technische universität
dortmund

Binary Temporal Convolutional Networks for the Logistic Activity Recognition Challenge (LARa) dataset

Bachelor Thesis

Emmanuel Kembou Nguefack

August 22, 2022

Supervisors:
Prof. Dr.-Ing. Gernot A. Fink
Fernando Moya, M.Sc.

Fakultät für Informatik
Technische Universität Dortmund
http://www.cs.uni-dortmund.de

CONTENTS

1 INTRODUCTION

Understanding the lifestyle and behavior of human beings requires the observation of their movements. This is one of the main goals of artificial intelligence research. These last years have witnessed the growing number of techniques and approaches that rely on the artificial intelligence model to understand human behavior [NRMR+20]. One of these state-of-the-art approaches is HAR. It is a part of deep learning which consists in recognizing humans' actions or movement through different sources, e.g. videos, images, sensor data [BBS13]. The ability to recognize human behavior is important in various areas of life. Therefore, HAR is deployed in several domains such as logistics, health, technology, security, and communications.

One way to predict these human activities is to use data labeled from sensors as an input of HAR machine learning model. Similarly to neuronal networks,HAR machine learning networks are previously trained with specific training data to recognize different activities. One of the neural networks that can improve the training process is the convolutional neural network according to [NRMR+20]. By its structure, this network is well suited to effectively recognize human activities using sensor data. However, deploying a method for HAR using deep learning is not an easy task as it faces a great challenge. Running a CNN-based HAR method can only be executed on larger devices with huge GPUs which require more capacity and financial investment and not everybody can afford these state-of-the-art GPUs. Therefore, several works have been launched to reduce the memory consumption of the HAR machine. The main goal is to make HAR work on the smallest devices (including smartphones or other end devices) to not only improve this area of machine learning, but to make the benefits of the HAR accessible to all sectors of human activities.

Many works have been proposed for solving HAR problems. Therefore, [NZ21] proposed achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training but this could lead to the reduction in the precision of the parameters and data, which easily hurt a model's task accuracy. However, a state-of-the-art results model was proposed by [CB16]. The idea is to drastically reduce the input data from 32 fp to 1 bit, i.e. to reduce 32 times the memory consumption of the HAR model called Binarization. The latter is a 1-bit quantization where data can only have two possible values, namely -1 or $+1$. So, it consists in updating the weight of the layer from full-precision models number to 1 or -1. This thesis therefore aims at using this state-of-the-art method for solving HAR issues. The LARa dataset proposed by [NRMR+20] is used in this work.

This work deals with the neural network described in [NRMR+20], which is used to detect human activities in a warehouse called TCNN. The main goal of this work is to binarize the convolution layers of the tCNN, which has been previously trained in the LARa dataset to solve the HAR, then analyze its performance compared to normal tCNN. The Layers of the tCNN will be trained independently. However, before presenting relatively good performance compared with the baseline or state of the art of this experiment, this thesis will begin by giving a brief description of how the HAR works, then the related work trying to solve the difficulties of HAR, and then use the binarization method, to train tCNN on LARa dataset while giving a brief overview of the LARa and tCNN architecture. It will be therefore a question of observing if the drastic diminution of the number of bits will have an impact on the network's performance.

2 FUNDAMENTALS

2.1 HAR

Human activity recognition (HAR) assigns human action labels to signals of movements. It could be defined as the pattern recognition of identifying the specific movement or action of a person based on sensor data.
The application of Human Activity Recognition extends to various domains, including medicine (monitoring disease progression and rehabilitation support through humans' movement), sports (understanding and interpreting athletes' behavior and actions, or tracking performance with fitness trackers) or smart homes (increasing the house security. HAR can also be used in the industry or logistics to understand workers' actions and behavior. The resulting optimizations and improvements through analytics help industries to improve their productivity or the working environment. However, HAR is a very challenging task to master [BBS13].
Due to the high level of *intra- and inter-class variability* in human actions, HAR appears to be a particularly challenging field. This is because the same activity may be performed differently by different people, while some fundamentally different activities may have very similar characteristic properties in sensor data. One of the expensive and tedious issues is the *Ground Truth annotation*. For HAR, some annotations are done in real time and the labeling manually. Moreover, motion data recorded from one sensor is often more difficult to interpret than data from other sensors. It is also difficult to classify a movement according to its characteristics because the activities are performed differently in different environments, depending on different contexts and reasons. This diversity of physical activities remains a specific issue for HAR. Another difficulty is that, when creating a balanced data set, it must always be considered that the movements are not always the same for all humans because some activities occur more frequently for some than others.[BBS13] talks about *class imbalance* which depends on the level of activity to be recognized by a particular HAR system and requires recording additional training data. Another major issue is *Data Collection and Experiment Design*. Properly designing and conducting a HAR experiment can be underestimated at first sight and data collection may require more equipment and logistics. Other challenges are *Variability in Sensor Characteristics* and *Tradeoffs in*

Human Activity Recognition System Design. Some sensors are particularly sensitive to the environment. The possibility of errors or hardware failures, sensor drifts cannot be excluded.
The first step of HAR Methods [RNMR+19] is *data representation*. Different data representations in Human Activity Recognition can be recorded using various sensors and cameras. This includes sensors integrated in IMUs set on the human body, e.g. on the hands, legs, head, and torso. Acceleration sensors are placed for example on the waist to record the acceleration measurement, and Motion Capturing System like OMOCAP are deployed for the local human joint poses. These movements are often typical activities performed indoors, such as walking, standing, etc. There could also be additional activities like those performed in logistics, e.g. picking and packing(items). The subjects are taken to a laboratory where these devices are installed on parts of their bodies Figure2.1.1. Then they perform different movements that will be classified and assigned to activity classes. During the experiment, a series of sensor curves will be drawn for each subject, describing the activities during the experiment. It is also important to note that the number of records and times are annotated and included in the dataset.
After the representation, it is time for the *pre-processing* [RNMR+19]. Factors that may influence the data are simply filtered and eliminated. These factors are related to the different characteristics of sensors. Thus, different filters are applied for smoothing the signal, reducing or eliminating random noise and separating the acceleration components due to the gravity. [RNMR+19] also talk about normalizing the extracted Data -for example to the range- before the training stage.
Segmentation refers to extracting a sequence of the pre-processed data that are plausible to represent a human activity. For HAR, they mostly use the "sliding-window" for creating segments, which are moved over the time-series data by a certain step to extract a segment or being processed by a classifier [RNMR+19]. In [RNMR+19], there will also be other approaches for segmenting sequences using additional measurements or events for some additional parts of the body, e.g. eyes.
Divided into two main groups - statistical features and application-based -, the *Feature Extraction* is the standard pattern recognition method, which allows representing data in a compact manner for the later classification stages[RNMR+19].
A *feature reduction* is deployed for reducing the dimensionality of the feature space, keeping the discriminant properties of the features. Therefore facilitating classification [RNMR+19].
Another method of HAR is *classification*. There are several classification models mentioned in [RNMR+19].

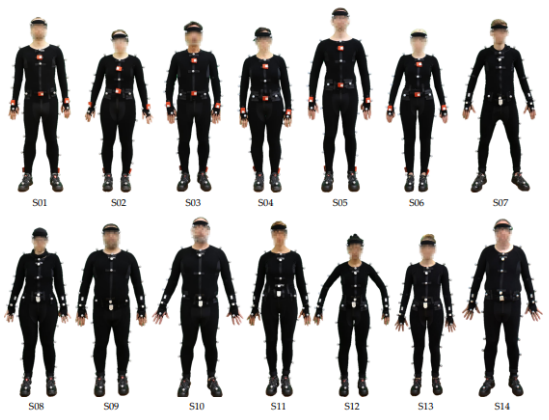


Figure 2.1.1: A Graphic of subjects with sensors on their bodies. Taken from [NRMR+20].

2.2 CLASSIFICATION

Classification is the process of predicting the class of given data points. There are therefore one or more classifications of the activity including the associated pseudo-probability as output. Examples of classifiers include Naïve Bayes, k-Nearest Neighbor (KNN), Support Vector Machines (SVMs), Random Forests, Dynamic Time Warping (DTW), and Hidden Markov Models (HMMs) [RNMR+19]. According to [BBS13], while the best results for all available data are achieved by SVM , the worst performance is held by naive Bayes, k-NN hast lowest recall, the k-NN and HMM classifiers, however, achieved the higher precision. One of the most used models is the Hidden Markov Model (HMM), which is trained by [LC11] per axis of pre-processed acceleration measurements merged with a weighted sum. The hidden state at time t depends on the previous state at time t-1. The observed variable at time t depends on the state at time t. The goal is to find the joint probability distribution[Sar14]. Therefore, the aim of the classification is to find a description that is as suitable as possible for the movement. In addition to the simple loss classification , the precision, the recall and the accuracy are metrics of such classification methods, which indicate the correct classification . The average metric across all object classes is referred to as mean precision and mean accuracy.
Deep Learning overcomes some issues regarding computation and adaptability of handcrafted features. It is a state-of-the-art method for solving HAR, which combines extraction and classification. The accuracy here is better because, their features are directly learned from data.

2.3 NEURAL NETWORKS

Just as many inventions in engineering were inspired by real life, e.g. planes and birds, so was the machine learning. Machine learning engineers based themselves on the human brain (Figure2.3.1) (responsible for learning) to create a network called the neural network. The brain consists in several neurons which transfer information in the form of electrical impulses. These neurons are composed of three main parts (Figure2.3.1): dendrites, cell body and axon. Dendrites are the main conducting wires through which information from the outside passes. They absorb the information and pass it onto the cell body, which is made up of the control center, that processes information received from the dendrites. The axon is the common wire that carries the output signal from the cell body to other neurons. The first artificial neuron in computer science was published in 1943 by Warren S. Mc Culloch and Walter Pitts [MP43] and was represented by a simple electrical circuit. The McCulloch-Pitts neuron can receive n binary signals as input and output a single binary signal.

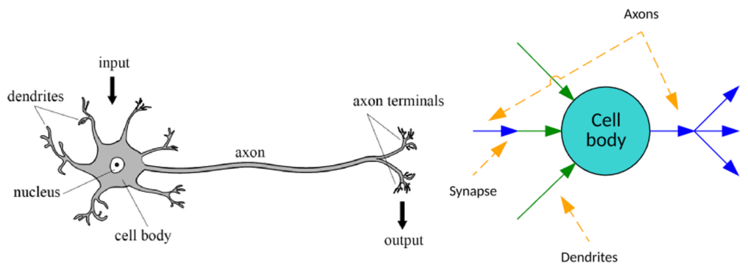


Figure 2.3.1: Structure of a biological neuron taken from [NGLK18] and [Jon17] respectively.

2.3.1 *Perceptron*

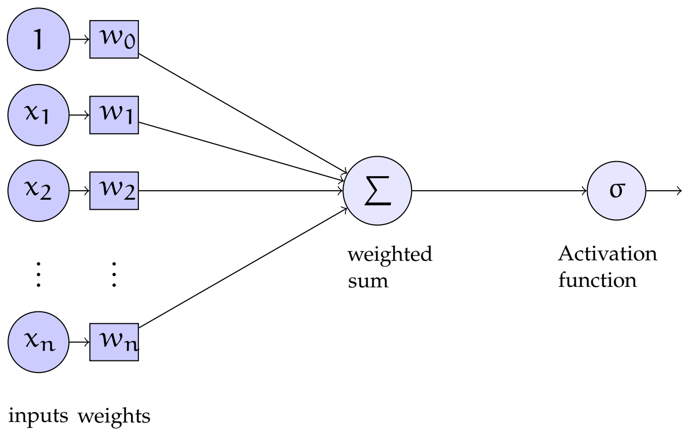


Figure 2.3.2: Representation of a Basic Perceptron Neural Network.

In 1958, Frank Rosenblatt created the perceptron [Ros58] , a simple neural model that could be used to classify data into two sets. Based on the McCulloch-Pitts neuron, the perceptron is an ANN consisting of only a single neuron, which can solve linear separable problems. Figure 2.3.1 shows a simple perceptron that includes n input vector $x \in \mathbb{R}^n$, the associated weights $w \in \mathbb{R}^n$ and a bias weight. The perceptron sums the products of the inputs and their associated weights and bias, and then applies that result through an activation function. Figure2.3.1 , the activation function is a function with the binary output $f(x) \in \mathbb{B}$. If this output is greater than or equal to 1, then the output is 1, otherwise, the output is 0.

In the literature, there are different variants that are equivalent in terms of functionality, but differ in their presentation (e.g. if the bias is used). The advantages of incorporating the bias into the weight vector is necessary during the optimization of NNs. This thesis consider a model of n binary input vectors $1, ..., X_n$ and exactly the same number of weights $W_1, ..., W_n$, that are multiply together and sum up. The input vector and the weight vector look like $x = [1x_1x_2...x_m]$ and $[w = w_0w_1w_2...w_m]$. Thus, $w^T x = w_0 + w_1x_1 + w_2x_2 + ... + w_mx_m$. Considering a as activation,

$$a = \sum_{i=1}^{n} w_ix_i + b = w^Tx + b \tag{2.3.1}$$

Mathematically , the bias b can actually be incorporated into the weight vector, i.e. $W_1 = b$ and set $x_0 = +1$ for all of our inputs. However, it will not influence the model, but it will be helpful for the further process.

$$a = \sum_{i=0}^{n} w_i x_i = w^T x \tag{2.3.2}$$

The activation function σ is applied to the activation in equation2.3.2 to obtain the output activation z.

$$z = \sigma(a) = \sigma\left(\sum_{i=0}^{n} w_i x_i\right) = \sigma\big(w^T x\big). \tag{2.3.3}$$

The calculation of the output values can be described with the following function:

$$\sigma(a) = \begin{cases} 1, & \text{if } a \geqslant 0 \\ 0, & \text{if otherwise} \end{cases} \tag{2.3.4}$$

Perceptron training follows this general rule. First, the weights of the network are initialized to a random set of values. Then are iterated over a training set until no further errors occur. Then apply a training vector to the network and execute the network in order to get an output value. This output will be subtracted from the desired output , thus the error value is obtained. This error, together with a learning rate, is then used to adjust the weight multiplied by the input associated with the given weight. This process will be repeated many times until the error becomes minimal or zero.

2.3.2 *MLP*

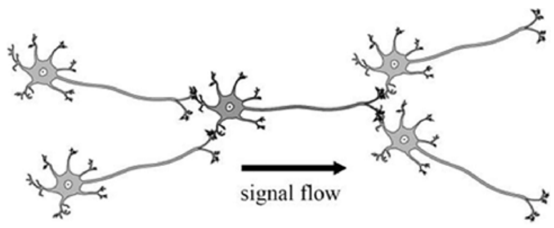


Figure 2.3.3: Structure of a biological neuron connected together taken from [NGLK18].

A multi-layer perceptron or MLP is an ANN consisting of several sequentially arranged perceptrons, the neurons are arranged in several layers and the neurons within a layer are not connected to each other. As modeled on the human brain (Figure2.3.3), every neuron in a layer is linked to every neuron in the following layer and there are three types of layers: the input layer, which gives the input vector $x \in \mathbb{R}^n$, the hidden layers, which adjust the data taken from the input layers and the output layer, which provides the final result $\hat{y} \in \mathbb{R}$ of the computation. In Figure 2.3.4, there are 3 input layers and a single output layer. But in reality, it depends on the model of the framework in which the network is used because it can be many.There can be an infinite number of hidden layers and from 2 layers, is already possible to talk of a deep network. Each layer can also contain an infinity of neurons. Thus, the number of neurons or layers is mostly arbitrary. So the MLP can contain at least 3 layers (1 input, 1 hidden and 1 output).
The activation of the neuron can be determined in a similar way to that of the simple single neuron or perceptron. For example, the equation2.3.5 presents the activation of a neuron j (of the layer k) receiving a weight coming from the neuron i of the previous layer $k-1$. Where k represents the layer, j the neuron and i the neuron of the previous layer.

$$a_j^{(k)} = \sum_{i=0}^{n^{(k-1)}} w_{i,j}^{(k)} z_i^{(k-1)} = w_j^{(k)T} z^{(k-1)}, \tag{2.3.5}$$

Applying the activation function σ to the activation in equation2.3.2) will give the output of the same neuron:

$$z_j^{(k)} = \sigma^{(k)}\left(a_j^{(k)}\right) = \sigma^{(k)}\left(\sum_{i=0}^{n^{(k-1)}} w_{i,j}^{(k)} z_i^{(k-1)}\right) = \sigma^{(k)}\left(w_j^{(k)T} z^{(k-1)}\right), \tag{2.3.6}$$

Thus, the application of the activation function σ on a layer k will give a vector set, which is the combination of all the outputs of the previous neurons. However, these previous neurons also receive the weights combined with the outputs of all the preceding neurons except the first layer, which only receives inputs as demonstrated in the following Figure.

