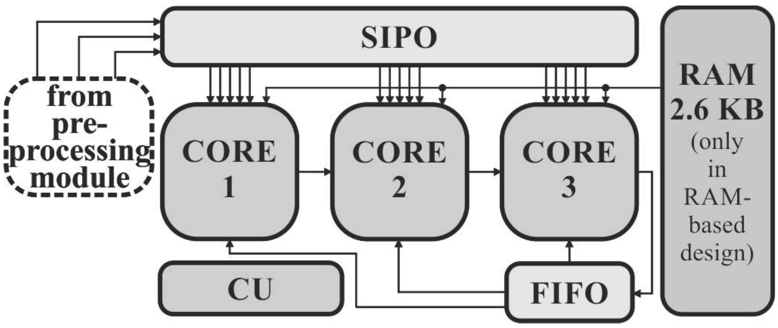


Figure V.8 *Block diagram of the proposed HBN accelerator. The RAM module is present in the RAM-based design only. The structure of the cores is different for the two versions.*

Another aspect of the proposed design is scalability. If higher throughput were required by the target application, the number of cores can be increased with a very low design effort. Indeed, the strategy of storing the model weights locally in each core mitigates the bandwidth-related issues that normally would arise when scaling up the design (Chen, 2019). Thus, we can expect that the performance varies linearly with the number of cores. For example, by doubling the number of cores we can expect a doubling of power, area, and throughput.

Architecture of the cores in the FIFO-based HBN accelerator

The block diagram of each core in the FIFO-based HBN accelerator is shown in Figure V.9 In the FIFO-based design, each core embeds 800 bytes of FIFO memories in which weights, biases, and partial results are stored. Each core locally stores all the parameters needed to run the model. Also, output activations from CONV layers are locally reused in each core. Thus, the design takes advantage of a "flattened" memory hierarchy, where there is no need to execute high-cost access operations to higher levels in a memory hierarchy. To make this possible, FIFOs must be initialized during the system start-up by an external data stream. Successively, FIFOs work as a circular buffer, carefully managed by a Control Unit (CU). In particular, in the design in Figure V.9, the "FIFO_w" structures store the weights of the model, whereas "the FIFO_b" structures store the biases. The structure of the circular FIFOs is represented in Figure V.10 for both the "FIFO_w" and the "FIFO_b" structures. At the startup of the system, the CU sets the LDP signal to 1 so that FIFOs are loaded with the parameters of the model by an external stream of data. During the normal operation of the systems, the CU sets the LDP signal to 0, so that, each time that a parameter is read and used, it is sent back to the first element of the FIFO. By doing so, there is no need to access a higher memory hierarchy level to re-load the parameters.

As shown in Figure V.9, two different "FIFO_w" and "FIFO_b" modules are needed in each core and used when the circuit implements a CONV layer or a FC layer, respectively. Indeed, CONV layers must be processed 16 times to get new input for the FC layers. Thus, considering that in FIFO structures we cannot have random accesses to the memory locations, we should have had to swipe all the weights of the CONV and the FC layers even when the latter would have not been useful. This would have been a drawback for the design because the number of weights of the FC layers requires 77% of the total memory required to store the network parameters, as reported in Table IV.2. In particular, considering that each weight in the HBN is represented by a single bit and that each CONV layer has a filter with dimension proportional to 5 (5 or 5×8), FIFOs for CONV layers have dimensions 80×5 bits, which corresponds to 50 bytes, while those for FC layers are 608x5 bits, which corresponds to 380 bytes. The same applies to "FIFO_b" but, considering that biases are coded with FI 16-bits, FIFOs require 16×16 bits (32 bytes) and 64×16 bits (128 bytes), respectively. "FIFO_o" stores the output activations of each layer. Each one of the above output FIFO is divided into up to 5 blocks, in order to provide up to 5 different output activations in parallel to the PE, designed to perform a dot product between vectors of length 5 in one cycle. In Figure V.9, the 3 "FIFO_o" memories store the output activations of the first stage, the second

CONV layer and the Max-Pool layer, respectively. Each axis is processed separately in CONV layers, thus the memory for the output activations is locally associated with each core. As shown in Figure V.8, a unique external FIFO memory is also used to store the output activations of the first FC layer, since in this case, all the input activations from the previous layer cannot be separated. Considering that the scheme in Figure V.8 iteratively implements all the layers of the HBN, the complex signal routing is managed by the devoted CU, which in turn has been implemented with a Finite State Machine (FSM) having a state for each layer.

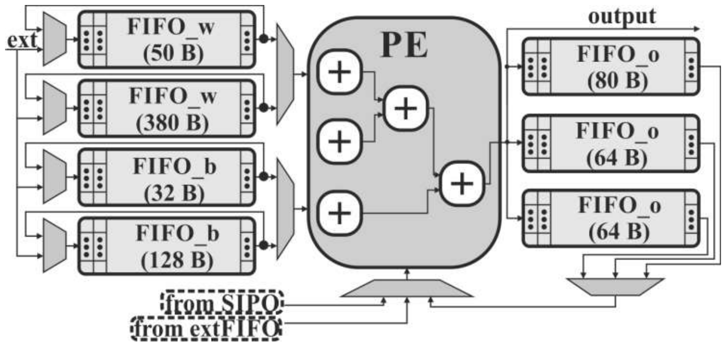


Figure V.9 *Block diagram of a core in the FIFO-based design. In this case, weights and biases are stored in FIFO memories locally. FIFO_w are the FIFOs where weights are stored, whereas FIFO_b are the FIFOs where biases are stored. Output activations from CONV layers are stored in FIFO_o and are re-used locally in each core.*

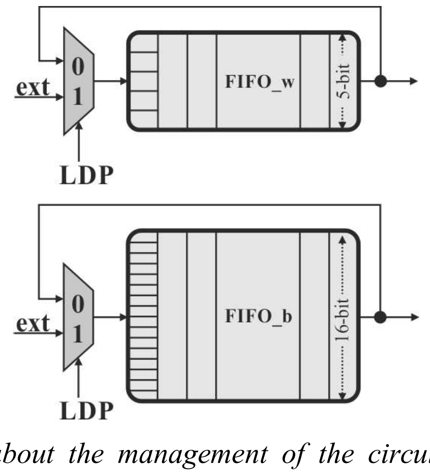


Figure V.10 *Detail about the management of the circular FIFOs. At the startup of the system, the CU sets the LDP signal to 1, and FIFOs are loaded with parameters by an external stream of data. During normal operations, the CU sets the LDP signal to 0 so that each parameter is sent back to the first element of the FIFO after having been used.*

Architecture of the core in the RAM-based HBN accelerator

The block diagram of each core in the RAM-based HBN accelerator is shown in Figure V.11. In the RAM-based design, a RAM has been instantiated to store weights and biases in place of FIFOs. The RAM is external to the core, as represented in Figure V.8. This choice is advantageous in terms of power dissipation since it avoids the data shifts that FIFOs do at each read operation, although the highest advantage is obtained with the availability of on-chip distributed RAM typical of FPGAs. The cores of the RAM-based HNN accelerator in Figure V.11 have a similar structure to the FIFO-based ones, but a RAM module of 31x696 bits, which corresponds to 2.63 KB, reduces the FIFOs dimensions to 200 bytes. The most significant 15 bits of each word of the RAM are used to store weights, therefore again each core receives 5 binarized weights at each cycle. The remaining 16 bits are used for the biases. The proposed architecture has been prototyped with a Xilinx Artix-7 FPGA and, for ease of comparison also with standard cells (std_cells) implementation. In the latter case, a larger SRAM of 32x704 bits has been instantiated due to the limitations of the memory compiler.

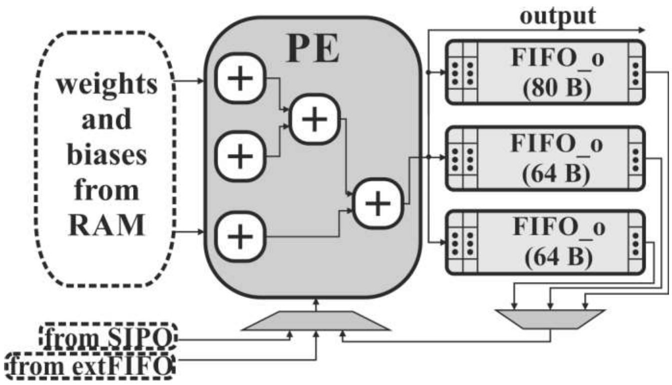


Figure V.11 *Block diagram of a core in the FIFO-based design. In this case, weights and biases are stored in a RAM, which is external and shared by each core. Output activations from CONV layers are stored in FIFO_o and are re-used locally in each core.*

V.2.2 ***Architecture of the processing element***

The Processing Element (PE) is made up of a 3-levels 16-bits fixed-point adder-tree and the circuitry to implement the activation functions (sign function (26) and ReLU (21)).

Adder Tree

Thanks to binarization multipliers are not required to process the layers in the HBN. In layers where inputs and weights are both binarized, a XNOR-popcount circuitry could have been used. However, this would have been an additional circuitry, which would have required additional resources and, hence, power. Thus, even for these layers, the processing is performed in the PE by using the adder tree. Nevertheless, it must be considered that the sign of operand changes when the weight is −1. This has been taken into account by using the circuitry in Figure V.12 for the implementation of the adders in the first level of the adder tree. The truth table for the signals *s_A*, *s_S*, *C_in*, and *RC* is reported in Table V.4. Only when all the weights are −1 (0 in binary) the result needs a further sum up of +1 to provide the correct results. This is accomplished by propagating a carry bit (*RC*) to the next level of the adder tree.

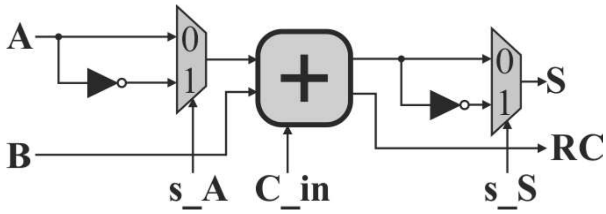


Figure V.12 *Circuitry for the sign management for the first level of the adder tree in the PE.*

Table V.4 *Signals for the sign management in the adder-tree.*

Required Operation (weights)	*s_A*	*C_in*	*s_S*	*RC*	*Result (S)*
A+B (1, 1)	0	0	0	0	A+B
A−B (1, 0)	1	0	1	0	A−B
−A+B (0, 1)	1	1	0	0	−A+B
−A−B (0, 0)	0	0	1	1	−A−B−1

Non-linearities implementation

A small circuitry has been deployed in each core to evaluate the non-linear functions, namely the ReLU function (21) and the sign function (26). The circuitry is represented in Figure V.13. The result of both the ReLU function and the sign function (used to implement the binarization) depends on the sign of the input operand, which is the sign "AT_RES". The sign is deduced by looking at the Most Significant Bit (MSB). Thus, the multiplexer "M_ReLU" implements the ReLU function by selecting between "AT_RES" and 0 based on “MSB(AT_RES)”. Instead, the multiplexer “M_BIN” implements the binarization by selecting between +1 or −1 (represented in two’s complement 16-bits fixed point coding, with an 8-bit fractional part) based on “MSB(AT_RES)”. The signal “s_NL” is then used to select the desired non-linearity, and the signal “s_RES” is used to choose if applying the selected non-linearity or not. Finally, the non-linearities are simply performed using MUXs.

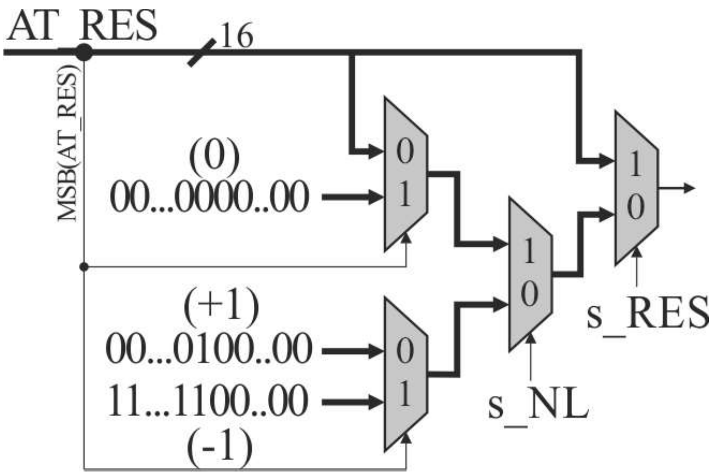


Figure V.13 *Block diagram of the circuitry for the implementation of ReLU function and binarization.*

V.3 Results

The performance of the HW accelerator, which consists of the pre-processing module and the HBN accelerator, has been measured both with FPGA and CMOS standard cells.

*V.3.1 **Results from FPGA implementation***

In Table V.5, the results from the FPGA implementation are summarized for both the FIFO-based and the RAM-based designs, and the proposed design has been compared to state-of-the-art works (Jafari, 2019), (Gaikwad, 2019), which are oriented to custom HW implementation. The FPGA takes advantage of the presence of RAM since the LUT utilization is reduced by about 3% compared to the FIFO-based counterpart, namely from 31.7% to 28.8% of the total number of the available LUTs on the FPGA. Analogously, the FF utilization is reduced by about 2.3% since 1017 FFs are used to store weights and biases of the HBN. The maximum clock frequency for both designs is 41 MHz, corresponding to a maximum Output Data Rate (ODR) of the sensor of 3.2 kHz However, when the ODR of the sensor is set to 25 Hz, normally used in HAR systems, considering that 12600 clock cycles are needed to process a data, the minimum clock frequency required for real-time operation is 315 kHz, indicated as the operating frequency (OpFreq) in Table V.5. At this frequency, considering that 16 input samples are required to obtain a prediction, the delay (Delay@OpFreq) is equal to 16×(1/25 Hz) = 640 ms. As shown in Table V.5, the value is higher than the ones obtained by Jafari *et al.* (2019) and Gaikwad *et al.* (2019). However, this is justified by the higher operating frequencies used in those works, which is not actually required in the proposed HAR system. However, if the proposed FPGA implementation would work at its maximum frequency, the delay will decrease to 5 ms, which is lower than the one achieved by Jafari *et al.* (2019). The delay obtained by Gaikwad *et al.* (2019) is orders of magnitude lower than the one achieved by the proposed solution. However, this is justified by the execution of a simpler model that can only achieve 94.6% accuracy. In contrast, the proposed design can achieve 99.5% accuracy (this result refers to the classification of 5 classes from the PAMAP2 dataset by using configuration 1 and hand16g combination, Table IV.5).

To estimate the power consumption, the power tool provided by Vivado has been set up at a high level of confidence by using Switching Activity Interchange Format (SAIF) files generated from post-implementation simulations. The power estimation returns for both designs a total power consumption of 72.04 mW at the OpFreq. This is almost all composed of static power, equal to the quiescent power dissipation of the FPGA. Dynamic power is under the sensitivity of the tool, which returns a generic <1 mW. Therefore, for the FPGA implementation, there are no significant differences between the two designs, and the RAM-based design could be preferred since it takes advantage of the primitives of the FPGA, while LUTs and FFs can be saved for other purposes. This should be considered in the economy of the whole system both in terms of power consumption and area occupation. A lower number of resources is required by our designs, although we do not instantiate DSP modules in order to provide results that are independent of the specific target platform, and for a fair comparison with std_cell implementations. The total RAM requirement for our RAM-based design is equivalent to 1 BRAM and 0 for the FIFO-based, while Jafari *et al.* (2019) use a significant number of BRAMs. The power consumption has been compared at the OpFreq of each system. The proposed design shows a reduction of the power consumption of 37% and 70% compared to the one obtained by Jafari *et al.* (2019) and Gaikwad *et al.* (2019) respectively. However, to make comparisons independent from the OpFreq, the normalized dynamic power consumption has been compared, where the proposed designs achieve a reduction of 70% compared to the results obtained by Gaikwad *et al.* (2019).

Table V.5 *Results from FPGA implementation of the proposed HW accelerator. The HW accelerator is made up of the pre-processing module and the HBN accelerator. The results are compared with state-of-the-art solutions as well.*

	Proposed FIFO-based design	Proposed RAM-based design	Jafari *et al.* (2019)	Gaikwad *et al.* (2019)
Platform	Artix-7	Artix-7	Artix-7	Artix-7
Accuracy	99.5%	99.5%	98.0%	94.6%
Dynamic Power [µW/MHz]	137	134	460	N.A.
Static Power [mW]	72	72	71	N.A.
Total Power @ OpFreq [mW]	72.04	72.04	116	241
#slices	2093	1856	982	N.A.
#LUTs	6601	5988	N.A.	3466
#FFs	5272	4299	N.A.	569
#DSPs	0	0	3	81
#BRAMs	0	1	14	0
Max Frequency [MHz]	41	41	N.A.	N.A.
Max Sensor ODR [kHz]	3.2	3.2	N.A.	N.A.
Delay@OpFreq [ms]	640	640	14.8	2.7×10^{-4}
Minimum Delay [ms]	5	5	N.A.	N.A.
Energy per inference [mJ]	46	46	N.A.	N.A.

V.3.2 ***Results from CMOS standard cells synthesis***

In Table V.6, the results of synthesis with TSMC CMOS 90 nm std_cells are summarized for both the FIFO-based and the RAM-based designs and compared with the solution in the paper by Jafari *et al.* (2019), which only presents ASIC results. The power consumption has been estimated by extracting Value Change Dump (VCD) files from post-synthesis simulations using Standard Delay Format (SDF) files. The power consumption has been estimated using Cadence Joules. The dynamic power consumption of the RAM-based design is 2.8 times higher compared to the FIFO-based design, while the leakage power is 1.8 times lower. However, the dynamic power consumption is negligible at the OpFreq. Therefore, despite the shifting of the data in the FIFOs does not represent an issue for the dynamic power consumption, the best solution to reduce the power consumption at the considered frequencies is the RAM-based design. On the contrary, being the memory distributed in the FIFO-based design, the maximum frequency is 1.6 times higher than the one of the RAM-based design. This could be considered for applications in which a high throughput is the first specification. Moreover, to verify the scaling capabilities and the actual impact of leakages, the proposed design has been synthesized with TSMC CMOS 65 nm low-power (LP) high-voltage-threshold (HVT). The HVT feature allows to strongly reduce the leakage power, at the cost of reduced speed. The results are reported in Table V.6. Unfortunately, the lack of a memory compiler for the 65 nm technology prevented the possibility to synthesize a RAM-based design with the more shrunk technology. Results show that the power consumption is only 6.3 μW at the OpFreq, which is 3 orders of magnitude lower than the above results. Despite this, 86% of the total power is leakage power. The overall area occupation is 0.20 mm^2. A detail of the various components of the design is shown in Figure V.14 and Figure V.15. In Figure V.14, it is shown that most of the area is required for the HNN accelerator module. In particular, each core occupies 26% of the architecture, whereas the pre-processing module occupies 16% of the total area. The remaining area is mainly used for the external FIFO (see Figure V.8), the SIPO, and the CU. Also, in Figure V.15 the breakdown of the various modules in each core is shown. 96.9% of the area occupation in each core is due to FIFO memories, and only the remaining 3.1% is required to implement the PE. In fact, thanks to weight binarization, the PE has been implemented avoiding multipliers, whose implementation requires large and power-hungry circuits. In Figure V.14 and Figure V.15, also a breakdown of the power consumption has been reported. Considering that power dissipation is mostly due to leakages, the power consumption breakdown exactly follows the one of the area occupation. In Table V.6 the proposed design has also been compared to one proposed by Jafari *et al.* (2019). To have a fair comparison of power consumption, we have considered results at the same throughput. In the work of Jafari *et al.* (2019) a throughput of 67 label/s is achieved at a clock frequency of 100 MHz. In the proposed solution, 202k clock cycles are required to produce a label. Thus, a lower clock frequency of (67×202k) Hz = 13.5 MHz is required to have a throughput of 67 label/s. At this frequency, the FIFO-based design and the RAM-based design dissipate 2.54 mW and 1.52 mW, which corresponds to a reduction of 7.3 times and 12.2 times respectively compared to the power consumption obtained by Jafari *et al.* (2019). The delay required to produce a label and the area occupation of the proposed designs are almost the same as the one obtained by Jafari *et al.* (2019). The maximum frequency achieved by Jafari *et al.* (2019) is 5.4 times higher and 8.8 times higher than the FIFO-based design and the RAM-based design, respectively. Despite this, the proposed design is able to provide a 1.4 times higher throughput in the case of the FIFO-based design.

It is interesting to note that the proposed design is not just competitive in terms of total power consumption, but in terms of the dynamic one. Indeed, the dynamic power consumption is up to 42 times lower than the one achieved by Jafari *et al.* (2019). This suggests that the proposed design might be reused for other applications as well, where higher throughput is required.

Table V.6 *Results from CMOS standard cell synthesis of the proposed HW accelerator. The HW accelerator is made up of the pre-processing module and the HBN accelerator. The results are compared with a state-of-the-art solution as well.*

	Proposed FIFO-based design		Proposed RAM-based design	Jafari *et al.* (2019)
Technology	CMOS 65 nm LP HVT	CMOS 90 nm GP	CMOS 90 nm GP	CMOS 65 nm
Dynamic Power [µW/MHz]	2.6	3.1	8.8	111
Leakage Power [mW]	5.4×10^{-3}	2.5	1.4	7.4
Total Power @ OpFreq [mW]	6.3×10^{-3}	2.5	1.4	N.A.
Total Power @ 67 labels/s [mW]	N.A.	2.54	1.52	18.5
Area [mm^2]	0.20	0.36	0.39	0.40
Max Frequency [MHz]	105	158	97	857
Max Throughput [label/s]	N.A.	784	480	574
Energy [µJ]	N.A.	38	23	274
Max Sensor ODR [kHz]	8.7	12.5	7.7	N.A.

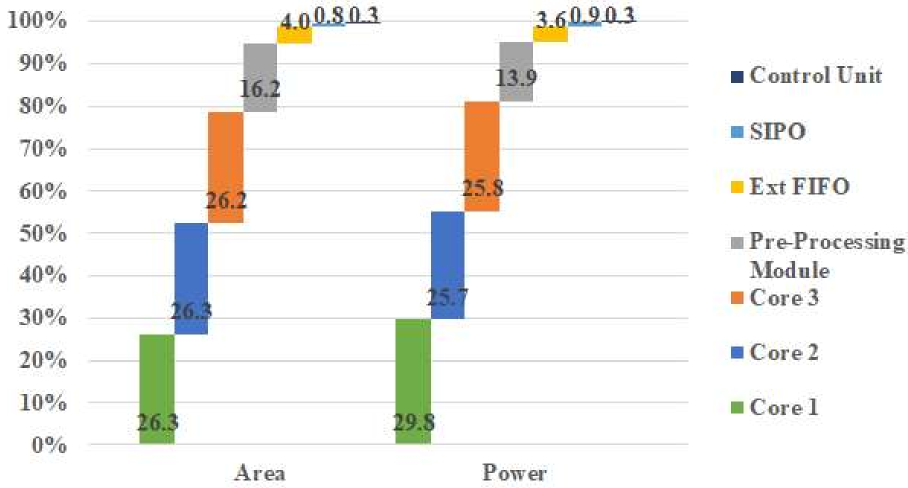


Figure V.14 *Breakdown of the area occupation and the power consumption of the various submodules of the proposed HW accelerator. All values refer to the FIFO-based version for the HBN accelerator.*

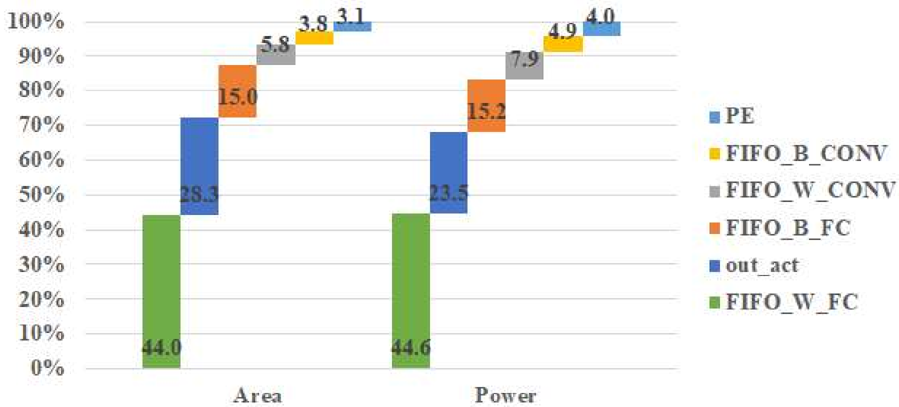


Figure V.15 *Breakdown of the area occupation and the power consumption of the components in a core of the FIFO-based HBN accelerator.*

V.4 FPGA-based demo board

To prove the real-time operation of the proposed HW accelerator, this has been deployed on an Artix-7 FPGA, and a FPGA-based demo board has been realized. The HAR system prototype is shown in Figure V.16. A small Digilent CMOD A7-35T has been used to implement the entire circuitry, while the X-NUCLEO-IKS01A1 (STMicroelectonics, 2015), which mounts the LSM6DSO IMU (STMicroelectronics, 2018), is used as the 3-axis accelerometer. The STM32F411RE microcontroller is used to manage the data transfer between IMU and FPGA and to display the processed results. The internal 12 MHz clock of the FPGA board has been used to synchronize the HW accelerator, while the microcontrollers used their own clock. Thus, an asynchronous handshake protocol has been implemented with the microcontroller to manage the data transfer between the FPGA-board and the sensor. A dedicated CU has been implemented on the FPGA side to manage the signals used to implement the data transfer. Also, the available pins on the FPGA board were not enough to allow the transfer of all data at the same time. Indeed, the proposed design is fed with data input from the 3-axis accelerator, which sums up to 3×16 bits = 48 bits. The output of the HW accelerator consists of 3×16 bits = 48 bits because each core provides a 16-bit output. Also, additional pins are required for the implementation of the handshake protocol, which in turn sum up to the 96 pins required for input and output data. Thus, an input buffer and an output buffer have been designed and implemented with the FPGA. The microcontroller sends the input data by packing it 4-bits at once, which are progressively stored in the input buffer on the FPGA. When the input buffer is full, data starts to be elaborated by the HW accelerator. The results of the data processing are stored in the output buffer, where they are packed 4-bits at once and sent to the microcontroller. Thus, only 8 bits are required for the input and output data.

Measurements on the CMOD board return a maximum current of about 100 mA in both cases, which is increased by the additional components of the board.

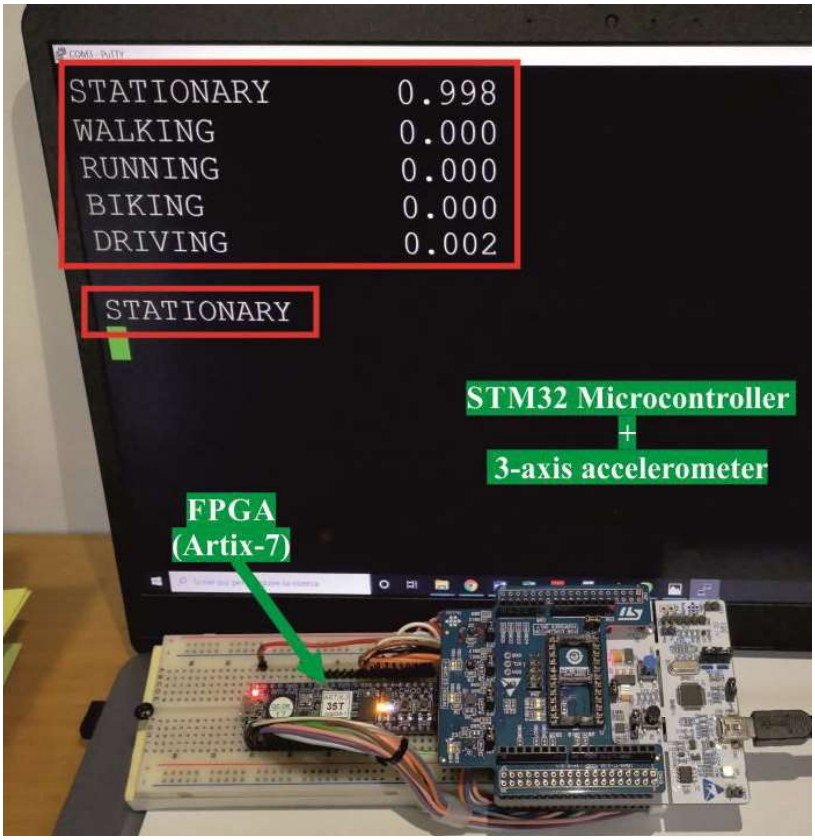


Figure V.16 *FPGA-based demo board. The scores for each one of the 5 classes and the consequent classification are printed to video in real-time.*

Conclusions

In this work, an ultra-low power HAR system has been proposed. The design has been carried out starting from the model to its HW implementation. IS-HAR has been selected as a case study, where the input data comes from inertial sensors.