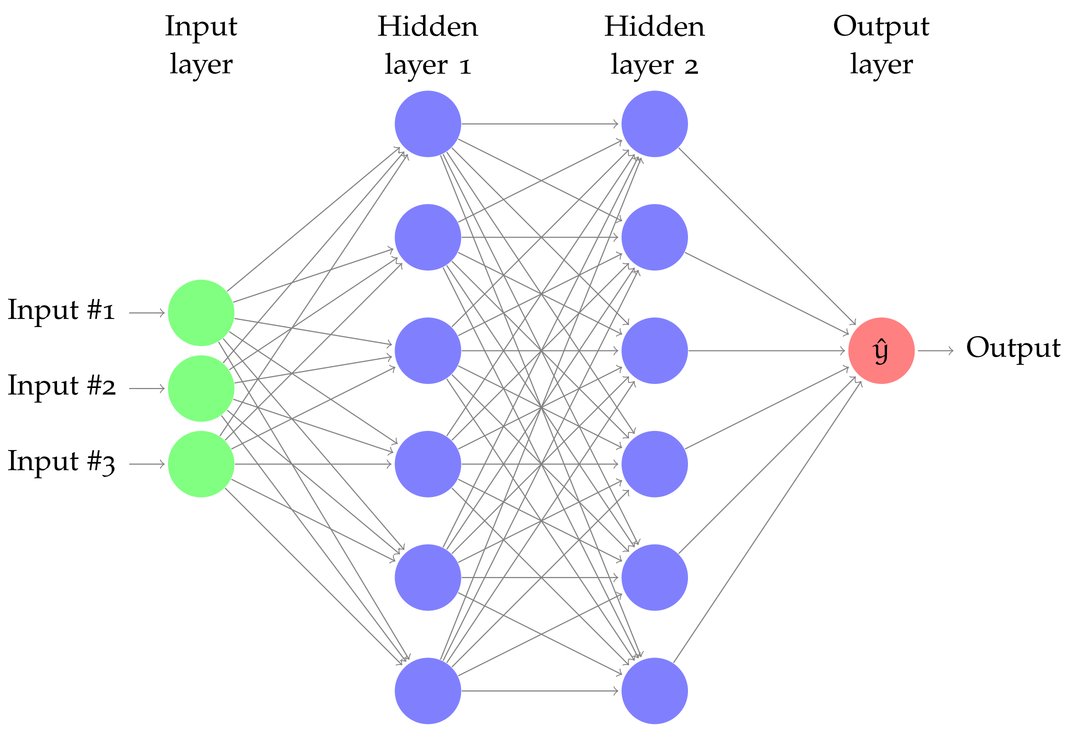


Figure 2.3.4: An example of MLP. The edges transfer the weights or output data from a neuron i to a neuron j of the next layer k. The nodes represent neurons.

Furthermore, there is a large number of activation functions.

2.3.2.1 *Activation function*

When the neurons calculate the weighted sum of the input values plus the bias (Equation 2.3.1), they are passed to the activation function, which checks whether the calculated value is above the required threshold. If the calculated value is greater than the required threshold, the activation function is activated, and an output value is calculated $(\sigma(a))$. This output value is transmitted to the next layers (depending on the complexity of the network) and will help neural networks to update the weight of the neurons. Thus, the activation function is a mathematical Equation that is activated under certain conditions.
Many activation functions have been proposed, but 2 most popular will be described in detail: sigmoid and tanh.

SIGMOID Historically, the sigmoid function is the oldest and most popular activation function. It is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.3.7}$$

Considering the equation2.3.1, i.e. if the variable a are first set to the weighted sum of the inputs,and then transmitted to the sigmoid function, the output z will be obtained as follows:

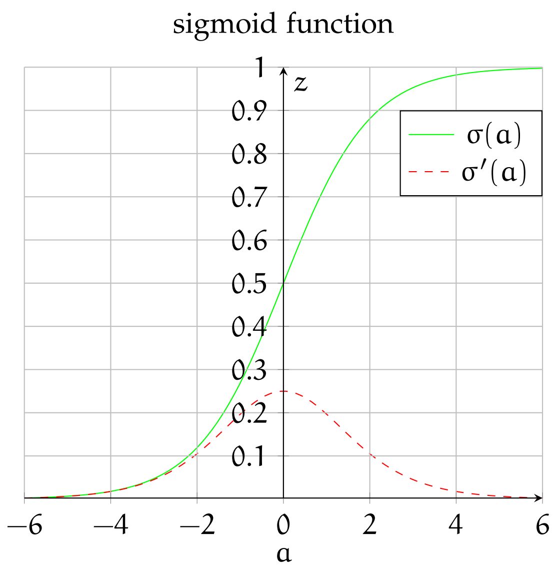
$$z = \sigma(a) = \frac{1}{1+e^{-a}} \tag{2.3.8}$$

The sigmoid function produces an S-shaped curve [Figure2.3.5]. Although non-linear in nature, it does not take into account slight variations in the inputs, which leads to similar results. The sigmoid function is continuously differentiable and the derivative is a function of the primitive $\sigma'(a) = \sigma(a)(1-\sigma(a))$ i.e. $z' = z(1-z)$. However, the sigmoid function poses a problem called vanishing gradient [Hoc98] [Dat20] [HBFS01]: when training on deeper neural networks the gradient would become smaller and smaller after multiple derivations,but only very small changes can be made based on those gradient, which leads to stagnation of the optimization of a neural network. As a result, the loss stops decreasing and the network does not get trained properly.

TANH FUNCTION The hyperbolic tangent became preferred over the sigmoid function because it gives better performance for multi-layer neural networks. The tanh function is defined as

$$z = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.3.9}$$

The tanh function has all the properties of the sigmoid function because it can be expressed as $\tanh(x) = 2 * \mathrm{sigmoid}(2x) - 1$. It gives negative, positive and zero as output(range of $(-1,1)$), so it solves the "not a zero-centered activation function "problem of the sigmoid function and produces zero centered output, thereby facilitating the back-propagation process.The gradient of tanh can be calculated using $\tanh'(x) = (1-\tanh(x)^2)$ otherwise $z = 1-z^2$. This is still a non-linear activation function (also belongs to the sigmoidal group of function), and that is why it still presents vanishing gradient problems similar to the sigmoids, when training on a large number of epochs. [Hoc98] [Dat20]. Two major problems arise from these mostly used activation function: the vanishing gradient problem (by sigmoid and tanh) and the dead neuron problem(ReLU) [Dat20].



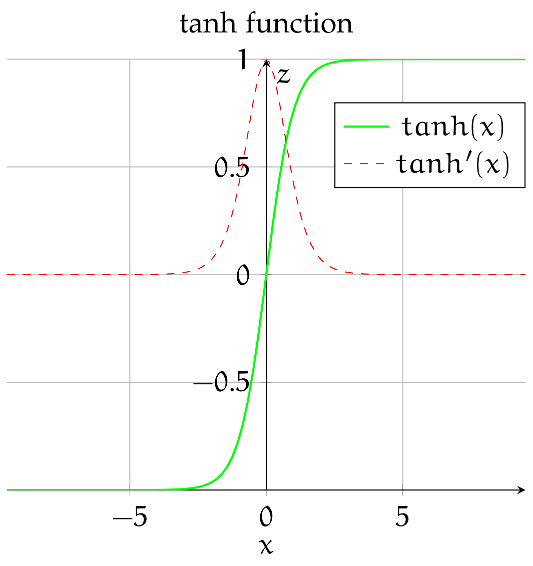


Figure 2.3.5: left: Graph of the tanh function, right: Graph of the gradient of tanh function. The graphics are inspired from stackexchange

2.3.2.2 *Backpropagation and Optimization*

BACKPROPAGATION Backpropagation is a process of the backward propagation of errors within a network. It is used in supervised learning in neural networks to calculate the mathematical derivatives of the error function with respect to the weights of the network. An input vector is applied to the network and propagated forward from the input layer to the hidden layer, which recognizes features in the input space and then to the output layer for a solution. The error value is propagated backward through the weights of the network beginning with the output neurons through the hidden layer and to the input layer. However, Backpropagation can be very efficient and has been used since the mid-1980s. Moreover, during backpropagation, the weight and other parameters are updated according to the gradient, allowing for learning/training of the data in a finite number of iterations. Thus, the Backpropagation method is used to calculate the error gradient for each neuron, from the last layer to the first (Figure 2.3.6).

The error for every output node is calculated independently of each other and the sum is dropped. The parameters are denoted as follows: x: input (feature vector), y: target output, so for a training set, there will be a set of input-output pairs, (x_i, y_i). An MLP f(x; W) whose parameters W are to be optimized, an error function $C(\hat{z}, z)$ (or shortly

C) that specifies the error for an input-output pair $(x, y) \in X$, the bias $b_j^{(k)}$ understood as a weight $w_{0,j}^{(k)}$ with constant input 1, W containing only the weights $w_{i,j}^{(k)}$. Therefore only the partial derivatives $\frac{\partial C}{\partial w_{ij}}$ have to be formed. Figure2.3.6 presents a view over Back and Forward propagation during training.

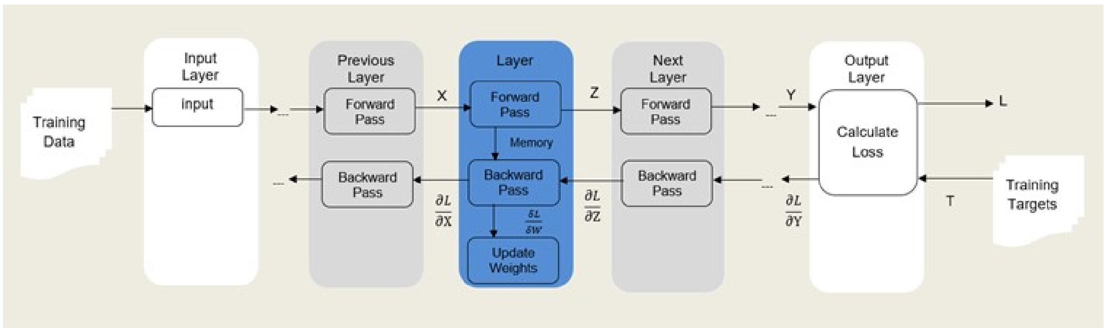


Figure 2.3.6: Backward- and forward-propagation during training of Neural Networks, inspired from [Aka20]

For each individual component gradient are given by:

$$\frac{\partial C}{\partial w_{ij}}, \tag{2.3.10}$$

can be calculated by the chain rule for the differentiation

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}}, \tag{2.3.11}$$

or more

$$\frac{\partial C}{\partial w_{ij}} = \underbrace{\frac{\partial C}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j}}_{=\sigma_j} \cdot \frac{\partial a_j}{\partial w_{ij}}, \tag{2.3.12}$$

with

$$\sigma_j = \frac{\partial C}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j}$$

so the components of $\frac{\partial C}{\partial w_{ij}}$ are given by

$$\frac{\partial C}{\partial w_{ij}} = \sigma_j \cdot \frac{\partial a_j}{\partial w_{ij}} = \sigma_j \cdot z_i, \tag{2.3.13}$$

Knowing that $z_j^{(k)} = \sigma^{(k)}\big(a_j^{(k)}\big)$ from Equation 2.3.6, the following equation can be written

$$\sigma_j = \frac{\partial C}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j} = C'(\hat{z}_j, z_j)\Phi'(a_j), \tag{2.3.14}$$

However, performing the equation 2.3.13 for each weight separately is inefficient.Thus on the jth neuron of the kth layer Equation2.3.14 will be rewritten as

$$\sigma_j^{(k)} = C'(\hat{z}_j, z_j)\Phi'^{(k)}(a_j^{(k)}), \tag{2.3.15}$$

The first term on the right, $C'(\hat{z}_j, z_j)$, only measures how fast the cost is changing as a function of the jth output activation. If, for example, C does not highly depend on a particular output neuron, j, then $\sigma_j^{(k)}$ will be small, which is what is needed. The second term on the right, $\Phi'^{(k)}(a_j^{(k)})$, measures how fast the activation function ϕ is changing at $a_j^{(k)}$.

Let's consider two possible paths that $\sigma_j^{(k)}$ can influence, the error C $(o_1^{(k)} and o_2^{(k)})$ and, according to the generalized chain rule, let's differentiate the two paths individually then add them

$$\frac{\partial C}{\partial z_i^{(k-1)}} = \sigma_1^{(k)} \cdot w_{i,1}^{(k)} + \sigma_2^{(k)} \cdot w_{i,2}^{(k)}, \tag{2.3.16}$$

then the $\frac{\partial C}{\partial z_j^{(k)}}$ components given by

$$\frac{\partial C}{\partial z_j^{(k)}} = \sum_{j=1}^{N^{k+1}} \sigma_j^{(k+1)} w_{ij}^{(k+1)} \tag{2.3.17}$$

Using 2.3.17 will give:

$$\sigma_j^{(k)} = \frac{\partial C}{\partial z_j^{(k)}} \cdot \frac{\partial z_j^{(k)}}{\partial a_j^{(k)}} = \sum_{j=1}^{N^{k+1}} \big(\sigma_j^{(k+1)} w_{ij}^{(k+1)}\big)\Phi'^{(k)}(a_j^{(k)}) \tag{2.3.18}$$

by using the following equations 2.3.15 and 2.3.18, the neuron error will be

$$\sigma_j^{(k)}, k = 1, ..., M.$$

$$\sigma_j^{(k)} = \begin{cases} C'(\hat{z}_j, z_j)\Phi'^{(k)}(a_j^{(k)}), & \text{if k=M} \\ \sum_{j=1}^{N^{k+1}} \big(\sigma_j^{(k+1)} w_{ij}^{(k+1)}\big)\Phi'^{(k)}(a_j^{(k)}), & \text{otherwise} \end{cases} \tag{2.3.19}$$

OPTIMIZATION Optimization is a branch of mathematics used to solve problems by determining the best element of a set according to certain predefined criteria. For example, the Adam and the Root Mean Square Przationon (RMSProp) optimization methods(presented below) were used in [RNMR+19] for training the CNNs, tCNNs, and RNNs. The network parameters can be optimized by minimizing the cross-entropy loss function using mini-batch gradient descent with the RMSProp update rule[MRGF+18], which adjusts the learning rate for each weight depending on the moving average of the squared gradients [Bus18]. The parameters of the networks are updated by minimizing the categorical cross entropy using the stochastic gradient descent with the RMSProp update rule. The RMSProp update rule is given by:

$$\mathbb{E}[g^2]_{new} = \gamma \mathbb{E}[g^2]_{old} + (1-\gamma)(\frac{\partial C}{\partial w})^2 \tag{2.3.20}$$

The aim of RMSProp is to keep a moving average of the squared gradient for each weight. However, it adjusts the learning rate for each weight depending on the moving average of the squared gradients [HSS12][Bus18][Rud16]

$$w_{new} = w_{old} - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_{new}}} \cdot \frac{\partial C}{\partial w} \tag{2.3.21}$$

RMSProp could consistently offer the best results with the widest tolerance to the learning rate setting after experiments with multiple per-parameter learning rate updates [OR16][MRGF+18]. E[g] is the moving average of squared gradients, $\frac{\partial C}{\partial w}$ the gradient of the cost function with respect to the weight, η learning rate and γ moving average parameter. [HSS12] suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.
Apart from storing an exponentially decaying average of past squared gradients v_t, like RMSprop, Adaptive Moment Estimation (Adam) also keeps an exponentially decaying average of past gradients m_t, similar to momentum[Rud16]:

$$m_{new} = \gamma_1 m_{old} + (1-\gamma_1))(\frac{\partial C}{\partial w}), \quad v_{new} = \gamma_2 \cdot v_{old} + (1-\gamma_2))(\frac{\partial C}{\partial w})^2 \tag{2.3.22}$$

However, defined as an Optimization Technique Algorithm for Gradient Descent, Adam is used to speed up the gradient descent algorithm by considering the "exponentially weighted average" of the gradients[Spa22].
The Adam update rule is given by [Rud16]:

$$w_{new} = w_{old} - \frac{\eta}{\sqrt{\hat{v}_{old}}} \hat{m}_{old}. \tag{2.3.23}$$

Where $\hat{m}_t = \frac{m_t}{1-\gamma_1^t}$, $\hat{v}_t = \frac{m_t}{1-\gamma_2^t}$ and γ_1 has 0.9 as default value, 0.999 for γ_2, which are both the decay rates. [Rud16] gives more details and improvement on Adam.
The loss function or cost function is established to solve the optimization problem during training. One of the most commonly used lSss funcEions (is)the Mean Squared Error (MSE), which measures the average of tIe squares ofs the errors. In other words, it is the average squared difference between the output values $\hat{y}$ and the label value y.

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.3.24}$$

The closer the y and $\hat{y}$ values are, the smaller the MSE. Hence the idea of minimizing the MSE through adjustments to y and $\hat{y}$. Therefore, it will suffice to bring the output value $\hat{y}$ closer to the expected value to minimize the function.
The cross entropy is the average number of bits needed to encode data coming from a source with distribution p when the model q is used [KA21]. It is used for multi-class and multi-label classifications. Given by

$$CE(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{2.3.25}$$

with and $y \in \mathbb{B}^c$ the target attribute representation and $\hat{y} \in \mathbb{B}^c$ the output of the architecture. CE is used to minimize the loss during training by updating model weight. A The authors in [NRMR+20] trained their model using the binary-cross entropy loss for binary classification given by

$$BCE(y, \hat{y}) = -\frac{1}{N} \sum_{j=1}^{N} y_j \log(\hat{y}_i) + (1 - y_j) \log(1 - \hat{y}_i) \tag{2.3.26}$$

2.3.3 *Convolutional Neural Networks*

AS initially described in Section 2.3.2, MLP are made up of neurons that have learnable weighs and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-lineaity function. Each neuron has an activation function and the network has a loss function on the last (fully connected) layer. In 1959, the authors of [HW59] developed the visual cortex of cats. In the process they discovered two types of cells that contributed to recognition. In 1989, based on these two insights, Yann LeCun developed the first convolutional neural network (CNN). The ConvNet architecture is therefore analogous to the connectivity model of neurons in the human brain. Convnet is a Deep Learning algorithm which can take in an input image, videos or sequences assign learnable weights and biases to various aspects/objects in this input in order to differentiate one from the other. ConvNet is a sequence of layers built in three main types: Convolutional Layer Pooling Layer, and Fully-Connected Layer. A closer view of the convolutional layers is given in Figure2.3.7.