A custom HW-friendly algorithm has been developed to solve the device-orientation problem in 3-axis accelerometers. The algorithm has been used as a possible pre-processing stage in the proposed system. The proposed solution allows implementing filtering and vector rotation with a lower number of arithmetic operators, avoiding complex trigonometric functions, and reducing the number of normalization. The algorithm has been implemented with Fixed-Point coding to reduce the number of required resources. The word-length has been sized to obtain an optimal tradeoff between the number of bits and precision, resulting in a 24-bit Fixed-Point coding, where the 8 MSBs are the integer part.

Also, a new HBN model has been proposed to achieve the classification of human activities. The HBN exploits the advantages of BNNs, but it brings an improvement in terms of accuracy without affecting the size of the model. Three different configurations have been proposed for the HAR system, based on the types of input sensors and the presence of pre-processing operations. The accuracy of the system has been measured for each configuration, where the HBN has been trained with data from 2 public datasets and 1 custom dataset. The results show an accuracy of up to 99% in classifying 5 human activities. The pre-processing operations bring an advantage in terms of accuracy only when the sensor is located at parts of the body that are subject to relevant rotational movements, such as hand and ankle.

A custom HW accelerator has been designed to implement both the pre-processing operations and the HBN model. A pre-processing module has been designed to implement the pre-processing operations. The architecture can be configured to perform either filtering operations or vector rotation operations. Then, the HBN accelerator has been designed to execute the HBN model. It is made up of 3 cores, each one processing one axis of the sensor individually. Two different versions of the design have been investigated. The first version is the FIFO-based design, where weights and biases are stored in FIFOs that are local to each core. The second version is the RAM-based design, where weights and biases are stored in a shared RAM. The results show that the RAM-based design allows achieving lower power consumption but at the cost of a lower maximum frequency. The proposed design has been both synthesized with CMOS standard cells and implemented with FPGA. The results from synthesis with TSMC CMOS 65 nm LP-HVT standard cells show a power consumption of only 6.3 μW, which is orders of magnitude lower than the custom HW implementations proposed in the literature. Also, the design has been implemented with TSMC CMOS 90 nm GP standard cells, and also in this case the power consumption is up to 12 times lower than state-of-the-art solutions.

The proposed HAR system has been also deployed on FPGA in order to realize a demo board. The demo board has allowed showing the real-time operation of the system.

In conclusion, during the Ph.D. project, the combination of reduced-precision NN models and custom HW design has been widely investigated. The results show that this allows integrating the classification stage in the sensor node, thanks to a low area occupation and power consumption in the order of tens of μW. Also, some pre-processing features can be integrated as well with low impact on the system performance. However, the advantage introduced by the pre-processing operations should be assessed based on the sensor position. Thus, the results from this Ph.D. project can be a starting point for the industrial development of efficient AI-based edge computing devices.

Considering that the maximum frequency of the proposed system is far higher than the operating frequency, future works might aim to extend the proposed design to other applications that require higher throughputs, such as anomaly detection for industrial machines. Even in this case, inertial sensors are used to sample data, but the sampling frequency is in the range of tens of kHz rather than tens of Hz.

References

Abdi, H., Williams, L. J. (2010) Principal component analysis. In *WIREs Comp Stat*, 2, pp. 433-459.
Doi: https://doi.org/10.1002/wics.101

Abobakr, A., Hossny, M., Nahavandi, S. (2018) A Skeleton-Free Fall Detection System From Depth Images Using Random Decision Forest. In *IEEE Systems Journal*, 12, pp. 2994-3005.
Doi: 10.1109/JSYST.2017.2780260

Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J. L. (2012) Human Activity Recognition on Smartphones Using a Multiclass Hardware-Friendly Support Vector Machine. In *Ambient Assisted Living and Home Care*, pp 216-223.
Doi: 10.1007/978-3-642-35395-6_30

Bankman, D., Yang, L., Moons, B., Verhelst, M., Murmann, B. (2018) An always-on 3.8μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28nm CMOS. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pp. 222-224.
Doi: 10.1109/ISSCC.2018.8310264

Banos, O., Galvez, J. M., Damas, M., Pomeras, H., Rojas, I. (2014) Window Size Impact in Human Activity Recognition. In *Sensors*, 14, pp. 6474-6479.
Doi: 10.3390/s140406474

Bao, L., Intille, S. S. (2004) Activity Recognition from User-Annotated Acceleration Data. In *Pervasive Computing*, pp. 1-17.
Doi: 10.1007/978-3-540-24646-6_1

Bengio, Y., Léonard, N., Courville, A. (2013) Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation [Online]. Available at: https://arxiv.org/abs/1308.3432

Bianchi, V., Bassoli, M., Lombardo, G., Fornacciari, P., Mordonini, M., De Munari, I. (2019) IoT Wearable Sensor and Deep Learning: An Integrated Approach for Personalized Human Activity Recognition in a Smart Home Environment, 6, pp. 8553-8562.
Doi: 10.1109/JIOT.2019.2920283

Bisio, I., Delfino, A., Lavagetto, F., Sciarrone, A. (2016) Enabling IoT for In-Home Rehabilitation: Accelerometer Signals Classification Methods for Activity and Movement Recognition. In *IEEE Internet of Things Journal*, 4, pp. 135-146.
Doi: 10.1109/JIOT.2016.2628938

Blanke, U., Schiele, B., Kreil, M., Lukowicz, P., Sick, B., Gruber, T. (2010) All for one or one for all? Combining Heterogeneous Features for Activity Spotting. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pp. 18-24.
Doi: 10.1109/PERCOMW.2010.5470597.

Blott, M., Preußer, T., Fraser, N. J., Gambardella, G., O'brien, K., Umuroglu, Y., Leeser, M., Vissers, K. (2018) FINN-*R*: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. In *ACM Transactions on Reconfigurable Technology and Systems*, 11.
Doi: 10.1145/3242897

Bulling, A., Ward, J. A., Gellersen, H., Troster, G. (2011). Eye Movement Analysis for Activity Recognition Using Electrooculography. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33, pp. 741-754.
Doi: 10.1109/TPAMI.2010.86

Bulling, A., Blanke, U., Schiele, B. (2014) A tutorial on human activity recognition using body-worn inertial sensors. In *ACM Computing Surveys*, 46.
Doi: 10.1145/2499621

Chen, L. L., Zhang, J., Zou, J. Z., Zhao, C. J., Wang, G. S. (2014) A framework on wavelet-based nonlinear features and extreme learning machine for epileptic seizure detection. In *Biomedical Signal Processing and Control*, 10, pp 1-10.
Doi: 10.1016/j.bspc.2013.11.010

Chen, Y., Yang, T., Emer, J., Sze, V. (2019) Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. In *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9, pp. 292-308.
Doi: 10.1109/JETCAS.2019.2910232

Chinimilli, P. T., Redkar, S., Zhang, W. (2017) Human Activity Recognition Using Inertial Measurement Units and Smart Shoes. In *2017 American Control Conference (ACC)*, pp. 1462-1467.
Doi: 10.23919/ACC.2017.7963159.

Cilibero, M., Ordonez Morales, F. J., Gjoreski, H., Roggen, D., Mekki, S., Valentin, S. (2017) High reliability Android application for multidevice multimodal mobile data acquisition and annotation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*.
Doi: 10.1145/3131672.3136977

Cola, G., Avvenuti, M., Vecchio, A. (2017) Real-Time Identification Using Gait Pattern Analysis on a Standalone Wearable Accelerometer. In *The Computer Journal*, 60, pp. 1173-1186.
Doi: 10.1093/comjnl/bxw111

Courbariaux, M., Bengio, Y., David, J. P. (2015) BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS)*, 2, pp. 3123-3131.
Doi: 10.5555/2969442.2969588

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y. (2016) Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or −1 [Online]. Available at: https://arxiv.org/abs/1602.02830

Cybenko, G. (1989) Approximation by superpositions of a sigmoidal function. In *Mathematics of Control, Signals and Systems*, 2, pp. 303-314.
Doi: 10.1007/BF02551274

Dai, J. S. (2015) Euler–Rodrigues formula variations, quaternion conjugation and intrinsic connections. In *Mechanism and Machine Theory*, 92, pp. 144-152.
Doi: 10.1016/j.mechmachtheory.2015.03.004

David, J. P., Kalach, K., Tittley, N. (2007) Hardware Complexity of Modular Multiplication and Exponentiation. In *IEEE Transactions on Computers*, 56, pp. 1308-1319.
Doi: 10.1109/TC.2007.1084

De, P., Chatterjee, A., Rakshit, A. (2018) Recognition of Human Behavior for Assisted Living Using Dictionary Learning Approach. In *IEEE Sensors Journal*, 18, pp. 2434-2441.
Doi: 10.1109/JSEN.2017.2787616

Deng, J., Dong, W., Socher, R., Li, L. J., Kai, L., Fei-Fei, L. (2009) ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248-255.
Doi: 10.1109/CVPR.2009.5206848.

Dinarević, E. C., Husić, J. B., Baraković, S. (2019) Step by Step Towards Effective Human Activity Recognition: A Balance between Energy Consumption and Latency in Health and Wellbeing Applications. In *Sensors*, 19, pp. 5206-5233.
Doi: 10.3390/s19235206

Emokpae, L. E., Emokpae, R. N., Emokpae, B. (2018) Flex Force Smart Glove Prototype for Physical Therapy Rehabilitation. In *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1-4.
Doi: 10.1109/BIOCAS.2018.8584774

Eskofier *et al.* (2016) Recent machine learning advancements in sensor-based mobility analysis: Deep learning for Parkinson's disease assessment. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 655-658.
Doi: 10.1109/EMBC.2016.7590787

Finkelstein, A., Almog, U., Grobman, M. (2019) Fighting Quantization Bias With Bias [Online]. Available at: https://arxiv.org/abs/1906.03193

Florentino-Liaño, V., O'Mahony, N., Artés-Rodríguez, A. (2012) Human activity recognition using inertial sensors with invariance to sensor orientation. In *2012 3rd International Workshop on Cognitive Information Processing (CIP)*, pp. 1-6.
Doi: 10.1109/CIP.2012.6232914

Fraser, N. J., Umuroglu, Y., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K. (2017) Scaling Binarized Neural Networks on Reconfigurable Logic. In *Proceedings of the 8th Workshop and 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, pp. 25-30.
Doi: 10.1145/3029580.3029586

Gaikwad, N. B., Tiwari, V., Keskar, A., Shivaprakash, N. C. (2019) Efficient FPGA Implementation of Multilayer Perceptron for Real-Time Human Activity Classification. In *IEEE Access*, 7, pp. 26696-26706.
Doi: 10.1109/ACCESS.2019.2900084

Gao, X., Luo, H., Wang, Q., Zhao, F., Ye, L., Zhang, Y. (2019) A Human Activity Recognition Algorithm Based on Stacking Denoising Autoencoder and LightGBM. In *Sensors*, 19, p. 947.
Doi: 10.3390/s19040947

Ghasemzadeh, M., Samragh, M., Koushanfar, F. (2018) ReBNet: Residual Binarized Neural Network. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 57-64.
Doi: 10.1109/FCCM.2018.00018

Ginting, A., Wahyunggoro, O. (2018) Attitude Control of Quadrotor Using PD Plus Feedforward controller on SO(3). In *International Journal of Electrical and Computer Engineering*, 8, pp. 566-575.
Doi: 10.11591/ijece.v8i1.pp566-575

Grigorescu, S., Trasnea, B., Cocias, T., Macesanu, G. (2019) A survey of deep learning techniques for autonomous driving. In *Journal of Field Robotics*, 37, pp. 362-386.
Doi: 10.1002/rob.21918

Guo, P., Ma, H., Chen, R., Li, P., Xie, S., Wang, D. (2018) FBNA: A Fully Binarized Neural Network Accelerator. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 51-513.
Doi: 10.1109/FPL.2018.00016

Hanai, Y., Hori, Y., Nishimura, J., Kuroda, T. (2009) A versatile recognition processor employing Haar-like feature and cascaded classifier. In *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pp. 148-149,149a.
Doi: 10.1109/ISSCC.2009.4977351

He, K., Zhang, X., Ren, S., Sun, J. (2016) Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778.
Doi: 10.1109/CVPR.2016.90

Hinton *et al.*(2012) Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. In *IEEE Signal Processing Magazine*, 29, pp. 82-97.
Doi: 10.1109/MSP.2012.2205597

Horowitz, M. (2014) Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*.
Doi: 10.1109/ISSCC.2014.6757323

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y. (2016) Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS)*, pp. 4114–4122.
Doi: 10.5555/3157382.3157557

IEEE (2019) IEEE Standard for Floating-Point Arithmetic. In *IEEE Std 754-2019 (Revision of IEEE 754-2008)*.
Doi: 10.1109/IEEESTD.2019.8766229.

Ioffe, S., Szegedy, C. (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, 37, pp. 448-456 [Online]. Available at: http://proceedings.mlr.press/v37/ioffe15.html

Jafari, A., Ganesan, A., Thalisetty, C. S. K., Sivasubramanian, V., Oates, T., Mohsenin, T. (2019) SensorNet: A Scalable and Low-Power Deep Convolutional Neural Network for Multimodal Data Classification. In *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66, pp. 274-287.
Doi: 10.1109/TCSI.2018.2848647

Janota, A., Šimák, V., Nemec, D., Hrbček, J. (2015) Improving the Precision and Speed of Euler Angles Computation from Low-Cost Rotation Sensor Data. In *Sensors*, 15, pp. 7016-7039.
Doi: 10.3390/s150307016

Jiang, W., Yin, W. (2015) Human Activity Recognition Using Wearable Sensors by Deep Convolutional Neural Networks. In *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 1307-1310.
Doi: 10.1145/2733373.2806333

Jimenez, A. R., Seco, F. (2018) Multi-Event Naive Bayes Classifier for Activity Recognition in the UCAmI Cup. In *Proceedings*, 2, p. 1264.
Doi: 10.3390/proceedings2191264

Jahn, A., Bachmann, M., Wenzel, P., David, K. (2017) Focus on the User: A User Relative Coordinate System for Activity Detection. In *Modeling and Using Context. CONTEXT 2017. Lecture Notes in Computer Science,* 10257, pp 582-595.
Doi: 10.1007/978-3-319-57837-8_47

Kang, W. J., Shiu, J. R., Cheng, C. K., Lai, J. S., Tsao, H. W., Kuo, T. S. (1995) The Application of Cepstral Coefficients and Maximum Likelihood Method in EMG Pattern Recognition. In *IEEE Transactions on Biomedical Engineering*, 42, pp. 777-785.
Doi: 10.1109/10.398638

Kingma, D. P., Ba, J. (2015) Adam: A Method for Stochastic Optimization [Online]. Available at: https://arxiv.org/abs/1412.6980

Kodali, S., Hansen, P., Mulholland, N., Whatmough, P., Brooks, D., Wei, G. Y. (2017) Applications of Deep Neural Networks for Ultra Low Power IoT. In *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 589-592.
Doi: 10.1109/ICCD.2017.102.

Kok, M., Hol, J. D., Schön, T. B. (2017) Using Inertial Sensors for Position and Orientation Estimation.
Doi: 10.1561/2000000094.

Krizhevsky, A. (2009) Learning Multiple Layers of Features from Tiny Images [Online]. Available at: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012) ImageNet classification with deep convolutional neural network. In *Neural Information and Processing Systems (NIPS)* [online]. Available at:
https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Kunze, K., Barry, M., Heinz, E. A., Lukowicz, P., Majoe, D., Gutknecht, J. (2006) Towards Recognizing Tai Chi - An Initial Experiment Using Wearable Sensors. In *3rd International Forum on Applied Wearable Computing 2006*, pp. 1-6 [Online]. Available at:
https://ieeexplore.ieee.org/document/5758288

Lasagne (2018) lasagne Documentation [Online]. Available at:
https://lasagne.readthedocs.io/en/latest/user/tutorial.html

LeCun, Y., Bengio, Y. and Hinton, G. (2015) Deep Learning. In *Nature*, 521, pp. 436-444.
Doi: 10.1038/nature14539

Lester, J., Choudhury, T., Kern, N., Borriello, G. (2005) A Hybrid Discriminative/Generative Approach for Modeling Human Activities. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pp. 766-772 [Online]. Available at:
https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.444.5829&rep=rep1&type=pdf

Li, D., Zhao, D., Zhang, Q., Chen, Y. (2019) Reinforcement Learning and Deep Learning Based Lateral Control for Autonomous Driving [Application Notes]. In *IEEE Computational Intelligence Magazine*, 14, pp. 83-98.
Doi: 10.1109/MCI.2019.2901089

Li, Y., Liu, Z., Liu, W., Jiang, Y., Goh, W. L., Yu, H., Ren. F. (2019) A 34-FPS 698-GOP/s/W Binarized Deep Neural Network-Based Natural Scene Text Interpretation Accelerator for Mobile Edge Computing. In *IEEE Transactions on Industrial Electronics*, 66, pp. 7407-7416.
Doi: 10.1109/TIE.2018.2875643

Lin, X., Zhao, C., Pan, W. (2017) Towards Accurate Binary Convolutional Neural Network. In *Advances in Neural Information Processing Systems*, 30, pp. 345-353 [Online]. Available at:
https://papers.nips.cc/paper/2017/hash/b1a59b315fc9a3002ce38bbe070ec3f5-Abstract.html

McCarthy, J. (2007) What is Artificial Intelligence? [Online]. Available at: http://jmc.stanford.edu/articles/whatisai.html

Mitra, S. K. (2000) *Digital Signal Processing: A Computer Based Approach*. 2nd edn. New York: Mc-Graw Hill Education.

Mizell, D. (2003) Using Gravity to Estimate Accelerometer Orientation. In *Seventh IEEE International Symposium on Wearable Computers, 2003. Proceedings*, pp. 252-253.
Doi: 10.1109/ISWC.2003.1241424

Nakahara, H., Yonekawa, H., Sasao, T., Iwamoto, H., Motomura, M. (2016) A memory-based realization of a binarized deep convolutional neural network. In *2016 International Conference on Field-Programmable Technology (FPT)*, pp. 277-280.
Doi: 10.1109/FPT.2016.7929552

Nakahara, H., Fujii, T., Sato, S. (2017) A fully connected layer elimination for a binarizec convolutional neural network on an FPGA. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1-4.
Doi: 10.23919/FPL.2017.8056771

Nassif, A. B., Shanin, I., Attili, I. (2019) Speech Recognition Using Deep Neural Networks: A Systematic Review. In *IEEE Access*, 7, pp. 19143-19165.
Doi: 10.1109/ACCESS.2019.2896880

Nicosia, A., Pau, D., Giacolone, D., Plebani, E., Bosco, A., Iacchetti, A. (2018) Efficient light harvesting for accurate neural classification of human activities. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1-4.
Doi: 10.1109/ICCE.2018.8326103.

Normani, N., Urru, A., Abraham, L., Walsh, M., Tedesco, S., Cenedese, A., Susto, G. A., O'Flynn, B. (2018) A Machine Learning Approach for Gesture Recognition with a Lensless Smart Sensor System. In *2018 IEEE 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pp. 136-139.
Doi: 10.1109/BSN.2018.8329677

Ordóñez, F. J., Roggen, D. (2016) Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. In *Sensors*, 16, p. 115.
Doi: 10.3390/s16010115

Qin, H., Gong, R., Liu, X., Shen, M., Wei, Z., Yu, F., Song, F. (2020) Forward and Backward Information Retention for Accurate Binary Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2250-2259 [Online]. Available at: https://arxiv.org/abs/1909.10788

Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A. (2016) XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Computer Vision – ECCV 2016*.
Doi: 10.1007/978-3-319-46493-0_32

Rault, T., Bouadballah, A., Challal, Y., Marin, F. (2017) A survey of energy-efficient context recognition systems using wearable sensors for healthcare applications. In *Pervasive and Mobile Computing*, 37.
Doi: 10.1016/j.pmcj.2016.08.003

Ravi, N., Dandekar, N., Mysore, P., Littman, M. L. (2007) Activity Recognition from Accelerometer Data. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pp. 1541-1546 [Online]. Available at: https://www.aaai.org/Papers/IAAI/2005/IAAI05-013.pdf

Reiss, A., Stricker, D. (2012) Introducing a New Benchmarked Dataset for Activity Monitoring. In *2012 16th International Symposium on Wearable Computers*, pp. 108-109.
Doi: 10.1109/ISWC.2012.13

Richards, R. K. (1956) *Arithmetic operations in digital computers*. 4th pr. Princeton: D. Van Nostrand Company Inc.

Simons, T., Lee, D. J. (2019) A Review of Binarized Neural Networks. In *Electronics*, 8, p. 661.
Doi: 10.3390/electronics8060661

Simonyan, K., Zisserman, A. (2014) Very Deep Convolutional Networks for Large-Scale Image Recognition [Online]. Available at: https://arxiv.org/abs/1409.1556

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

STMicroelectronics (2015) X-NUCLEO-IKS01A1 Motion MEMS and environmental sensor expansion board for STM32 Nucleo [Online]. Available at: https://www.st.com/resource/en/datasheet/x-nucleo-iks01a1.pdf

STMicroelectronics (2018) iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope [Online]. Available at: https://www.st.com/resource/en/datasheet/lsm6dsm.pdf

Sze, V., Chen, Y. H., Yang, T. J., Emer, J. S. (2017) Efficient Processing of Deep Neural Networks: A Tutorial and Survey. In *Proceedings of the IEEE*, 105, pp. 2295-2329.
Doi: 10.1109/JPROC.2017.2761740

Tompson, J., Jain, A., LeCun, Y., Bregler, C. (2014) Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation. In *Neural Information and Processing Systems (NIPS)* [online]. Available at: https://arxiv.org/abs/1406.2984

Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K. (2017) FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65-74.
Doi: 10.1145/3020078.3021744

Ustev, Y. E., Incel, O. D., Ersoy, C. (2013) User, device and orientation independent human activity recognition on mobile phones: challenges and a proposal. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing*, pp. 1427-1436.
Doi: 10.1145/2494091.2496039

Vaidyanathan, P., Mitra S. K., Neuvo, Y. (1986) A new approach to the realization of low-sensitivity IIR digital filters. *In IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34, pp. 350-361.
Doi: 10.1109/TASSP.1986.1164829

Valavi, H., Ramadge, P. J., Nestler, E., Verma, N. (2018) A Mixed-Signal Binarized Convolutional-Neural-Network Accelerator Integrating Dense Weight Storage and Multiplication for Reduced Data Movement. In *2018 IEEE Symposium on VLSI Circuits*, pp. 141-142.
Doi: 10.1109/VLSIC.2018.8502421

VanKasteren, T., Noulas, A., Englebienne, G., Krose, B. (2008) Accurate activity recognition in a home setting. In *Proceedings of the 10th international conference on Ubiquitous computing*, pp. 1-9.
Doi: 10.1145/1409635.1409637

Wu, Z., Sun, Z., Zhang, W., Chen, Q. (2016) A Novel Approach for Attitude Estimation Based on MEMS Inertial Sensors Using Nonlinear Complementary Filters. In *IEEE Sensors Journal*, 16, pp. 3856-3864.
Doi: 10.1109/JSEN.2016.2532909

Xian, Y., Rong, X., Yang, X., Tian, Y. (2017) Evaluation of Low-Level Features for Real-World Surveillance Event Detection. In *IEEE Transactions on Circuits and Systems for Video Technology*, 27, pp. 624-634.
Doi: 10.1109/TCSVT.2016.2589838

Xilinx (2020) 7 Series FPGAs Data Sheet: Overview [Online]. Available at:
https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

Yang, L., He, Z., Fan, D. (2018) A Fully Onchip Binarized Convolutional Neural Network FPGA Impelmentation with Accurate Inference. In *Proceedings of the International Symposium on Low Power Electronics and Design*.
Doi: 10.1145/3218603.3218615

Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., Hua, X. (2019) Quantization Networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7300-7308.
Doi: 10.1109/CVPR.2019.00748

Yin, S., Ouyang, P., Zheng, S., Song, D., Li, X., Liu, L., Wei, S. (2018) A 141 UW, 2.46 PJ/Neuron Binarized Convolutional Neural Network Based Self-Learning Speech Recognition Processor in 28NM CMOS. In *2018 IEEE Symposium on VLSI Circuits*, pp. 139-140.
Doi: 10.1109/VLSIC.2018.8502309

Yu, H., Cang, S., Wang, Y. (2016) A review of sensor selection, sensor devices and sensor deployment for wearable sensor-based human activity recognition systems. In *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*, pp. 250-257.
Doi: 10.1109/SKIMA.2016.7916228

Zhang, L., Wu, X., Luo, D. (2015) Recognizing Human Activities from Raw Accelerometer Data Using Deep Neural Networks. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 865-870.
Doi: 10.1109/ICMLA.2015.48

Zhang, D., Yang, J., Ye, D., Hua, G. (2018) LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks [Online]. Available at: https://arxiv.org/abs/1807.10029

Zhao, B., Lu, H., Chen, S., Liu, J., Wu, D. (2017) Convolutional neural networks for time series classification. In *Journal of Systems Engineering and Electronics*, 28, pp. 162-169.
Doi: 10.21629/JSEE.2017.01.18

Zhou, Y., Redkar, S., Huang, X. (2017) Deep learning binary neural network on an FPGA. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 281-284.
Doi: 10.1109/MWSCAS.2017.8052915

Zinnen, A., Blanke, U., Schiele, B. (2009) An Analysis of Sensor-Oriented vs. Model-Based Activity Recognition. In *2009 International Symposium on Wearable Computers*, pp. 93-100.
Doi: 10.1109/ISWC.2009.32.

Appendix A
Confusion Matrixes for the HBN

In this section, the confusion matrixes of the HBN model for all the results presented in paragraphs IV.3.2 and IV.3.3 are detailed. The results have been obtained with *k*-fold cross-validation, with *k* = 5. Thus, for each combination of configuration and position, all 5 confusion matrixed are reported.

Confusion matrixes for 5 classes on the PAMAP2 dataset

In this paragraph, the confusion matrixes obtained when testing the HBN model to classify 5 activities for the PAMAP2 dataset are reported. In the following, the list of the human activities used in this paragraph is specified:

1. stationary
2. walking
3. running
4. cycling
5. rope jumping

Conf 1 - 3D accelerometer (with pre-processing)

Position: ankle16g

Actual	Predicted class				
class	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.840	0.000	0.000	0.160
3	0.000	0.000	95.067	0.000	4.933
4	0.000	0.000	0.000	99.515	0.485
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.884%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	91.852	0.000	8.148
4	0.516	0.065	0.065	98.839	0.516
5	0.000	0.138	0.069	0.000	99.793

Average Recall: 98.097%

Actual class	Predicted class 1	2	3	4	5
1	99.931	0.000	0.000	0.069	0.000
2	0.000	99.812	0.000	0.000	0.188
3	0.000	0.000	97.357	0.000	2.643
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.420%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.923	0.000	0.077	0.000
3	0.000	0.000	90.960	0.000	9.040
4	0.000	0.000	0.000	98.160	1.840
5	0.000	0.207	0.138	0.000	99.655

Average Recall: 97.740%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	89.412	0.000	10.588
4	0.000	0.000	0.000	98.320	1.680
5	0.000	0.312	0.000	0.000	99.688

Average Recall: 97.484%

Mean Average Recall: 98.325%
Standard Deviation: 0.808

Position: ankle6g

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	1.130	0.000	98.609	0.261
5	0.000	0.000	0.067	0.000	99.933

Average Recall: 99.708%

Actual class	Predicted class 1	2	3	4	5
1	99.951	0.000	0.000	0.049	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	99.923	0.000	0.077
4	0.065	0.000	0.000	98.452	1.484
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.665%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.929	0.000	0.000	0.071
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.188	0.000	99.812

Average Recall: 99.948%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.926	0.000	0.000	0.074
3	0.000	0.000	99.652	0.000	0.348
4	0.059	0.353	0.000	97.000	2.588
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.316%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.655	0.000	0.345	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.067	0.467	0.000	99.467	0.000
5	0.000	0.125	1.125	0.062	98.688

Average Recall: 99.562%

Mean Average Recall: 99.640%
Standard Deviation: 0.230

Position: hand16g

Actual class	Predicted class 1	2	3	4	5
1	99.684	0.316	0.000	0.000	0.000
2	0.000	98.421	0.000	1.579	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.176	0.471	0.000	99.353	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.492%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.357	99.214	0.000	0.286	0.143
3	0.000	0.000	100.000	0.000	0.000
4	0.095	1.429	0.000	98.476	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.538%

Actual class	Predicted class				
	1	2	3	4	5
1	98.645	1.355	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	99.833	0.000	0.167
4	0.000	0.897	0.000	99.103	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.516%

Actual class	Predicted class				
	1	2	3	4	5
1	99.938	0.062	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.500	0.562	0.000	98.938	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.775%

Actual class	Predicted class				
	1	2	3	4	5
1	99.481	0.519	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.857	0.857	0.000	98.286	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.553%

Mean Average Recall: 99.575%
Standard Deviation: 0.114

Position: hand6g

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.080	99.920	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.324	0.486	0.000	99.189	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.822%

Actual class	Predicted class				
	1	2	3	4	5
1	98.485	1.394	0.000	0.121	0.000
2	0.414	98.069	0.000	1.517	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.250	0.750	0.000	99.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.111%

Actual class	Predicted class				
	1	2	3	4	5
1	99.714	0.286	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	99.862	0.000	0.138
4	0.000	0.722	0.000	99.278	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.771%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.421	99.579	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	2.190	0.286	0.000	97.524	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.421%

Actual class	Predicted class				
	1	2	3	4	5
1	99.294	0.706	0.000	0.000	0.000
2	0.000	99.920	0.000	0.080	0.000
3	0.000	0.000	100.000	0.000	0.000
4	1.154	0.615	0.000	98.231	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.489%

Mean Average Recall: 99.522%
Standard Deviation: 0.288

Position: chest16g

Actual class	Predicted class				
	1	2	3	4	5
1	98.667	0.000	0.000	1.333	0.000
2	0.000	95.655	0.000	4.345	0.000
3	0.000	0.000	99.929	0.000	0.071
4	0.083	0.000	0.000	99.917	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.834%

Actual class	Predicted class				
	1	2	3	4	5
1	99.545	0.000	0.000	0.455	0.000
2	0.000	96.000	0.000	4.000	0.000
3	0.000	0.000	98.667	0.000	1.333
4	5.704	6.889	0.000	87.407	0.000
5	0.000	0.065	0.452	0.000	99.484

Average Recall: 96.221%

Actual class	Predicted class				
	1	2	3	4	5
1	99.407	0.000	0.000	0.593	0.000
2	0.000	98.207	0.000	1.793	0.000
3	0.000	0.000	93.826	0.000	6.174
4	0.647	0.176	0.000	99.176	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.123%

Actual class	Predicted class				
	1	2	3	4	5
1	99.576	0.000	0.000	0.424	0.000
2	0.000	87.724	0.000	12.276	0.000
3	0.000	0.000	99.739	0.000	0.261
4	0.914	0.000	0.000	99.086	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 97.225%

Actual class	Predicted class				
	1	2	3	4	5
1	99.154	0.000	0.000	0.846	0.000
2	0.000	89.862	0.000	10.138	0.000
3	0.000	0.000	83.471	0.000	16.529
4	0.167	0.000	0.000	99.833	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 94.464%

Mean Average Recall: 96.973%
Standard Deviation: 1.711

Position: chest6g

Actual class	Predicted class				
	1	2	3	4	5
1	98.968	0.000	0.000	1.032	0.000
2	0.000	99.000	0.000	1.000	0.000
3	0.000	0.000	99.517	0.000	0.483
4	0.000	0.069	0.000	99.931	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.483%

Actual class	Predicted class 1	2	3	4	5
1	98.000	0.000	0.000	2.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.516	0.000	99.484

Average Recall: 99.497%

Actual class	Predicted class 1	2	3	4	5
1	98.971	0.000	0.000	1.029	0.000
2	0.000	85.833	0.000	14.167	0.000
3	0.000	0.000	96.516	0.000	3.484
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 96.264%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	87.625	0.000	11.562	0.812
3	0.000	0.000	97.231	0.000	2.769
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.057	99.943

Average Recall: 96.960%

Actual class	Predicted class 1	2	3	4	5
1	99.037	0.000	0.000	0.963	0.000
2	0.000	96.690	0.000	3.310	0.000
3	0.000	0.000	100.000	0.000	0.000
4	2.615	0.000	0.000	97.385	0.000
5	0.000	0.000	0.500	0.000	99.500

Average Recall: 98.522%

Mean Average Recall: 98.145%
Standard Deviation: 1.475

Conf 2 - 3D accelerometer (no preprocessing)

Position: ankle16g

Actual class	Predicted class 1	2	3	4	5
1	99.800	0.000	0.000	0.200	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.364	2.182	0.000	97.455	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.451%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.938	0.000	0.062	0.000
3	0.000	0.000	95.143	0.000	4.857
4	0.000	3.760	0.000	94.240	2.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 97.864%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.935	0.000	0.065	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.485	20.485	0.000	76.970	2.061
5	0.000	0.323	1.161	0.000	98.516

Average Recall: 95.084%

Actual class	Predicted class				
	1	2	3	4	5
1	95.625	0.000	0.000	4.375	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	99.576	0.000	0.424
4	0.276	3.172	0.000	95.241	1.310
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.088%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	4.686	2.457	0.000	92.000	0.857
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.400%

Mean Average Recall: 97.778%
Standard Deviation: 1.624

Position: ankle6g

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.459	0.000	0.000	0.541
3	0.000	0.000	100.000	0.000	0.000
4	2.286	0.000	0.000	97.000	0.714
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.292%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	95.455	0.000	4.545
4	0.000	4.412	0.000	95.059	0.529
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.103%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	99.846	0.154
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.969%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.697	0.000	0.303	0.000
3	0.000	0.000	100.000	0.000	0.000
4	3.543	0.457	0.000	94.229	1.771
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.785%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	99.939	0.000	0.061
4	1.143	16.667	0.000	80.762	1.429
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 96.140%

Mean Average Recall: 98.458%
Standard Deviation: 1.465

Position: hand16g

Actual class	Predicted class				
	1	2	3	4	5
1	98.182	1.091	0.000	0.727	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.769	0.000	99.231	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.483%

Actual class	Predicted class				
	1	2	3	4	5
1	98.560	1.440	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	8.160	1.840	0.000	90.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 97.712%

Actual class	Predicted class				
	1	2	3	4	5
1	86.071	2.000	0.000	11.929	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	1.400	0.600	0.000	98.000	0.000
5	0.000	0.000	1.037	0.000	98.963

Average Recall: 96.607%

Actual class	Predicted class				
	1	2	3	4	5
1	98.364	0.273	0.000	1.364	0.000
2	0.148	99.037	0.000	0.667	0.148
3	0.000	0.000	100.000	0.000	0.000
4	0.688	0.062	0.000	99.250	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.330%

Actual class	Predicted class				
	1	2	3	4	5
1	97.667	2.222	0.000	0.111	0.000
2	14.880	85.120	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	1.806	0.000	0.000	98.194	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 96.196%

Mean Average Recall: 97.866%
Standard Deviation: 1.513

Position: hand6g

Actual class	Predicted class				
	1	2	3	4	5
1	96.733	3.267	0.000	0.000	0.000
2	0.080	99.920	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	1.652	0.783	0.000	97.565	0.000
5	0.000	0.000	3.500	0.000	96.500

Average Recall: 98.144%

Actual class	Predicted class				
	1	2	3	4	5
1	99.929	0.071	0.000	0.000	0.000
2	0.061	99.939	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	2.846	0.000	0.000	97.154	0.000
5	0.000	0.000	0.778	0.000	99.222

Average Recall: 99.249%

Actual class	Predicted class				
	1	2	3	4	5
1	98.741	1.259	0.000	0.000	0.000
2	1.143	98.857	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	2.500	0.312	0.000	97.188	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.957%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.538	99.462	0.000	0.000	0.000
3	0.000	0.000	99.926	0.000	0.074
4	3.189	0.649	0.000	96.162	0.000
5	0.000	0.000	1.538	0.000	98.462

Average Recall: 98.802%

Actual class	Predicted class				
	1	2	3	4	5
1	99.290	0.000	0.000	0.710	0.000
2	0.000	99.920	0.000	0.000	0.080
3	0.000	0.000	100.000	0.000	0.000
4	5.385	0.000	0.000	94.615	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.765%

Mean Average Recall: 98.783%
Standard Deviation: 0.405

Position: chest16g

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	95.655	0.000	4.345	0.000
3	0.000	0.000	100.000	0.000	0.000
4	3.333	0.000	0.000	96.667	0.000
5	0.000	0.000	0.519	0.000	99.481

Average Recall: 98.361%

Actual class	Predicted class 1	2	3	4	5
1	97.636	0.000	0.000	2.364	0.000
2	1.071	98.929	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.313%

Actual class	Predicted class 1	2	3	4	5
1	97.630	0.000	0.000	2.370	0.000
2	1.241	79.931	0.000	18.828	0.000
3	0.000	0.000	100.000	0.000	0.000
4	2.588	0.000	0.000	97.412	0.000
5	0.000	0.000	0.065	0.000	99.935

Average Recall: 94.982%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.500	0.000	99.500

Average Recall: 99.900%

Actual class	Predicted class 1	2	3	4	5
1	98.154	0.000	0.000	1.846	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.452	0.000	99.548

Average Recall: 99.540%

Mean Average Recall: 98.419%
Standard Deviation: 2.004

Position: chest6g

Actual class	Predicted class 1	2	3	4	5
1	99.548	0.000	0.000	0.452	0.000
2	0.000	92.400	0.000	7.600	0.000
3	0.000	0.000	100.000	0.000	0.000
4	13.034	0.000	0.000	86.966	0.000
5	0.000	0.400	0.160	0.080	99.360

Average Recall: 95.655%

Actual	Predicted class				
class	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	91.655	0.000	8.345	0.000
3	0.000	0.000	100.000	0.000	0.000
4	1.879	0.000	0.000	98.121	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 97.955%

Actual	Predicted class				
class	1	2	3	4	5
1	99.771	0.000	0.000	0.229	0.000
2	0.000	99.833	0.000	0.167	0.000
3	0.000	0.000	98.258	0.000	1.742
4	0.759	0.000	0.000	99.241	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.421%

Actual	Predicted class				
class	1	2	3	4	5
1	99.250	0.000	0.000	0.750	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	97.923	0.000	2.077
4	1.630	0.074	0.000	98.296	0.000
5	0.000	0.000	0.629	0.000	99.371

Average Recall: 98.968%

Actual	Predicted class				
class	1	2	3	4	5
1	98.667	0.000	0.000	1.333	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.077	0.000	0.000	99.923	0.000
5	0.000	0.000	0.571	0.000	99.429

Average Recall: 99.604%

Mean Average Recall: 98.320%
Standard Deviation: 1.621

Conf 3 - 3D accelerometer + 3D gyroscope

Position: ankle16g

Actual	Predicted class				
class	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	98.963	1.037
5	0.000	0.000	1.724	0.000	98.276

Average Recall: 99.448%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	5.111	0.000	94.889

Average Recall: 98.978%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 100.000%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	2.667	0.000	97.333

Average Recall: 99.467%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.867	0.000	0.000	99.133	0.000
5	0.000	0.000	0.444	0.000	99.556

Average Recall: 99.738%

Mean Average Recall: 99.526%
Standard Deviation: 0.381

Position: ankle6g

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.914	0.000	99.086	0.000
5	0.000	0.000	1.333	0.000	98.667

Average Recall: 99.551%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	1.034	0.000	98.966

Average Recall: 99.793%

Actual class	Predicted class				
	1	2	3	4	5
1	99.688	0.000	0.000	0.312	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.148	0.000	0.000	99.852	0.000
5	0.000	0.000	2.000	0.000	98.000

Average Recall: 99.508%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	2.769	0.000	97.231

Average Recall: 99.4462%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.143	0.000	0.000	99.000	0.857
5	0.000	0.000	1.778	0.000	98.222

Average Recall: 99.444%

Mean Average Recall: 99.548%
Standard Deviation: 0.144

Position: hand16g

Actual class	Predicted class				
	1	2	3	4	5
1	99.931	0.000	0.000	0.069	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.986%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 100.000%

Actual class	Predicted class				
	1	2	3	4	5
1	99.857	0.000	0.000	0.143	0.000
2	0.071	98.857	0.000	1.071	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.897	0.000	0.000	99.103	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.563%

Actual class	Predicted class				
	1	2	3	4	5
1	97.714	0.000	0.000	2.286	0.000
2	0.061	99.818	0.000	0.121	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.506%

Actual class	Predicted class				
	1	2	3	4	5
1	99.926	0.000	0.000	0.074	0.000
2	0.483	99.517	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.182	0.000	0.000	99.818	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.852%

Mean Average Recall: 99.782%
Standard Deviation: 0.233

Position: hand6g

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.897	0.000	0.000	99.103	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.821%

Actual class	Predicted class 1	2	3	4	5
1	99.533	0.000	0.000	0.467	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	1.657	0.000	0.000	98.343	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.575%

Actual class	Predicted class 1	2	3	4	5
1	98.875	0.000	0.000	1.125	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.775%

Actual class	Predicted class 1	2	3	4	5
1	99.714	0.057	0.000	0.229	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	1.083	0.000	0.000	98.917	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.726%

Actual class	Predicted class 1	2	3	4	5
1	99.429	0.171	0.000	0.400	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.886%

Mean Average Recall: 99.757%
Standard Deviation: 0.117

Position: chest16g

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 100.000%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.276	0.000	0.000	99.724	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.945%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 100.000%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 100.000%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.590	0.000	0.410	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.918%

Mean Average Recall: 99.973%
Standard Deviation: 0.039

Position: chest6g

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.733	0.000	0.000	99.267	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.853%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.188	0.000	0.000	99.812	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.962%

Actual class	Predicted class 1	2	3	4	5
1	99.714	0.000	0.000	0.286	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.943%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.214	99.786	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.957%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	100.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.000	0.000	100.000	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 100.000%

Mean Average Recall: 99.943%
Standard Deviation: 0.054

Confusion matrixes for 12 classes on the PAMAP2 dataset

In this paragraph, the confusion matrixes obtained when testing the HBN model to classify all 12 standard activities for the PAMAP2 dataset are reported. In the following, the list of the human activities used in this paragraph is specified:

1. lying
2. sitting
3. standing
4. walking
5. running
6. cycling
7. Nordic walking

8. ascending stairs
9. descending stairs
10. vacuum cleaning
11. ironing
12. rope jumping

Conf 2 - 3D accelerometer (no pre-processing)

Position: ankle16g

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	99.805	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.195	0.000	0.000
2	2.222	32.346	62.716	0.000	0.000	0.000	0.000	0.000	0.000	2.716	0.000	0.000
3	0.000	10.215	81.900	0.179	0.000	0.179	0.000	0.000	0.000	7.527	0.000	0.000
4	0.000	0.000	0.166	35.655	4.478	7.463	27.197	4.809	11.774	1.161	0.000	7.297
5	0.000	0.000	0.000	0.188	88.701	0.188	0.377	0.000	7.910	0.000	0.000	2.637
6	0.000	0.000	3.457	4.444	2.222	56.790	5.679	1.728	5.185	17.037	0.000	3.457
7	0.000	0.000	0.188	35.782	7.156	4.143	32.768	2.072	12.618	1.130	0.000	4.143
8	0.000	0.176	0.529	11.464	5.820	23.810	8.995	9.171	14.991	6.526	0.000	18.519
9	0.000	0.000	0.546	5.647	16.393	1.457	8.015	1.821	51.913	2.550	0.000	11.658
10	0.404	2.424	16.364	2.828	0.000	6.465	0.808	1.616	0.404	67.677	0.000	1.010
11	0.000	1.613	78.853	0.000	0.000	0.358	0.000	0.000	0.000	19.176	0.000	0.000
12	0.000	0.000	0.000	3.145	3.145	0.210	0.629	1.048	7.757	1.887	0.000	82.180

Average Recall: 54.310%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	99.790	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.210
2	4.520	23.917	56.497	0.000	0.000	8.851	0.188	0.000	0.000	6.026	0.000	0.000
3	0.000	5.461	78.343	0.000	0.565	4.331	0.000	0.000	0.000	11.299	0.000	0.000
4	0.000	0.000	0.000	76.329	0.483	1.691	17.874	0.000	2.174	0.966	0.000	0.483
5	0.000	0.000	0.000	0.473	89.835	0.236	1.655	0.000	4.019	0.000	0.000	3.783
6	0.000	0.000	0.000	10.764	2.257	63.194	6.076	0.347	0.868	13.368	0.000	3.125
7	0.000	0.000	0.000	61.953	2.020	2.525	29.293	0.000	2.357	0.168	0.000	1.684
8	0.000	0.000	1.149	35.824	8.621	14.559	12.261	1.149	13.985	1.724	0.000	10.728
9	0.000	0.000	0.000	23.679	21.494	1.821	7.104	0.000	29.144	2.186	0.000	14.572
10	0.195	4.288	12.281	3.899	0.585	14.620	1.559	0.000	0.000	62.573	0.000	0.000
11	0.000	5.197	52.688	0.000	0.000	1.971	0.538	0.000	0.358	39.247	0.000	0.000
12	0.000	0.000	0.198	5.159	4.762	0.198	0.794	0.000	2.976	0.794	0.000	85.119

Average Recall: 53.224%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	11.111	24.670	56.685	0.000	0.000	1.883	0.000	0.000	0.000	3.390	2.260	0.000
3	0.000	5.926	83.704	0.000	0.000	0.000	0.370	0.000	0.000	7.037	2.963	0.000
4	0.000	0.000	0.000	28.460	4.094	2.339	56.725	4.094	1.365	1.365	0.000	1.559
5	0.000	0.000	0.000	0.364	89.071	0.364	4.007	0.364	4.007	0.000	0.000	1.821
6	0.000	2.254	0.483	9.501	0.483	52.657	8.213	9.340	1.932	13.527	0.000	1.610
7	0.000	0.000	0.000	23.868	8.230	2.058	59.259	1.440	3.086	0.412	0.000	1.646
8	0.000	0.383	0.000	17.625	2.682	16.858	10.728	27.778	6.130	3.831	0.000	13.985
9	0.000	0.214	0.000	6.624	18.803	0.855	9.615	3.632	47.863	1.709	0.000	10.684
10	0.000	6.043	14.230	0.585	0.195	4.483	2.144	1.170	0.585	64.717	5.458	0.390
11	0.000	6.197	69.231	0.000	0.000	0.000	0.214	0.214	0.000	17.521	6.624	0.000
12	0.000	0.000	0.000	2.914	2.186	0.000	2.368	1.457	3.461	2.550	0.000	85.064

Average Recall: 55.822%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	6.327	38.580	50.309	0.000	0.000	0.309	0.154	0.000	0.000	2.623	1.698	0.000
3	0.000	5.442	82.313	0.000	0.000	0.000	0.680	0.000	0.000	6.122	5.442	0.000
4	0.000	0.000	0.546	13.843	2.732	0.911	61.931	3.643	9.107	0.000	0.000	7.286
5	0.000	0.000	0.000	0.000	84.615	0.000	1.197	0.171	3.419	0.000	0.000	10.598
6	0.000	0.188	0.753	3.390	2.448	35.405	12.053	6.780	6.968	21.281	1.130	9.605
7	0.000	0.000	0.000	12.500	7.870	0.926	65.278	3.472	3.704	0.694	0.000	5.556
8	0.000	0.000	0.000	6.584	3.086	4.733	20.576	22.840	19.959	1.440	0.412	20.370
9	0.000	0.198	0.000	3.373	5.952	0.000	17.857	1.984	46.627	1.984	0.000	22.024
10	0.206	2.058	7.202	0.617	0.206	5.144	3.909	1.029	1.440	67.284	10.494	0.412
11	0.000	2.083	59.259	0.000	0.000	0.000	0.231	0.000	0.000	21.065	17.361	0.000
12	0.000	0.000	0.000	0.182	2.368	0.000	1.639	1.275	9.472	0.911	0.000	84.153

Average Recall: 54.858%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	99.020	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.980
2	5.769	24.359	57.479	0.000	0.000	1.923	0.427	0.000	0.000	9.402	0.641	0.000
3	0.000	4.167	81.151	0.000	0.000	1.190	0.595	0.000	0.000	12.897	0.000	0.000
4	0.000	0.000	0.000	29.293	2.222	0.606	58.586	1.818	6.263	0.606	0.000	0.606
5	0.000	0.000	0.000	0.000	84.848	0.000	8.687	0.202	3.434	0.000	0.000	2.828
6	0.000	0.222	0.889	8.444	3.556	48.222	9.556	4.889	1.556	19.778	0.444	2.444
7	0.000	0.000	0.000	25.370	7.963	0.185	61.296	0.741	3.704	0.000	0.000	0.741
8	0.206	0.412	0.000	11.934	7.202	10.494	26.955	18.724	12.346	2.675	0.000	9.053
9	0.000	0.000	0.000	7.143	9.524	0.198	23.214	0.595	54.762	1.190	0.000	3.373
10	0.000	1.058	8.642	1.587	0.529	6.878	4.762	0.705	0.353	74.956	0.353	0.176
11	0.000	2.998	60.847	0.000	0.000	0.000	0.000	0.000	0.000	36.155	0.000	0.000
12	0.000	0.000	0.000	0.397	1.984	0.000	5.556	1.190	12.302	2.381	0.000	76.190

Average Recall: 54.402%

Mean Average Recall: 54.310%
Standard Deviation: 1.109

Position: ankle6g

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	7.345	25.989	55.932	0.000	0.000	0.000	0.000	0.000	0.000	6.403	4.331	0.000
3	0.000	1.932	74.396	0.000	0.000	0.000	0.161	0.000	0.000	7.085	16.425	0.000
4	0.000	0.000	0.000	69.615	2.041	0.000	19.501	4.989	0.227	0.454	0.000	3.175
5	0.000	0.000	0.000	0.000	97.917	0.000	0.231	0.000	0.694	0.000	0.000	1.157
6	0.000	4.007	2.186	9.290	3.279	44.080	4.554	10.383	2.550	16.940	1.275	1.457
7	0.000	0.000	0.327	56.863	4.902	0.817	23.039	7.516	1.961	2.124	0.000	2.451
8	0.000	0.192	0.575	29.693	9.195	6.513	12.069	25.479	3.065	5.747	0.000	7.471
9	0.000	0.000	0.741	11.852	44.444	0.556	4.815	9.630	12.037	4.444	0.000	11.481
10	0.000	7.475	4.848	1.414	0.202	0.808	0.808	0.808	0.000	74.141	9.495	0.000
11	0.000	3.778	44.000	0.000	0.000	0.000	0.000	0.000	0.000	29.333	22.889	0.000
12	0.000	0.000	0.000	3.704	8.497	0.871	1.307	1.089	1.089	1.525	0.871	81.046

Average Recall: 54.219%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	5.455	26.869	53.333	0.000	0.000	0.000	0.202	0.000	0.000	14.141	0.000	0.000
3	0.000	7.265	77.991	0.000	0.000	1.068	0.641	0.214	0.000	12.821	0.000	0.000
4	0.000	0.000	0.000	44.618	0.000	0.868	48.090	5.903	0.000	0.347	0.000	0.174
5	0.000	0.000	0.000	0.529	94.709	0.000	1.940	0.529	1.587	0.000	0.000	0.705
6	0.000	0.992	0.595	7.738	0.992	56.944	8.929	11.508	0.000	9.127	0.000	3.175
7	0.000	0.000	0.000	41.808	0.188	0.565	53.861	1.883	0.565	0.753	0.000	0.377
8	0.000	0.000	0.000	17.234	3.175	17.687	19.501	33.333	0.907	2.948	0.000	5.215
9	0.000	0.000	0.000	16.880	24.573	0.214	15.385	12.607	9.188	1.709	0.000	19.444
10	0.000	5.761	10.700	2.058	0.000	4.115	3.292	1.029	0.000	72.222	0.000	0.823
11	0.000	3.351	57.143	0.000	0.000	0.176	0.176	0.000	0.000	38.977	0.000	0.176
12	0.000	0.000	0.000	6.771	3.472	0.174	2.257	1.042	2.951	0.868	0.000	82.465

Average Recall: 54.350%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	8.961	33.154	45.341	0.000	0.000	1.075	0.000	0.000	0.000	7.527	3.943	0.000
3	0.000	12.667	54.222	0.000	0.000	0.444	0.000	0.000	0.000	23.333	9.333	0.000
4	0.000	0.000	0.000	15.984	0.585	0.195	79.532	0.390	0.780	1.754	0.000	0.780
5	0.000	0.000	0.000	0.000	90.675	0.198	0.992	0.000	6.548	0.000	0.000	1.587
6	0.166	5.141	0.166	8.292	0.332	45.937	7.794	0.829	1.493	24.046	0.829	4.975
7	0.000	0.000	0.000	10.082	0.000	0.206	85.185	0.412	2.675	0.617	0.000	0.823
8	0.000	0.185	0.000	21.852	0.741	10.000	29.259	5.926	6.852	7.593	0.370	17.222
9	0.000	0.000	0.000	9.167	6.944	0.000	8.889	0.278	57.500	3.333	0.000	13.889
10	0.163	7.680	2.778	0.817	0.000	3.595	2.451	0.000	0.817	74.673	5.229	1.797
11	0.000	9.524	29.960	0.000	0.000	0.198	0.198	0.000	0.198	43.254	16.667	0.000
12	0.000	0.000	0.000	4.242	0.404	0.000	3.232	0.404	9.091	2.626	0.000	80.000

Average Recall: 54.994%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	4.082	28.571	57.370	0.000	0.000	0.000	0.227	0.000	0.000	7.256	2.494	0.000
3	0.000	5.848	84.211	0.000	0.000	0.000	0.000	0.000	0.195	6.823	2.924	0.000
4	0.000	0.000	0.000	56.497	3.766	3.578	18.079	0.188	12.053	0.377	0.000	5.461
5	0.000	0.000	0.000	0.751	92.793	0.000	0.901	0.000	4.204	0.000	0.000	1.351
6	0.000	0.210	3.354	12.369	2.516	58.071	1.677	0.629	1.258	14.885	2.306	2.725
7	0.000	0.000	0.000	52.778	6.151	2.976	24.008	0.000	8.333	0.397	0.000	5.357
8	0.000	0.000	0.896	34.409	3.405	24.373	7.348	0.538	14.516	3.943	0.000	10.573
9	0.000	0.000	0.000	15.620	16.425	3.221	3.382	0.000	54.911	1.771	0.000	4.670
10	0.000	0.889	9.778	0.667	0.000	2.000	0.000	0.000	0.444	76.222	10.000	0.000
11	0.000	2.976	68.056	0.000	0.000	0.794	0.000	0.000	0.000	19.643	8.532	0.000
12	0.000	0.000	0.000	5.241	3.354	0.210	2.516	0.000	6.918	1.677	0.000	80.084

Average Recall: 55.370%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	5.735	21.505	61.828	0.000	0.000	2.151	0.179	0.000	0.000	8.423	0.179	0.000
3	0.000	6.215	72.316	0.000	0.000	0.565	0.000	0.000	0.000	20.904	0.000	0.000
4	0.000	0.000	0.000	44.834	0.195	0.195	49.318	2.534	0.000	0.780	0.000	2.144
5	0.000	0.000	0.000	0.000	88.647	0.483	2.899	0.242	3.623	0.000	0.000	4.106
6	0.000	0.231	0.000	12.037	0.231	49.769	12.269	5.324	0.231	15.046	0.000	4.861
7	0.000	0.000	0.000	37.556	0.667	0.667	54.889	1.778	0.889	0.222	0.000	3.333
8	0.000	0.192	0.000	21.456	0.958	8.812	26.628	21.264	2.490	4.598	0.000	13.602
9	0.000	0.000	0.000	10.943	20.539	1.178	17.677	4.545	23.737	2.020	0.000	19.360
10	0.185	1.111	3.889	2.222	0.000	6.296	5.926	0.370	0.000	79.444	0.000	0.556
11	0.000	0.546	55.009	0.000	0.000	0.729	0.000	0.000	0.000	43.352	0.364	0.000
12	0.000	0.000	0.000	1.736	2.257	0.000	2.778	0.174	1.389	1.562	0.000	90.104