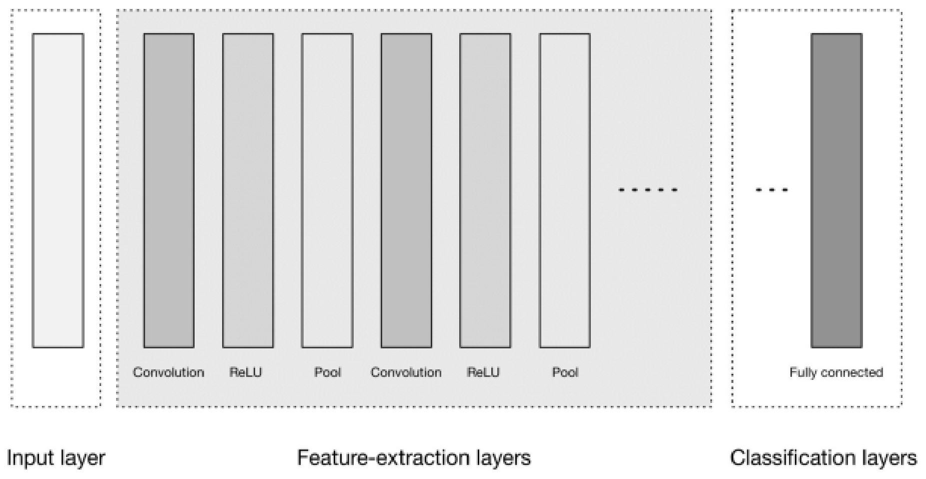


Figure 2.3.7: Schematic diagram of a High-level ConvNet(CNN) architecture taken from [PG17]

The architecture presented by the Figure2.3.7 is composed of an input layer to supply the network with input, followed by a convolutional layer, then follows a ReLU activation function, and after that a pool layer. The last layer is the classification layer or the fully connected layer.

2.3.3.1 *Convolution-Layer*

Convolutions are used to handle inputs. When considering an n-dimensional input, the input can be encoded with matrices having values. These values will correspond to a pixel and our input will be composed of those numbers. While sliding over the data input represented in matrices, the values obtain are multiply by the kernel values. Kernels or filters are learnable matrices with multidimensional weights, i.e. the weights can also be learned during the training of the network. The kernel values (or weights) are initialized and then updated by gradient backpropagation. The kernel is moved step by step over the n-dimensional input by computing the dot product of the kernel and the input elements covered at each step, and this is then turned into a weighted sum whose weights are the kernel values. Figure2.3.8 illustrates this in more details. In short, the result of a convolution is obtained by convolving the input values with the values (weights) of the kernel. This result will be called the "Featuremap", which gives information on the location of the features in the image: the higher the value, the more the corresponding place t in the image is similar to the feature. The convolutional layer can be set by adjusting the filter size F and the stride S. A stride S is a parameter that indicates the number of steps the matrix moves through after each operation or simply the size of the steps through which the matrix is shifted. For example, for $S = 1$, the matrix moves by one step. A filter of size $F \times F$ applied to an input containing N channels is a volume of size $F \times F \times N$ that convolves an input of size $L \times L \times N$ and produces an output feature map of size 0×01 given by the formula $0 \times 0 \times K$ where K is the number of filters applied. Zero-padding is a technique that adds P zeros to each side of the input boundaries. This margin helps control the spatial dimension of the output volume. Thus it is a technique that allows us to preserve the original input size. A correct formula for calculating the number of neurons is given by $(W - F + 2P)/S + 1$ according to [GBC16] The convolution of a feature for a 2-D input I with a two-dimensional kernel K of size m × n can be expressed by the following formula [GBC16]

$$S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n) \tag{2.3.27}$$

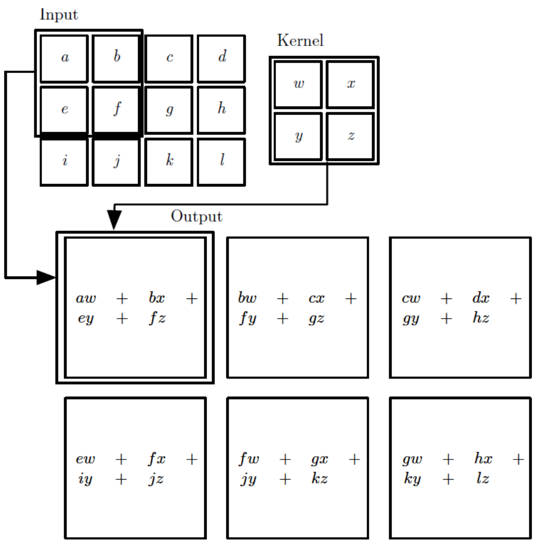


Figure 2.3.8: Graphic of a convolution operation applied to a 2-D tensor. The inputs are represented by alphabetic letters from a to l, the Kernel from w to z. The output is the feature map of input and Kernel. image taken from [GBC16]

A Convolution incorporates three important methods that -an help improve a machine-learning system, namely sparse interactions,parameter sharing and equivariant [GBC16].

2.3.3.2 *Pooling-Layer*

a convolutional layer is usually followed by a pooling layer. In a ConvNet architecture, a pooling layer is regularly inserted between successive conv layers. Its function is to control overfitting, by progressively reducing the spatial size of the representation and by reducing the number of parameters and computation in the network [LKBS20]. Hence, pooling is usually referred to as a downsampling operation. Here, the kernel itself does not contain any entries as in convolutional layers but is only needed to mark the areas to be summarized. In particular, the most commonly used types of pooling are max and average pooling, where the maximum and average values are taken, respectively. The most commonly used is the Max pooling because Max pooling keeps the detected features. To achieve this, the image is cut and then the maximum value is saved within each cell. In general, small square cells is used. The most common choices are adjacent cells of size 2×2 pixels which do not overlap, the choice of the size and stride of a pooling kernel can strongly impact the performance of the network and reducing the feature map size by 75% [LKBS20], the stride is normally set to 2, thus ensuring that each element appears at most once in a pooling operation. The maximum values are located less accurately in the feature maps obtained after pooling than in those received as input - which is actually a great benefit! In fact, to recognize a motorcycle for example, its handbrake lever or the mirrors do not need to be located as precisely as possible: knowing that they are located approximately next to the throttle is enough!

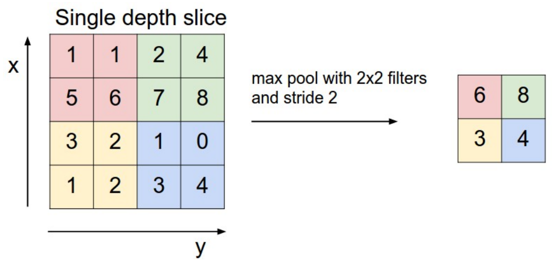


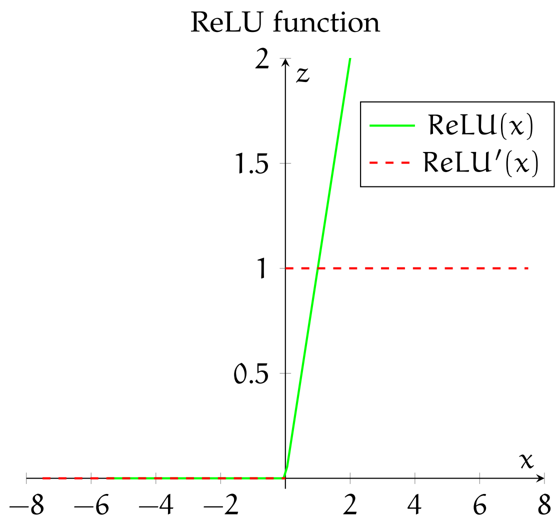
Figure 2.3.9: Example of a Downsampling operation –Every Max operation would in this case be taking a max over 4 numbers (with filter size 2, stride 2). Figure taken from [LKBS20].

2.3.3.3 *Fully-Connected-Layers*

Fully connected layers are typically present at the end of CNN architectures and can be used to classify input extracted from conv layers. FCs thus allow to determine the link ben the position of the input and the class[NRMR+20]. The FC layer receives a vector as input and produces a new vector as output. It does this by applying a linear combination and eventually an activation function to the values received as input. The fact that each input value is connected with all output values explains the term fully connected. The last fully connected layer is used to classify the input image of the network: it returns a vector $(x_1 x_2 ... x_N)$, where N is the number of classes in our image classification problem. Each element of the vector indicates the percentage of the input image that belongs to a class. For example, if the problem is to discriminate bikes and motorbikes, the output vector will be of size 2: the first element gives the probability of belonging to the "bike" class and the second to the "motorbike" class. Thus, the vector [0.20.8] means that the image has an 80% chances of being a motorbike. To calculate full probabilities, the Fully-connected layer therefore multiplies each input element by a weight, sums, and then applies an activation function. [NRMR+20] uses the softmax and sigmoid function for its experiment. The sigmoid function is generally used if the class number is equal to 2 (binary) and softmax if it is greater (multi-classification) [Bas20]. More details about these functions can be found in section2.3.2.1 and section2.3.3.5.

2.3.3.4 *Activation function*

After each convolution operation, a CNN performs a ReLU (Rectified Linear Unit) transformation on the feature map, which brings a non-linearity into the model [LKBS20]. Rectified Linear Units, or ReLUs, are a type of activation function that are linear in the positive dimension, but zero in the negative dimension. It is defined as $RELU(x) = max(0, x)$, where x is the input of the activation function. It is currently, the most successful and widely used activation function [Dat20] that enabled the fully supervised training of state-of-the-art deep networks. Thanks to the RELU the Deep networks are more easily optimized and it has become the default activation function used across the deep learning community [RZL17]. Unlike sigmoid functions, linearity in the positive dimension has the attractive property that it prevents non-saturation of gradients. The ReLU function is not differentiable at $x = 0$ and basically pushes the negative values to zero; it offers better performance in neural networks than the sigmoid and tanh activation functions and could run six times faster than sigmoid/tanh in terms of number of epochs required to train a network according to [Dat20]. The author of [Dat20] further states that there is no vanishing gradient problem in the ReLU because the derivative is exactly 0 or 1 every time. However, it suffers from the dead neuron, which is already mentioned at section 2.3.2.1. [Dat20] defines two new variants of the ReLU called the Leaky ReLU or LReLU and the parametric ReLU or PReLU to solve the dead neuron problem.



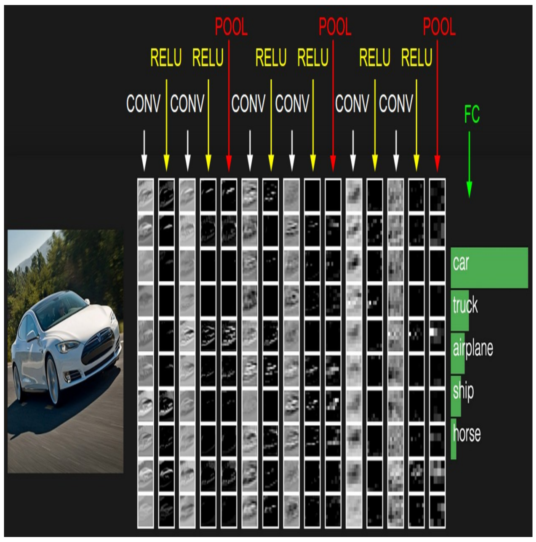


Figure 2.3.10: left the schematic Graph of the ReLU function and right the activations of an example CNN architecture. Image of architecture taken from [LKBS20]

2.3.3.5 *classification*

Fully-Connected-Layers usually use a Softmax or Sigmoid activation function to classify the inputs in the best way [NRMR⁺20].

SOFTMAX ACTIVATION FUNCTION Softmax activation function is similar to the Sigmoid function. It is a Sigmoid activation function that takes vectors of real numbers $x \in \mathbb{R}^n$ as inputs, and normalizes them into a probability distribution $p \in \mathbb{R}^n$ proportional to the exponential of the input numbers. The difference between the two functions is that not only one element should be considered, but all output data should be included. That is why all values in the denominator are summed: $p = (p_1, p_2, ..., p_n) \quad \text{where} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$

The Softmax function is given by:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \qquad \text{for} \quad i = 1, ..., N \tag{2.3.28}$$

After applying Softmax, each element will be in the range of 0 to 1. A Softmax activation is used in the Fully-Connected-Layer to classify the data [NRMR⁺20]. Idea is to map the non-normalized output of data to the probability distribution for output classes. With Softmax the sum of all the output probabilities is equal to one.

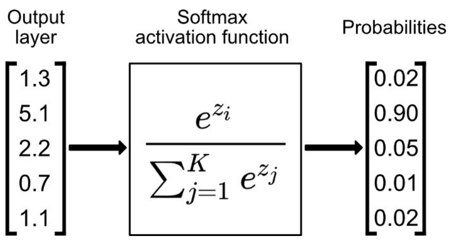

Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

Figure 2.3.11: Example of a softmax activation function diagram taken from [Rad20]

OUTPUT AND LOSS The loss function in a neural network is the difference between the predicted output and the actual output. Idea is to minimize the Loss for gut classification. Therefore the gradients(between the Loss and input) are derived to update the weights. For the multiclass classification for example, Convolutional neural networks are trained with either the log-loss or cross-entropy .
The cross-entropy is given by

$$L_{CE} = -\sum_{c=1}^{N} y_{i,c} \log(\hat{y}_{i,c}) \tag{2.3.29}$$

With $N > 2$ the number of classes, y the binary indicator $(0 or 1)$ if class label c is the correct classification for observation i. -therefore out of the whole sum only one term with $y_c = 1$ will actually be added - and $\hat{y}$ model's prediction of class c, i.e, the output of the softmax for class c.

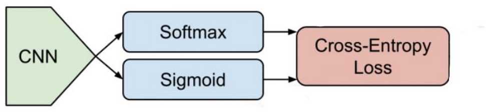


Figure 2.3.12: Cross-entropy function using the activation from [Gom18]

If the Softmax function with cross-Entropy Loss is used as the output layer, the Softmax function is derived or if the sigmoid with Cross-Entropy Loss is used as the output layer, the sigmoid function is derived. The derivation is to compute the gradient (between the Loss and the input) Equation 2.3.10.

2.3.3.6 *Overfitting and Regularization*

Overfitting or high variance in machine learning models happens when the accuracy of the dataset used to train the model (the training dataset), is higher than the accuracy of the tests. With regard to loss, overfitting appears when your model has a higher error in the test set and a low error in the training set. To treat Overfitting, a technique called Regularization is used, which optimizes a model by disfavoring complex models, thus minimizing losses and complexity. This makes the neural network simpler. One way to achieve this goal is to minimize the following function:

$$\min_{U,V} = \left\| \left(X - WY^T \right) \right\|_F^2 \tag{2.3.30}$$

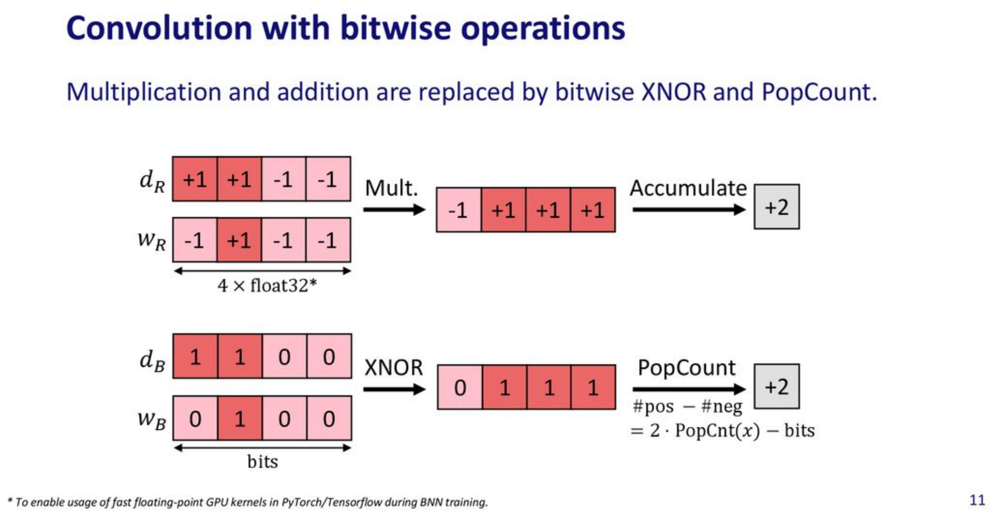
This formulation allows optimization by matrix factorization, leading to a structured factorization of X, where $\|.\|_F^2$ is the Frobenius norm, $X \in \mathbb{R}^{n\times m}$ designates the input data, $W \in \mathbb{R}^{m\times d}$ the weight matrix and $Y \in \mathbb{R}^{n\times d}$ the target labels. The Frobenius norm establishes a similarity between X and WY^T.
The regularization techniques are various as follows **regularization based on Data augmentation (l2)**, by bringing changes in the training data for CNNs producing an update $\theta \leftarrow \theta - \eta \nabla_\theta J_{train}^{old}(\theta) - \eta\lambda\theta$. The term $\eta\lambda\theta$ in the update leads the parameters θ slightly towards 0, adding some decay in the weights with each update, **Regularization L1** of CNNs $\theta \leftarrow \theta - \eta \nabla_\theta J_{train}^{old}(\theta) - \eta\lambda \cdot \mathrm{sign}(\theta)$. Which puts more probability feature close to 0 and the **Dropout**. [SP22] and [OZCR20] bring more details on the regularization of CNNs.