Average Recall: 53.906%

Mean Average Recall: 54.568%
Standard Deviation: 0.598

Position: hand16g

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	81.548	18.452	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	7.292	52.431	36.285	0.000	1.215	0.694	0.347	0.000	0.000	0.174	0.000	1.562
3	5.201	7.092	77.778	3.546	0.946	2.364	0.473	1.182	0.000	0.236	0.236	0.946
4	0.242	0.000	8.937	60.628	0.725	1.932	8.937	18.116	0.000	0.000	0.000	0.483
5	0.000	0.000	0.000	0.000	93.968	0.317	3.016	0.000	0.000	0.000	0.000	2.698
6	0.174	4.514	5.729	6.250	0.868	79.167	2.604	0.000	0.000	0.000	0.000	0.694
7	0.000	2.614	1.089	5.447	3.050	3.486	65.577	8.715	0.000	0.000	0.218	9.804
8	0.000	0.926	13.519	30.185	3.333	2.222	8.333	39.074	0.000	0.556	0.000	1.852
9	4.630	3.009	4.167	8.565	10.417	34.954	17.130	6.481	0.926	0.000	1.157	8.565
10	0.673	2.862	11.111	22.222	2.020	12.795	15.825	24.579	0.168	0.168	0.000	7.576
11	4.586	24.868	25.573	1.235	1.587	30.159	5.291	3.527	0.000	0.176	1.235	1.764
12	0.000	0.000	0.210	1.258	12.998	1.887	4.822	4.612	0.000	0.000	0.210	74.004

Average Recall: 52.209%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	96.595	3.047	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.358
2	12.159	56.813	29.140	0.000	1.048	0.419	0.000	0.000	0.000	0.000	0.000	0.419
3	13.757	2.822	75.485	4.409	0.000	0.529	0.176	0.882	0.000	1.764	0.000	0.176
4	0.000	0.206	2.469	72.016	0.617	1.029	8.848	10.494	0.412	3.909	0.000	0.000
5	0.000	0.000	0.000	0.000	96.914	0.000	0.206	0.000	0.000	0.000	0.000	2.881
6	0.179	8.602	3.226	8.781	2.867	72.581	1.971	1.075	0.000	0.000	0.000	0.717
7	0.000	4.520	0.188	6.215	5.273	10.923	57.627	2.825	0.000	0.565	0.000	11.864
8	0.206	2.469	12.551	30.864	1.235	1.029	14.609	19.342	3.086	13.580	0.000	1.029
9	6.080	2.096	3.774	6.709	10.692	39.623	13.836	5.660	4.822	1.677	0.000	5.031
10	0.794	5.026	6.349	22.487	1.323	10.847	15.344	8.995	2.116	20.635	0.000	6.085
11	6.046	25.980	21.895	5.556	2.778	23.693	2.778	3.595	0.000	2.288	0.654	4.739
12	0.000	0.000	0.000	0.347	2.257	0.694	3.993	0.000	0.521	2.083	0.000	90.104

Average Recall: 55.299%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	97.421	2.183	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.397
2	15.556	51.852	25.185	0.000	0.926	1.111	0.741	0.926	0.370	2.222	0.185	0.926
3	7.533	3.766	70.245	2.825	0.000	1.883	0.565	3.955	0.188	9.040	0.000	0.000
4	0.000	0.000	4.406	50.192	1.341	3.640	4.598	31.992	0.000	3.065	0.000	0.766
5	0.000	0.000	0.000	0.000	94.856	0.000	1.029	0.000	0.000	0.000	0.000	4.115
6	0.000	4.321	2.881	1.440	0.000	84.979	4.321	0.000	0.206	0.000	0.206	1.646
7	0.000	0.000	0.192	5.747	3.257	0.383	77.011	4.981	0.000	0.575	0.000	7.854
8	0.741	0.556	2.407	12.037	2.037	2.222	22.222	45.185	0.000	8.889	0.000	3.704
9	3.351	2.469	1.411	1.587	8.818	43.739	24.691	7.055	0.176	3.880	0.176	2.646
10	0.766	2.490	3.831	12.261	2.107	12.069	24.904	20.690	0.000	18.391	0.000	2.490
11	5.556	15.598	12.393	1.496	2.778	44.231	4.915	3.419	0.855	4.274	0.641	3.846
12	0.198	0.000	0.000	0.198	1.984	0.595	1.786	0.794	0.000	2.381	0.000	92.063

Average Recall: 56.918%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	97.737	2.263	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	19.923	79.119	0.000	0.766	0.000	0.000	0.000	0.192	0.000	0.000	0.000	0.000
3	4.487	50.000	0.000	20.085	0.214	1.923	0.214	23.077	0.000	0.000	0.000	0.000
4	0.176	1.587	0.000	54.497	0.353	7.760	14.286	20.811	0.000	0.000	0.000	0.529
5	0.000	0.000	0.000	0.000	96.032	0.397	1.389	0.198	0.000	0.000	0.000	1.984
6	2.614	5.882	0.000	2.397	1.089	84.749	2.832	0.000	0.000	0.000	0.000	0.436
7	0.992	10.317	0.000	5.556	3.770	10.714	60.913	5.159	0.000	0.000	0.000	2.579
8	1.029	4.733	0.000	27.366	2.675	2.675	14.198	45.062	1.235	0.000	0.000	1.029
9	5.947	2.347	0.000	8.764	6.729	40.689	16.119	9.546	6.416	0.000	0.000	3.443
10	1.130	3.578	0.000	17.137	0.942	17.891	19.209	35.782	0.753	0.000	0.000	3.578
11	6.709	40.671	0.000	6.080	0.419	28.721	6.080	9.015	0.000	0.000	0.839	1.468
12	0.182	0.000	0.000	0.364	7.286	0.546	0.729	1.457	0.182	0.000	0.000	89.253

Average Recall: 51.218%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	77.401	22.599	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	4.487	81.624	5.128	0.000	1.068	5.556	0.000	0.427	0.000	0.000	0.214	1.496
3	3.125	25.694	55.382	5.208	0.000	5.382	0.868	3.646	0.000	0.347	0.174	0.174
4	0.505	0.168	1.852	62.626	0.505	4.882	8.081	15.993	0.000	5.051	0.000	0.337
5	0.000	0.000	0.000	0.000	94.549	0.000	2.096	0.000	0.000	0.000	0.000	3.354
6	0.000	3.770	0.397	4.762	0.595	85.119	5.357	0.000	0.000	0.000	0.000	0.000
7	0.179	1.434	0.000	0.717	1.075	8.065	74.014	3.763	0.000	1.075	0.000	9.677
8	0.377	0.753	2.260	36.911	0.942	1.507	6.968	31.827	0.000	14.878	0.000	3.578
9	1.496	0.427	1.923	7.692	2.564	46.154	13.034	9.402	0.000	8.333	0.214	8.761
10	0.896	5.914	12.366	15.233	1.075	11.290	18.817	13.082	0.000	16.846	0.179	4.301
11	5.778	34.000	15.778	0.222	1.556	30.667	3.778	4.222	0.000	1.556	0.889	1.556
12	0.000	0.000	0.000	0.419	3.983	1.887	4.612	0.000	0.000	2.096	0.000	87.002

Average Recall: 55.607%

Mean Average Recall: 54.250%
Standard Deviation: 2.420

Position: hand6g

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	97.893	2.107	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	14.747	60.404	18.788	0.202	1.212	0.404	0.000	0.000	0.000	3.232	0.000	1.010
3	4.733	2.058	87.449	0.206	0.000	1.235	0.000	2.263	0.206	1.852	0.000	0.000
4	0.000	0.000	2.534	39.571	0.390	1.949	11.306	41.326	0.390	2.339	0.000	0.195
5	0.000	0.000	0.000	0.000	96.057	0.358	1.434	0.000	0.358	0.000	0.000	1.792
6	0.195	7.407	5.458	1.949	1.170	81.287	1.754	0.195	0.000	0.000	0.195	0.390
7	0.842	10.606	0.168	0.842	1.684	8.754	65.488	10.438	0.000	0.000	0.000	1.178
8	0.427	0.214	3.632	18.590	3.205	0.855	13.675	45.940	4.915	7.265	0.000	1.282
9	5.263	0.585	1.754	5.653	9.357	38.791	15.010	12.671	4.094	2.339	0.000	4.483
10	2.004	4.189	3.825	10.565	0.364	16.211	17.304	18.033	2.004	20.036	0.000	5.464
11	8.386	21.593	26.625	1.468	1.258	27.463	1.677	3.564	0.000	4.612	1.677	1.677
12	0.000	0.000	0.000	1.587	5.754	2.183	2.381	0.000	1.190	2.381	0.000	84.524

Average Recall: 57.035%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	75.772	24.074	0.000	0.000	0.000	0.000	0.000	0.000	0.154	0.000	0.000	0.000
2	3.241	86.343	7.176	0.000	0.926	0.463	0.000	0.000	0.463	0.926	0.463	0.000
3	5.128	16.410	69.231	1.197	0.342	1.197	0.855	2.735	0.855	1.197	0.513	0.342
4	0.159	0.317	4.127	54.603	2.222	0.952	6.349	25.079	1.270	2.063	2.063	0.794
5	0.000	0.000	0.000	0.000	98.925	0.000	0.358	0.000	0.000	0.000	0.000	0.717
6	0.000	3.704	3.009	4.167	0.000	82.639	3.241	0.231	0.231	0.000	2.778	0.000
7	0.000	1.449	0.000	2.415	4.106	5.556	64.251	8.937	7.246	0.242	0.242	5.556
8	0.000	1.361	8.390	19.728	5.215	0.454	6.122	51.474	0.000	2.494	1.134	3.628
9	0.709	2.364	0.709	2.128	19.385	27.660	8.511	17.730	9.456	2.600	3.546	5.201
10	0.370	3.333	5.185	15.185	2.407	14.259	15.556	19.074	1.296	17.963	2.222	3.148
11	2.867	24.373	14.695	1.075	3.943	22.401	3.047	9.857	3.226	4.659	8.423	1.434
12	0.000	0.000	0.000	0.753	11.676	0.565	0.753	0.565	0.000	1.130	0.000	84.557

Average Recall: 58.636%

Actual class	Predicated class 1	2	3	4	5	6	7	8	9	10	11	12
1	78.723	21.277	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	2.381	92.659	2.778	0.000	0.595	0.397	0.000	0.595	0.198	0.000	0.198	0.198
3	7.819	27.160	51.852	7.819	0.000	0.206	0.206	3.909	0.000	0.000	1.029	0.000
4	0.227	0.227	1.814	82.086	0.000	0.000	0.680	14.059	0.000	0.000	0.000	0.907
5	0.000	0.000	0.000	0.000	95.455	0.000	0.673	0.000	0.337	0.000	0.000	3.535
6	0.182	2.732	0.729	3.643	2.004	81.967	2.732	0.000	1.457	0.000	4.189	0.364
7	0.000	2.951	0.174	18.056	2.257	2.431	53.299	5.208	0.694	0.000	0.174	14.757
8	0.926	0.000	4.259	45.370	3.148	0.741	5.000	32.778	0.185	0.000	0.000	7.593
9	2.593	0.556	2.593	12.407	7.407	30.926	13.333	8.519	5.926	0.185	0.556	15.000
10	0.629	4.822	5.451	27.254	0.629	9.644	15.933	25.367	1.258	0.000	1.677	7.338
11	5.370	30.741	10.556	1.481	4.444	23.333	3.704	3.889	1.111	0.000	12.963	2.407
12	0.000	0.000	0.000	0.766	2.682	0.000	1.341	0.575	0.000	0.000	0.000	94.636

Average Recall: 56.862%

Actual class	Predicated class 1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	16.204	44.907	34.877	0.000	0.772	1.852	0.000	0.154	0.000	0.926	0.000	0.309
3	7.654	3.951	78.025	2.963	0.000	1.975	0.741	1.481	0.000	3.210	0.000	0.000
4	0.505	0.000	9.428	50.337	0.337	4.040	7.407	17.340	0.000	10.101	0.000	0.505
5	0.000	0.000	0.000	0.000	90.703	0.227	0.454	0.000	0.000	0.227	0.000	8.390
6	0.000	5.556	6.548	3.175	0.000	82.937	1.786	0.000	0.000	0.000	0.000	0.000
7	0.565	2.825	1.695	3.013	0.942	9.793	69.303	6.026	0.000	0.753	0.000	5.085
8	0.179	0.179	16.667	16.846	1.613	1.434	15.412	30.466	0.000	14.695	0.000	2.509
9	2.604	0.347	10.590	8.507	2.083	41.319	17.708	4.340	0.000	7.465	0.347	4.688
10	0.195	3.509	7.018	10.526	1.170	15.205	14.620	11.891	0.000	32.359	0.000	3.509
11	6.584	19.547	24.691	1.852	2.058	29.835	4.527	3.292	0.000	5.144	0.000	2.469
12	0.202	0.000	0.000	0.404	1.414	1.414	0.808	0.404	0.000	1.818	0.000	93.535

Average Recall: 56.048%

Actual class	Predicated class 1	2	3	4	5	6	7	8	9	10	11	12
1	77.231	22.404	0.000	0.000	0.364	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	7.879	70.909	19.192	0.000	0.202	0.606	0.000	0.000	0.000	0.000	0.404	0.808
3	4.831	9.662	70.048	4.992	0.000	6.119	0.805	1.127	1.288	0.805	0.161	0.161
4	0.000	0.000	5.432	69.383	0.741	2.222	7.160	11.605	0.000	3.210	0.000	0.247
5	0.000	0.000	0.000	0.000	98.582	0.000	0.000	0.000	0.000	0.000	0.000	1.418
6	0.171	6.325	4.274	6.154	2.222	78.974	1.538	0.000	0.000	0.000	0.171	0.171
7	0.427	0.427	0.000	5.128	4.060	4.701	75.427	3.205	0.000	1.923	0.214	4.487
8	0.694	1.562	6.424	34.028	2.083	2.604	13.021	24.306	0.000	11.806	0.000	3.472
9	8.429	1.916	0.766	10.345	9.004	41.571	10.153	5.364	0.192	2.107	0.766	9.387
10	0.992	3.770	6.746	24.802	0.397	10.317	19.444	9.325	0.000	21.429	0.397	2.381
11	5.364	24.330	17.816	2.874	0.958	30.268	3.640	4.981	0.000	5.939	1.149	2.682
12	0.000	0.000	0.000	0.000	4.023	0.192	4.789	0.000	0.000	0.958	0.000	90.038

Average Recall: 57.011%

Mean Average Recall: 54.310%
Standard Deviation: 0.985

Position: chest16g

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	97.484	0.000	0.000	0.000	0.419	0.000	0.000	0.000	0.000	2.096	0.000	0.000
2	5.903	17.882	60.069	0.694	0.000	0.000	0.000	0.174	0.000	0.000	15.278	0.000
3	0.000	0.370	58.704	2.407	0.000	0.000	0.000	0.926	0.000	0.370	37.222	0.000
4	0.000	0.000	2.626	30.707	0.202	0.000	49.495	9.293	7.677	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	90.395	0.000	0.000	0.000	0.377	0.000	0.000	9.228
6	0.000	0.000	6.944	0.000	0.595	23.214	0.397	9.325	1.389	5.159	52.976	0.000
7	0.000	0.364	0.000	20.765	0.000	0.000	62.295	2.368	14.026	0.000	0.000	0.182
8	0.000	0.000	4.444	15.960	0.808	0.000	13.333	53.333	7.071	2.222	2.626	0.202
9	0.000	0.000	8.230	11.728	2.263	0.000	21.811	7.613	34.156	0.000	3.704	10.494
10	0.000	0.000	6.557	0.364	0.000	5.647	0.364	7.832	0.000	32.787	46.448	0.000
11	0.000	0.000	15.443	0.753	0.000	0.000	0.000	2.448	0.000	1.318	80.038	0.000
12	0.000	0.218	0.436	0.000	4.357	0.000	0.436	1.089	2.397	0.000	0.654	90.414

Average Recall: 55.951%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	98.077	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.923	0.000	0.000
2	0.206	53.086	0.000	2.263	0.000	0.206	0.412	0.000	0.000	1.852	41.975	0.000
3	0.000	21.088	0.000	1.814	0.000	0.000	0.000	0.000	0.000	0.680	76.417	0.000
4	0.000	0.000	0.000	20.468	2.534	0.585	52.827	19.883	3.509	0.195	0.000	0.000
5	0.000	0.000	0.000	0.000	95.993	0.000	0.729	0.000	0.729	0.000	0.000	2.550
6	0.000	0.000	0.000	0.000	0.332	63.350	0.000	6.302	0.829	8.126	21.061	0.000
7	0.000	0.000	0.000	9.793	1.318	0.188	77.778	5.273	5.273	0.000	0.188	0.188
8	0.000	0.000	0.000	5.128	8.889	2.051	11.453	57.607	12.308	2.564	0.000	0.000
9	0.000	0.753	0.000	3.955	8.475	3.766	14.313	15.443	40.866	1.695	1.507	9.228
10	0.000	1.029	1.235	0.412	0.206	15.432	0.412	4.321	0.000	43.827	33.128	0.000
11	0.000	2.881	0.000	3.086	0.000	0.000	0.412	0.000	0.000	7.819	85.802	0.000
12	0.000	0.000	0.000	0.975	5.068	0.000	1.559	1.754	1.559	0.390	0.195	88.499

Average Recall: 60.446%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	98.851	0.000	0.000	0.000	0.192	0.000	0.000	0.000	0.000	0.958	0.000	0.000
2	1.613	26.703	1.434	0.000	0.000	0.538	0.000	0.000	0.000	0.000	69.713	0.000
3	0.000	5.008	1.565	0.000	0.000	0.313	0.469	2.034	0.000	0.313	90.297	0.000
4	0.000	0.000	0.000	9.977	8.163	7.710	48.299	15.646	6.803	0.227	0.000	3.175
5	0.000	0.000	0.000	0.000	88.679	0.000	1.048	0.000	0.000	0.000	0.000	10.273
6	0.000	0.000	0.000	0.236	0.000	95.035	0.000	2.600	0.000	0.473	1.418	0.236
7	0.000	0.000	0.000	5.018	7.168	1.075	61.649	5.556	8.423	0.179	0.000	10.932
8	0.000	0.000	0.000	4.918	6.011	18.761	6.011	50.820	8.015	4.554	0.364	0.546
9	0.000	0.000	0.000	1.821	10.018	26.047	7.286	12.750	22.404	0.000	0.182	19.490
10	0.000	0.444	2.667	0.222	1.111	29.556	0.000	10.889	0.000	45.778	9.333	0.000
11	0.000	0.000	1.949	0.195	0.000	6.628	0.000	3.314	0.000	0.780	87.135	0.000
12	0.000	0.000	0.000	0.000	6.433	1.365	1.559	0.975	0.975	0.000	0.000	88.694

Average Recall: 56.441%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	77.273	21.212	0.000	0.000	0.000	0.000	0.000	1.515	0.000	0.000	0.000	0.000
2	0.000	28.571	0.595	1.587	0.000	0.000	0.000	1.190	0.198	0.000	67.857	0.000
3	0.000	3.909	2.058	3.909	0.000	0.000	0.000	6.173	0.617	0.000	83.333	0.000
4	0.000	0.000	1.471	39.706	0.000	0.000	30.882	8.987	18.301	0.000	0.654	0.000
5	0.000	0.000	0.000	0.000	97.670	0.000	0.000	0.000	0.000	0.000	0.000	2.330
6	0.000	0.000	9.644	17.400	0.210	12.788	0.419	3.145	15.933	0.000	40.461	0.000
7	0.000	0.000	0.000	16.667	0.000	0.000	50.595	2.976	29.762	0.000	0.000	0.000
8	0.000	0.000	0.966	22.464	0.725	0.000	8.937	50.725	14.976	0.000	0.966	0.242
9	0.000	0.000	1.533	15.517	2.682	0.000	16.475	3.065	51.724	0.000	1.916	7.088
10	0.000	0.000	2.675	3.704	0.412	5.556	0.000	23.663	0.823	33.128	30.041	0.000
11	0.000	0.000	1.677	4.822	0.000	0.000	0.000	7.338	0.210	0.000	85.954	0.000
12	0.000	0.179	0.000	0.538	7.706	0.000	0.896	0.358	2.867	0.000	0.896	86.559

Average Recall: 51.396%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	98.276	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.724	0.000	0.000
2	1.111	31.778	10.667	0.444	0.000	0.222	0.000	2.444	0.222	0.000	53.111	0.000
3	0.000	2.564	17.094	1.282	0.000	0.000	0.000	6.410	0.855	0.214	71.581	0.000
4	0.000	0.000	0.000	14.176	2.107	0.766	62.644	13.410	6.513	0.000	0.000	0.383
5	0.000	0.000	0.000	0.000	78.205	0.214	0.427	0.000	0.000	0.000	0.000	21.154
6	0.000	0.000	2.293	0.353	0.529	78.307	1.411	5.644	0.882	0.353	9.347	0.882
7	0.000	0.000	0.000	8.163	2.268	0.680	62.812	7.483	16.780	0.000	0.000	1.814
8	0.000	0.000	0.000	5.461	8.286	1.318	22.411	53.484	5.273	0.000	0.188	3.578
9	0.000	0.000	0.404	1.616	4.646	3.636	18.788	5.657	34.141	0.000	0.202	30.909
10	0.000	0.000	7.843	1.797	0.000	14.869	0.490	23.856	0.000	33.170	17.974	0.000
11	0.000	0.174	6.424	1.215	0.000	0.347	0.000	7.986	0.000	1.215	82.639	0.000
12	0.000	0.000	0.000	1.296	3.889	0.370	0.926	0.926	2.037	0.000	0.370	90.185

Average Recall: 56.189%

Mean Average Recall: 56.085%
Standard Deviation: 3.208

Position: chest6g

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	93.376	6.624	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	5.556	30.000	0.000	2.222	0.000	0.000	0.317	2.063	3.492	0.000	56.349	0.000
3	0.000	0.839	0.839	4.403	0.000	0.000	0.000	4.403	2.935	0.629	85.954	0.000
4	0.000	0.000	0.000	58.945	0.000	0.000	9.416	20.151	2.072	0.000	9.416	0.000
5	0.000	0.000	0.000	0.000	94.624	0.000	0.000	0.000	0.000	0.000	0.000	5.376
6	0.000	0.000	0.377	4.143	0.188	16.949	0.000	0.377	10.546	0.377	67.043	0.000
7	0.000	0.000	0.606	41.212	0.000	0.000	26.465	9.697	8.889	0.000	13.131	0.000
8	0.000	0.000	0.000	34.222	0.889	0.000	0.444	45.556	1.778	5.333	11.111	0.667
9	0.000	0.000	0.546	23.497	3.279	0.546	4.736	5.647	38.434	0.000	18.215	5.100
10	0.000	0.000	0.218	10.022	0.000	5.447	0.000	11.983	2.832	35.076	34.423	0.000
11	0.000	0.192	0.000	5.747	0.000	0.000	0.192	5.747	2.490	1.341	84.291	0.000
12	0.000	0.575	0.000	0.766	6.513	0.000	0.000	0.958	1.916	0.000	0.383	88.889

Average Recall: 51.120%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	97.694	0.000	0.000	0.000	0.210	0.000	0.000	0.000	0.000	2.096	0.000	0.000
2	4.167	33.532	22.421	0.000	0.000	0.000	0.000	0.000	0.000	0.794	39.087	0.000
3	0.000	5.065	48.856	2.124	0.000	0.000	0.000	0.000	0.000	2.614	41.340	0.000
4	0.000	0.000	0.000	75.505	0.000	0.000	9.091	9.343	5.808	0.253	0.000	0.000
5	0.000	0.000	0.000	0.000	86.667	0.000	0.202	0.000	0.808	0.000	0.000	12.323
6	0.000	0.000	0.717	0.896	0.538	52.509	0.000	6.093	0.896	18.817	18.817	0.717
7	0.000	0.000	0.000	50.912	0.166	0.166	38.474	2.985	6.136	0.000	0.000	1.161
8	0.000	0.000	0.000	37.100	2.260	0.188	6.780	41.431	8.851	3.390	0.000	0.000
9	0.000	0.000	0.926	18.750	1.389	1.157	9.722	6.481	53.241	1.852	1.157	5.324
10	0.000	0.000	6.011	1.639	0.000	16.940	0.000	4.918	0.000	52.459	18.033	0.000
11	0.000	0.206	28.807	2.263	0.000	0.206	0.000	0.000	0.000	11.111	57.407	0.000
12	0.000	0.000	0.000	0.729	2.732	0.182	0.729	0.000	4.007	0.000	0.729	90.893

Average Recall: 60.722%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	88.889	4.873	0.000	0.000	0.195	0.000	0.000	0.000	0.000	6.043	0.000	0.000
2	0.000	41.348	18.215	0.000	0.000	0.000	0.000	0.364	0.000	3.279	36.794	0.000
3	0.000	20.202	27.273	7.071	0.000	0.000	0.000	1.414	0.000	4.040	39.798	0.202
4	0.000	0.000	0.179	60.573	0.000	2.151	13.978	19.713	2.509	0.538	0.358	0.000
5	0.000	0.000	0.000	0.000	95.473	0.000	0.000	0.412	0.000	0.000	0.000	4.115
6	0.000	0.000	0.179	0.538	0.000	44.265	2.330	13.799	0.358	2.330	36.022	0.179
7	0.000	0.000	0.000	22.222	0.000	0.839	60.797	8.805	6.709	0.000	0.419	0.210
8	0.000	0.000	0.000	7.051	1.282	2.991	14.530	60.043	6.410	4.915	2.778	0.000
9	0.000	1.093	3.097	12.386	2.004	1.275	16.758	6.193	51.002	0.546	3.279	2.368
10	0.000	0.000	0.444	1.556	0.000	13.333	0.000	7.778	0.000	42.667	34.222	0.000
11	0.000	0.185	2.222	2.037	0.000	0.185	0.000	4.074	0.000	6.852	84.444	0.000
12	0.000	0.364	0.000	2.004	6.740	0.000	0.546	0.729	0.729	0.000	1.093	87.796

Average Recall: 62.048%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	84.722	15.278	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.000	29.140	44.864	1.468	0.000	0.210	0.000	0.210	0.629	3.145	20.335	0.000
3	0.000	9.111	41.111	4.667	0.000	1.778	0.000	1.111	0.889	2.444	38.889	0.000
4	0.000	0.000	0.000	74.074	0.185	0.000	22.222	1.667	1.852	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	93.957	0.000	0.390	0.390	1.365	0.000	0.000	3.899
6	0.000	0.000	0.000	1.691	0.000	51.208	3.865	7.246	0.000	1.208	32.850	1.932
7	0.000	0.000	0.000	44.444	0.000	0.000	49.630	0.556	5.370	0.000	0.000	0.000
8	0.000	0.000	0.000	24.521	1.341	0.000	35.249	32.567	3.640	2.682	0.000	0.000
9	0.000	0.000	0.000	19.397	0.753	0.377	30.320	2.448	41.808	0.188	2.825	1.883
10	0.000	0.000	0.000	1.389	0.347	9.375	0.174	3.819	0.000	49.479	35.243	0.174
11	0.000	1.212	2.424	2.222	0.000	0.000	0.000	2.222	0.000	4.848	87.071	0.000
12	0.000	0.000	0.000	1.852	6.584	0.000	3.704	1.029	6.173	0.823	0.823	79.012

Average Recall: 59.482%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	99.161	0.839	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	0.473	42.317	0.473	1.891	0.000	0.236	0.000	0.473	0.000	0.000	54.137	0.000
3	0.000	16.667	0.926	7.222	0.000	0.000	0.000	0.741	0.741	0.000	73.704	0.000
4	0.000	0.000	0.000	40.073	0.000	0.182	32.969	20.765	5.647	0.000	0.364	0.000
5	0.000	0.000	0.000	0.000	81.871	0.000	0.000	0.000	0.195	0.000	0.000	17.934
6	0.000	0.383	0.000	7.471	0.383	35.441	0.383	5.172	10.345	0.000	40.421	0.000
7	0.000	0.000	0.000	12.607	0.000	0.000	56.410	11.325	19.017	0.000	0.427	0.214
8	0.000	0.163	0.000	7.190	1.307	2.124	10.294	56.699	17.647	0.817	2.451	1.307
9	0.000	0.000	0.000	10.153	1.533	0.383	13.027	7.088	58.621	0.000	2.107	7.088
10	0.000	0.000	0.000	4.918	0.546	10.018	0.000	11.475	0.364	39.344	33.333	0.000
11	0.000	0.926	0.000	10.185	0.000	0.370	0.185	1.667	0.000	1.852	84.815	0.000
12	0.000	0.000	0.000	0.629	4.403	0.000	0.839	1.258	3.564	0.000	0.629	88.679

Average Recall: 57.030%

Mean Average Recall: 58.080%
Standard Deviation: 4.308

Conf 3 - 3D accelerometer + 3D gyroscope

Position: ankle16g

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	95.940	0.000	1.068	0.000	0.000	0.000	0.000	0.000	0.000	1.923	1.068	0.000
2	5.674	20.804	50.355	0.000	0.000	0.473	0.000	0.000	0.000	15.603	7.092	0.000
3	0.000	9.357	73.294	0.585	0.000	0.585	0.780	0.000	0.000	7.602	7.797	0.000
4	0.000	0.000	0.000	47.670	1.434	0.179	48.387	0.179	1.971	0.179	0.000	0.000
5	0.000	0.000	0.000	0.185	95.370	0.370	0.741	0.185	3.148	0.000	0.000	0.000
6	0.000	0.958	0.766	8.046	0.575	75.287	2.874	0.958	4.789	4.023	1.724	0.000
7	0.000	0.000	0.000	34.116	1.408	0.000	62.441	0.000	1.878	0.156	0.000	0.000
8	0.000	0.000	0.000	3.640	1.341	7.854	7.854	56.322	2.107	5.747	0.000	15.134
9	0.000	0.000	0.000	10.758	10.229	4.586	8.995	2.116	58.907	1.058	0.000	3.351
10	0.000	5.653	6.433	1.559	0.000	7.018	3.899	5.263	0.975	56.725	11.696	0.780
11	0.000	1.307	64.924	0.000	0.000	0.436	0.000	0.436	0.000	17.429	15.468	0.000
12	0.000	0.000	0.000	0.641	3.846	0.000	0.214	2.137	1.923	1.709	0.000	89.530

Average Recall: 62.313%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	88.525	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.375	5.100	0.000
2	7.692	28.205	52.137	0.000	0.000	0.171	0.000	0.171	0.000	6.838	4.786	0.000
3	0.000	3.846	71.154	0.214	0.000	0.000	0.000	0.427	0.000	6.838	17.521	0.000
4	0.000	0.000	0.000	47.443	3.175	0.176	38.095	4.409	6.349	0.000	0.000	0.353
5	0.000	0.000	0.000	0.397	91.071	0.000	0.397	0.992	6.944	0.000	0.000	0.198
6	0.000	2.778	1.111	3.889	0.185	71.481	0.741	2.037	3.704	12.963	0.926	0.185
7	0.000	0.000	0.000	32.037	2.407	0.741	57.222	5.000	2.407	0.000	0.000	0.185
8	0.000	0.000	0.000	0.694	2.315	3.009	9.722	61.111	6.944	0.231	0.000	15.972
9	0.000	0.000	0.000	3.578	4.896	6.026	4.143	1.507	75.518	2.072	0.000	2.260
10	0.000	1.743	7.190	1.307	0.218	2.614	3.050	5.882	0.218	70.806	5.011	1.961
11	0.000	0.694	51.389	0.000	0.000	0.521	0.000	0.694	0.000	27.951	18.750	0.000
12	0.000	0.000	0.000	0.000	1.587	0.000	0.454	2.041	1.814	1.134	0.000	92.971

Average Recall: 64.521%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	93.791	0.163	0.817	0.000	0.000	0.000	0.000	0.000	0.000	5.065	0.163	0.000
2	8.081	31.515	51.919	0.000	0.000	1.616	0.000	0.000	0.000	3.232	3.636	0.000
3	0.000	6.989	80.645	0.179	0.000	0.179	0.179	0.000	0.000	6.631	5.197	0.000
4	0.000	0.000	0.000	26.325	4.615	0.000	55.385	2.735	9.402	0.000	0.000	1.538
5	0.000	0.000	0.000	0.000	93.617	0.000	2.128	0.000	3.783	0.000	0.000	0.473
6	0.000	1.940	0.353	3.527	0.176	67.725	2.822	3.351	4.762	14.109	0.705	0.529
7	0.000	0.000	0.000	13.333	4.667	0.222	64.444	8.000	7.333	0.000	0.000	2.000
8	0.000	0.000	0.202	1.212	3.636	3.434	7.273	69.293	4.242	2.020	0.000	8.687
9	0.000	0.000	0.000	3.205	11.966	2.778	4.701	3.205	67.521	0.641	0.000	5.983
10	0.000	6.289	10.901	0.210	0.000	1.677	2.096	1.887	0.000	74.214	2.725	0.000
11	0.000	3.060	63.768	0.000	0.000	1.288	0.161	0.000	0.000	25.121	6.602	0.000
12	0.000	0.000	0.000	0.454	4.308	0.000	0.454	0.907	1.587	1.814	0.000	90.476

Average Recall: 63.847%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	9.444	33.333	46.667	0.370	0.000	0.000	0.000	0.370	0.000	3.333	6.481	0.000
3	0.000	8.249	69.697	0.673	0.000	1.178	0.168	0.168	0.000	9.933	9.933	0.000
4	0.000	0.000	0.000	74.074	0.463	1.389	21.991	0.231	1.620	0.000	0.000	0.231
5	0.000	0.000	0.000	1.080	93.364	0.000	0.926	0.463	1.543	0.000	0.000	2.623
6	0.000	2.067	0.517	3.101	1.034	72.610	0.517	1.292	0.775	16.279	0.258	1.550
7	0.000	0.000	0.000	41.026	1.068	0.427	54.274	1.282	1.282	0.214	0.000	0.427
8	0.000	0.000	0.556	4.074	2.222	3.333	4.630	47.778	1.296	4.259	0.000	31.852
9	0.000	0.000	0.000	13.082	8.244	3.943	4.122	1.434	65.591	0.896	0.000	2.688
10	0.436	7.843	5.447	2.397	0.000	8.932	1.307	2.832	0.218	62.527	6.100	1.961
11	0.000	3.205	43.376	0.427	0.000	0.427	0.000	0.000	0.000	29.274	23.291	0.000
12	0.000	0.000	0.000	0.457	4.566	0.000	0.152	0.000	1.065	2.588	0.000	91.172

Average Recall: 65.643%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	85.965	7.407	2.729	0.000	0.000	0.000	0.000	0.000	0.000	0.585	3.314	0.000
2	7.037	34.815	51.667	0.185	0.000	1.111	0.000	0.370	0.000	0.556	4.259	0.000
3	0.000	4.535	83.673	0.000	0.000	0.227	0.000	0.000	0.000	4.989	6.576	0.000
4	0.000	0.000	0.000	76.620	2.546	0.000	19.676	0.694	0.463	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	95.299	0.000	0.000	1.068	3.632	0.000	0.000	0.000
6	0.000	3.880	0.882	5.644	0.176	73.369	0.000	0.176	9.347	4.586	1.940	0.000
7	0.000	0.000	0.000	63.103	0.629	0.419	31.447	2.935	1.258	0.000	0.000	0.210
8	0.000	0.168	1.178	6.397	1.347	9.764	4.040	51.178	6.397	4.209	0.505	14.815
9	0.000	0.000	0.000	7.843	8.061	3.268	0.000	0.000	77.996	0.871	0.000	1.961
10	0.296	8.741	9.481	2.519	0.000	13.037	1.333	5.037	0.444	44.741	14.222	0.148
11	0.000	5.011	54.684	0.000	0.000	4.357	0.218	0.000	0.000	17.647	17.647	0.436
12	0.000	0.000	0.000	0.353	5.291	0.000	0.000	1.235	1.411	3.175	0.353	88.183

Average Recall: 63.411%

Mean Average Recall: 63.947%
Standard Deviation: 1.242

Position: ankle6g

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	9.950	33.333	52.405	0.000	0.000	0.332	0.000	0.332	0.000	1.990	1.658	0.000
3	0.000	5.914	84.946	0.179	0.000	1.971	0.000	0.000	0.000	4.122	2.867	0.000
4	0.617	0.000	0.000	79.424	1.646	0.000	14.609	1.440	2.058	0.206	0.000	0.000
5	0.000	0.000	0.000	0.000	97.386	0.000	0.000	0.000	2.397	0.000	0.000	0.218
6	0.988	1.728	2.469	8.642	0.494	71.111	0.000	0.741	4.691	7.654	0.988	0.494
7	0.000	0.000	0.000	59.649	3.899	0.000	32.554	1.559	2.144	0.000	0.000	0.195
8	0.473	0.000	1.182	1.891	0.946	5.910	3.073	65.485	5.910	0.946	0.000	14.184
9	0.000	0.000	0.000	7.906	10.256	4.060	0.000	0.427	73.077	1.282	0.000	2.991
10	0.000	1.507	15.819	0.942	0.188	9.793	1.130	5.650	0.377	54.802	9.228	0.565
11	0.322	1.610	66.667	0.000	0.161	1.610	0.000	0.483	0.000	20.612	8.535	0.000
12	0.000	0.000	0.000	0.626	2.034	0.156	0.000	2.660	1.095	1.095	0.156	92.175

Average Recall: 66.069%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	100.00	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	10.870	29.952	51.449	0.000	0.000	0.242	0.000	0.242	0.000	1.449	5.797	0.000
3	0.000	6.130	72.605	0.192	0.000	0.192	0.192	0.383	0.000	5.364	14.943	0.000
4	0.000	0.000	0.000	42.424	0.505	0.000	50.842	1.515	4.040	0.673	0.000	0.000
5	0.000	0.000	0.000	0.358	92.473	0.000	0.717	0.000	5.376	0.000	0.000	1.075
6	0.000	1.587	1.429	3.016	0.000	65.238	1.905	0.952	4.921	8.571	11.429	0.952
7	0.000	0.000	0.000	29.690	1.457	0.000	61.202	3.279	3.643	0.000	0.000	0.729
8	0.000	0.000	0.370	4.815	1.481	6.481	7.222	55.000	1.111	4.815	0.000	18.704
9	0.000	0.000	0.242	2.174	5.072	0.966	4.831	0.483	78.261	1.691	0.483	5.797
10	0.253	1.768	5.556	1.010	0.000	2.778	4.545	2.020	0.253	63.384	18.182	0.253
11	0.000	2.604	52.778	0.000	0.000	1.910	0.347	0.174	0.000	14.062	28.125	0.000
12	0.000	0.000	0.000	0.000	2.825	0.188	0.565	0.377	0.565	2.637	0.188	92.655

Average Recall: 65.110%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	94.359	0.000	3.248	0.000	0.000	0.000	0.000	0.000	0.000	1.026	1.368	0.000
2	13.148	27.037	55.000	0.000	0.000	0.000	0.000	0.000	0.000	0.556	4.259	0.000
3	0.000	3.013	83.239	0.000	0.000	0.753	0.000	0.000	0.000	4.708	8.286	0.000
4	0.000	0.000	0.000	52.675	0.823	0.412	39.712	0.617	5.350	0.000	0.000	0.412
5	0.000	0.000	0.000	0.000	98.788	0.000	0.202	0.000	0.808	0.000	0.000	0.202
6	0.000	1.587	0.000	0.595	0.198	86.905	0.000	0.198	0.595	9.325	0.595	0.000
7	0.000	0.000	0.000	30.135	2.357	0.337	59.428	2.189	4.882	0.168	0.000	0.505
8	0.000	0.000	0.463	2.315	2.546	5.324	12.037	37.500	7.176	5.324	0.000	27.315
9	0.000	0.000	0.000	2.778	11.806	1.736	6.424	1.736	70.833	2.778	0.000	1.910
10	0.546	7.650	6.557	0.364	0.000	7.104	0.911	1.821	1.093	61.749	11.658	0.546
11	0.000	0.231	54.861	0.000	0.000	3.472	0.000	0.000	0.000	23.148	18.287	0.000
12	0.000	0.000	0.000	0.641	4.274	0.000	0.855	0.000	2.350	2.778	0.000	89.103

Average Recall: 64.992%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	99.177	0.000	0.206	0.000	0.000	0.000	0.000	0.000	0.000	0.206	0.412	0.000
2	8.081	28.620	55.724	0.000	0.000	0.168	0.000	0.000	0.000	0.168	7.239	0.000
3	0.000	9.293	75.354	0.606	0.000	1.616	0.000	0.000	0.000	2.424	10.707	0.000
4	0.000	0.000	0.000	70.598	0.513	0.513	24.274	1.026	2.222	0.855	0.000	0.000
5	0.000	0.000	0.000	0.000	93.762	0.000	1.170	0.195	4.678	0.000	0.000	0.195
6	0.000	0.546	1.639	8.925	0.000	71.038	0.364	0.364	4.007	10.200	2.732	0.182
7	0.000	0.000	0.000	55.991	0.218	1.089	35.294	3.922	3.050	0.436	0.000	0.000
8	0.000	0.000	0.171	5.812	3.248	10.427	2.222	51.966	4.103	5.299	0.000	16.752
9	0.000	0.000	0.000	15.079	10.119	4.365	3.373	2.976	59.921	0.000	0.000	4.167
10	1.389	4.167	4.563	2.381	0.000	13.095	1.190	4.762	0.992	58.730	8.333	0.397
11	0.000	0.855	53.846	0.000	0.000	2.350	0.000	0.000	0.000	17.308	25.427	0.214
12	0.000	0.000	0.000	0.444	5.778	0.444	0.667	2.000	0.000	1.778	0.000	88.889

Average Recall: 63.231%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	99.104	0.000	0.896	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	7.176	35.417	48.148	0.000	0.000	0.000	0.231	0.463	0.000	2.315	6.250	0.000
3	0.000	9.434	75.681	0.419	0.000	1.048	0.000	0.210	0.000	3.983	9.224	0.000
4	0.000	0.000	0.000	56.738	2.600	0.000	37.825	0.236	1.655	0.473	0.000	0.473
5	0.000	0.000	0.000	0.179	96.416	0.000	1.075	0.000	1.792	0.000	0.000	0.538
6	0.202	2.828	0.606	3.838	0.202	70.909	0.808	0.404	9.899	7.475	2.626	0.202
7	0.000	0.000	0.000	32.265	2.137	0.214	62.821	0.855	1.496	0.000	0.000	0.214
8	0.000	0.168	0.337	2.694	3.367	5.556	4.882	45.118	3.872	2.694	0.842	30.471
9	0.000	0.000	0.000	8.213	5.475	1.771	3.865	0.483	75.523	2.093	0.322	2.254
10	0.663	5.307	9.950	2.156	0.000	7.297	4.478	3.648	0.332	44.113	21.559	0.498
11	0.210	2.516	58.491	0.000	0.000	1.468	0.000	0.210	0.000	16.352	20.755	0.000
12	0.000	0.000	0.000	0.000	4.321	0.000	0.000	0.823	1.235	3.086	0.000	90.535

Average Recall: 64.428%

Mean Average Recall: 64.766%
Standard Deviation: 1.041

Position: hand16g

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	74.491	25.352	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.156	0.000	0.000
2	7.088	68.199	17.816	0.000	0.192	1.724	0.575	0.000	0.958	2.299	0.766	0.383
3	5.144	5.144	72.222	1.235	0.000	1.029	0.412	2.263	2.469	2.469	7.202	0.412
4	0.427	0.000	1.282	80.769	0.855	0.855	4.487	8.333	1.496	0.641	0.641	0.214
5	0.000	0.000	0.000	0.000	98.380	0.000	0.000	0.000	0.231	0.000	0.000	1.389
6	0.195	0.975	9.552	0.390	0.195	77.778	0.585	0.390	3.314	0.195	6.433	0.000
7	0.000	0.412	0.823	2.469	2.263	0.206	90.947	1.440	1.029	0.206	0.206	0.000
8	0.358	0.000	8.423	16.487	1.075	0.717	6.452	49.821	4.301	5.914	4.659	1.792
9	5.838	0.377	2.448	4.331	1.507	22.222	4.708	0.942	42.373	4.520	7.721	3.013
10	1.440	0.823	2.469	8.642	0.617	11.728	12.140	12.140	15.226	29.630	2.881	2.263
11	4.938	6.878	19.224	5.115	0.705	12.875	2.646	4.938	7.584	8.289	26.102	0.705
12	0.000	0.000	0.000	0.397	1.587	0.000	0.595	0.794	1.190	1.190	0.000	94.246

Average Recall: 67.080%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	96.017	3.774	0.000	0.000	0.000	0.000	0.000	0.000	0.210	0.000	0.000	0.000
2	24.713	56.513	15.326	0.000	0.000	0.192	0.958	0.000	1.149	0.383	0.192	0.575
3	7.568	0.322	71.981	1.288	0.000	2.415	0.483	1.288	2.254	4.509	7.407	0.483
4	0.000	0.000	0.444	77.111	0.667	0.444	7.556	3.778	7.556	1.333	0.444	0.667
5	0.000	0.000	0.000	0.000	97.175	0.000	0.188	0.000	0.565	0.000	0.000	2.072
6	0.206	0.000	5.144	0.823	1.235	78.601	2.058	0.617	5.350	0.000	5.967	0.000
7	0.202	0.000	0.000	3.232	2.626	0.202	90.707	1.616	1.212	0.202	0.000	0.000
8	0.454	0.000	1.814	18.367	0.000	3.401	8.844	43.764	9.070	7.256	6.349	0.680
9	5.031	0.000	0.000	2.935	3.354	12.369	6.499	0.000	61.845	2.516	4.193	1.258
10	0.817	0.000	2.288	5.719	1.471	5.556	14.379	11.601	13.399	39.052	3.758	1.961
11	5.848	9.552	7.602	1.559	0.390	8.382	2.339	3.119	20.273	5.458	31.579	3.899
12	0.000	0.000	0.000	0.353	2.998	0.000	0.000	0.000	1.587	0.882	0.000	94.180

Average Recall: 69.877%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	95.322	3.899	0.000	0.000	0.000	0.000	0.000	0.000	0.585	0.195	0.000	0.000
2	14.657	39.480	39.243	0.000	0.473	1.418	0.236	0.000	1.655	0.000	2.837	0.000
3	11.111	0.595	70.040	0.992	0.000	1.389	0.992	3.373	1.786	1.389	8.333	0.000
4	0.195	0.000	0.195	69.786	0.195	0.195	9.747	13.840	0.780	0.390	4.678	0.000
5	0.000	0.000	0.000	0.000	98.182	0.000	0.404	0.202	0.202	0.000	0.000	1.010
6	0.000	0.222	1.778	0.000	0.444	87.111	0.000	0.222	4.444	0.000	5.778	0.000
7	0.626	0.000	0.000	2.191	0.782	0.313	92.332	1.565	0.782	0.782	0.156	0.469
8	0.741	0.370	0.556	16.852	0.185	0.000	4.630	49.630	5.556	10.370	9.815	1.296
9	2.407	0.000	0.000	3.333	2.037	9.074	10.185	0.741	51.111	8.148	10.556	2.407
10	0.473	0.709	1.182	4.019	0.000	5.201	21.513	11.348	14.657	33.097	6.383	1.418
11	3.013	3.390	11.488	3.013	1.507	8.286	1.507	2.448	14.124	10.734	39.171	1.318
12	0.000	0.000	0.000	0.161	1.932	0.000	0.000	0.805	0.805	0.805	0.000	95.491

Average Recall: 68.396%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	98.866	1.134	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	16.239	48.376	28.205	0.000	0.171	1.026	0.684	0.000	0.855	0.513	3.590	0.342
3	10.913	6.349	68.254	1.190	0.000	1.786	0.397	0.992	1.786	2.778	5.357	0.198
4	0.182	0.000	0.182	75.410	0.000	0.182	4.736	12.568	0.911	3.643	0.911	1.275
5	0.000	0.000	0.000	0.000	97.374	0.000	0.202	0.000	0.000	0.000	0.000	2.424
6	0.174	0.347	2.778	2.083	0.174	80.382	0.521	0.347	7.639	0.000	4.861	0.694
7	0.383	0.000	0.192	9.770	1.724	1.149	77.203	2.874	4.023	1.533	0.000	1.149
8	0.000	0.000	1.556	23.111	0.222	1.556	7.111	42.000	7.111	11.556	4.444	1.333
9	6.591	0.000	0.188	1.695	3.013	7.721	4.520	0.565	58.380	5.650	6.026	5.650
10	0.397	0.595	3.571	8.532	0.397	5.159	14.286	12.897	10.714	37.302	3.175	2.976
11	6.173	8.818	9.877	1.587	1.235	6.173	3.880	1.411	15.520	6.349	33.862	5.115
12	0.641	0.000	0.000	0.641	2.991	0.000	0.000	0.214	1.068	1.709	0.000	92.735

Average Recall: 67.512%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	99.610	0.000	0.000	0.000	0.195	0.000	0.000	0.000	0.195	0.000	0.000	0.000
2	31.801	53.448	9.387	0.000	0.383	0.383	0.383	0.000	1.149	0.958	1.916	0.192
3	5.128	35.684	47.009	1.068	0.000	1.923	0.000	0.641	1.709	1.068	5.556	0.214
4	0.166	0.000	0.000	78.275	0.498	0.663	5.307	9.453	0.829	1.161	2.322	1.327
5	0.000	0.000	0.000	0.000	97.101	0.000	0.644	0.000	0.000	0.000	0.000	2.254
6	2.151	0.358	3.047	1.971	0.896	77.419	0.896	1.613	3.763	0.000	7.885	0.000
7	0.000	0.000	0.000	7.710	2.268	2.268	82.766	2.268	1.587	0.680	0.227	0.227
8	0.000	0.000	4.209	25.758	0.505	2.525	6.566	44.613	7.407	5.051	1.515	1.852
9	5.556	0.000	0.000	4.960	3.571	12.103	6.746	0.595	53.571	7.341	2.579	2.976
10	1.971	0.896	3.584	12.545	0.179	6.631	16.129	11.111	11.111	27.599	5.914	2.330
11	8.395	2.222	16.296	0.741	3.457	3.457	2.222	3.210	9.136	3.210	45.926	1.728
12	0.000	0.000	0.000	0.494	3.457	0.000	0.000	0.000	0.741	1.728	0.000	93.580

Average Recall: 66.743%

Mean Average Recall: 67.922%
Standard Deviation: 1.256

Position: hand6g

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	96.599	3.401	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	16.179	56.725	21.637	0.000	0.000	0.585	0.585	0.195	0.000	2.729	0.975	0.390
3	4.127	0.317	78.889	1.429	0.000	1.429	0.159	3.968	1.587	0.794	7.143	0.159
4	0.222	0.000	0.000	79.778	0.444	0.000	8.222	8.444	0.667	1.556	0.667	0.000
5	0.000	0.000	0.000	0.000	99.247	0.000	0.377	0.000	0.188	0.000	0.000	0.188
6	1.341	0.000	3.065	2.490	0.383	78.544	0.766	0.766	4.789	0.000	7.663	0.192
7	0.182	0.000	0.000	5.829	1.093	0.000	88.342	3.461	0.911	0.000	0.182	0.000
8	0.210	0.000	1.048	16.562	2.516	2.306	7.966	57.442	4.193	2.516	3.774	1.468
9	5.556	0.383	0.192	5.939	1.724	11.111	12.644	2.682	49.234	2.682	5.556	2.299
10	1.471	0.327	2.778	9.150	1.797	7.516	13.562	17.647	13.399	24.510	5.719	2.124
11	8.772	2.924	7.018	2.144	0.975	9.162	2.924	7.018	9.162	6.043	41.715	2.144
12	0.000	0.000	0.000	0.694	0.926	0.000	0.926	0.463	0.926	1.620	0.000	94.444

Average Recall: 70.456%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	96.881	2.534	0.000	0.000	0.000	0.000	0.000	0.000	0.390	0.195	0.000	0.000
2	15.556	46.263	33.131	0.000	1.010	1.212	0.606	0.404	0.404	0.202	1.010	0.202
3	9.091	2.222	75.556	0.606	0.000	4.040	0.404	2.020	0.808	2.828	2.424	0.000
4	0.182	0.000	3.097	77.960	0.000	0.546	3.825	8.925	0.546	1.639	3.097	0.182
5	0.000	0.000	0.000	0.390	94.737	0.000	1.170	0.195	2.144	0.000	0.000	1.365
6	0.000	0.000	8.135	0.595	0.595	83.135	0.595	1.587	1.786	0.000	3.571	0.000
7	0.390	0.000	0.585	13.255	0.585	0.390	78.947	4.288	1.170	0.195	0.000	0.195
8	0.163	0.000	5.065	21.569	0.327	0.980	4.412	50.163	2.451	5.556	8.007	1.307
9	3.175	0.595	0.397	6.349	1.786	8.730	7.738	1.190	53.175	7.143	7.540	2.183
10	1.887	1.887	11.111	11.530	0.210	7.128	10.063	14.885	10.273	27.673	2.306	1.048
11	6.709	11.321	18.449	2.516	2.096	12.579	2.096	5.031	14.675	3.983	18.449	2.096
12	0.000	0.000	0.000	1.667	1.481	0.000	0.185	0.000	1.111	2.407	0.000	93.148

Average Recall: 66.341%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	93.303	6.697	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	15.152	57.172	25.859	0.000	0.000	0.000	0.404	0.000	0.202	0.000	0.808	0.404
3	7.176	2.546	78.472	0.926	0.694	2.546	0.926	0.926	0.694	3.241	1.620	0.231
4	0.412	0.000	0.000	65.844	0.000	2.675	13.374	9.053	4.733	2.675	0.823	0.412
5	0.000	0.000	0.000	0.000	94.747	0.000	1.818	0.606	0.000	0.000	0.000	2.828
6	0.202	0.404	2.828	0.202	0.202	86.465	1.414	0.202	3.434	0.000	4.646	0.000
7	0.000	0.000	0.000	1.565	0.782	0.156	94.679	0.939	0.626	0.000	0.000	1.252
8	0.000	0.000	2.424	10.707	1.212	5.859	11.919	49.697	6.869	7.273	3.434	0.606
9	3.704	0.161	0.000	1.932	0.322	20.773	5.958	0.483	52.818	5.314	5.636	2.899
10	0.202	0.404	3.434	5.051	0.202	12.323	13.535	10.101	9.293	38.586	5.859	1.010
11	3.783	4.255	21.040	0.473	0.236	18.440	3.073	2.364	11.820	5.201	26.478	2.837
12	0.000	0.000	0.000	0.000	1.307	0.000	0.000	0.000	1.089	0.436	0.000	97.168

Average Recall: 69.619%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	92.172	7.071	0.000	0.000	0.253	0.000	0.000	0.000	0.505	0.000	0.000	0.000
2	18.182	26.094	34.007	0.000	0.842	1.684	0.000	0.000	1.010	2.357	15.657	0.168
3	10.185	0.741	73.519	1.481	0.000	2.963	0.926	2.593	1.296	3.519	2.222	0.556
4	0.000	0.000	0.000	76.882	0.179	0.538	7.706	10.215	2.867	0.896	0.358	0.358
5	0.000	0.000	0.000	0.000	96.539	0.000	0.911	0.000	0.364	0.000	0.000	2.186
6	0.000	0.000	4.643	0.995	0.000	84.245	0.663	2.985	5.307	0.000	0.995	0.166
7	0.483	0.242	0.000	5.314	0.483	0.483	86.232	4.831	0.966	0.242	0.000	0.725
8	0.694	0.463	2.083	15.509	0.463	4.167	9.491	48.611	9.491	4.167	0.926	3.935
9	8.176	0.419	0.210	3.354	2.306	7.757	8.386	1.468	61.216	2.306	2.096	2.306
10	1.341	0.383	6.130	6.705	0.766	7.280	18.582	15.517	9.962	29.119	1.916	2.299
11	4.225	3.286	22.379	1.252	0.782	11.268	2.347	9.077	12.520	9.546	21.909	1.408
12	0.427	0.000	0.000	0.000	0.427	0.000	1.282	0.000	0.214	0.214	0.000	97.436