2.4 BINARY NEURAL NETWORKS

Binary networks are deep neural networks that use binary values as activation and weights, instead of full-precision values. And inside the networks all the weight and activation are binary, i.e. 32-bit floating-point input data are updated to logical -1 or logical 1. Thus, the X-nor operation works similarly to the multiplication operation. So, the multiplication will be replaced here by the X-nor (Figure2.4.1), which reduces the execution time and this very suitable for the GPUs, otherwise this can be implemented very efficiently on the Datase. The BBNs features are very compact and powerful. Thus no need to use powerful GPUs and memory for training and storing the model which cannot be supported by small devices or graphic card(because there is no more need to deal with floats.) instead, simple GPU can now be used to run the computations. Furthermore, the BBNs are cheaper to store and to compute the features(weight and activation are now represented in a single bit but no more in 32-bit floating-point and this is a big win fort the memory).
The architecture of a BBN is similar to the DDN architecture, except,the fact that the features are binarized to either -1 or $+1$. But one problem to be encountered is that,according to section2.3.2.2, normally for each inputs (in the learning process) NNs

weights and activations are multiplied and then accumulated in order to have a single value called gradient (the idea here is to determine the derivation between the output and the real values and from this error determine the gradient in order to update the weight: optimizing the parameter and for the gut result). In our case,however, weights and activations are represented by either -1 or $+1$. And the problem is that to obtain these gradients the sign function output has to be derived. But if the sign function output is derived, the result will be Zero because the sign function is not continued and differentiable. Thus,there will be no more Backpropagation and, therefore, no learning in the networks. In 2016, [CHS+16] used the straight-through estimator, to solve this problem. This means during the training, the Networks are based on 32 bits or simply data with large numbers, and when the network is used -during the forward pass, weight and activation will be binarized using the sign function. This means the output is based on binary numbers, but when the error is being computed and its result has to be given through the networks, real numbers can be used (or simply the 32 bit numbers). Therefore, the straight-through estimator is used by [CHS+16] and [HWG+19] for the Backpropagation, which makes the experiment faster.

This thesis considers the fact that because of the XNOR function, the logical 0 represents -1 and the logical 1 represents $+1$ which leads XNOR into a simple multiplication and then float addition will also be replaced with bit counting[Lin17]. The multiplication is replaced in Figure2.4.1 above by the X-nor. Because X-nor in the binary data works similarly to the multiplication operation in the 32-bit floating-point input. Thus, this reduces memory consumption and the number of operations.



Convolution with bitwise operations

Multiplication and addition are replaced by bitwise XNOR and PopCount.

* To enable usage of fast floating-point GPU kernels in PyTorch/Tensorflow during BNN training.

Figure 2.4.1: Convolution with bit-wise operation from [Bru20]

2.4.1 *Preliminary (Backward, forward, ...)*

FORWARD PASS [CB16] has proposed two kinds of binarization:the Deterministic and the Stochastic Binarization.
The first binarization function is deterministic:

$$x^b = \text{Sign}(x) = \begin{cases} +1, & \text{if } x \geqslant M \\ -1, & \text{otherwise} \end{cases} \tag{2.4.1}$$

where x^b is the binarized variable (weight or activation), which can be determined and x the real-valued variable. It is very simple to implement and works very well in practice.
The second binarization function is stochastic:

$$x^b = \begin{cases} +1, & \text{with probability} \quad p = \sigma(x) \\ -1, & \text{with probability} \quad 1-p \end{cases} \tag{2.4.2}$$

where σ is the hard sigmoid function:

$$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(1, \frac{x+1}{2})). \tag{2.4.3}$$

The deterministic binarization function is the mostly used function because although stochastic binarization is more attractive than the deterministic, it is more difficult to implement. Stochastic binarization requires the hardware to randomize bits during quantization.

BACKWARD PASS The real-valued gradients of the weights are accumulated in real-valued variables, although our BNN method uses binary weights and activation. In the simplest case, the gradient ϕ is obtained by replacing the binarization during the backward-pass with the "Straight-Through Estimator"(STE) [HWG+19]:

$$\Phi(L, x) = \frac{\partial L}{\partial x^b} \approx \frac{\partial L}{\partial x} \tag{2.4.4}$$

which is simply $g_r = g_q 1_{\|r\| \leqslant 1}$, with g_q estimator of the gradient $\frac{\partial L}{\partial q}$ and $q = Sign(r)$ [CB16]

2.4.2 *Training*

[CB16] uses the $Htanh(x) = Clip(x, -1, 1) = max(-1, min(1, x))$ for the activation and the Shift Based Batch Normalization to accelerate the training and reduces the impact of the weights. This work will train the Network using the binary-cross entropy loss for binary classification reasons given by

$$BCE(y, \hat{y}) = -\frac{1}{N} \sum_{j=1}^{N} y_j log(\hat{y}_i) + (1 - y_j) log(1 - \hat{y}_i) \tag{2.4.5}$$

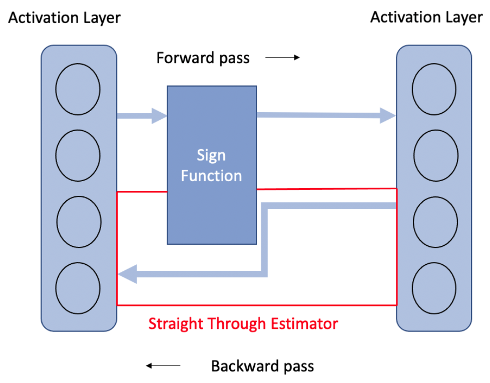


Figure 2.4.2: Training of the BNNs using the Sign function in the Forward propagation and the Straight-Throug Estimator in the Backpropagation. Taken from [Var21]

[CB16] provides 5 algorithms for the training of the networks. More details on training and optimization have already been discussed in section 2.3.2.2.

2.4.3 *Optimization*

Despite the difficulties caused by the combination pseudo-gradient and latent weights,known DNN methods can still apply to BNNs, including various optimizers (Momentum, Adam, Adamax ...) and regularizers like regularization L2 and weight decay regularization. [HWG+19] [CB16]

3 RELATED WORKS

In the previous chapter, the Fundamentals, and the functionality of MLPs were discussed. But there is still a problem with memory consumption during implementation. In this part, some solutions brought by some authors to overcome this problem will be discussed. This part has 3 sections: section3.1 elaborates the idea of the authors of [NZ21] who proposed Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training, then section3.2 focuses on training neural networks with weights and activations constrained to $+1$ or -1 proposed by [CB16]. Finally, Section4.1 describes the tCNN architecture.

This chapter discusses related works in the field of deep learning, paying attention to the following key aspects: Performing deep learning at the edge for real-time inference is essential for many fields of application. However, devices have limited memory, computing resources, and power. To bridge the gap between research and the production value of machine learning,there has been a growing emphasis on making models that are lighter and more efficient. The purpose of such research is to make them runnable on edge devices and mobile devices. This thesis explores two major research works, namely those by [NZ21] and [CB16]. In their research, [NZ21] focused on reducing precision or 8-bit integer numbers instead of training the models in floating-point 32-bit arithmetic, which reduces the memory size of the model by a factor of 4 and increases the throughput by a factor of 2 to 4 faster. Recently, [CB16] suggested a 1-bit quantization where data can only have two possible values generally called Binarization. Before approaching the architecture of the tCCN in Section3.3, these both works previously mentioned above respectively in section3.1 and section3.2 will first be investigated.

3.1 ACHIEVING FP32 ACCURACY FOR INT8 INFERENCE USING QUANTIZATION AWARE TRAINING.

Quantization is used to improve latency and resource requirements of Deep Neural Networks during inference.

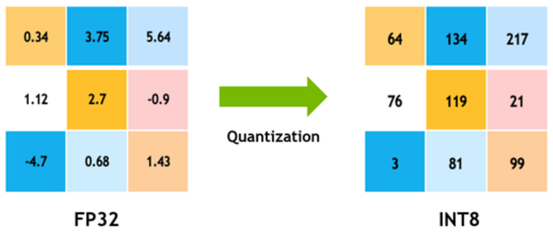


Figure 3.1.1: Example of FP 32 accuracy achievement for INT8 inference Using Quantization. Figure taken from [NZ21].

Because of the significantly large computational capacity, most models are trained in 32-bit floating point arithmetic. But these models can, however, require larger power budgets and can take more time to predict results. Thus, resulting in a slow response in real time and a negative impact on the user's experience. To help lighten the computing budget, without affecting the structure and number of model parameters, [NZ21] proposed to run the inference with a lower precision by quantizing models with a popular approach called 8-bit integer representation for weights and tensors such as TensorRT 8.0.
Model 8-bit quantization is a deep learning method in which both weight and activations data are updated from 32 floating-point representation to 8-bit integers to reduce both computing and memory requirements.
According to [NZ21], 32-bit floating-point is also known as Large dynamic range represented in the interval $[-3.4e^{38}, 3.40e^{38}]$, where parameters and data have most of their distribution mass around zero. However, an 8-bit integer representation can represent only 256 different values distributed uniformly or nonuniformly. [NZ21] recommends using a uniform representation because it enables high-throughput parallel or vectorized integer math pipelines. The update of the representation of a floating-point tensor (x_{fp}) to an 8-bit representation (x_{q8}) is obtained by mapping the floating-point tensor's dynamic range to $[-128, 127]$ using a scale-factor :

$$x_{q8} = \mathrm{Clip}\Big(\mathrm{Round}\big(\frac{x_{fp}}{\mathrm{scale}}\big)\Big)$$

Where *Round* is a function that applies some rounding-policy from round rational numbers to integers; *Clip* is a function that clips outliers that fall outside the $[-128, 127]$ interval. [NZ21] uses symmetric [1] quantization to represent or quantify both activation

1 dynamic-range ist given by $[-amax, amax]$. -amax and amax are given below.

data and model weights where *amax* is the element with the largest absolute value to represent. The quantization scale is given by

$$scale = \frac{2 \cdot amax}{256} \qquad where \quad amax = max\big(abs(x_{fp})\big).$$

Floating-point values that are outside the dynamic range are clipped to the min/max value of the dynamic range.
A Parameter data from 32-bit floats to 8-bit integers has several benefits such as a less storage space requirement by models, smaller parameter updates , higher cache utilization, it results in $4\times$ data reduction, which saves power and reduces the produced heat. Memory-limited layers benefit from reduced bandwidth requirement. However, [NZ21] notes that the larger the range used to represent 8-bit integers, the higher the probability to deal with rounding errors of floating-point values and a lower precision of the parameters. A model's task accuracy can be easily impacted by data, given that the process can be lossy, i.e. it may involve the loss of data or information. Nevertheless, even though a smaller dynamic range reduces those rounding errors, it introduces a clipping error.
To overcome the impact of the loss of precision on the task accuracy, [NZ21] proposed two categories of quantization: post-training quantization (PTQ) or quantization-aware training (QAT).

The PTQ is performed following the training of a high-precision model [NZ21] [WJZ+20]. With PTQ, quantizing the weights is easy but quantizing the activations is more challenging. [WJZ+20] evaluated these quantization parameters on a variety of neural network tasks and models, which have multiple types of network architectures: convolutional feed forward networks, recurrent networks, and attention-based networks. The relative accuracy change from int32 to int8 is computed by $(acc_{int8} - acc_{fp32})/acc_{fp32}$. An operation is quantized by quantizing all of its inputs but most of the other layers, such as softmax and batch normalization, are not quantized [Dat20];. However,the output of a quantized operation is not quantized to int8 because the following operation may require precision higher than 8 bit. According to [Dat20], maximum calibration is sufficient to maintain accuracy when quantizing weights to int8. [WJZ+20]recommended QAT for acceptable task accuracy or to improve the accuracy of quantized models by considering the quantization error in the forward and backward passe during the training phase. Furthermore the author in [WJZ+20] runs QAT by inserting Fake Quantization nodes for the weights of the Layer. During Fake Quantization data are quantized, but then immediately dequantized so that the operation computed remains in float-point precision. The quantized and dequantized operation are use to an approximate the input $(\hat{x} \approx x)$ where x ist the input and $\hat{x} = dequantize(quantize(x, b, s), b, s)$, $s = \frac{255}{\alpha - \beta}$. The authors used the Fake Quantization in the forward-pass and in the backward pass. However, the weights' gradients is used to update the floating-point weights. They also use a straight-through estimator (STE) to handle the quantization gradient. STE approximates the derivative of the fake quantization function to be 1 for inputs in the representable range $[\beta, \alpha]$:

$$\frac{\partial \hat{x}}{\partial x} = \begin{cases} 0, & y < \beta \\ 1, & \beta \leqslant y \leqslant \alpha \\ 0, & y > \alpha \end{cases} \tag{3.1.1}$$

TQ is simple and does not involve the training pipeline, which also makes it the faster method. However, QAT almost always produces better accuracy, and is sometimes the only acceptable method.

Moreover, [Dat20] recommend the following procedure(steps) and flow chart(Figure3.1.2) to quantize a pre-trained neural network.

- Scale quantization(with per-column/per-channel granularity or a symmetric integer range), scale quantization with tensor granularity, and maximum calibration are required to quantize the 8-bit quantization, activations, and weight respectively.
- PTQ: quantize all the computationally intensive layers (convolution, linear, matrix multiplication, etc.) and run activation calibration including max, entropy(for Clustering) and 99.99%, 99.999% percentile(to represent the distribution percentage). If none of the calibrations gives the desired accuracy continue to partial quantization or QAT.
- Partial Quantization: perform sensitivity analysis to identify the most sensitive layers and leave them in floating-point. If the effect on computational performance is not acceptable or an acceptable accuracy cannot be reached, continue to QAT.
- QAT: start from the best calibrated and quantized model. Use QAT to fine-tune for around 10% of the original training schedule with an annealing learning rate schedule starting at 1% of the initial training learning rate.

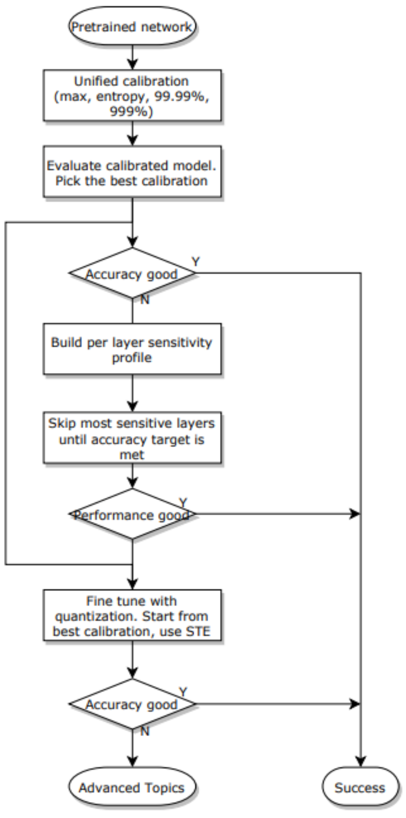

Pretrained network
Unified calibration (max, entropy, 99.99%, 999%)
Evaluate calibrated model. Pick the best calibration
Accuracy good
Y
N
Build per layer sensitivity profile
Skip most sensitive layers until accuracy target is met
Performance good
Y
Fine tune with quantization. Start from best calibration, use STE
Accuracy good
Y
N
Advanced Topics
Success

Figure 3.1.2: Flow chart of quantization workflow recommended and explained by the authors in [Dat20]

In Summary, Performing deep learning, by using reduced precision or 8-bit integer numbers at the edge for real-time inference is essential for many fields of application. However, it may result in a lower accuracy model. What happens if the number of bits is significantly reduced, i.e. from 32-bit floats to just 1-bit integers?

3.2 BNNS: TRAINING NEURAL NETWORKS WITH WEIGHTS AND ACTIVATIONS CONSTRAINED TO $+1$ OR -1

Thanks to DNNs, artificial intelligence (AI) is able to perform more tasks than ever before starting from real-time object detection, image recognition and natural language processing. But despite this, deep neural networks are characterized by their high memory consumption and ability to ruin the battery life of devices during training. These tasks require high-power intensive devices like GPU because of its thousands of multiplications and additions of floating-point numbers. The deployment of these models on limited resources like mobile or embedded devices is a challenging task. Therefore, the Binarization of the Neural Networks is known to be one of the most successful methods for solving the high memory problem of deep neural network models.

3.2.1 *The BNN Architecture*

[CB16] introduces a method to train Binarized Neural Networks (BNNs). The authors define the BNNs as DNN (explained above at section2.3.2) with Weights and Activations Constrained to $+1$ or -1 at run-time. The authors of [HCS+16] first proposed a method to train a BNN, then they implemented two experiments on Torch7 and Theano Framework respectively to show that BNN(DNN with binary weight and activation) can achieve near-state-of-the-art result on MNIST; CIFAR-10 and SVHN while challenging ImageNet dataset. Furthermore, they proved that during the forward pass, BNNs drastically reduce memory consumption. However, the authors of replaced most arithmetic operations with bitwise operations. This potentially increases in power efficiency and could reduce the time complexity. The authors also programmed a binary matrix multiplication GPU which makes it possible to run our MNIST BNN seven times faster than with an unoptimized GPU kernel, without suffering any loss in classification accuracy.