Average Recall: 66.165%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	99.479	0.347	0.000	0.000	0.000	0.000	0.000	0.000	0.174	0.000	0.000	0.000
2	19.287	48.008	27.883	0.210	0.419	0.419	0.000	0.210	1.048	0.419	2.096	0.000
3	11.317	1.029	76.132	1.852	0.206	2.263	0.000	0.823	1.029	1.852	3.292	0.206
4	0.000	0.000	1.481	70.185	0.370	1.296	12.037	6.111	2.407	2.593	2.222	1.296
5	0.000	0.000	0.000	0.000	97.980	0.000	0.404	0.000	0.202	0.000	0.000	1.414
6	0.436	0.654	7.407	4.139	1.525	71.678	0.871	1.089	8.061	0.000	4.139	0.000
7	0.214	0.000	0.000	7.265	3.419	0.214	77.991	5.342	4.487	0.000	0.427	0.641
8	0.179	0.179	4.480	18.817	3.943	0.896	7.168	37.097	8.423	7.885	6.989	3.943
9	4.357	0.000	0.000	2.614	1.525	10.458	4.793	0.218	68.192	1.307	3.704	2.832
10	0.839	0.839	3.774	13.627	1.887	9.015	10.273	9.644	15.304	26.415	5.241	3.145
11	5.172	8.429	16.667	2.682	2.299	6.897	1.724	1.724	11.303	4.406	35.057	3.640
12	0.296	0.000	0.000	0.000	1.333	0.000	0.000	0.000	0.000	2.963	0.296	95.111

Average Recall: 66.944%

Mean Average Recall: 67.905%
Standard Deviation: 1.990

Position: chest16g

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	95.906	1.365	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.729	0.000	0.000
2	0.000	26.415	59.539	0.000	0.000	5.451	0.000	0.000	0.000	2.725	5.870	0.000
3	0.176	1.235	86.243	0.176	0.000	4.056	0.000	0.000	0.882	2.646	4.586	0.000
4	0.000	0.000	0.000	40.042	1.048	1.677	40.252	11.950	1.677	0.839	0.000	2.516
5	0.000	0.000	0.000	0.000	99.034	0.000	0.000	0.242	0.000	0.000	0.000	0.725
6	0.000	0.000	2.124	0.000	0.000	86.111	0.490	2.288	1.797	3.105	3.922	0.163
7	0.000	0.000	0.000	8.286	0.942	1.883	73.823	2.637	5.085	0.000	0.000	7.345
8	0.000	0.000	0.000	5.115	1.587	4.586	10.935	65.256	5.291	5.291	0.705	1.235
9	0.000	0.000	0.000	1.646	2.058	0.823	16.255	4.527	67.695	1.029	0.000	5.967
10	0.000	0.000	3.030	0.000	0.000	8.081	0.202	2.828	1.818	50.505	33.535	0.000
11	0.156	0.313	14.085	0.469	0.000	12.520	0.000	1.408	1.252	9.233	60.563	0.000
12	0.000	0.483	0.000	0.483	4.589	0.000	0.000	2.657	2.415	1.208	0.000	88.164

Average Recall: 69.980%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	98.148	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.852	0.000	0.000
2	31.197	0.000	63.248	0.000	0.000	1.709	0.000	0.000	0.000	1.496	2.350	0.000
3	1.134	0.000	83.220	2.494	0.227	2.268	0.227	0.680	0.227	4.308	4.535	0.680
4	0.000	0.000	0.000	45.062	0.000	1.389	35.185	12.037	6.019	0.309	0.000	0.000
5	0.000	0.000	0.000	0.000	98.384	0.000	0.000	1.010	0.202	0.000	0.000	0.404
6	0.000	0.000	2.397	0.654	0.218	66.231	1.307	14.815	1.525	2.614	10.240	0.000
7	0.000	0.000	0.168	11.785	0.000	0.505	71.549	5.051	10.438	0.337	0.000	0.168
8	0.000	0.000	0.000	3.831	0.958	1.341	6.897	80.268	4.981	1.533	0.000	0.192
9	0.000	0.000	1.034	4.134	0.258	2.584	23.514	6.202	59.690	1.292	0.000	1.292
10	0.000	0.000	1.411	0.529	0.176	6.702	0.353	10.053	0.529	62.257	17.989	0.000
11	0.412	0.000	22.016	0.412	0.000	7.819	0.412	1.235	0.000	6.996	60.700	0.000
12	0.000	0.000	0.000	0.156	3.286	0.000	2.191	1.565	3.756	0.313	0.000	88.732

Average Recall: 67.853%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	97.506	0.454	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.041	0.000	0.000
2	0.198	58.929	18.651	1.984	0.000	10.516	0.198	0.000	4.365	4.167	0.992	0.000
3	1.587	30.864	38.448	10.229	0.000	4.938	0.176	0.705	2.822	3.880	6.349	0.000
4	0.000	0.000	0.000	65.278	0.000	3.770	17.857	5.754	7.143	0.198	0.000	0.000
5	0.000	0.000	0.000	0.000	97.636	0.000	0.000	0.473	0.236	0.000	0.000	1.655
6	0.000	1.525	1.307	0.871	0.000	78.214	0.436	0.871	2.179	2.179	12.200	0.218
7	0.000	0.000	0.805	30.113	0.000	2.254	51.691	3.543	10.950	0.000	0.000	0.644
8	0.000	0.364	0.911	16.576	0.729	10.565	8.197	56.102	3.643	1.821	1.093	0.000
9	0.000	0.342	5.812	5.812	1.538	0.342	9.402	0.171	72.650	0.000	1.538	2.393
10	0.218	0.218	1.307	1.961	0.000	7.843	0.000	4.139	0.436	61.002	22.876	0.000
11	0.000	2.778	6.349	0.595	0.000	15.476	0.000	1.389	0.000	9.921	63.294	0.198
12	0.347	0.000	0.000	1.562	0.174	0.000	0.868	0.347	1.042	1.389	0.000	94.271

Average Recall: 69.585%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	84.211	15.497	0.000	0.000	0.146	0.000	0.000	0.000	0.000	0.146	0.000	0.000
2	0.565	32.203	64.218	0.377	0.000	0.753	0.188	0.000	0.188	0.188	1.318	0.000
3	0.390	1.949	90.253	0.585	0.000	0.780	0.000	1.170	0.390	1.754	2.729	0.000
4	0.000	0.000	0.595	43.056	0.000	2.381	43.254	5.556	4.563	0.000	0.595	0.000
5	0.000	0.000	0.000	0.000	97.937	0.000	0.000	1.270	0.317	0.000	0.000	0.476
6	0.000	0.000	3.486	0.000	0.000	78.867	0.436	3.704	3.050	2.397	7.843	0.218
7	0.000	0.000	0.542	20.325	0.271	0.813	63.957	3.252	10.569	0.000	0.271	0.000
8	0.000	0.000	0.000	2.116	1.587	5.026	10.582	71.429	5.820	3.175	0.265	0.000
9	0.000	0.163	3.758	3.758	0.163	2.778	14.706	2.778	69.935	0.163	0.000	1.797
10	0.000	0.000	4.736	0.182	0.000	7.650	0.000	6.557	0.000	59.016	21.676	0.182
11	0.000	0.210	20.964	0.000	0.000	9.434	0.210	0.839	0.000	5.660	62.683	0.000
12	0.000	0.000	0.206	0.000	1.852	0.000	2.469	0.206	1.852	0.617	0.000	92.798

Average Recall: 70.529%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	98.039	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.961	0.000	0.000
2	0.829	29.353	63.184	0.000	0.000	0.332	0.000	0.166	0.332	2.819	2.985	0.000
3	1.029	0.823	87.449	1.029	0.000	0.412	0.000	0.823	0.412	2.058	5.967	0.000
4	0.000	0.000	0.222	60.889	0.000	2.667	27.333	4.667	3.333	0.222	0.222	0.444
5	0.000	0.000	0.000	0.000	97.712	0.000	0.163	0.163	0.163	0.000	0.000	1.797
6	0.000	0.000	5.051	0.842	0.168	78.956	0.337	3.872	0.168	1.515	8.586	0.505
7	0.000	0.000	0.000	20.261	0.436	0.436	75.381	1.743	1.525	0.000	0.000	0.218
8	0.000	0.000	0.000	10.394	2.688	11.649	7.885	57.706	3.584	3.943	0.896	1.254
9	0.000	0.000	1.170	3.509	0.975	2.339	16.764	1.170	65.887	0.390	0.585	7.212
10	0.000	0.000	2.534	0.780	0.000	5.848	0.000	7.018	0.975	50.292	31.189	1.365
11	0.000	0.000	19.078	1.258	0.000	2.096	0.000	2.725	0.000	5.031	69.602	0.210
12	0.000	0.000	0.000	0.214	2.137	0.427	1.496	0.427	1.068	0.214	0.000	94.017

Average Recall: 72.107%

Mean Average Recall: 70.011%
Standard Deviation: 1.541

Position: chest6g

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	97.175	0.942	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.883	0.000	0.000
2	0.546	20.401	59.199	0.546	0.000	10.018	0.000	0.182	0.000	2.368	6.740	0.000
3	0.000	0.926	78.148	1.296	0.000	8.704	0.000	0.926	0.000	0.370	9.630	0.000
4	0.000	0.000	0.377	58.004	0.000	6.968	22.411	7.156	4.896	0.188	0.000	0.000
5	0.000	0.000	0.000	0.000	99.383	0.000	0.000	0.000	0.000	0.000	0.000	0.617
6	0.000	0.000	0.946	1.182	0.473	81.797	0.236	7.329	0.946	0.709	6.383	0.000
7	0.000	0.000	0.000	20.947	0.000	2.732	65.209	2.732	8.379	0.000	0.000	0.000
8	0.000	0.000	1.533	5.939	0.958	12.835	9.579	64.176	2.490	1.916	0.575	0.000
9	0.000	0.000	1.449	5.797	1.208	2.657	12.077	2.174	71.256	0.000	0.966	2.415
10	0.000	0.000	2.293	0.705	0.000	9.700	0.000	7.760	0.353	40.212	38.977	0.000
11	0.000	0.322	13.849	2.738	0.000	19.646	0.000	3.221	0.161	7.568	52.496	0.000
12	0.218	0.654	0.000	0.654	2.397	2.179	0.871	0.871	1.307	0.218	0.000	90.632

Average Recall: 68.241%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	96.768	0.606	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.626	0.000	0.000
2	2.998	24.691	64.903	0.000	0.000	0.176	0.000	0.000	0.000	0.176	7.055	0.000
3	0.210	1.887	84.906	0.000	0.000	1.258	0.000	0.419	0.000	0.210	11.111	0.000
4	0.000	0.000	0.258	64.341	0.000	0.775	19.897	6.460	8.269	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	99.042	0.000	0.000	0.000	0.000	0.000	0.000	0.958
6	0.000	0.000	1.093	0.000	0.000	76.321	0.000	10.018	0.364	2.186	9.836	0.182
7	0.000	0.000	0.000	28.341	0.000	0.322	53.784	3.865	13.688	0.000	0.000	0.000
8	0.000	0.000	0.358	5.018	0.717	2.330	6.810	75.806	8.065	0.538	0.000	0.358
9	0.000	0.663	1.327	4.975	0.166	2.985	9.453	4.478	74.461	0.332	0.000	1.161
10	0.000	0.000	2.268	0.454	0.000	7.256	0.000	8.617	0.227	65.533	15.646	0.000
11	0.000	0.000	21.248	0.000	0.000	1.365	0.000	2.534	0.000	5.458	69.396	0.000
12	0.000	0.218	0.218	0.436	3.486	0.218	0.000	0.654	1.743	0.654	0.000	92.375

Average Recall: 73.119%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	97.872	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.128	0.000	0.000
2	1.258	54.927	20.545	0.000	0.000	15.094	0.000	0.000	1.048	0.210	6.918	0.000
3	0.347	13.889	66.319	0.000	0.000	10.764	0.000	0.347	0.521	1.215	6.597	0.000
4	0.000	0.000	0.206	37.654	0.000	2.675	44.239	4.115	10.905	0.206	0.000	0.000
5	0.000	0.000	0.000	0.000	98.659	0.000	0.000	0.766	0.000	0.000	0.000	0.575
6	0.000	0.176	0.176	0.705	0.000	95.944	0.000	0.176	0.353	1.587	0.882	0.000
7	0.000	0.000	0.000	15.254	0.000	1.695	72.881	3.390	6.780	0.000	0.000	0.000
8	0.000	0.000	0.444	6.444	0.000	8.667	14.889	64.889	3.333	1.333	0.000	0.000
9	0.000	0.538	0.358	1.254	0.717	4.659	21.864	2.688	65.771	1.075	0.000	1.075
10	0.000	0.000	1.212	0.606	0.000	17.374	0.404	2.626	0.404	63.434	13.939	0.000
11	0.000	0.444	11.111	0.000	0.000	23.556	0.444	0.889	0.667	5.778	57.111	0.000
12	0.000	0.000	0.000	0.000	0.913	0.913	0.609	0.761	3.653	0.152	0.000	92.998

Average Recall: 72.372%

Actual	Predicated class											
class	1	2	3	4	5	6	7	8	9	10	11	12
1	99.824	0.176	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	1.818	31.919	64.646	0.202	0.000	0.808	0.000	0.000	0.000	0.606	0.000	0.000
3	0.000	0.444	87.333	2.889	0.000	2.444	0.222	2.889	0.000	1.111	2.667	0.000
4	0.000	0.000	0.168	52.189	0.000	3.872	23.737	15.152	4.882	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	99.234	0.000	0.000	0.575	0.000	0.000	0.000	0.192
6	0.000	0.000	0.000	0.000	0.192	88.123	0.000	3.448	0.000	5.172	3.065	0.000
7	0.000	0.000	0.517	9.302	0.000	3.101	66.150	7.235	13.437	0.000	0.258	0.000
8	0.000	0.000	0.188	3.955	1.507	6.403	4.520	78.154	4.708	0.565	0.000	0.000
9	0.000	0.000	1.457	3.279	2.368	0.546	14.754	3.643	64.663	0.182	0.911	8.197
10	0.000	0.000	2.222	1.010	0.000	14.343	0.202	9.091	0.606	56.364	15.758	0.404
11	0.168	0.000	14.646	0.673	0.000	16.330	0.000	3.872	0.337	11.448	52.525	0.000
12	0.000	0.000	0.000	0.000	1.029	1.029	1.852	0.412	1.852	0.000	0.000	93.827

Average Recall: 72.525%

Actual class	Predicated class											
	1	2	3	4	5	6	7	8	9	10	11	12
1	81.834	16.578	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.587	0.000	0.000
2	0.000	56.162	10.707	0.000	0.000	0.808	0.202	0.000	0.606	1.212	30.303	0.000
3	0.195	22.807	40.936	0.585	0.000	0.585	0.780	1.559	1.170	0.780	30.604	0.000
4	0.000	0.000	0.347	42.014	0.000	8.333	40.278	6.944	1.910	0.174	0.000	0.000
5	0.000	0.000	0.000	0.000	97.175	0.000	0.000	1.130	0.000	0.000	0.000	1.695
6	0.000	0.192	0.000	0.192	0.000	82.759	0.000	8.812	0.192	0.192	7.471	0.192
7	0.000	0.000	0.000	13.131	0.000	0.606	78.788	4.040	3.030	0.000	0.000	0.404
8	0.000	0.000	0.000	4.023	0.192	23.946	8.429	58.238	0.958	3.257	0.575	0.383
9	0.000	0.000	0.436	1.525	0.218	5.011	27.887	1.743	57.734	1.307	1.743	2.397
10	0.000	0.000	0.171	0.171	0.000	9.915	0.000	5.128	0.855	58.120	25.470	0.171
11	0.000	0.494	6.420	0.494	0.000	4.938	0.247	3.457	0.000	7.901	76.049	0.000
12	0.000	0.000	0.192	0.383	1.533	0.575	0.958	0.958	0.958	0.000	0.192	94.253

Average Recall: 68.672%

Mean Average Recall: 70.986%
Standard Deviation: 2.331

Confusion matrixes for 5 classes on the SHL dataset

In this paragraph, the confusion matrixes obtained when testing the HBN model to classify 5 activities for the SHL dataset are reported. In the following, the list of the activities used in this paragraph is specified:

1. still
2. walk
3. run
4. bike
5. car

Conf 1 - 3D accelerometer (with pre-processing)

Position: Bag

Actual class	Predicted class				
	1	2	3	4	5
1	98.393	0.000	0.000	0.000	1.607
2	0.000	99.679	0.000	0.000	0.321
3	0.000	0.166	99.734	0.100	0.000
4	3.161	5.842	0.160	63.185	27.651
5	1.471	0.000	0.000	0.000	98.529

Average Recall: 91.904%

Actual class	Predicted class				
	1	2	3	4	5
1	98.729	0.000	0.000	0.000	1.271
2	0.572	98.979	0.082	0.368	0.000
3	0.000	0.143	99.572	0.250	0.036
4	2.384	1.269	0.115	70.396	25.836
5	5.287	0.000	0.000	0.000	94.713

Average Recall: 92.478%

Actual class	Predicted class				
	1	2	3	4	5
1	90.164	0.225	0.000	0.000	9.611
2	0.000	99.224	0.245	0.000	0.531
3	0.000	0.034	99.415	0.344	0.206
4	1.702	1.147	0.000	80.022	17.129
5	0.654	0.000	0.000	0.621	98.725

Average Recall: 93.510%

Actual class	Predicted class				
	1	2	3	4	5
1	96.742	0.000	0.000	0.000	3.258
2	0.000	99.066	0.000	0.047	0.887
3	0.000	0.000	99.922	0.078	0.000
4	0.676	9.162	0.203	67.579	22.380
5	3.240	0.000	0.000	0.000	96.760

Average Recall: 92.014%

Actual class	Predicted class				
	1	2	3	4	5
1	96.182	0.172	0.000	0.241	3.406
2	0.000	98.039	0.205	0.380	1.375
3	0.000	0.447	98.865	0.378	0.310
4	0.245	10.609	0.420	73.144	15.581
5	3.318	0.000	0.000	0.754	95.928

Average Recall: 92.432%

Mean Average Recall: 92.467 %
Standard Deviation: 0.635

Position: Hand

Actual class	Predicted class				
	1	2	3	4	5
1	92.285	0.932	0.000	0.032	6.750
2	0.428	98.467	0.000	1.105	0.000
3	0.000	0.399	99.601	0.000	0.000
4	10.364	0.160	0.000	84.514	4.962
5	27.598	1.422	0.000	2.892	68.088

Average Recall: 88.591%

Actual class	Predicted class				
	1	2	3	4	5
1	97.386	0.290	0.000	0.000	2.324
2	2.042	95.507	0.000	2.451	0.000
3	0.000	0.570	99.430	0.000	0.000
4	4.306	0.192	0.000	94.156	1.346
5	41.737	0.315	0.000	1.506	56.443

Average Recall: 88.584%

Actual class	Predicted class 1	2	3	4	5
1	94.536	1.543	0.000	0.096	3.825
2	1.634	97.345	0.041	0.490	0.490
3	0.344	0.516	99.140	0.000	0.000
4	8.213	2.812	0.000	86.681	2.294
5	26.242	3.529	0.000	2.320	67.908

Average Recall: 89.122%

Actual class	Predicted class 1	2	3	4	5
1	95.083	0.935	0.000	0.452	3.529
2	0.840	95.565	0.000	3.595	0.000
3	0.000	0.000	100.000	0.000	0.000
4	6.660	1.318	0.000	90.500	1.521
5	34.996	1.194	0.000	0.426	63.384

Average Recall: 88.906%

Actual class	Predicted class 1	2	3	4	5
93.258	1.720	0.000	0.447	4.575	93.258
1.024	97.191	0.263	1.317	0.205	1.024
0.000	0.516	99.174	0.000	0.310	0.000
2.871	1.120	0.000	94.433	1.576	2.871
35.897	1.131	0.000	0.867	62.104	35.897

Average Recall: 89.232%

Mean Average Recall: 88.887%
Standard Deviation: 0.297

Position: Hips

Actual class	Predicted class 1	2	3	4	5
1	96.496	0.000	0.000	0.000	3.504
2	0.178	99.822	0.000	0.000	0.000
3	0.000	0.133	99.867	0.000	0.000
4	7.043	1.761	0.000	81.713	9.484
5	11.029	0.000	0.000	1.765	87.206

Average Recall: 93.021 %

Actual class	Predicted class 1	2	3	4	5
1	94.481	0.000	0.000	0.000	5.519
2	0.000	97.426	0.163	2.124	0.286
3	0.000	0.535	98.895	0.570	0.000
4	1.499	1.153	0.000	92.234	5.113
5	12.325	0.000	0.000	0.070	87.605

Average Recall: 94.128%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.654	98.611	0.000	0.735	0.000
3	0.000	0.275	99.622	0.103	0.000
4	13.356	0.074	0.000	86.385	0.185
5	99.281	0.000	0.000	0.000	0.719

Average Recall: 77.067%

Actual class	Predicted class 1	2	3	4	5
1	92.821	0.000	0.000	0.000	7.179
2	0.000	98.273	0.000	1.167	0.560
3	0.000	0.039	99.961	0.000	0.000
4	3.516	1.859	0.000	85.632	8.993
5	13.598	0.000	0.000	0.000	86.402

Average Recall: 92.618%

Actual class	Predicted class 1	2	3	4	5
1	88.510	0.000	0.000	0.000	11.490
2	0.029	98.654	0.000	0.615	0.702
3	0.000	0.000	99.690	0.000	0.310
4	1.261	2.556	0.315	86.275	9.594
5	11.275	0.000	0.000	0.000	88.725

Average Recall: 92.371%

Mean Average Recall: 89.841%
Standard Deviation: 7.172

Position: Torso

Actual class	Predicted class 1	2	3	4	5
1	91.868	0.000	0.000	0.000	8.132
2	0.071	99.465	0.071	0.107	0.285
3	0.000	0.000	100.000	0.000	0.000
4	16.647	17.447	0.360	52.741	12.805
5	23.627	0.000	0.000	1.863	74.510

Average Recall: 83.717%

Actual class	Predicted class 1	2	3	4	5
1	96.550	0.000	0.000	0.000	3.450
2	0.490	97.018	1.062	1.185	0.245
3	0.000	0.535	99.465	0.000	0.000
4	6.228	7.882	0.038	78.739	7.113
5	26.366	0.000	0.000	1.961	71.674

Average Recall: 88.689%

Actual class	Predicted class 1	2	3	4	5
1	97.814	0.000	0.000	0.000	2.186
2	0.694	98.448	0.041	0.817	0.000
3	0.000	0.172	99.759	0.000	0.069
4	11.506	14.650	0.259	67.629	5.956
5	18.497	0.000	0.000	2.255	79.248

Average Recall: 88.580%

Actual class	Predicted class 1	2	3	4	5
1	98.431	0.000	0.000	0.000	1.569
2	0.700	98.926	0.000	0.373	0.000
3	0.000	0.000	100.000	0.000	0.000
4	8.688	1.217	0.203	84.517	5.375
5	28.730	0.000	0.000	3.836	67.434

Average Recall: 89.862%

Actual class	Predicted class 1	2	3	4	5
1	90.643	0.000	0.000	0.000	9.357
2	0.907	97.483	0.205	0.995	0.410
3	0.000	0.000	99.725	0.000	0.275
4	4.307	10.714	0.035	76.576	8.368
5	24.133	0.000	0.000	0.226	75.641

Average Recall: 88.014%

Mean Average Recall: 87.772%
Standard Deviation: 2.364

Conf 2 - 3D accelerometer (no preprocessing)

Position: Bag

Actual class	Predicted class 1	2	3	4	5
1	96.721	0.000	0.000	1.221	2.057
2	0.392	99.216	0.000	0.392	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	1.040	0.000	98.960	0.000
5	5.000	0.000	0.000	0.000	95.000

Average Recall: 97.979%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	98.856	0.000	1.144	0.000
3	0.000	0.036	99.964	0.000	0.000
4	0.115	0.154	0.000	99.731	0.000
5	1.786	0.000	0.000	0.000	98.214

Average Recall: 99.353%

Actual class	Predicted class 1	2	3	4	5
1	94.278	0.032	0.000	0.000	5.689
2	0.368	98.325	0.000	1.307	0.000
3	0.000	0.000	99.656	0.344	0.000
4	0.370	2.960	0.000	96.670	0.000
5	3.333	0.000	0.000	0.000	96.667

Average Recall: 97.119%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.747	98.553	0.000	0.700	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.338	4.970	0.000	94.692	0.000
5	2.472	0.000	0.000	0.000	97.528

Average Recall: 98.155%

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.468	98.595	0.000	0.936	0.000
3	0.000	0.000	99.690	0.310	0.000
4	0.000	3.852	0.000	96.148	0.000
5	1.923	0.000	0.000	0.113	97.964

Average Recall: 98.479%

Mean Average Recall: 98.217%
Standard Deviation: 0.810

Position: Hand

Actual class	Predicted class 1	2	3	4	5
1	87.207	1.221	0.000	0.161	11.411
2	8.342	77.718	0.820	9.840	3.280
3	0.000	0.000	100.000	0.000	0.000
4	0.000	4.162	0.000	95.838	0.000
5	19.853	0.000	0.000	0.588	79.559

Average Recall: 88.064%

Actual class	Predicted class 1	2	3	4	5
1	81.409	2.542	0.000	0.000	16.049
2	6.577	77.492	1.511	8.415	6.005
3	0.713	0.143	99.144	0.000	0.000
4	0.192	4.537	0.000	95.271	0.000
5	12.675	1.436	0.000	0.665	85.224

Average Recall: 87.708%

Actual class	Predicted class				
	1	2	3	4	5
1	81.581	0.289	0.096	0.289	17.743
2	3.350	77.737	5.065	12.173	1.675
3	0.344	0.000	99.656	0.000	0.000
4	0.518	4.403	0.000	94.303	0.777
5	15.131	0.523	0.000	0.033	84.314

Average Recall: 87.518%

Actual class	Predicted class				
	1	2	3	4	5
1	80.241	1.026	0.000	0.181	18.552
2	4.435	81.466	4.155	9.197	0.747
3	0.000	0.000	100.000	0.000	0.000
4	0.406	5.781	0.169	93.509	0.135
5	12.788	0.426	0.000	0.000	86.786

Average Recall: 88.400%

Actual class	Predicted class				
	1	2	3	4	5
1	80.667	1.342	0.000	0.378	17.613
2	4.741	76.149	4.009	10.945	4.156
3	0.069	0.000	99.931	0.000	0.000
4	0.210	0.175	0.000	99.615	0.000
5	8.446	0.189	0.000	0.000	91.365

Average Recall: 89.545%

Mean Average Recall: 88.247%
Standard Deviation: 0.801

Position: Hips

Actual class	Predicted class				
	1	2	3	4	5
1	95.661	0.579	0.000	3.761	0.000
2	0.000	99.964	0.000	0.036	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.680	1.200	0.000	98.119	0.000
5	5.000	0.000	0.000	0.000	95.000

Average Recall: 97.749%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.000	99.265	0.327	0.408	0.000
3	0.000	0.713	99.287	0.000	0.000
4	3.883	0.846	1.461	93.810	0.000
5	1.786	0.000	0.000	0.000	98.214

Average Recall: 98.115%

Actual class	Predicted class				
	1	2	3	4	5
1	99.518	0.000	0.000	0.482	0.000
2	0.613	99.265	0.082	0.041	0.000
3	0.069	0.344	99.587	0.000	0.000
4	1.443	0.370	0.000	97.003	1.184
5	3.333	0.000	0.000	0.000	96.667

Average Recall: 98.408%

Actual class	Predicted class				
	1	2	3	4	5
1	98.793	0.000	0.000	1.207	0.000
2	0.700	98.973	0.000	0.327	0.000
3	0.000	0.118	99.882	0.000	0.000
4	1.995	0.778	0.338	94.861	2.028
5	2.174	0.000	0.000	0.000	97.826