However, The architecture of a BNN is similar to the traditional DNN architecture, except weights and activations are binarized to either $+1 or -1$. Figure 3.2.1

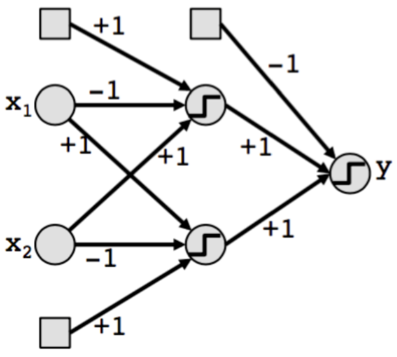


Figure 3.2.1: An example of Binarized Neural Network taken from [Nat18]

3.2.2 *Binarized Neural Networks*

[CB16] proposed two different methods for the Binarization,respectly the Deterministic and Stochastic binarization.

3.2.2.1 *Deterministic and Stochastic Binarization*

The deterministic is as follows:

$$x^b = Sign(x) = \begin{cases} +1, & \text{if } x \geqslant M \\ -1, & \text{otherwise} \end{cases} \tag{3.2.1}$$

where x^b is the binarized variable (weight or activation), which is a determined value, and x the real-valued variable.
Secondly the stochastic binarization function is:

$$x^b = \begin{cases} +1, & \text{with probability} \quad p = \sigma(x) \\ -1, & \text{with probability} \quad 1-p \end{cases} \tag{3.2.2}$$

where σ is the hard sigmoid function[CB16]:

$$\sigma(x) = clip(\frac{x+1}{2}, 0, 1) = max(0, min(1, \frac{x+1}{2})). \tag{3.2.3}$$

Although the stochastic method is more efficient, the author of [HCS+16] opts for the use of the sign method instead. Because stochastic is difficult to be implemented.

3.2.2.2 *Gradient Computation*

The [HCS+16] training method can be considered as a variant of Dropout, in which both the activation and the weights are set to $+1$ and -1 instead of setting some of the activations to zero. The authors use binary weights and activations to calculate the parameter gradients, but they then accumulate the real-valued gradients of the weights into real-valued variables, which require the storage of more than one bit and which the stochastic gradient descent (SGD) needs to work properly. SGD explores the space of parameters in small and noisy steps [2], hence that noise is averaged out by the stochastic gradient contributions accumulated in each weight. The author of [HCS+16] assert that high precision is absolutely needed to maintain sufficient resolution for these accumulators.

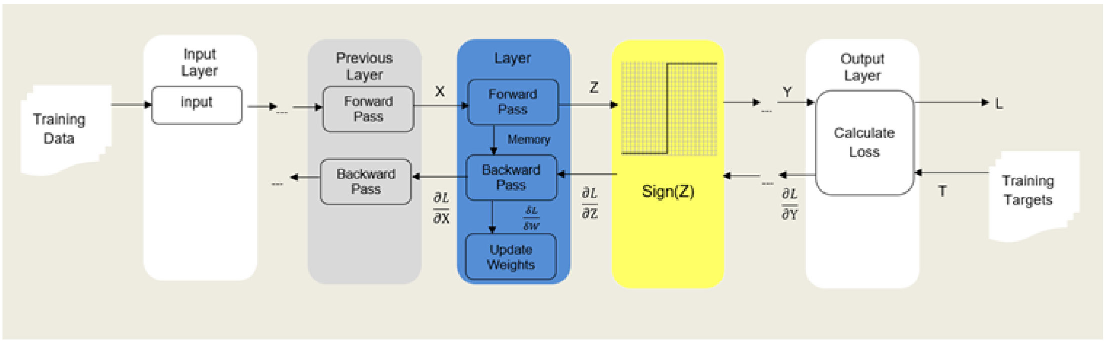


Figure 3.2.2: Training of binary Neural Networks, inspired from [Aka20]. X, Y. Z are the output of previous layers and input of the next layers. $\frac{\partial L}{\partial W}$ is the gradient

3.2.2.3 *Gradient Propagation*

To obtain the gradient,it is always common to derive the inputs. In this case, however,because of the sign function which is zero at the derivation there will be no Backpropagation and therefore no training of the networks. SGD is now useless since the precise gradient of the cost with respect to the quantities before the pre-activations or binarization would be zero. This is also the case with the stochastic method (Figure3.2.2 presents a view on training and gradient computation).
To overcome this problem,the authors of [HCS+16] therefore resorted to the **straight-through estimator** that takes into account the saturation effect, They considered the Sign function quantization

$$q = \mathrm{Sign}(r)$$

2 Provide a form of regularization that can help to generalize better

and they assumed that an estimator g_q of the gradient $\frac{\partial c}{\partial q}$ has been obtained through the straight-through estimator . Then, the straight-through estimator of $\frac{\partial c}{\partial r}$ is:

$$g_r = g_q 1_{|r| \leqslant 1}$$

Where r is the input. The concept of a direct estimator is to render the incoming and outgoing gradients of a threshold function identical, independently of the threshold function itself. The author specifies that this preserves the gradient information and cancels the gradient when r is too large, which significantly degrades performance. This prevents neurons with large activations ($|r| > 1$) from being updated. STE ist used in Algorithm2.

PARAMETER UPDATE While they used the Sign function to obtain binary activations for hidden units ,The authors of [CB16] use two operations to update the weights: They first force the real weight w^r to be in a range $[-1, 1]$ by clipping all the weights w^r outside $[-1, 1]$ during training. This is to prevent the real-valued weights from becoming very large without having any impact on the binary weights w^b. Then, secondly update real-valued weights to binary weights using $w^b = Sign(w^r)$. With $|w^r| > 1$.

3.2.2.4 *Normalization and Result*

The linear function hard tanh is an the activation function used by [CB16]. the Htanh(x) is given by

$$Htanh(x) = Clip(x, -1, 1) = max(-1, min(1, x)).$$

During the training, the authors of [CB16] used Batch Normalization (BN) and the AdaMax. This accelerated the training, reduced the impact of the weight's scale and regularized the model. To avoid multiplications, they suggested to use shift-based batch normalization(SBN) or shift-based AdaMax, which reduces the hardware demand. However the authors did not register any loss of the accuracy using them.

3.2.2.5 *Efficiency*

The authors carried out two sets of experiments, each based on a different framework, namely Torch7 and Theano. The BBN was successfully able to achieve nearly state-of-art performance with the smaller Dataset like MNIST and CIFAR-10 and SVHN. However, there was some degradation in the performance while training on larger Dataset like ImageNet. After many attempts, the authors improved this by using a few more Bit in the Neural Network and using AlexNet, GoogleNet. It was also noted that the MLP runs seven times faster on GPU at run-time with the XNOR kernel while using SIMD (single instruction, multiple data) within a register (SWAR) and there was not any loss registered in classification accuracy. The authors added that exploiting Filter Repetitions when using a ConvNet architecture with binary weights limited the number of unique filters by the filter size and reduced the number of the XNOR-popcount operations by three. According to [HCS+16], memory access is more energy hungry than the arithmetic operation, thus higher memory size means more energy consumption. The authors claimed that BNNs require 32 times smaller memory size and 32 times less memory accesses than DNNs because all the $32-\mathrm{bits}$ floating point numbers are now updated into 1-bit binary numbers. The update reduces the energy consumption efficiently, using binary numbers for multiplications and additions. Moreover, most of the $32-\mathrm{bit}$ floating point multiply-accumulations are replaced by $1-\mathrm{bit}$ XNOR-count operations. This could have an important impact on dedicated deep learning hardware. In summary, during training and run time (pass forward), BNNs considerably decrease memory size and accesses, and replace most arithmetic operations with bitwise operations. Consequently, significantly improving energy efficiency and dedicated hardware can reduce the time complexity by 60% as claimed by the authors of [HCS+16].

3.3 TEMPORAL CONVOLUTIONAL NEURAL NETWORK

It is complex to solve HAR problems. Moreover, people perform the same kind of activity in different ways, and these activities can manifest at different points in time. Several types of deep learning models, (e.g. Deep Belief Network (DBN)) have been implemented to solve the challenges and complexity associated with HAR.DBN neglects the available label information in feature extraction and does not use efficient signal processing units [YNS+15]. However, [RC15] proposed that it is important to consider the temporal dependence of nearby readings of time-series sensors during the classification. Due to their ability to recognize local dependencies between the sensor values and their ability to enable more robust recognition, a CNN, which uses temporal convolutions was proposed for solving HAR and is currently the very competitive method for the HAR problems [GLR+17, NRMR+20, RC15]. A CNN applied along the temporal dimension for each sensor was first proposed by [ZLL+14] for solving HAR and was later improved by [YNS+15]. [ZLL+14]'s CNN is made up of an input layer, whose values are fixed by the input data; (two) hidden layers whose values are derived from previous; effective signal processing units (such as convolution, pooling and rectifier) output layer whose values are derived from the last hidden layer and a Softmax classifier. like many other networks, the TCN learns using a set of weights in input

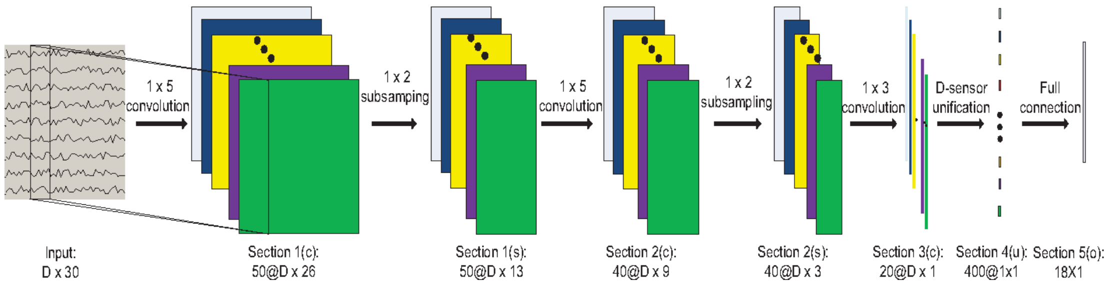


Figure 3.3.1: Graphic of a TCNN architecture. The alphabets "c", "s", "u", "o" in the parentheses of the layer tags refer to convolution, subsampling, unification and output operations respectively. The numbers before and after "@"refer to the number of feature maps and the dimension of a feature map in this layer. Taken from [YNS+15]

.

Nevertheless, solving HAR is made of challenges due to multiple channels of time series signal such as applying processing units in CNN along temporal dimension, sharing or unifying the units in CNN among multiple sensors. To face these problems, [YNS+15] improved the CNN of [ZLL+14] by introducing a TCN for the classification of time-series Data in HAR. The architecture of this TCN is organized into five sections. The first two sections consist in a convolution layer with kernels, a RELU activation function that maps the output of the previous layer, followed by a max-pooling layer with a normalization layer, a max-pooling layer applied over a range of local temporal level and a normalization layer for the values of the computed feature maps from the previous layer.
The third section of the network only consists in a convolution layer,a RELU layer and a normalization layer. There is no more pooling layer, since the time dimension of the feature map after the convolution layer is the size one, thus the maximum of only one value would always be the value itself. If max-pooling is performed on the temporal level, only one single value would be considered at a time, since only one value exists on the temporal level.

The fourth section consists in a fully connected layer, that normalizes the values in the previous layer. Thus, it is used to unify the computed feature maps from section 3 of all sensors. This is also followed by the RELU layer and a normalization layer. The last section is a fully connected network layer, used to map the latent features into the

output classes and followed by a softmax activation function used to classify the data. Furthermore, the Entropy cost function is used for the training of the TCN. Nevertheless, two different datasets were used for different purposes. The first is called the " **Opportunity Activity Recognition**"and is deployed for the whole body movement, the second one is the "**Hand Gesture**"focused on the hand movement. They have both the same architecture but different features, sizes, kernels, and datasets. The author of [YNS+15] compared TCN to four baseline: the **SVN**, the **KNN**, the **MV**, and the **DBN** but, was better than all the four by 5%,with or without smoothing settings. There were no improvements using the magnitudes of Fourier transform of the raw data as inputs. The author believes that his proposed TCN for studying the multichannel time series sensors, can serve as a competitive tool of feature learning and classification for the HAR problem.
Nevertheless, the author of [NRMR+20] proposed a new TCN to solve HAR but by making here some minor changes. The new TCN is called a tCNN and will be explain in section4.1.

Nevertheless, this section focused on the work related to solving a HAR problem which is binarizing the neural network for HAR to limit memory consumption. Thus, after their experiments, the authors of [CB16] assert to have had a state-of-the-art result with binarization. Similarly, the authors of [YNS+15] stated after experiments that the CNN method, which is a Deep Learning algorithm capable of classifying images and videos in a very accurate way, is more efficient than the other state-of-the-art methods. In the last part of this section, a new version of the state-of-the-art CNN designed by [NRMR+20] and called the tCNN was presented. The next Section is about mixing these two works, i.e., binarizing the tCNN during the training to have a state-of-the-art result as in the baseline.

4 METHOD

In Chapter 3 , three works dealing with binary networks for solving HAR were discussed, including Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training of section3.1, This work suggested that to reduce the memory consumption of the HAR,it is necessary to quantize the data (e.g., the weight) from 32 floating point to 8-bit. Although this method has achieved better results,it contains some data loss. The second work, i.e. BNNs: Training Neural Networks with Weights and Activations Constrained to $+1$ or -1 of section3.2, pointed out that to reduce the memory consumption of the HAR, it is necessary to reduce the weight and activation representation by $32\times$, i.e., from 32 floating point to 8-bit. Lastly, section3.3 introduced the temporal Convolutional Neural Network later explained in section4.1. This wok will combine the ideas in tCNN and BNNs to develop its method. The method proposed in this thesis aims at binarizing the tCNN for solving HAR problems. Thus, the first part will consist in reconstituting the Temporal Convolutional Neural Networks (tCNN) based on the related work, and ultimately focus on the training of these updated networks. Then later in the experiment, the inputs will be classified and interpreted according to two different classifiers Softmax and attributes (sigmoid) for Mocap and Mbientlab respectively, without forgetting the performance.

4.1 TEMPORAL CONVOLUTIONAL NEURAL NETWORK FOR HAR USING LARA DATASET

Due to its multichannel time-series, a variation of CNN has been deployed for solving HAR problems and many of them have brought better results such as the tCN proposed by [YNS+15]. Based on [YNS+15], the author of [NRMR+20] also proposed a TCN to solve HAR using the LARa dataset but by making here some minor changes: Unlike the [YNS+15] model, the newly proposed tCNN is an end-to-end architecture composed of feature extractors and a classifier(softmax or sigmoid) but has no downsampling operation and no normalization layers then according to the author [NRMR+20], they could affect negatively the performance of the network. The new architecture has N number of sequence channels with sequence length L.

CONVOLUTIONAL LAYERS the [NRMR+20]'s tCNN contains four convolutional layers. The convolutional layers are composed of 64 filters of size [5×1], performing convolutions along the time axis and extracting the temporal features from the data.

FULLY CONNECTED LAYERS [NRMR+20] uses three fully [1] -connected layers for his tCNN. These FC-layers are made up of two fully connected layers (which contain 128 units) and a classifier (Sotfmax or Sigmoid). Two Dropout[2] layers were applied to the first and second fully connected layers respectively .

LAST FULLY CONNECTED-LAYERS the last fully connected layers compile the data extracted by previous layers to form the final output. Depending on the task[3], two different output layers from the end of the tCNN. The Softmax and the Sigmoid.

CLASSIFICATION the tCNN will use a softmax layer or a sigmoid layer for the classification [NRMR+20].

- SoftMax Layer: The last fully connected layer with a softmax function is called the Softmax layer and is intended for the class(activity) results [NRMR+20]. The Softmax layer has $C = 8$ units of activity classes. In this context, the number of output units depends on the number of classes.
- Sigmoid layer: The last fully connected layer with a sigmoid function is called the sigmoid layer and is intended for the attribute classification [NRMR+20]. It contains 19 units of Attribute representations. Here, the number of output units depends on the number of attributes.The author in [LMRA+21] used a sigmoid layer, to compute a attribute representation from an input sequence, which, according to them, have proven to be advantageous to HAR, as they used the $\text{tCNN}_{\text{attribute}}$.

1 all neurons are connected to all inputs and all outputs
2 The dropout step involves removing some neurons randomly during the learning process. This avoids the network from depending too much on single neurons and forces all neurons to learn to generalize better.
3 The are two different classification functions according to two representations: the activity or class representation classified by Softmax, and the attributes representation classified by the Sigmoid.

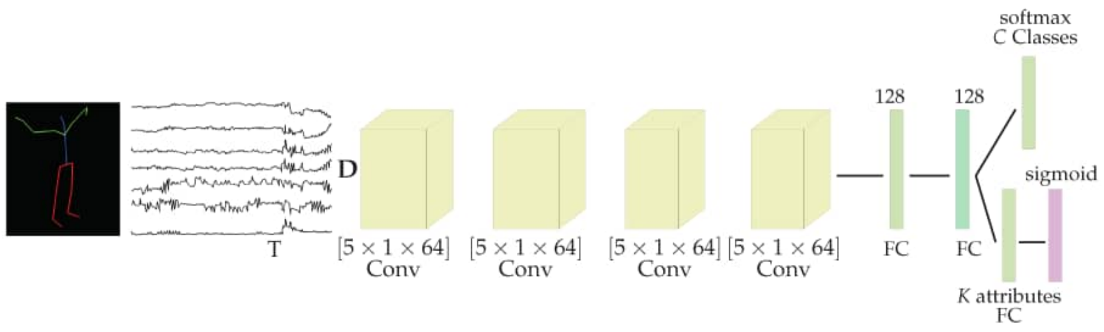


Figure 4.1.1: A Graphic of the Temporal Convolutional Neural Network (tCNN) architecture (The Dropout are missing),taken from [NRMR+20]

.

4.1.1 *Training of Temporal Convolutional Neural Network for HAR using LARa*

First of all, it should be noted that, for each sensor channel, the authors of [NRMR+20] said to have normalized the input sequences in the range $[0, 1]$ While also including some Gaussian noise, which simulates sensor's inaccuracy. The noise will have as parameters $\mu = 0$ and $\sigma = 0.01$.
The training procedures entail three parts: training, validation, and testing. The training set contains recordings from the eight subjects while the validation and testing sets contains the recordings from three subjects respectively. The Validation set is used for the early stopping approach and will assist in finding correct training hyperparameters. To prevent the network from overfitting and the training data memorization, during training, the weights of the best testing set result are copied and returned as output value at the end of the training.
The authors trained the tCNN using the batch gradient-descent with RMSProp update rule with a batch size of 400, an RMS decay of 0.9,and a learning rate of 1×10^{-5}. As far as the cost function is concerned, the tCNN architecture is trained using the **binary-cross entropy loss** for the sigmoid layer when predicting attributes and the **Cross-Entropy Loss** for the Softmax when predicting activities. Regarding the metrics (precision, overall accuracy, the weighted F1 score and recall), the author found that solving HAR using the attribute representation offers better results than using a softmax layer.

The following Figure4.1.2 gives more view on the training of tCCN, At the forward pass, the classifier receives the input from the previous layers and computes the Loss (error or difference between the predicted and the real output) then at the back pass a gradient is computed according to the Loss (derivative of Loss over the input). Then this gradient is used to adjust the weight. Thus, the weights are updated and optimized. This will be done through epochs until the Loss is minimized.

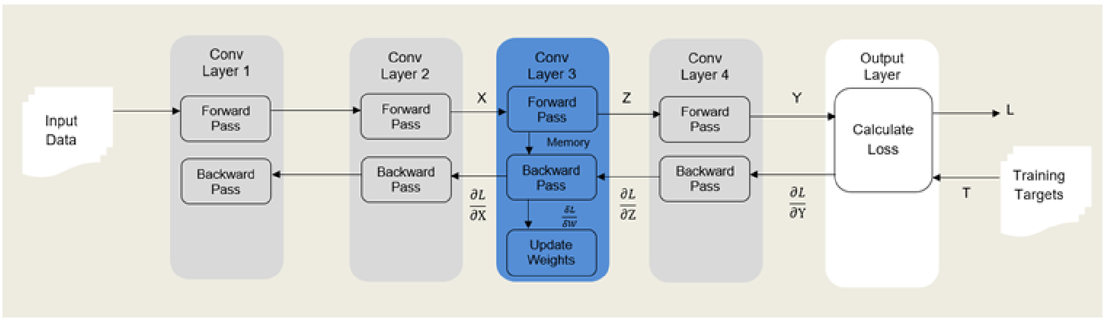


Figure 4.1.2: Graphic of the Training of the tCNN proposed by [NRMR+20]. the Networks has 4 Conv layers each composed of 64 filters of size $[5 \times 1]$, zwei Fc-layers of 128 filters and a classifier (sofmax or sifmoid depending on the task). Image inspired from [Aka20]

Nevertheless,the tCCN that was previously trained on LARa dataset is going to be trained again, then the weight will be binarized for Conv Layers. Following [NRMR+20], LARa OMoCap is divided into three sets: training, validation and testing. The following Figure4.1.3 presents the BtCNN at training.

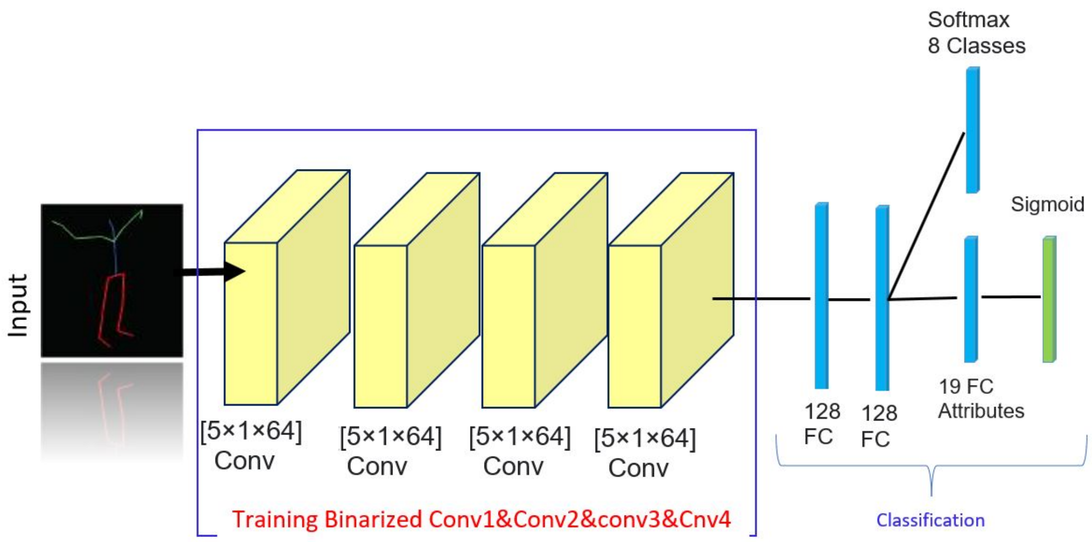


Figure 4.1.3: Graphic of BtCNN during training.Image inspired from [NRMR+20].

4.2 THE CLASSIFIERS

Regarding the classification, this thesis will consider the Softmax classifier, the Sigmoid or the k-nearest neighbor (NN). Each depending on the task. In the LARa method for solving HAR, the input sequence is $[T = 200, D = 126]$ and the slide $S = 25$ [NRMR+20]. Therefore,the tCNN will use this input sequence to calculate the activity class and the binary-attribute representation. This means that our data are arranged in two representations: The activity classes with unit $C = 8$ which is classified by the **Softmax** and the attributes which are classified by the **Sigmoid** and contains K = 19 attributes. It should also be noted that the last fully connected layers have 128 units each. The predictions of the different classes and attributes will therefore be gathered in a table called confusion matrix. The author of [NRMR+20] also used the NN for the representation of the attributes and based on the performance results, the author asserts that the attribute representation was able to classify the inputs the best compared to Softmax, especially for classes like handling and synchronization. However, the early stopping technique is used during the validation.

4.3 BINARY TEMPORAL CONVOLUTIONAL NEURAL NETWORK FOR HAR

Instead of full-precision values, the networks of this thesis use binary values for activations and weights. Binary values in BNNs can either mathematically be logical value 0 and logical value 1 (Single beat). But this thesis will use -1 or 1 as it is more usual.

4.3.1 *Architecture*

the architecture of the now binarized tCNN will be henceforth called BtCNN. It has the same configuration as the original tCNN of [NRMR+20]. No change has been made regarding the architecture explained in section4.1. The only difference here (which is the aim of this work) is that the layers have been binarized. It means that the inputs have all been binarized, both weight and activation. The BtCNN will work with binarized data according to the recommendations of section3.2. Therefore,a new function called Sign function explained in section 3.2.2 will be added to the network configuration. The architecture of the original tCNN has been explained in section 4.1.

4.3.2 *Training Binary tCNN*

However,the method consists in training 10 epochs on a training dataset. Still following [NRMR+20], the architecture is trained using the batch gradient-descent with RMSProp update rule, three learning rates of 10^{-4}, 10^{-5}, and 10^{-6} and a batch of size 581 at the validation.The work continues by implementing and experimenting binary conv layers.
This thesis proposes a BtCNN to reduce memory consumption and to reduce the number of operations. The reason why this work uses -1 and 1 instead of the traditional 0 and 1 is that the XNOR will replace the multiplication for the operations. By applying XNOR on -1 and 1, the results are the same as those of the multiplication in binary representation. Therefore, it will be preferable to use the XNOR and consequently -1 and 1, which considerably reduces the number of operations (or multiplication). Furthermore, the BtCNN training of this work is similar to the tCNN except that the features are binarized. So, this work will use binary values during the training procedure (single Bit precision). Following the [CHS+16], this thesis uses the deterministic binary function for the training as the values are deterministic. The deterministic binary (sign)function is presented in Equation2.4.1. The procedure will be similar to the traditional training of tCNN but during the forward Pass, the sign function is applied, then the Loss between the actual value and the prediction is calculated using the Loss function Equation 2.3.26. However, the real values are preserved and will be used during the backpropagation to compute the gradient. Because the output of the sign function cannot be derived. Figure4.3.1 gives more view of the training procedure of BtCNN, The BtCNN has $C = 4$ number of Conv layers and the 4 layers are binarized during training independently. The network has 4 configurations and the 4 configurations are trained and independently. For all the layers to be binarized, the weights and activations are binarized. More information on the configurations is given in Experiment in section5.3
The training in the following Figure4.3.1 is similar to the training of Figure 4.1.2 in section 4.1.1 with the difference that weights are binarized with a sign function a the forward pass. But at the backward pass, real numbers are used to facilitate the gradient computation.

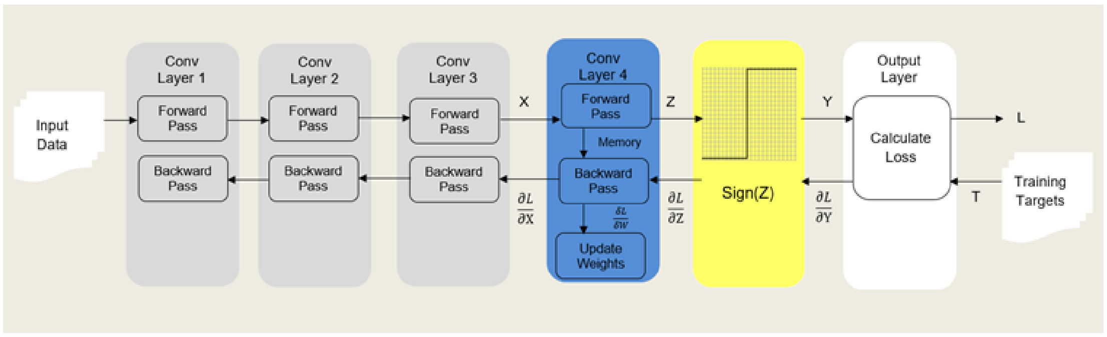
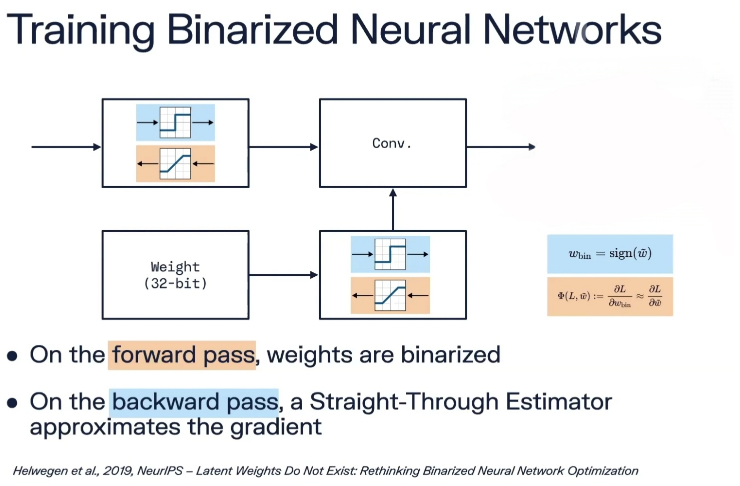


Figure 4.3.1: Training of the binary tCNN (BtCNN). At the Forward pass, the sign function is used for binarization. At the Backward, the loss and the gradient are computed using real value. The tCNN was proposed by [NRMR+20]. The figure inspired from [Aka20]

The gradient is the derivative of the Loss with respect to the weight ($\frac{\partial Loss}{\partial X}$). The Loss is the difference between the actual input and the prediction. So, the algorithm will use the computed loss to calculate the gradient and in the Backward pass, the weight will be updated by subtracting the gradient multiplied by the learning rate(Algorithm1and 2). Figure4.3.1 shows more view. However,the derivation of the Sign function is zero almost everywhere making it difficult for the backpropagation. This,is due to the fact that the gradients of the cost are zero(even using the stochastic function) leading to no training.
Thus, instead of using the gradient of the sign function, this work will use an estimator Method called "Straight-through estimator"(STE) proposed by [CHS+16] to preserves the gradient's information . However, this proxy derived at the backward pass is nothing but a function used to evaluate an unknown parameter related to a probability law such as its expected value or its variance. The STE is explained in section 3.2.2.3. Nevertheless, the weight of the Network will be updated to produce a smallest score for that class. This will be done over several epochs until the minimal loss [CHS+16].


Conv.
Weight (32-bit)
$w_{\text{bin}} = \text{sign}(\tilde{w})$
$\Phi(L, \tilde{w}) := \frac{\partial L}{\partial w_{\text{bin}}} \approx \frac{\partial L}{\partial \tilde{w}}$

Training Binarized Neural Networks

- On the forward pass, weights are binarized
- On the backward pass, a Straight-Through Estimator approximates the gradient

Helwegen et al., 2019, NeurIPS – Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization

Figure 4.3.2: Example of Diagram of the Forwardpropagation and the Backwardpropagation,taken from [HWG+19]. At the Forward pass, the inputs are binarized with the sign function and at the Backward, the real is used to compute the error. STE is used to approximate the gradient.

Moreover, in the **Forward Pass**,Algorithm1 is used to train the networks with a Binary function called sign function. Where s are activations before BatchNorm, a are activations after BatchNorm, C number of layers and the Binarize() method is the sign function. Batchnorm() takes the output from the first layer, normalizes it, and passes it as an input to the next layer.

Algorithm 1 Binary training: Forward propagation

```
for $k = 1$ to C do                                  ▷ C: Number of Conv layers
    $W_k^b \leftarrow \text{Binarize}(W_k)$           ▷ signum function
    $s_k \leftarrow a_{k-1}^b W_k^b$                  ▷ $s_k$ : activations before BatchNorm
    $a_k \leftarrow \text{BatchNorm}(s_k, \theta_k)$  ▷ $a_k$: activations after BatchNorm
    if $k < C$ then                                   ▷ me
        $a_k^b \leftarrow \text{Binarize}(a_k)$       ▷ Binarized $a_k$
    end if
end for
```

In the **Backward Pass**. The weights are updated using the gradient and the Learning rate. The gradients are therefore maintained non-binary. Algorithm2 gives more details on the Backpropagation. Where E is the cost function, λ is the learning rate, and C is the number of layers. $\circ$ indicates element-wise multiplication used to copy the gradient information through STE. and Clip() to prevent vanishing and exploding gradients, BackBatchNorm() takes the output from the first layer, normalizes it, and passes it as an input to the next layer. Update() specifies how to update the parameters when their gradients are known, using shift-based AdaMax.

Algorithm 2 Binary training: Backward propagation

```
Please note that the gradients are not binary.

Compute $g_{a_C} = \frac{\partial E}{\partial a_C}$ knowing $a_L$ and $a^*$          ▷ E: Cost function.
for $k = C$ to 1 do                                                                   ▷ C: Number of Conv layers
    if $k < C$ then
        $g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|A_K| \leqslant 1}$                    ▷ STE
    end if                                         ▷ $s_k$ / $a_k$ : activations before/after BatchNorm.
    $(g_{s_k}, g_{\theta_k}) \leftarrow \mathrm{BackBatchNorm}(g_{a_k}, s_k, \theta_k)$
    $g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b$                                          ▷ $a_k^b$: Binarized $a_k$
    $g_{W_k^b} \leftarrow g_{s_k}^\top a_{k-1}^b$
end for
(Accumulating the parameters gradients)
for $k = 1$ to C do
    $\theta_k^{t+1} \leftarrow \mathrm{Update}(\theta_k, \eta, g_{\theta_k})$         ▷ $\eta$: Learning rate
    $W_k^{t+1} \leftarrow \mathrm{Clip}(\mathrm{Update}(W_k, \gamma_k \eta, g_{W_k^b}), -1, 1)$
    $\eta^{t+1} \leftarrow \lambda \eta$                                              ▷ $\lambda$: drope rate.
end for
```

At last, this section presented the method, the first part focused on the architecture of the tCNN and the second part on the binarization of the BtCNN networks during the training process. The question now is to know whether this experiment produces state-of-the-art results.

5 EXPERIMENTAL EVALUATION

the previous section presented firstly the tCCN and its architecture then secondly the BtCNN which is the binary tCNN. Therefore, the binarization method and the training of BtCNN have been explained. Moreover, this section will evaluate the performance of the BtCNN. To do this, a set of training and tests will be carried out on the LARa dataset. The LARa dataset is presented in the first part of this section. Then will follow the presentation of the set of metrics necessary to compare the performances. They are three key metrics to measure the performance of the network. The Accuracy, the F1-mean Score, and the F1-weighted Score. Then the different configurations are explained as well as their different binarization. Furthermore, the performance of these configurations will be listed and compared using metrics. Last but not least, the results obtained will be compared to the baseline results to conclude if the results produced by this experiment are the state of the art.