Average Recall: 98.067%

Actual class	Predicted class				
	1	2	3	4	5
1	98.796	0.000	0.000	1.204	0.000
2	0.263	99.239	0.000	0.498	0.000
3	0.000	1.032	98.624	0.000	0.344
4	2.486	0.420	0.000	95.308	1.786
5	1.923	0.000	0.000	0.000	98.077

Average Recall: 98.009%

Mean Average Recall: 98.070%
Standard Deviation: 0.236

Position: Torso

Actual class	Predicted class				
	1	2	3	4	5
1	89.939	0.000	0.000	0.000	10.061
2	0.285	99.572	0.000	0.143	0.000
3	0.000	0.000	100.000	0.000	0.000
4	16.527	0.000	0.000	83.473	0.000
5	0.882	0.000	0.000	0.000	99.118

Average Recall: 94.420%

Actual class	Predicted class				
	1	2	3	4	5
1	97.967	0.000	0.000	0.000	2.033
2	1.593	97.917	0.000	0.490	0.000
3	0.000	0.000	100.000	0.000	0.000
4	5.113	0.000	0.038	94.848	0.000
5	7.948	0.000	0.000	0.000	92.052

Average Recall: 96.557%

Actual class	Predicted class 1	2	3	4	5
1	90.325	0.000	0.000	0.000	9.675
2	0.817	98.897	0.000	0.286	0.000
3	0.000	0.344	99.656	0.000	0.000
4	12.172	0.000	0.000	87.828	0.000
5	0.294	0.033	0.000	0.000	99.673

Average Recall: 95.276%

Actual class	Predicted class 1	2	3	4	5
1	93.876	0.000	0.000	0.000	6.124
2	1.354	98.366	0.000	0.280	0.000
3	0.000	0.000	100.000	0.000	0.000
4	8.891	0.000	0.000	90.264	0.845
5	0.128	0.000	0.000	0.000	99.872

Average Recall: 96.476%

Actual class	Predicted class 1	2	3	4	5
1	98.555	0.000	0.000	1.445	0.000
2	0.819	97.747	0.000	1.434	0.000
3	0.275	0.000	99.725	0.000	0.000
4	4.132	0.000	0.000	95.868	0.000
5	1.735	0.000	0.000	0.000	98.265

Average Recall: 98.032%

Mean Average Recall: 96.152%
Standard Deviation: 1.376

Conf 3 - 3D accelerometer + 3D gyroscope

Position: Bag

Actual class	Predicted class 1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.071	99.073	0.071	0.677	0.107
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.480	0.000	99.520	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 99.719%

Actual class	Predicted class 1	2	3	4	5
1	97.422	0.436	0.000	0.000	2.142
2	0.000	98.039	0.000	1.961	0.000
3	0.000	0.143	99.857	0.000	0.000
4	0.000	1.423	0.769	97.809	0.000
5	0.000	0.000	0.000	0.000	100.000

Average Recall: 98.625%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.531	97.549	0.000	1.920	0.000
3	0.000	0.000	99.690	0.310	0.000
4	0.000	0.888	0.000	99.112	0.000
5	2.320	1.013	0.000	0.000	96.667

Average Recall: 98.604%

Actual class	Predicted class				
	1	2	3	4	5
1	99.910	0.000	0.000	0.000	0.090
2	0.700	98.366	0.000	0.934	0.000
3	0.000	0.000	100.000	0.000	0.000
4	0.000	4.598	0.000	95.402	0.000
5	2.046	0.512	0.000	0.000	97.442

Average Recall: 98.224%

Actual class	Predicted class				
	1	2	3	4	5
1	99.209	0.344	0.000	0.000	0.447
2	0.410	98.361	0.000	1.229	0.000
3	0.310	0.000	99.690	0.000	0.000
4	0.000	2.941	0.000	97.059	0.000
5	1.923	0.000	0.000	0.000	98.077

Average Recall: 98.479%

Mean Average Recall: 98.730%
Standard Deviation: 0.575

Position: Hand

Actual class	Predicted class				
	1	2	3	4	5
1	91.000	0.900	0.000	0.000	8.100
2	4.670	93.226	0.250	0.214	1.640
3	0.000	0.000	100.000	0.000	0.000
4	0.000	0.200	0.000	99.800	0.000
5	20.588	0.245	0.000	0.245	78.922

Average Recall: 92.590%

Actual class	Predicted class				
	1	2	3	4	5
1	89.216	1.017	0.000	0.182	9.586
2	5.882	92.770	0.000	0.286	1.062
3	0.000	0.000	99.608	0.392	0.000
4	0.000	0.192	0.000	99.808	0.000
5	15.091	0.945	0.000	0.000	83.964

Average Recall: 93.073%

Actual class	Predicted class 1	2	3	4	5
1	95.436	1.639	0.000	0.257	2.668
2	1.062	97.712	0.408	0.735	0.082
3	0.310	0.000	99.690	0.000	0.000
4	0.037	3.108	0.000	96.855	0.000
5	15.294	0.980	0.000	0.000	83.725

Average Recall: 94.684%

Actual class	Predicted class 1	2	3	4	5
1	92.157	2.353	0.000	0.000	5.490
2	2.334	97.292	0.000	0.000	0.373
3	0.000	0.196	99.804	0.000	0.000
4	0.000	4.936	0.000	95.064	0.000
5	14.962	0.213	0.000	0.000	84.825

Average Recall: 93.828 %

Actual class	Predicted class 1	2	3	4	5
1	89.061	1.926	0.000	0.103	8.910
2	2.692	90.284	0.059	1.141	5.824
3	0.310	0.138	99.415	0.138	0.000
4	0.035	0.455	0.000	99.510	0.000
5	10.483	0.189	0.000	0.000	89.329

Average Recall: 93.520%

Mean Average Recall: 93.539%
Standard Deviation: 0.793

Position: Hips

Actual class	Predicted class 1	2	3	4	5
1	99.357	0.000	0.000	0.643	0.000
2	0.000	99.786	0.000	0.214	0.000
3	0.000	0.233	99.767	0.000	0.000
4	5.042	0.000	0.000	94.958	0.000
5	5.000	0.000	0.000	0.000	95.000

Average Recall: 97.774%

Actual class	Predicted class 1	2	3	4	5
1	98.947	0.000	0.000	1.053	0.000
2	0.000	99.183	0.000	0.817	0.000
3	0.000	1.640	98.360	0.000	0.000
4	0.923	0.000	0.884	97.116	1.077
5	1.786	0.000	0.000	0.000	98.214

Average Recall: 98.364 %

Actual class	Predicted class				
	1	2	3	4	5
1	98.907	0.000	0.000	1.093	0.000
2	0.245	99.551	0.000	0.204	0.000
3	0.000	0.241	99.656	0.103	0.000
4	4.772	0.000	0.000	94.932	0.296
5	2.614	0.000	0.000	0.000	97.386

Average Recall: 98.086%

Actual class	Predicted class				
	1	2	3	4	5
1	97.074	0.000	0.000	2.926	0.000
2	0.280	98.599	0.000	1.120	0.000
3	0.000	0.000	100.000	0.000	0.000
4	1.285	1.623	0.169	95.909	1.014
5	2.174	0.000	0.000	0.000	97.826

Average Recall: 97.882%

Actual class	Predicted class				
	1	2	3	4	5
1	100.000	0.000	0.000	0.000	0.000
2	0.176	99.298	0.000	0.527	0.000
3	0.000	1.170	98.590	0.000	0.241
4	0.735	0.560	0.000	97.339	1.366
5	1.923	0.000	0.000	0.000	98.077

Average Recall: 98.661%

Mean Average Recall: 98.153%
Standard Deviation: 0.362

Position: Torso

Actual class	Predicted class				
	1	2	3	4	5
1	91.096	0.000	0.000	0.000	8.904
2	0.285	99.643	0.000	0.071	0.000
3	0.000	0.000	100.000	0.000	0.000
4	15.006	0.000	0.000	83.954	1.040
5	1.225	0.000	0.000	0.000	98.775

Average Recall: 94.694%

Actual class	Predicted class				
	1	2	3	4	5
1	95.606	0.000	0.000	0.000	4.394
2	0.204	98.897	0.000	0.899	0.000
3	0.000	0.000	99.929	0.000	0.071
4	4.575	0.000	0.000	95.425	0.000
5	3.396	0.000	0.000	0.000	96.604

Average Recall: 97.292%

Actual class	Predicted class				
	1	2	3	4	5
1	86.950	0.000	0.000	0.000	13.050
2	0.572	99.101	0.000	0.327	0.000
3	0.000	0.344	99.656	0.000	0.000
4	10.100	0.000	0.000	89.382	0.518
5	0.229	0.098	0.000	0.000	99.673

Average Recall: 94.952%

Actual class	Predicted class				
	1	2	3	4	5
1	97.888	0.000	0.000	0.151	1.961
2	0.093	96.919	0.000	2.241	0.747
3	0.000	0.000	100.000	0.000	0.000
4	6.525	0.000	0.000	92.732	0.744
5	3.282	0.000	0.000	0.000	96.718

Average Recall: 96.851%

Actual class	Predicted class				
	1	2	3	4	5
1	95.975	0.000	0.000	0.860	3.165
2	0.322	97.893	0.000	1.785	0.000
3	0.000	0.000	100.000	0.000	0.000
4	3.992	0.000	0.000	96.008	0.000
5	1.244	0.000	0.000	0.000	98.756

Average Recall: 97.726 %

Mean Average Recall: 96.303%
Standard Deviation: 1.389

Confusion matrixes for 8 classes on the SHL dataset

In this paragraph, the confusion matrixes obtained when testing the HBN model to classify 5 activities for the SHL dataset are reported. In the following, the list of the activities used in this paragraph is specified:

1. still
2. walk
3. run
4. bike
5. car
6. bus
7. train
8. subway

Conf 1 - 3D accelerometer (with pre-processing)

Position: Bag

Actual class	Predicted class							
	1	2	3	4	5			
1	89.521	0.000	0.000	0.000	0.000	0.000	3.600	6.879
2	0.000	99.572	0.107	0.321	0.000	0.000	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	0.120	11.164	0.800	62.305	9.004	0.360	9.684	6.563
5	0.245	0.000	0.000	0.588	66.618	0.098	2.157	30.294
6	4.261	0.339	0.000	0.377	22.021	24.661	18.326	30.015
7	5.689	0.032	0.000	0.579	0.257	0.771	87.046	5.625
8	7.252	0.000	0.000	0.560	3.517	0.000	25.366	63.305

Average Recall: 74.129%

Actual class	Predicted class							
	1	2	3	4	5			
1	96.659	0.000	0.182	0.000	0.290	0.000	0.690	2.179
2	0.041	97.794	0.613	1.552	0.000	0.000	0.000	0.000
3	0.000	0.606	99.394	0.000	0.000	0.000	0.000	0.000
4	1.769	12.880	0.154	61.669	19.454	0.077	1.230	2.768
5	7.038	0.000	0.000	0.105	86.275	0.210	2.801	3.571
6	4.813	0.000	0.036	0.321	32.121	25.455	14.082	23.173
7	5.184	0.000	0.000	0.266	5.284	1.595	84.380	3.290
8	14.706	0.032	0.000	0.633	17.932	0.032	17.615	49.051

Average Recall: 75.085%

Actual class	Predicted class							
	1	2	3	4	5			
1	98.554	0.096	0.000	0.129	0.064	1.125	0.032	0.000
2	0.245	99.183	0.327	0.000	0.000	0.245	0.000	0.000
3	0.000	0.000	99.759	0.000	0.000	0.241	0.000	0.000
4	6.844	0.703	0.592	79.467	8.213	3.885	0.296	0.000
5	16.307	0.000	0.000	0.882	76.536	0.980	0.719	4.575
6	27.378	0.000	0.000	7.081	42.338	7.771	3.558	11.874
7	34.143	0.000	0.000	0.000	1.194	0.554	43.223	20.887
8	41.594	0.000	0.000	0.321	4.307	13.404	10.511	29.862

Average Recall: 66.794%

Actual class	Predicted class							
	1	2	3	4	5			
1	96.259	0.000	0.000	0.000	0.302	0.000	0.965	2.474
2	0.000	98.506	0.000	0.700	0.373	0.000	0.000	0.420
3	0.000	1.569	97.686	0.745	0.000	0.000	0.000	0.000
4	1.826	12.238	0.034	65.213	13.185	0.203	2.637	4.665
5	3.410	0.000	0.000	0.000	74.552	0.000	0.725	21.313
6	3.695	0.038	0.000	0.113	34.766	22.964	16.742	21.682
7	5.362	0.184	0.000	0.000	6.863	0.521	82.935	4.136
8	11.702	0.062	0.000	0.000	11.640	0.156	22.658	53.782

Average Recall: 73.987%

Actual class	Predicted class							
	1	2	3	4	5			
1	92.363	0.413	0.000	0.138	0.034	0.138	3.681	3.234
2	0.029	98.917	0.088	0.615	0.117	0.000	0.205	0.029
3	0.000	0.688	97.867	1.135	0.310	0.000	0.000	0.000
4	0.000	9.839	2.031	75.665	7.423	0.000	3.817	1.225
5	3.167	0.000	0.000	0.867	68.891	3.017	8.220	15.837
6	4.885	0.034	0.000	1.617	41.211	9.150	25.112	17.991
7	1.783	0.802	0.000	0.000	2.941	0.936	92.781	0.758
8	10.784	0.000	0.000	0.039	14.510	0.000	38.941	35.725

Average Recall: 71.420%

Mean Average Recall: 72.283%
Standard Deviation: 3.355

Position: Hand

Actual class	Predicted class							
	1	2	3	4	5			
1	77.338	1.671	0.000	0.096	0.225	0.000	8.807	11.861
2	0.000	98.289	0.713	0.856	0.000	0.000	0.000	0.143
3	0.000	0.233	99.767	0.000	0.000	0.000	0.000	0.000
4	6.683	0.920	0.000	85.714	1.040	0.000	0.360	5.282
5	22.206	4.118	0.000	4.951	47.255	3.725	13.039	4.706
6	18.665	3.205	0.000	2.715	14.442	30.807	24.925	5.241
7	20.958	2.443	0.000	0.354	1.221	0.129	66.442	8.454
8	35.885	4.046	0.000	2.272	3.610	0.187	36.383	17.616

Average Recall: 65.404%

Actual class	Predicted class							
	1	2	3	4	5			
1	82.934	2.941	0.000	2.251	4.067	1.089	1.924	4.793
2	0.286	95.384	0.000	4.330	0.000	0.000	0.000	0.000
3	0.000	0.535	99.465	0.000	0.000	0.000	0.000	0.000
4	2.499	0.961	0.000	94.733	1.115	0.192	0.000	0.500
5	23.915	1.996	0.000	5.917	54.867	5.917	6.373	1.015
6	23.708	0.963	0.000	4.349	13.654	46.275	9.875	1.176
7	23.895	2.060	0.000	0.432	4.453	5.184	60.685	3.290
8	37.318	4.807	0.000	4.016	14.168	4.048	27.577	8.065

Average Recall: 67.801%

Actual class	Predicted class							
	1	2	3	4	5			
1	70.653	2.732	0.000	0.996	1.736	1.221	18.869	3.793
2	0.327	98.570	0.490	0.245	0.000	0.000	0.368	0.000
3	0.000	0.344	99.415	0.000	0.000	0.000	0.241	0.000
4	6.104	2.368	0.000	89.345	0.740	0.000	0.629	0.814
5	19.967	4.575	0.000	4.739	56.405	5.686	8.170	0.458
6	9.840	5.737	0.000	6.318	8.351	39.288	27.560	2.905
7	16.070	1.662	0.000	0.384	0.554	3.410	77.877	0.043
8	33.623	2.154	0.000	2.700	14.401	2.507	41.466	3.150

Average Recall: 66.838%

Actual class	Predicted class							
	1	2	3	4	5			
1	81.388	0.905	0.000	0.483	1.599	0.000	10.860	4.766
2	0.700	97.386	0.000	1.914	0.000	0.000	0.000	0.000
3	0.000	0.235	99.725	0.039	0.000	0.000	0.000	0.000
4	4.632	3.076	0.000	90.095	0.980	0.034	0.101	1.082
5	32.822	1.748	0.000	2.046	45.993	1.108	12.958	3.325
6	17.949	1.508	0.000	2.866	20.098	20.852	31.335	5.392
7	19.240	0.582	0.000	0.153	1.317	0.061	77.819	0.827
8	37.379	2.054	0.000	0.436	10.831	0.467	40.025	8.808

Average Recall: 65.258%

Actual class	Predicted class							
	1	2	3	4	5			
1	83.832	1.754	0.000	0.275	2.270	0.447	2.993	8.428
2	1.668	96.956	0.059	1.141	0.117	0.000	0.000	0.059
3	0.000	2.167	97.558	0.000	0.069	0.000	0.000	0.206
4	1.576	0.945	0.000	95.763	0.035	0.000	0.000	1.681
5	31.523	1.282	0.000	2.225	48.190	1.584	5.995	9.201
6	33.643	2.133	0.000	2.546	12.384	30.822	11.386	7.086
7	26.471	2.094	0.000	0.490	4.590	2.406	49.599	14.349
8	42.627	2.157	0.000	0.314	14.784	2.157	20.667	17.294

Average Recall: 65.002%

Mean Average Recall: 66.060%
Standard Deviation: 1.207

Position: Hips

Actual class	Predicted class							
	1	2	3	4	5			
1	98.264	0.000	0.000	0.161	0.643	0.161	0.418	0.354
2	0.000	99.857	0.036	0.107	0.000	0.000	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000	0.000	0.000	0.000
4	8.443	1.240	0.000	81.593	3.681	3.882	1.000	0.160
5	23.775	0.000	0.000	0.196	64.706	6.275	1.176	3.873
6	18.100	0.113	0.000	2.903	23.944	39.894	4.600	10.445
7	33.398	0.000	0.000	0.386	15.976	0.386	35.455	14.401
8	36.321	0.000	0.000	0.965	21.786	2.832	6.660	31.435

Average Recall: 68.901%

Actual class	Predicted class							
	1	2	3	4	5			
1	97.313	0.000	0.000	0.036	1.344	0.799	0.182	0.327
2	0.000	99.060	0.000	0.776	0.000	0.163	0.000	0.000
3	0.000	0.749	99.144	0.107	0.000	0.000	0.000	0.000
4	2.038	5.075	0.654	82.507	2.653	6.882	0.192	0.000
5	21.709	0.000	0.000	0.000	61.064	7.878	1.155	8.193
6	16.649	0.000	0.000	0.463	15.579	60.820	1.497	4.991
7	25.523	0.000	0.000	0.133	6.979	0.366	51.811	15.188
8	25.838	0.285	0.000	0.316	16.856	7.938	11.765	37.002

Average Recall: 73.590%

Actual class	Predicted class							
	1	2	3	4	5			
1	91.803	0.000	0.000	0.000	1.157	0.579	6.236	0.225
2	0.286	99.510	0.000	0.204	0.000	0.000	0.000	0.000
3	0.000	0.206	99.656	0.138	0.000	0.000	0.000	0.000
4	5.475	0.888	0.185	84.388	1.517	4.550	2.960	0.037
5	6.471	0.000	0.000	0.098	77.451	0.882	14.346	0.752
6	4.720	0.036	0.000	0.871	38.126	32.462	20.370	3.413
7	15.814	0.000	0.000	0.298	12.532	0.171	70.716	0.469
8	21.826	0.643	0.000	0.000	27.901	2.443	43.009	4.179

Average Recall: 70.021%

Actual class	Predicted class							
	1	2	3	4	5			
1	95.385	0.000	0.000	0.121	2.262	1.478	0.603	0.151
2	0.327	99.533	0.000	0.000	0.000	0.140	0.000	0.000
3	0.000	0.196	99.804	0.000	0.000	0.000	0.000	0.000
4	4.970	7.978	0.000	80.325	3.685	2.941	0.000	0.101
5	23.231	0.000	0.000	0.085	69.096	3.026	0.298	4.263
6	13.122	0.226	0.000	0.943	29.299	44.910	4.336	7.164
7	20.006	0.000	0.000	1.042	8.915	0.092	52.298	17.647
8	22.969	0.124	0.000	0.031	26.393	4.949	12.481	33.053

Average Recall: 71.801%

Actual class	Predicted class							
	1	2	3	4	5			
1	96.113	0.000	0.000	0.000	2.786	0.516	0.378	0.206
2	0.000	98.361	0.000	1.200	0.234	0.205	0.000	0.000
3	0.000	0.275	99.278	0.172	0.275	0.000	0.000	0.000
4	2.486	4.832	0.210	85.679	3.326	3.256	0.175	0.035
5	23.454	0.000	0.000	0.000	72.021	1.395	1.998	1.131
6	7.602	0.000	0.000	0.550	32.852	38.837	13.829	6.330
7	11.720	0.000	0.000	0.178	9.225	0.045	78.209	0.624
8	21.098	0.000	0.000	0.667	21.882	3.176	40.667	12.510

Average Recall: 72.626%

Mean Average Recall: 71.388%
Standard Deviation: 1.911

Position: Torso

Actual class	Predicted class							
	1	2	3	4	5			
1	92.382	0.000	0.000	0.000	1.832	0.000	0.064	5.722
2	0.178	99.822	0.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	14.646	17.047	0.000	59.144	5.722	1.761	0.040	1.641
5	26.912	0.000	0.000	3.431	62.549	1.078	0.000	6.029
6	41.742	0.377	0.000	3.356	18.439	12.293	7.730	16.063
7	22.179	0.257	0.000	0.000	1.318	0.000	72.742	3.504
8	39.932	0.000	0.000	0.000	8.248	0.934	5.384	45.503

Average Recall: 68.054%

Actual class	Predicted class							
	1	2	3	4	5			
1	91.830	0.000	0.000	0.000	3.195	0.000	0.000	4.975
2	0.000	99.101	0.041	0.817	0.000	0.041	0.000	0.000
3	0.000	0.570	99.430	0.000	0.000	0.000	0.000	0.000
4	4.614	12.764	0.000	71.972	6.036	4.191	0.077	0.346
5	25.105	0.000	0.000	1.576	60.749	1.401	0.000	11.169
6	31.230	0.357	0.000	3.494	17.219	18.075	5.276	24.349
7	19.807	0.233	0.000	0.000	0.897	0.000	74.111	4.952
8	28.336	0.253	0.000	0.253	6.673	2.119	8.602	53.763

Average Recall: 71.129%

Actual class	Predicted class							
	1	2	3	4	5			
1	85.085	0.514	0.000	0.000	14.401	0.000	0.000	0.000
2	0.286	98.815	0.000	0.449	0.449	0.000	0.000	0.000
3	0.000	0.206	99.690	0.034	0.069	0.000	0.000	0.000
4	6.622	16.796	0.000	65.668	5.956	4.846	0.111	0.000
5	20.654	0.000	0.000	2.974	71.275	4.248	0.850	0.000
6	36.347	1.670	0.000	10.857	20.407	28.431	2.288	0.000
7	44.928	0.000	0.000	0.000	14.578	5.627	34.868	0.000
8	51.109	0.000	0.000	0.129	41.144	5.079	2.539	0.000

Average Recall: 60.479%

Actual class	Predicted class							
	1	2	3	4	5			
1	92.217	0.000	0.000	0.000	7.481	0.000	0.000	0.302
2	0.140	99.346	0.000	0.000	0.187	0.000	0.000	0.327
3	0.000	0.118	99.882	0.000	0.000	0.000	0.000	0.000
4	7.268	20.926	0.270	60.920	5.815	4.733	0.000	0.068
5	22.080	0.000	0.043	2.387	74.552	0.639	0.000	0.298
6	40.988	0.302	0.000	2.225	31.523	14.555	2.903	7.504
7	20.558	0.613	0.000	0.000	5.484	0.582	68.842	3.922
8	27.949	0.000	0.000	0.124	22.378	4.482	7.937	37.130

Average Recall: 68.431%

Actual class	Predicted class							
	1	2	3	4	5			
1	96.354	0.000	0.000	0.894	2.752	0.000	0.000	0.000
2	0.468	98.390	0.000	0.410	0.732	0.000	0.000	0.000
3	0.000	0.000	99.759	0.000	0.069	0.172	0.000	0.000
4	3.852	5.112	0.140	83.929	3.116	2.871	0.980	0.000
5	78.092	0.000	0.000	0.452	15.385	1.961	4.110	0.000
6	62.779	0.103	0.000	5.435	8.875	8.187	14.620	0.000
7	60.027	0.000	0.000	0.045	1.381	0.000	38.547	0.000
8	85.098	0.000	0.000	0.784	3.569	1.176	9.373	0.000

Average Recall: 55.069%

Mean Average Recall: 64.632%
Standard Deviation: 6.653

Conf 2 - 3D accelerometer (no pre-processing)

Position: Bag

Actual	Predicted class							
class	1	2	3	4	5			
1	82.964	0.000	0.000	0.000	0.000	3.922	0.000	13.115
2	0.000	99.251	0.000	0.250	0.321	0.000	0.000	0.178
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	0.000	0.960	0.000	97.759	0.000	0.000	0.360	0.920
5	5.000	0.000	0.000	0.000	86.765	8.235	0.000	0.000
6	2.036	0.000	0.000	0.000	1.621	96.342	0.000	0.000
7	0.000	0.000	0.000	0.129	0.000	0.000	99.743	0.129
8	0.000	0.000	0.000	0.218	0.000	0.000	0.871	98.911

Average Recall: 95.217%

Actual	Predicted class							
class	1	2	3	4	5			
1	80.828	0.000	0.000	0.000	0.000	9.913	0.000	9.259
2	0.449	79.167	0.000	0.163	0.000	18.750	0.000	1.471
3	0.000	0.214	99.679	0.000	0.000	0.107	0.000	0.000
4	0.000	2.230	0.000	97.770	0.000	0.000	0.000	0.000
5	1.786	0.000	0.000	0.000	74.020	24.195	0.000	0.000
6	0.000	0.000	0.000	0.000	1.747	98.253	0.000	0.000
7	0.000	0.166	0.000	0.000	0.000	0.000	99.834	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	100.00

Average Recall: 91.194%

Actual	Predicted class							
class	1	2	3	4	5			
1	81.871	0.000	0.000	0.000	0.000	9.932	0.000	8.197
2	0.531	98.080	0.000	1.389	0.000	0.000	0.000	0.000
3	0.000	0.000	99.656	0.344	0.000	0.000	0.000	0.000
4	0.000	3.330	0.000	96.633	0.000	0.000	0.000	0.037
5	4.608	0.000	0.000	0.098	75.033	20.261	0.000	0.000
6	2.651	0.000	0.000	0.000	0.182	97.168	0.000	0.000
7	0.000	0.000	0.000	7.417	0.000	0.000	92.327	0.256
8	0.000	0.000	0.000	0.707	0.000	0.000	0.129	99.164

Average Recall: 92.492%

Actual	Predicted class							
class	1	2	3	4	5			
1	80.090	0.000	0.000	0.000	0.000	9.140	0.000	10.769
2	0.000	75.817	1.120	0.373	0.000	22.689	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	0.000	5.612	0.000	94.151	0.000	0.000	0.237	0.000
5	2.174	0.000	0.000	0.000	62.958	34.868	0.000	0.000
6	0.000	0.000	0.000	0.000	1.508	98.492	0.000	0.000
7	0.000	0.092	0.000	0.000	0.000	0.000	99.908	0.000
8	0.000	0.000	0.000	0.280	0.000	0.000	1.712	98.008

Average Recall: 88.678%

Actual class	Predicted class							
	1	2	3	4	5			
1	90.437	0.000	0.000	0.000	0.000	6.054	0.000	3.509
2	0.000	97.717	0.000	0.615	0.000	1.551	0.000	0.117
3	0.000	0.000	99.656	0.000	0.000	0.344	0.000	0.000
4	0.000	3.396	0.000	95.448	0.000	0.000	0.210	0.945
5	1.923	0.000	0.000	0.038	77.225	20.814	0.000	0.000
6	0.000	0.000	0.000	0.000	0.482	99.518	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	100.00 0	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.941	99.059