5.1 DATASET

5.1.1 *LARa Dataset*

"Logistic Activity Recognition Challenge"(LARa) is a first freely accessible HAR dataset, which contains annotations of human actions and their attributes in the intra-logistics [NRMR+20]. The data recording was collected by having 14 subjects performing three real-world warehouse scenarios during a total of 758 minutes of recordings. LARA uses Optical marker-based Motion Capture (OMoCap) to track 39 markers worn by the subjects, six inertial measurement units (IMUs) from MbientLab dataset which are fixed on arms, legs, chest, and waist, with a focus on both triaxial linear and angular acceleration. An RGB camera dedicated to recording. The subjects undergo the scenario-based picking and packaging activities during which all the movements are recorded and their data labeled. This labeling is performed using of eight activity classes and nineteen attribute representations.

ACTIVITY CLASSES the dataset includes both periodic and static activities for the classification and are divided into eight $C = c_1, ..., c_8 \in \mathbb{N}^8$ classes, such as Handling, Standing, Walking, Cart, Synchronization and None. The following table, taken from [NRMR+20], indicates the meaning of these activities.

Activity Class	**Description**
C_1: Standing	The subject is either on his/her feet or taking smaller steps, and may or may not be holding an object in hands.
C_2: Walking	The subject performs steps while holding an item or hands-free.
C_3: Cart	The subject is walking with the cart to a new position with some exception such as placing boxes or collecting items.
C_4:Handling upwards	At least one hand reaches 80% of a body's total height(shoulder height) during the handling activity.
C_5:Handling centred	Handling items without bending, kneeling or raising the arms at the shoulder joint
C_6: Handling downwards	During Handling, the subjects are kneeling with their hands below the level of their knees.
C_7: Synchronization	Both hands are above the subject's head when the recording starts.
C_8: None	Will be ignored as it is reserved for Errors, Exceptions, Gaps and all unclassifiable activities.

Table 5.1.1: Activity Classes and their definition

The class C_5 then has the largest share of classes in the Dataset while the C_7 class has the smallest.

ATTRIBUTE REPRESENTATIONS there are $K = 19$ attributes $A \in \mathbb{B}^k$. Attributes are etymological descriptions of the class activities [NRMR+20]. It is seen as a bit map between sequential data and human activities. The following table, taken from [NRMR+20], clearly indicates the meaning of these attributes.

Group	Attributes	Description
Legs	A	**Gait Cycle**
	B	**Step** (where the feet leave the ground without a foot swing)
	C	**Standing Still**
Upper Body	A	**Upwards**
	B	**Centered**
	C	**Downwards**
	D	**No Intentional Motion** (steady position without doing anything)
	E	**Torso Rotation** (Rotation in the transverse plane)
Hands	A	**Right Hand**
	B	**Left Hand**
	C	**No Hand** (no holding nor for handling)
Item	A	**Bulky Unit** (Objects that cannot be held with the hands)
	B	**Handy Unit**(small Items that cannot be held with the hands)
	C	**Utility** (Use of the working tools)
	D	**Cart** (Handling & No Intentional Motion)
	E	**Computer** (e.g. mouse and keyboard)
	F	**No Item**
Data	A	**None** (None class)

Table 5.1.2: Attributes representation and their significations

[NRMR+20] used nearest neighbor (NN) approach during attributes prediction using the attributes representation and the output of the network to compute $c \in C$.

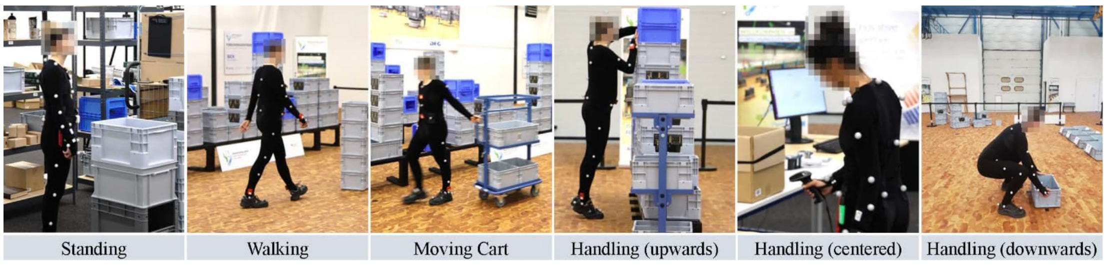


Figure 5.1.1: A Graphic of one Subject participating in a recording while performing activities and wearing Sensors suit. Image Taken from [NRMR+20]

.

TRAINING The tCNN, explained in session 4.1, was deployed for solving HAR using the LARa dataset. However, the classification performance with tCNN is state-of-the-art. The training of tCNN is explained in section 4.1.1.
According to [NRMR+20] , the LARa-MoCap's training set contains recordings from the eight subjects. However, the validation and testing sets own recordings from three subjects respectively. The validation set helps to determine proper training hyperparameters and is used for the early stopping approach. Thus, it will find the best training step or epoch and then stop the result. The tCNN is pre-trained for attributes prediction for all experiments. However, to determine the metrics, class as attributes predictions using tCNN with the softmax layer or a nearest neighbor (NN) approach are needed.

5.2 METRICS

Four different and individual metrics are going to be observed in this work. These give us more information on how the model behaves. These metrics(represented in Table 5.2.1) are **True Positive(TP)**, which is the amount of time that model correctly classifies a positive sample as positive, **False Negative(FN)**, which is the amount of time that model incorrectly classifies a positive sample as negative **False Positive(FP)**,which is the amount of time that model incorrectly classifies a negative sample as positive and **True Negative(TN)**, which is the amount of time that model correctly classifies a negative sample as negative.

Table 5.2.1: A Graphic of positive-negative-true-false-matrix. Image Taken from [Dil19]

		Actual	
		Positive	Negative
Pred	Positive	**True Positive**	**False positive**
	Negative	**False Positive**	***True negative***

Moreover, based on these parameters the scores or metric will be determined.

5.2.1 *Precision, Recall, F1-Scores and loss*

the precision:determines the correctness of a positive prediction. In other words, it determines the precision of the network with respect to an activity class. It is calculated using the following Equation:

$$P = \frac{\#True \quad Positive}{\#Total \quad predicted \quad Positive} = \frac{TP}{TP + FP}. \tag{5.2.1}$$

The recall: determines how many true positives get predicted out of all the positives in the Dataset.It is calculated using the following Equation:

$$R = \frac{\#True \quad Positive}{\#matching \quad elements} = \frac{TP}{TP + FN} \tag{5.2.2}$$

The Accuracy indicates the proximity of a computed feature to a standard or known value. In other words, how many of the computed activity classes match the predictions of the data. It is calculated using the following Equation:

$$Acc = \frac{\#correct \quad predictions}{\#all \quad predictions} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2.3}$$

The **F1-score** is the harmonic mean of the precision and recall. In other words, it balances the precision and the recall. The higher the accuracy of the predictions per class, the higher the F_1-score. The formula is given by:

$$F_1 - score = 2 \times \frac{Recall \times Precision}{Recall + Precision} = 2 \times \frac{R \times P}{R + P} \tag{5.2.4}$$

The **F1-mean** score is the arithmetic average of all F_1-score in the activity classes. The formula is given by:

$$mF_1 = \frac{1}{n}\sum_n F_{1_n}, \tag{5.2.5}$$

Where n stands for the number of classes.
Nevertheless, since all the activity classes have the same influence on the result, a problem of balance may occur. This problem is solved using a new F1-score which takes into account the size of classes' weights called **F1-weighted Score** and is given by:

$$F_1 - weighted = \frac{1}{n}\sum_n (F_{1_n} \times \text{Propotion\_of\_class})$$

or simply:

$$wF_1 = \sum_i^C 2 \times \frac{n_i}{N} \times \frac{P_i \times R_i}{P_i + R_i}, \tag{5.2.6}$$

Where n_i is the number of window samples of class $C_i \in C$. The accuracy becomes a less valuable metric when the data are unbalanced. Furthermore, a high recall results in poorer precision and vice versa . The low recall means that there is a higher number of FN than expected (labeled negative instead of positive). However, a high recall means that there is a high number of FP and this leads to a poorer accuracy. Nevertheless, a best result is achieved when F1, recall and precision are equal.
Loss functions are given by equations 2.3.25 and 2.3.26. Using the **binary-cross entropy loss** for the sigmoid layer when predicting attributes and the **Cross-Entropy Loss** for the Softmax when predicting activities.

5.3 EXPERIMENT AND RESULTS

the tCNN is a state-of-the-art network that has been deployed on the LARa dataset for the HAR problems. In the [NRMR+20] the authors are quite satisfied after finishing the experiment. The table 5.3.1 shows the result of their work. First of all, in this work, the tCCN that was previously trained on Lara dataset has been retrained to classify LARa-mocap and LARa-mbientlab and the results are presented in table 5.3.2and table 5.3.3 . Then the tCNN will be binarized to give a new tCNN called here BtCNN, for this purpose, the weights and biases of the new tCNN (BtCNN) will be binarized, all with three learning rate 10^{-4}, 10^{-5} and 10^{-6}. However, as it can be seen on the

Figure5.3.1, the networks will be binarized part after part. First the conv1 layer then the first two layers then the first three and finally all the four. Just like in [NRMR+20], this work will use the BtCNN to classify the LARa-Mbientlab and LARa-Mocap dataset with the softmax and sigmoid functions. To study the performance of BtCNN, the f1-score, f1_weighted and accuracy will be used as metrics. In addition, the memory consumption of the different configurations will be studied as well as the execution time.

5.3.1 *Baseline tCNN*

First of all, it should be noted that before the training, the input data are equipped with Gaussian noise of parameters $[\mu = 0, \sigma = 0,01]$ to simulate the inaccuracies of the sensors following [NRMR+20].

The authors of [NRMR+20] assert to have gotten state-of-the-art results after training the HAR using LARA dataset. In this subsection, the first step will be to train (under the same conditions) the tCNN previously trained on the LARa-dataset for the HAR. Table5.3.1 presents the results obtained by the authors. It is the overall accuracy and weighted F1 of HAR on the LARa OMoCap. Table5.3.2 and table5.3.3 represent the baseline. This means that they present the data of the training of HAR trained in this experience following the authors of [NRMR+20]. The first table is the performance of HAR on LARa-Mbientlab. It (HAR on LARa-Mbientlab) has been first trained with Softmax and later with Sigmoid or Vice versa. The second table is the performance of HAR on LARa-Mocap. It was also first trained with Softmax and then with Sigmoid or vice versa. mF1 ist the mean F1-scores, wF1 ist the weighted F1-score and Acc the accuracy. The best results are colored. Many authors consider wF1 as the best metric.

Metric	performance	
	Softmax	Attributes
Acc	0.690	0.751
wF1	0.644	0.736

Table 5.3.1: The overall accuracy and weighted F1 of HAR on the LARa OMoCap dataset from [NRMR+20]

It should be noted that the authors of [NRMR+20] only provided the best results of weighted F1-score and accuracy, unlike this work which also recorded the mean

F1-scores. They trained networks with just a single learning rate (10^{-4}) unlike this work that used three learning rate.

Table 5.3.2: Accuracies, F1-mean, and weighted F1 of HAR on the LARa mbientlab dataset.

Mbientlab						
		Activation	Learning rate	mF1	wF1	Accuracy
Baseline	Non-Binary	Attributes	$lr = 10^{-4}$	0.53 ± 0.004	0.666 ± 0.0035	0.68 ± 0.003
			$lr = 10^{-5}$	0.353 ± 0.035	0.554 ± 0.022	0.611 ± 0.012
			$lr = 10^{-6}$	0.125 ± 0.016	0.368 ± 0.019	0.493 ± 0.01
		Softmax	$lr = 10^{-4}$	0.666 ± 0.0028	0.707 ± 0.0008	0.719 ± 0.021
			$lr = 10^{-5}$	0.491 ± 0.025	0.629 ± 0.019	0.672 ± 0.01
			$lr = 10^{-6}$	0.298 ± 0.009	0.456 ± 0.005	0.566 ± 0.003

Table 5.3.3: accuracies, F1-mean, and weighted F1 of HAR on the LARa Mocap dataset.

Mocap						
		Activation	Learning rate	mF1	wF1	Accuracy
Baseline	Non-Binary	Attributes	$lr = 10^{-4}$	0.518 ± 0.008	0.633 ± 0.008	0.625 ± 0.006
			$lr = 10^{-5}$	0.213 ± 0.017	0.391 ± 0.02	0.471 ± 0.021
			$lr = 10^{-6}$	0.225 ± 0.015	0.355 ± 0.011	0.358 ± 0.013
		Softmax	$lr = 10^{-4}$	0.646 ± 0.004	0.678 ± 0.002	0.695 ± 0.0
			$lr = 10^{-5}$	0.19 ± 0.0	0.36 ± 0.0	0.49 ± 0.0
			$lr = 10^{-6}$	0.25 ± 0.01	0.409 ± 0.01	0.517 ± 0.005

Regarding the reconstruction of the baseline, the results obtained in this thesis and those of [NRMR+20] are not that different. In fact, the best result obtained by [NRMR+20] was 69% Accuracy with the Softmax and 75.1% Accuracy with the attributes during the LARA mocap dataset training. However, the result of this thesis observes a best result of 71.4% and 68.8% respectively according to table 5.3.3. This is not too different from the results obtained by [NRMR+20]. The binarization can be pursued in order to know how the performance behaves when the network are binarized during training, the time for the training and their memory consumption.

Moreover, Softmax has the best result in the two tables. E.g. LARa-mbientlab softmax has registered 71.9% and LARa-mocap 69.5% of Accuracy.

5.3.2 *BtCNN for HAR on LARa*

After obtaining the results at the state-of-the-art in the baseline. The method of this thesis is now going to be applied to measure the influence of binarization on the HAR performance. A part-after-part binarization of the network will follow. In other words, each time a certain part of the network will be binarized with a certain "Learning Rate" and it will consist in finding out which learning rate has the best prediction, which speed has the binarization and training due to the weight update, where the point of false binarization is and how the accuracy behaves. The figure5.3.1 shows the steps of the binarization process. The first convolution layer has been binarized, then the first 2 layers, then the first 3 layers and then all the 4 layers. Moreover, it tries to find out what happens when a certain part of the network is binarized and how far it goes. Which speed has the binarization and training when the weight is updated. Where is the point of false binarization. How does the accuracy behave. The Configurations are considered as follows:

- Conv[1] : only the first Conv layer is binarized during the training,
- Conv[1-2]: the first and the second Layer are binarized during the training,
- Conv[1-3]: Conv1, Conv2, Conv3 are binarized during the training,
- Conv[1-4]: Conv1, Conv2, Conv3 and Conv4 are binarized during the training.
- lr represents the learning rate
- Two LARa dataset: mocap and mbientlab.
- Two classifier for attribute representation and class activity respectively i.e., Sigmoid for attribute representation and Softmax class activity
- the best results are represented in blue or in green.

The networks are therefore named as follows: Conv[1-j], where $j \in \{1, 2, 3, 4\}$, representing the number of binary convolutional layers at instant j itself (in general the first j layers). lr is the learning rate. The results are presented in the tables and are represented according to the type of LARa and classifier.

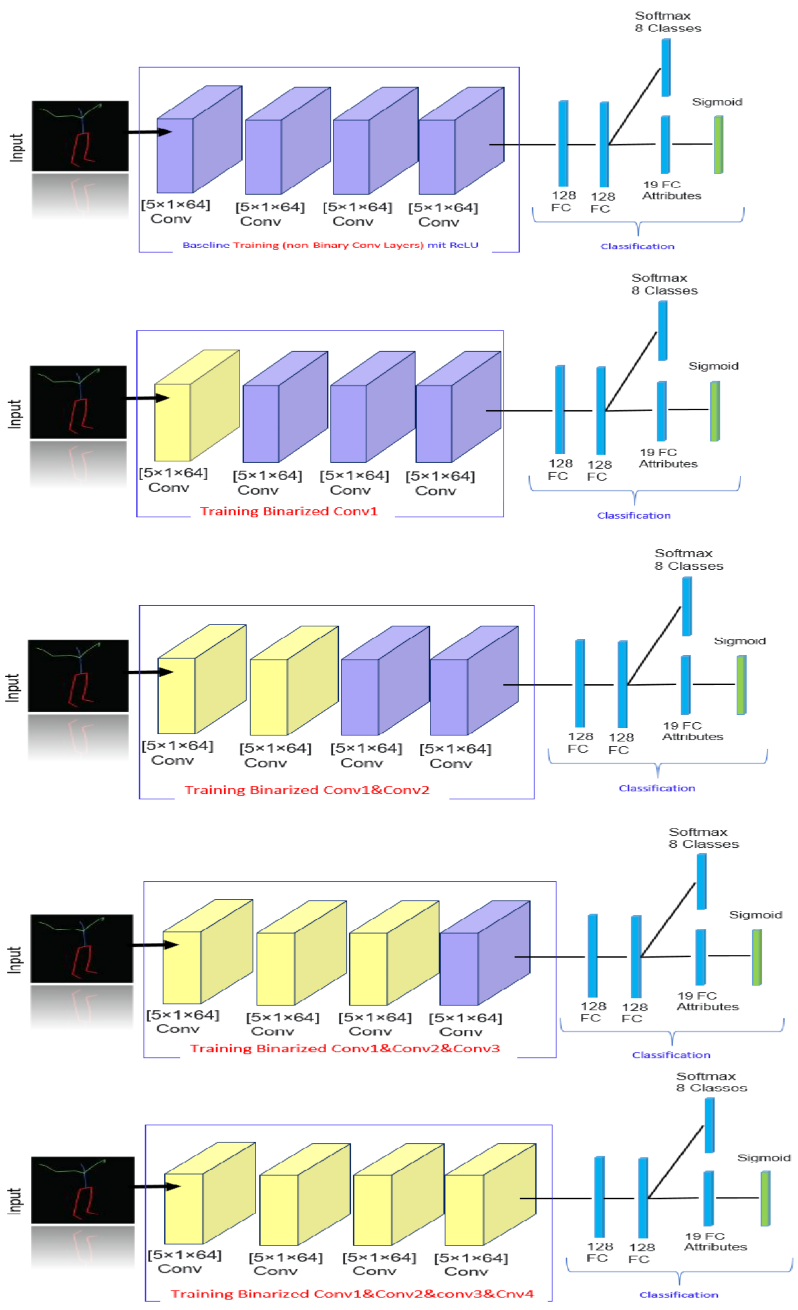


Figure 5.3.1: Training of tCNN proposed by [NRMR+20] The layers in yellow are the layers that are being trained/binarized while those in blue are still to be binarized. On the first figure, no layer is yet binarized (this is the original tCNN), on the second figure, the first layer is binarized (BtCNN), on the third figure, the first and second layers, on the fourth, the first three and on the fifth, all four layers

.