Average Recall: 94.883%

Mean Average Recall: 92.493%
Standard Deviation: 2.710

Position: Hand

Actual class	Predicted class							
	1	2	3	4	5			
1	68.177	0.289	0.129	0.000	5.625	0.675	22.244	2.861
2	1.426	68.770	2.068	17.469	4.207	1.283	1.569	3.209
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	3.201	1.681	0.000	95.118	0.000	0.000	0.000	0.000
5	14.755	0.049	0.000	0.294	83.480	0.000	1.422	0.000
6	2.489	0.000	0.000	1.056	0.000	78.582	10.106	7.768
7	1.639	0.064	0.000	0.000	6.332	0.000	91.385	0.579
8	15.780	3.859	0.000	1.183	1.214	2.148	32.088	43.729

Average Recall: 78.655%

Actual class	Predicted class							
	1	2	3	4	5			
1	70.733	0.182	0.145	0.000	5.338	4.357	17.611	1.634
2	0.735	63.194	6.863	11.438	6.944	2.247	7.149	1.430
3	0.000	0.000	99.430	0.036	0.000	0.535	0.000	0.000
4	0.346	4.844	0.000	94.771	0.000	0.000	0.000	0.038
5	14.671	0.105	0.000	0.000	82.003	0.000	3.221	0.000
6	1.462	0.570	0.000	0.285	0.357	72.478	13.226	11.622
7	6.215	0.000	0.000	0.000	11.167	0.000	79.827	2.792
8	19.481	2.119	0.190	0.443	1.550	1.645	16.129	58.444

Average Recall: 77.610%

Actual class	Predicted class							
	1	2	3	4	5			
1	69.495	1.221	0.000	0.257	4.436	1.286	19.994	3.311
2	0.776	77.042	2.369	16.258	1.471	0.776	0.000	1.307
3	0.310	0.000	99.450	0.000	0.000	0.241	0.000	0.000
4	0.518	4.440	0.000	94.303	0.000	0.185	0.000	0.555
5	16.699	2.157	0.000	0.065	78.072	0.294	2.712	0.000
6	2.505	1.525	0.000	1.053	0.000	71.750	11.946	11.220
7	0.469	0.384	0.000	0.000	6.650	0.341	91.858	0.298
8	24.365	3.118	0.000	0.321	1.093	1.189	20.701	49.212

Average Recall: 78.898%

Actual class	Predicted class							
	1	2	3	4	5			
1	59.457	1.056	0.000	0.090	6.486	0.513	31.101	1.297
2	1.074	80.719	2.708	12.325	0.327	2.007	0.373	0.467
3	0.000	0.000	99.765	0.000	0.000	0.235	0.000	0.000
4	0.000	4.767	0.237	94.726	0.000	0.000	0.000	0.270
5	13.683	0.639	0.000	0.000	85.507	0.000	0.000	0.171
6	0.377	2.074	0.000	0.754	0.377	73.416	10.935	12.066
7	3.125	0.306	0.000	0.000	18.290	0.000	77.941	0.337
8	18.705	7.314	0.000	0.000	5.011	1.089	22.596	45.285

Average Recall: 77.102%

Actual class	Predicted class							
	1	2	3	4	5			
1	61.576	0.894	0.000	2.167	6.742	0.034	23.942	4.644
2	1.200	77.319	2.634	12.028	3.746	2.634	0.205	0.234
3	0.000	0.000	99.656	0.000	0.000	0.344	0.000	0.000
4	0.000	1.891	0.000	97.654	0.175	0.175	0.000	0.105
5	9.201	0.679	0.000	0.000	89.517	0.075	0.528	0.000
6	1.548	1.238	0.000	0.757	1.238	59.064	16.340	19.814
7	1.203	0.713	0.045	0.089	6.105	1.070	88.592	2.184
8	13.922	1.294	0.157	0.353	2.235	1.333	22.824	57.882

Average Recall: 78.908%

Mean Average Recall: 78.234%
Standard Deviation: 0.828

Position: Hips

Actual class	Predicted class							
	1	2	3	4	5			
1	86.885	0.000	0.000	0.000	0.000	13.115	0.000	0.000
2	0.000	99.501	0.000	0.143	0.000	0.357	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	0.000	0.480	0.000	96.519	0.000	3.001	0.000	0.000
5	5.000	0.000	0.000	0.000	95.000	0.000	0.000	0.000
6	32.692	0.264	0.000	1.810	0.000	65.234	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	100.00	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	65.079	34.921

Average Recall: 84.758%

Actual class	Predicted class							
	1	2	3	4	5			
1	87.037	0.000	0.000	0.000	0.000	12.963	0.000	0.000
2	0.000	99.265	0.000	0.000	0.000	0.735	0.000	0.000
3	0.000	0.463	99.465	0.000	0.071	0.000	0.000	0.000
4	1.153	0.231	0.000	89.120	0.000	7.536	0.115	1.845
5	1.786	0.000	0.000	0.000	96.989	1.225	0.000	0.000
6	40.428	0.000	0.000	0.357	0.000	59.216	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	100.00	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	61.290	38.710

Average Recall: 83.725%

Actual class	Predicted class							
	1	2	3	4	5			
1	83.607	0.000	0.000	0.000	0.000	16.393	0.000	0.000
2	0.000	99.265	0.000	0.082	0.000	0.654	0.000	0.000
3	0.000	0.069	99.725	0.000	0.000	0.206	0.000	0.000
4	0.000	0.000	0.000	93.637	0.000	4.477	0.000	1.887
5	3.333	0.000	0.000	0.000	96.667	0.000	0.000	0.000
6	39.107	0.000	0.000	1.344	0.000	59.550	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	77.195	22.805
8	0.000	0.000	0.000	0.000	0.000	0.000	30.537	69.463

Average Recall: 84.889%

Actual class	Predicted class							
	1	2	3	4	5			
1	86.576	0.030	0.000	3.167	0.000	10.226	0.000	0.000
2	0.000	98.133	0.000	0.327	0.000	1.541	0.000	0.000
3	0.000	0.000	100.000	0.000	0.000	0.000	0.000	0.000
4	0.000	0.101	0.372	93.915	0.000	2.535	0.270	2.806
5	1.918	0.000	0.000	0.256	97.826	0.000	0.000	0.000
6	22.247	0.000	0.000	2.074	3.167	72.511	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	85.846	14.154
8	0.000	0.000	0.000	0.000	0.000	0.000	29.318	70.682

Average Recall: 88.186%

Actual class	Predicted class							
	1	2	3	4	5			
1	82.456	0.000	0.000	0.000	0.000	17.544	0.000	0.000
2	0.000	98.625	0.000	0.234	0.000	1.141	0.000	0.000
3	0.000	0.275	99.381	0.000	0.344	0.000	0.000	0.000
4	0.000	0.000	0.000	95.728	0.000	2.486	0.000	1.786
5	1.923	0.000	0.000	0.000	98.077	0.000	0.000	0.000
6	34.159	0.000	0.000	0.241	0.000	65.600	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	100.00	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	66.000	34.000

Average Recall: 84.233%

Mean Average Recall: 96.303%
Standard Deviation: 1.389

Position: Torso

Actual class	Predicted class							
	1	2	3	4	5			
1	96.175	0.000	0.000	0.000	0.546	3.279	0.000	0.000
2	0.321	99.572	0.000	0.107	0.000	0.000	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	15.286	0.000	0.000	84.714	0.000	0.000	0.000	0.000
5	0.637	0.000	0.000	0.000	86.373	4.951	0.000	8.039
6	23.454	0.452	0.000	0.075	11.916	54.940	6.938	2.225
7	11.636	0.000	0.000	0.000	3.922	14.497	64.834	5.111
8	22.876	0.000	0.000	0.000	13.072	13.725	7.874	42.453

Average Recall: 78.633%

Actual class	Predicted class							
	1	2	3	4	5			
1	92.593	0.000	0.000	0.000	0.036	6.863	0.000	0.508
2	1.225	98.243	0.000	0.286	0.000	0.245	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	4.191	0.000	0.000	95.194	0.308	0.308	0.000	0.000
5	1.821	0.000	0.000	0.000	90.651	1.786	0.000	5.742
6	5.062	0.250	0.000	0.000	24.670	53.832	12.692	3.494
7	8.408	0.066	0.000	0.000	17.647	9.305	64.008	0.565
8	13.662	0.949	0.000	0.253	26.407	16.477	15.655	26.597

Average Recall: 77.640%

Actual class	Predicted class							
	1	2	3	4	5			
1	92.993	0.000	0.000	0.000	0.000	7.007	0.000	0.000
2	1.021	98.652	0.000	0.286	0.000	0.041	0.000	0.000
3	0.034	0.206	99.690	0.000	0.000	0.069	0.000	0.000
4	10.729	0.000	0.000	89.271	0.000	0.000	0.000	0.000
5	0.817	0.000	0.000	0.000	93.301	1.993	0.229	3.660
6	29.085	2.070	0.000	0.000	17.284	30.138	9.586	11.837
7	18.755	0.000	0.000	0.000	9.037	5.627	57.374	9.207
8	32.883	0.000	0.000	0.000	25.522	9.579	11.443	20.572

Average Recall: 72.749%

Actual class	Predicted class							
	1	2	3	4	5			
1	86.154	0.000	0.000	0.000	3.409	10.437	0.000	0.000
2	0.980	98.319	0.000	0.000	0.000	0.700	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	6.863	0.034	0.000	90.297	0.000	1.724	0.000	1.082
5	0.000	0.000	0.000	0.000	90.452	8.312	0.000	1.236
6	3.771	0.415	0.000	0.075	2.677	78.544	10.897	3.620
7	10.815	0.000	0.000	0.000	7.261	28.431	39.246	14.246
8	7.656	0.000	0.000	0.000	29.381	34.298	6.754	21.911

Average Recall: 75.615%

Actual class	Predicted class							
	1	2	3	4	5			
1	95.631	0.000	0.000	0.138	0.550	3.509	0.000	0.172
2	0.468	98.244	0.000	1.083	0.000	0.088	0.000	0.117
3	0.275	0.000	99.725	0.000	0.000	0.000	0.000	0.000
4	4.867	0.000	0.000	95.133	0.000	0.000	0.000	0.000
5	1.357	0.000	0.000	0.000	81.523	3.356	0.075	13.688
6	20.330	0.619	0.000	0.103	12.006	53.044	4.919	8.978
7	16.176	0.000	0.000	0.000	8.021	16.578	48.708	10.517
8	21.922	0.000	0.000	0.000	8.431	8.431	9.843	51.373

Average Recall: 77.923%

Mean Average Recall: 76.512%
Standard Deviation: 2.383

Conf 3 - 3D accelerometer (with pre-processing)

Position: Bag

Actual class	Predicted class 1	2	3	4	5			
1	95.596	0.000	0.000	0.000	0.000	4.404	0.000	0.000
2	0.000	99.144	0.000	0.749	0.000	0.107	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	0.000	0.560	0.000	99.440	0.000	0.000	0.000	0.000
5	5.000	0.000	0.000	0.000	93.137	1.863	0.000	0.000
6	0.000	0.189	0.000	0.000	0.603	99.208	0.000	0.000
7	0.000	0.257	0.000	0.000	0.000	0.000	99.743	0.000
8	0.000	0.031	0.000	0.000	0.000	0.000	1.401	98.568

Average Recall: 98.105%

Actual class	Predicted class 1	2	3	4	5			
1	89.434	0.073	0.000	0.000	2.179	7.698	0.000	0.617
2	0.000	99.755	0.000	0.245	0.000	0.000	0.000	0.000
3	0.000	0.321	99.643	0.000	0.000	0.036	0.000	0.000
4	0.154	1.884	0.000	97.885	0.000	0.000	0.000	0.077
5	1.786	0.000	0.000	0.000	96.008	2.206	0.000	0.000
6	0.000	0.000	0.000	0.000	1.961	98.039	0.000	0.000
7	0.000	0.000	0.000	0.199	0.000	0.000	99.801	0.000
8	2.214	0.032	0.000	0.000	0.000	0.000	2.151	95.604

Average Recall: 97.021%

Actual class	Predicted class 1	2	3	4	5			
1	85.921	0.225	0.000	0.000	5.529	8.325	0.000	0.000
2	0.286	98.529	0.000	1.185	0.000	0.000	0.000	0.000
3	0.000	0.069	99.690	0.241	0.000	0.000	0.000	0.000
4	0.000	2.812	0.000	96.522	0.000	0.000	0.000	0.666
5	3.333	0.000	0.000	0.000	92.778	3.889	0.000	0.000
6	0.000	0.000	0.000	0.000	0.617	99.383	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	100.00	0.000
8	0.000	0.000	0.000	0.611	0.000	0.000	1.639	97.750

Average Recall: 96.322%

Actual class	Predicted class 1	2	3	4	5			
1	92.127	0.000	0.000	0.000	4.615	3.258	0.000	0.000
2	0.233	98.319	0.000	1.214	0.000	0.233	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	0.000	4.023	0.000	95.301	0.000	0.000	0.000	0.676
5	2.174	0.000	0.000	0.000	94.928	2.899	0.000	0.000
6	0.000	0.000	0.000	0.000	1.320	98.680	0.000	0.000
7	0.000	0.000	0.000	0.092	0.000	0.000	99.908	0.000
8	0.000	0.000	0.000	0.218	0.000	0.000	0.000	99.782

Average Recall: 97.381%

Actual class	Predicted class							
	1	2	3	4	5			
1	95.322	0.482	0.000	0.000	0.378	3.818	0.000	0.000
2	0.000	98.771	0.000	0.907	0.000	0.322	0.000	0.000
3	0.000	0.000	99.656	0.000	0.000	0.344	0.000	0.000
4	0.000	2.696	0.000	97.094	0.000	0.000	0.000	0.210
5	1.923	0.000	0.000	0.000	96.418	1.659	0.000	0.000
6	0.000	0.000	0.000	0.000	1.204	98.796	0.000	0.000
7	0.000	0.000	0.000	0.223	0.000	0.000	99.777	0.000
8	3.098	0.157	0.000	0.000	0.000	0.000	0.000	96.745

Average Recall: 97.822%

Mean Average Recall: 97.330%
Standard Deviation: 0.699

Position: Hand

Actual class	Predicted class							
	1	2	3	4	5			
1	71.681	2.764	0.000	0.000	4.436	0.289	16.136	4.693
2	0.570	94.367	0.428	0.677	1.711	0.463	0.000	1.783
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	0.320	0.280	0.000	99.400	0.000	0.000	0.000	0.000
5	13.824	0.539	0.000	0.294	84.755	0.000	0.588	0.000
6	2.451	0.189	0.000	2.112	0.000	74.057	9.842	11.350
7	2.122	0.836	0.000	0.000	6.493	0.000	82.064	8.486
8	22.129	1.836	0.000	1.369	2.552	0.685	16.060	55.369

Average Recall: 82.712%

Actual class	Predicted class							
	1	2	3	4	5			
1	66.231	3.595	0.000	0.000	4.139	3.740	20.261	2.033
2	0.735	97.467	0.490	0.408	0.000	0.735	0.123	0.041
3	0.000	0.000	99.465	0.000	0.000	0.535	0.000	0.000
4	0.000	0.692	0.000	98.847	0.000	0.423	0.000	0.038
5	11.870	1.576	0.000	0.000	82.458	0.000	4.027	0.070
6	1.141	0.963	0.036	0.071	0.000	79.073	13.547	5.169
7	0.100	0.399	0.000	0.000	6.181	0.000	91.758	1.562
8	17.805	2.593	0.000	0.380	0.854	1.202	20.462	56.705

Average Recall: 84.001%

Actual class	Predicted class							
	1	2	3	4	5			
1	67.181	4.564	0.000	0.225	5.014	1.189	18.611	3.214
2	0.000	96.528	0.858	1.838	0.041	0.449	0.000	0.286
3	0.034	0.275	99.690	0.000	0.000	0.000	0.000	0.000
4	0.000	0.296	0.000	99.260	0.000	0.000	0.000	0.444
5	13.464	1.503	0.000	0.196	81.732	0.196	2.908	0.000
6	1.997	1.743	0.000	1.380	0.000	76.507	11.765	6.609
7	0.426	1.364	0.000	0.000	4.007	0.128	92.796	1.279
8	22.758	3.311	0.000	2.154	0.579	0.096	18.611	52.491

Average Recall: 83.273%

Actual class	Predicted class 1	2	3	4	5			
1	59.487	5.189	0.000	0.000	4.706	0.271	26.184	4.163
2	0.654	96.125	0.093	0.840	0.000	2.288	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	0.135	0.879	0.000	98.546	0.000	0.439	0.000	0.000
5	12.106	0.256	0.000	0.000	81.969	0.085	5.584	0.000
6	0.943	1.546	0.000	1.282	0.000	75.943	10.935	9.351
7	3.186	0.674	0.000	0.000	9.589	0.000	78.646	7.904
8	18.643	1.867	0.000	0.622	6.225	1.120	22.004	49.518

Average Recall: 80.029%

Actual class	Predicted class 1	2	3	4	5			
1	65.600	5.573	0.000	0.000	6.570	0.138	20.674	1.445
2	1.054	91.045	0.059	1.902	4.097	0.819	0.176	0.849
3	0.000	0.413	99.243	0.000	0.000	0.344	0.000	0.000
4	0.000	0.560	0.000	99.300	0.000	0.000	0.000	0.140
5	7.919	0.264	0.000	0.000	87.029	0.113	4.676	0.000
6	1.961	1.479	0.034	2.339	0.000	60.062	16.443	17.681
7	2.406	1.961	0.000	0.401	3.699	0.267	90.775	0.490
8	16.235	0.824	0.000	0.471	2.431	0.471	21.569	58.000

Average Recall: 81.382%

Mean Average Recall: 82.279%
Standard Deviation: 1.582

Position: Hips

Actual class	Predicted class 1	2	3	4	5			
1	93.250	0.000	0.000	0.193	0.000	6.557	0.000	0.000
2	0.000	99.715	0.000	0.000	0.000	0.285	0.000	0.000
3	0.000	0.532	99.468	0.000	0.000	0.000	0.000	0.000
4	1.160	0.000	0.000	95.918	0.000	2.921	0.000	0.000
5	1.569	0.000	0.000	0.000	95.000	3.431	0.000	0.000
6	26.885	0.000	0.000	0.302	0.000	72.813	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	97.589	2.411
8	0.000	0.000	0.000	0.000	0.000	0.000	47.401	52.599

Average Recall: 88.294%

Actual class	Predicted class 1	2	3	4	5			
1	79.847	0.000	0.000	0.000	0.000	20.153	0.000	0.000
2	0.000	99.428	0.000	0.327	0.000	0.245	0.000	0.000
3	0.000	1.783	98.217	0.000	0.000	0.000	0.000	0.000
4	0.269	0.000	0.000	96.963	0.000	0.807	0.000	1.961
5	0.000	0.000	0.000	0.000	98.214	1.786	0.000	0.000
6	5.348	0.000	0.000	0.963	3.137	90.553	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	78.465	21.535
8	0.000	0.000	0.000	0.000	0.000	0.000	32.385	67.615

Average Recall: 88.663%

Actual	Predicted class							
class	1	2	3	4	5			
1	94.407	0.000	0.000	0.643	0.000	4.950	0.000	0.000
2	0.204	99.510	0.041	0.000	0.000	0.245	0.000	0.000
3	0.000	0.378	99.622	0.000	0.000	0.000	0.000	0.000
4	2.775	0.000	0.000	95.117	0.000	0.222	0.000	1.887
5	0.229	0.000	0.000	0.000	96.667	3.105	0.000	0.000
6	5.882	0.000	0.000	4.321	0.472	89.325	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	94.885	5.115
8	0.000	0.000	0.000	0.000	0.000	0.000	14.529	85.471

Average Recall: 94.376%

Actual	Predicted class							
class	1	2	3	4	5			
1	87.089	0.000	0.000	5.611	0.000	7.300	0.000	0.000
2	0.000	98.646	0.000	0.327	0.000	1.027	0.000	0.000
3	0.000	0.196	99.804	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	93.915	0.000	2.637	0.000	3.448
5	0.980	0.000	0.000	0.000	97.826	1.194	0.000	0.000
6	17.986	0.000	0.000	0.339	1.772	79.902	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	87.347	12.653
8	0.000	0.000	0.000	0.000	0.000	0.000	15.282	84.718

Average Recall: 91.156%

Actual	Predicted class							
class	1	2	3	4	5			
1	98.142	0.000	0.000	0.344	0.000	1.514	0.000	0.000
2	0.000	98.712	0.000	0.059	0.000	1.229	0.000	0.000
3	0.000	3.234	96.457	0.000	0.310	0.000	0.000	0.000
4	0.000	0.000	0.000	97.059	0.000	1.155	0.000	1.786
5	0.000	0.000	0.000	0.000	98.077	1.923	0.000	0.000
6	12.109	0.000	0.000	1.204	2.855	83.832	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	97.148	2.852
8	0.000	0.000	0.000	0.000	0.000	0.000	35.098	64.902

Average Recall: 91.791%

Mean Average Recall: 90.856%
Standard Deviation: 2.486

Position: Torso

Actual	Predicted class							
class	1	2	3	4	5			
1	84.378	0.000	0.000	0.257	7.040	6.075	0.000	2.250
2	0.000	99.537	0.000	0.107	0.000	0.357	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	9.684	0.000	0.000	83.673	0.000	4.442	0.000	2.201
5	0.000	0.000	0.000	0.000	93.627	5.196	0.000	1.176
6	0.641	0.000	0.000	0.226	11.312	71.003	5.204	11.614
7	6.750	0.032	0.000	0.000	4.854	17.840	64.642	5.882
8	4.108	0.747	0.000	0.031	33.302	14.846	6.816	40.149

Average Recall: 79.626%

Actual class	Predicted class							
	1	2	3	4	5			
1	86.275	0.000	0.000	0.363	0.000	7.807	0.000	5.556
2	0.000	98.938	0.000	1.062	0.000	0.000	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	2.499	0.000	0.000	96.732	0.000	0.769	0.000	0.000
5	0.035	0.000	0.000	0.000	71.674	9.944	0.035	18.312
6	1.462	0.071	0.000	0.000	2.852	74.866	16.257	4.492
7	7.644	0.000	0.000	0.000	0.399	21.967	64.706	5.284
8	11.037	0.759	0.000	0.253	7.211	22.138	13.694	44.908

Average Recall: 79.762%

Actual class	Predicted class							
	1	2	3	4	5			
1	91.578	0.000	0.000	0.129	0.000	8.197	0.000	0.096
2	0.000	98.897	0.000	0.327	0.000	0.776	0.000	0.000
3	0.000	0.206	99.656	0.000	0.000	0.138	0.000	0.000
4	7.325	0.000	0.000	90.011	0.000	2.590	0.000	0.074
5	0.000	0.000	0.000	0.000	84.935	9.869	0.000	5.196
6	3.123	0.508	0.000	0.000	7.662	81.409	5.011	2.288
7	9.165	0.000	0.000	0.000	2.899	23.743	60.273	3.922
8	7.650	0.000	0.000	0.032	16.940	33.269	12.279	29.830

Average Recall: 79.574%

Actual class	Predicted class							
	1	2	3	4	5			
1	85.882	0.000	0.000	0.000	10.226	3.529	0.000	0.362
2	0.000	98.273	0.000	0.887	0.000	0.840	0.000	0.000
3	0.000	0.000	100.00	0.000	0.000	0.000	0.000	0.000
4	6.051	0.034	0.000	90.061	0.000	2.366	0.000	1.487
5	0.000	0.000	0.000	0.000	92.413	6.394	0.000	1.194
6	3.243	0.566	0.000	0.151	5.279	79.751	5.581	5.430
7	10.938	0.000	0.000	0.000	9.865	30.576	37.163	11.458
8	2.832	0.187	0.000	0.031	34.765	33.987	7.034	21.164

Average Recall: 75.588%

Actual class	Predicted class							
	1	2	3	4	5			
1	82.215	0.000	0.000	0.172	13.691	1.926	0.000	1.995
2	0.263	97.951	0.000	1.493	0.000	0.029	0.000	0.263
3	0.310	0.000	99.690	0.000	0.000	0.000	0.000	0.000
4	1.751	0.000	0.000	95.238	0.000	0.525	0.000	2.486
5	0.000	0.000	0.000	0.000	93.552	3.469	0.000	2.979
6	3.234	1.926	0.000	0.654	11.524	56.828	3.406	22.429
7	10.918	0.000	0.000	0.000	5.793	28.164	49.332	5.793
8	7.569	1.059	0.000	0.000	22.980	12.275	10.118	46.000

Average Recall: 77.601%

Mean Average Recall: 78.430%
Standard Deviation: 1.822

Ringraziamenti

A conclusione di questo percorso vorrei dedicare un pensiero a tutte le persone che in qualche modo mi sono state vicino e mi hanno sostenuto. Questa tesi probabilmente non sarebbe altrimenti potuta esistere.

Ringrazio Romina, la persona che più di tutte mi ha accompagnato in questi ultimi tre anni (sappiamo che sono di più), con cui ho condiviso non solo le gioie ma anche le incertezze, le paure e, talvolta, lo sconforto che spesso hanno minato questo intenso percorso ormai volto al termine. Grazie per aver cacciato sempre via quei momenti.

Ringrazio i miei genitori, che nemmeno per un istante hanno dubitato di me e sempre hanno sostenuto e incoraggiato le mie scelte. Grazie per avermi trasmesso tutti i vostri valori, che ora formano anche il mio modo di essere uomo.

Ringrazio i miei amici-fratelli Aldo, Antonio, Francesco, Gerardo, Giovanni, Mattia. Grazie per tutte le risate e i momenti felici passati assieme.

Ringrazio l'allegra compagnia dell'aula studio: Carmine, Gio, Guido, Leo, Peppe, Rapesta, Simone. Anche se negli ultimi tre anni mi sono trasferito di un piano, lo spirito dell'*how we made* è sempre rimasto con me.

Ringrazio il prof. Licciardo, che prima di tutti ha puntato su di me e che ha reso possibile non solo la mia crescita professionale, ma anche umana.

Ringrazio Danilo che, insieme al prof. Licciardo, è stato mia guida e mentore a partire dai miei primi passi nel percorso di dottorato. Grazie per aver fatto restare sempre accesa in me la passione per la ricerca.

Ringrazio tutti quelli che più o meno stabilmente hanno condiviso con me le ore di lavoro al laboratorio di Microelettronica.

Ringrazio tutte le persone che ho incontrato al TIMA di Grenoble, in particolare Frédéric ed Olivier. Grazie per avermi guidato in quei mesi e per avermi dato nuovi spunti di riflessione. Ringrazio anche Breytner e Bruno per avermi accolto al TIMA non solo come collega ma anche come amico.

Ringrazio infine tutti i dottorandi del Dipartimento di Ingegneria Industriale con cui ho condiviso quest'esperienza, insieme ai docenti che ci hanno assistito.

Acknowledgements

At the end of this path, I would like to give a thought to all the people who have always been there to support me. Probably, this thesis could not have come into existence without them.

Thank you to Romina, who more than anyone else has accompanied me throughout the last three years (more than three to be honest). With her I shared not only the joys, but also the uncertainties, the fears, and sometimes the discouragement, which have often undermined the intensive journey that has reached its conclusion now. Thanks for always having thrown away those moments-

Thank you to my parents, who never doubted me, not even for a moment, and always supported and encouraged my choices. Thanks for having conveyed to me your values, which have shaped my way to be a man.

Thank you to my friends-brothers Aldo, Antonio, Francesco, Gerardo, Giovanni, Mattia. Thanks for all the laughter and the happy moments we shared.

Thank you to the "aula studio" company: Carmine, Gio, Guido, Leo, Peppe, Rapesta, Simone. Even though I moved downstairs, the *how we made* spirit has always been with me.

Thank you to prof. Licciardo, who before all has bet on me and has made it possible not only my professional growth, but also my growth from a human perspective.

Thank you to Danilo, who has been a guide and a mentor starting from my first steps in the Ph.D. adventure. Thanks for keeping alive in me the passion for the scientific research.

Thank you to all those who more or less permanently have shared with me many working hours at the Microelettronica laboratory.

Thank you to all the people who I met at TIMA, in Grenoble, in particular Frédéric and Olivier. Thanks for having guided me in those months and for having given me many new insights. I also would like to thank Breytner and Bruno for having welcome me at TIMA, not only as a colleague but as friend as well.

Finally, I would like to say thank you to all the Ph.D. students at the Dipartimento di Ingegneria Industriale, together with all the professors who always assisted us.