Figure 5.3.1 shows the reconstruction of the tCNN. The figure contains 5 Configurations. The first one is the original tCNN of [NRMR+20]. The second is the first configuration of our method i.e. Conv[1] explained above, the third is Conv[1-2], the fourth represents Con[1-3] and the fifth represents Conv[1-4]. Thus, the yellow color represents the binarized layer during the training. The configurations will be evaluated to obtain the layer or the configuration with state-of-the-art metrics. The first configuration has already been evaluated at the baseline and will not be discussed again.

ACCURACY AND SCORE First, it should be considered that, Table 5.3.4 presents the performance of the BtCNN solving HAR on the LARa mbientlab with the Attributes representation. Table 5.3.5 presents the performance of the BtCNN solving HAR on the LARa mbientlab With Softmax. Table 5.3.6 presents the performance of the BtCNN solving HAR on the LARa mocap With Softmax. Table 5.3.7 presents the performance of the BtCNN solving HAR on the LARa mocap With the Attributes representation.

Table 5.3.4: Binary training results of BtCNN on the LARa mbientlab using Attribute representation

Mbientlab							
		Learn. rate	Metrics	Conv1	Conv[1 − 2]	Conv[1 − 3]	Conv[1 − 4]
Binary training	Sigmoid(Attributes)	$lr = 10^{-4}$	mF1	0.527 ± 0.006	0.529 ± 0.002	0.069 ± 0.032	0.053 ± 0.35
			wF1	0.666 ± 0.004	0.669 ± 0.003	0.21 ± 0.150	0.143 ± 0.154
			Acc	0.688 ± 0.004	0.686 ± 0.003	0.347 ± 0.186	0.257 ± 0.198
		$lr = 10^{-5}$	mF1	0.272 ± 0.143	0.029 ± 0.020	0.057 ± 0.032	0.069 ± 0.032
			wF1	0.478 ± 0.111	0.024 ± 0.031	0.149 ± 0.150	$0,210 \pm 0.150$
			Acc	0.569 ± 0.071	0.076 ± 0.013	0.272 ± 0.185	0.348 ± 0.186
		$lr = 10^{-6}$	mF1	0.261 ± 0.080	0.065 ± 0.049	0.091 ± 0.033	0.069 ± 0.032
			wF1	0.472 ± 0.060	0.116 ± 0.136	0.280 ± 0.120	0.210 ± 0.150
			Acc	0.538 ± 0.041	0.155 ± 0.120	0.407 ± 0.147	0.348 ± 0.186

Table 5.3.4 presents the binary training performances of BtCNN on the LARa mbientlab using attribute representation. With the learning rate $lr = 10^{-4}$ in table 5.3.4, the first configurations Conv[1] and Conv[1-2] behave similarly and show relatively good results compared to the other configurations. However, the two other Conv[1-3]

and Conv[1-4] configurations drop and show rather low results. At learning rates $lr = 10^{-5}$ and $lr = 10^{-6}$, quite low results for almost all the configurations are observed this may be because the learning rate is high. The overall accuracy is 68.8% and the overall weighted F1 is 66.9% for 5.3.4. Therefore both the accuracy and weighted F1 thus show a gap of -7% compared to the result of the-state-of-the-art.

Table 5.3.5: Binary training results of BtCNN on the LARa mbientlab using Softmax

Mbientlab							
		Learn. rate	Metrics	Conv1	Conv[1 − 2]	Conv[1 − 3]	Conv[1 − 4]
Binary training	Softmax	$lr = 10^{-4}$	mF1	0.661 ± 0.006	0.648 ± 0.004	0.009 ± 0.0	0.095 ± 0.0
			wF1	0.703 ± 0.003	0.690 ± 0.004	0.333 ± 0.0	0.333 ± 0.0
			Acc	0.714 ± 0.004	0.706 ± 0.005	0.499 ± 0.0	0.499 ± 0.0
		$lr = 10^{-5}$	mF1	0.475 ± 0.069	0.095 ± 0.0	0.095 ± 0.00	0.095 ± 0.0
			wF1	0.614 ± 0.036	0.332 ± 0.0	0.333 ± 00	$0,333 \pm 0.0$
			Acc	0.672 ± 0.211	0.499 ± 0.0	0.449 ± 0.0	0.499 ± 0.0
		$lr = 10^{-6}$	mF1	0.449 ± 0.017	0.214 ± 0.059	0.096 ± 0.00	0.096 ± 0.00
			wF1	0.597 ± 0.014	0.406 ± 0.034	0.333 ± 0.00	0.333 ± 0.001
			Acc	0.657 ± 0.01	0.533 ± 0.017	0.449 ± 0.00	0.499 ± 0.00

Table5.3.5 presents the performance of the BtCNN solving HAR on the LARa mbientlab with softmax. This performance behaves the same way as the previous table with the learning rate at $lr = 10^{-4}$. In Table 5.3.5 the first configurations Conv[1] and Conv[1-2] show relatively good results compared to the other configurations. However, the two other Conv[1-3] and Conv[1-4] configurations drop and show low results. Unlike Table 5.3.4, compared to the other configurations,this table shows good scores for all the learning rates when binarizing the first layer. The overall accuracy is 71.4% and the overall weighted F1 is 70.3% for 5.3.5. Therefore, it overcomes the state-of-the-art performance with an accuracy of +2% and a weighted F1 of 5-6% compared to the result of the state-of-the-art.

Table 5.3.6 presents the Binary training performance of BtCNN on the LARa Mocap using Softmax. It behaves the same way as the previous table With the learning rate $lr = 10^{-4}$. In Table ref tab:mocapSoft . Like the tables above, the performance is high during the training of convolution layers 1 and 2. But the other configurations show low results. The overall accuracy is 64.6% and the overall weighted F1 is 65% for 5.3.6. Therefore, Table 5.3.6 shows a gap of −5% accuracy and -1% compared to the result of the state of the art.

Table 5.3.6: Binary training results of BtCNN on the LARa Mocap using Softmax

Mocap							
		Learn. rate	Metrics	Conv1	Conv[1 − 2]	Conv[1 − 3]	Conv[1 − 4]
Binary training	Softmax	$lr = 10^{-4}$	mF1	0.503 ± 0.008	0.55 ± 0.051	0.089 ± 0.0	0.089 ± 0.0
			wF1	0.632 ± 0.005	0.636 ± 0.009	0.281 ± 0.0	0.281 ± 0.0
			Acc	0.646 ± 0.006	0.650 ± 0.006	0.452 ± 0.0	0.452 ± 0.0
		$lr = 10^{-5}$	mF1	0.109 ± 0.02	0.127 ± 0.038	0.089 ± 0.0	0.089 ± 0.0
			wF1	0.297 ± 0.015	0.310 ± 0.03	0.281 ± 0.0	0.281 ± 0.0
			Acc	0.46 ± 0.008	0.468 ± 0.016	0.452 ± 0.0	0.452 ± 0.0
		$lr = 10^{-6}$	mF1	0.264 ± 0.007	0.103 ± 0.013	0.105 ± 0.016	0.089 ± 0.0
			wF1	0.421 ± 0.007	0.290 ± 0.008	0.284 ± 0.003	0.282 ± 0.0
			Acc	0.526 ± 0.001	0.447 ± 0.005	0.453 ± 0.0	0.452 ± 0.0

Table 5.3.7: Binary training results of BtCNN on the LARa Mocap using Attribute representation

Mocap							
		Learn. rate	Metrics	Conv1	Conv[1 − 2]	Conv[1 − 3]	Conv[1 − 4]
Binary training	Sigmoid(Attributes)	$lr = 10^{-4}$	mF1	0.518 ± 0.0	0.241 ± 0.22	0.021 ± 0.00	0.057 ± 0.032
			wF1	0.63 ± 0.0	0.295 ± 0.283	0.012 ± 0.00	0.149 ± 0.132
			Acc	0.620 ± 0.0	0.329 ± 0.248	0.081 ± 0.00	0.274 ± 0.178
		$lr = 10^{-5}$	mF1	0.025 ± 0.0	0.044 ± 0.0	0.021 ± 0.00	0.055 ± 0.034
			wF1	0.017 ± 0.0	0.028 ± 0.0	0.012 ± 0.00	0.147 ± 0.135
			Acc	0.097 ± 0.0	0.092 ± 0.0	0.081 ± 0.00	0.266 ± 0.185
		$lr = 10^{-6}$	mF1	0.206 ± 0.066	0.026 ± 0.0	0.055 ± 0.034	0.021 ± 0.00
			wF1	0.346 ± 0.046	0.017 ± 0.0	0.147 ± 0.135	0.012 ± 0.00
			Acc	0.394 ± 0.026	0.098 ± 0.001	0.267 ± 0.185	0.081 ± 0.00

Table 5.3.7 presents Binary training results of BtCNN on the LARa Mocap using Attribute representation. Unlike the other tables above, table 5.3.7 presents a relatively low performance. This means that almost all the results are below 50%. Only the binarization training of the first layer Con1 showed relatively good results at the learning rate of 10^4. The overall accuracy is 62% and the overall weighted F1 is 63% for 5.3.7.

In summary, With the learning rate $lr = 10^{-4}$ in Table 5.3.4, the first layers Conv[1] and Conv[1-2] behave similarly and show relatively good results. However, the two other Conv[1-3] and Conv[1-3] configurations deteriorate and show rather low results. Also, at learning rates $lr = 10^{-5}$. and $lr = 10^{-6}$., a quite low result of almost all the configurations is observed. The architecture shown in table 5.3.5 also behaves in the same way as the previous table with one difference: the first configuration Conv[1] shows better results with all learning rates. It is the table that shows the best result with a score of 71.14% for the accuracy. However, Table 5.3.6 behaves almost the same way as the Table 5.3.4. Nevertheless, Table 5.3.7 showed relatively poor results. Only the Conv[1] have a result above the average. It may be appropriate to stop the binarization at the level of Conv[1-2] because this is where the best results of the binarization occur.

Furthermore, the wF1 for the testing dataset (wF1 testing) of the best performance in the four tables above according to the learning rate are presented in tables 5.3.8 and 5.3.9

Table 5.3.8: Learning rate and wF1-testing results of best performance using BtCNN for LARa-mbientlab (results of best performance are highlighted in Table 5.3.5 and 5.3.4).

Learning rate and wF1 testing result [lr(wF1)]				
Mbientlab				
Classifier	Conv[1]	Conv[1-2]	Conv[1-3]	Conv[1-4]
Sigmoid	$10^{-4}(0.723)$	$10^{-4}(0.716)$	$10^{-6}(0.412)$	$10^{-6}(0.412)$
Softmax	$10^{-4}(0.755)$	$10^{-4}(0.738)$	$10^{-4}(0,41)$	$10^{-4}(0.412)$

Table 5.3.8 presents wF1 for the testing dataset (wF1 testing) and the learning rate at which a configuration could reach its best performance using LARa-Mbientlab. These best performances are highlighted in blue in the four tables above. For instance, the binarization of the first layer during training (Conv[1]) had the best performance

(Acc=71.4% and wF1= 70.3%) at the learning rate 10^4 for both the sigmoid function and the softmax function.
Moreover, Conv[1] presents the best wF1-testing result with 72.3% using sigmoid and 75.5% using softmax.

Table 5.3.9: Learning rate best results of BtCNN on the LARa-mocap (results are in Table 5.3.6 and 5.3.7).

Learning rate and wF1 testing result [lr(wF1)]				
Mocap				
classifier	Conv[1]	Conv[1-2]	Conv[1-3]	Conv[1-4]
Sigmoid	$10^{-4}(0.691)$	$10^{-4}(0.711)$	$10^{-6}(0.343)$	$10^{-4}(0,344)$
Softmax	$10^{-4}(0.694)$	$10^{-4}(0.72)$	$10^{-6}(0.36)$	$10^{-6}(0.35)$

Table 5.3.9 presents wF1 for the testing dataset (wF1 testing) and the learning rate at which a configuration could reach its best performance using LARa-mocap. These best performances are also presented in blue in the four tables above. For instance, the binarization of the first layer during training (Conv[1]) had the best performance (Acc=64.6%and wF1= 63.2%) at the learning rate 10^4 for both the sigmoid function and the softmax function.
Moreover, Conv[1-2] presents the best wF1-testing result with 71.1% using sigmoid and 72% using softmax.

In summary, The BtCNN using Softmax on the LARa mbientlab shows the highest score with 71.14% (Softmax) and 68.8% (Sigmoid) on the Conv[1] network and with a learning rate of lr=10^{-4}. However, the binarization of weights and activations during the training does not influence negatively impact the performance of the configurations 1 and 2 (i.e. Conv[1] and Conv[1-2]), which rather present a state-of-the- art results while the others configurations deteriate or drop in most of the cases.

PERIODS Furthermore, the time (in sec.) taken to test the BtCNN after training for each configuration was recorded and then placed in the following tables 5.3.11 to 5.3.13.

Table 5.3.10: Testing time of the BtCNN training for HAR on the LARa mbienlab in second.

mbientlab						
BNN	Softmax	Learning rate	Conv[1]	Conv[1-2]	Conv[1-3]	Conv[1-4]
		10^{-4}	10.97	11.124	10.99	10.90
		10^{-5}	10.90	11.023	11.006	11.078
		10^{-6}	10.96	11.07	11.048	10.973

Table 5.3.10 shows the testing time of each configuration using LARa- mbientlab and softmax. It is the time needed for a configuration to be tested. The configuration Conv[1-2] takes more time for testing while the first configuration Con[1] takes less time with 10.90 sec.

Table 5.3.11: Testing time of the BtCNN training for HAR on the LARa mbienlab in sec.

mbientlab						
BNN	Sigmoid	Learning rate	Conv[1]	Conv[1-2]	Conv[1-3]	Conv[1-4]
		10^{-4}	25.43	25.44	25.30	25.34
		10^{-5}	25.43	25.47	25.34	25.34
		10^{-6}	25.42	25.48	25.41	25.36

Table 5.3.11 shows the testing time of each configuration using LARa- mbientlab and sigmoid. It is the time taken for a configuration to be tested. The configuration Conv[1-2] takes more time for testing with 25.48 sec. while the first configuration takes less time with 25.30 sec.

Table 5.3.12: Testing time of the BtCNN training of HAR on the LARa Mocap in sec.

Mocap						
BNN	Sigmoid	Learning rate	Conv[1]	Conv[1-2]	Conv[1-3]	Conv[1-4]
		10^{-4}	148.38	149.28	148.97	148.82
		10^{-5}	153.61	155.62	153.73	151.34
		10^{-6}	147.94	149.52	148.79	149.20

Table 5.3.12 shows the testing value of each configuration using LARa-mocap and sigmoid. It is the time needed for a configuration to be tested. The configuration

Conv[1-3] takes more time for testing 153.73 sec while the configuration Conv[1] takes less time 147.94 sec.

Table 5.3.13: Testing time of the BtCNN training of HAR on the LARa Mocap in sec.

Mocap						
BNN	Softmax	Learning rate	Conv[1]	Conv[1-2]	Conv[1-3]	Conv[1-4]
		10^{-4}	114.78	112.92	114.69	112.71
		10^{-5}	116.18	112.97	112.80	112.36
		10^{-6}	112.25	118.21	112.69	112.32

Table 5.3.13 shows the testing value of each configuration using LARa-mocap and softmax. It is the time needed for a configuration to be tested. The configuration Conv[1-2] takes more time for testing 118.21 sec. while the configuration Conv[1] takes less time 112.25 sec.

In summary, The training with Sigmoid takes a little longer. However, it is clear that all the configurations show a slight difference in terms of duration. Nevertheless Conv[1] shows a slight difference with a shorter duration of 112.25 sec. and 10.90 sec for LARa-mbientlab using softmax, then 147.94 sec. for LARa-mocap using Sigmoid. But will be overtaken by Conv[1-3] with a score of 26.30 sec. in the sigmoid classification using the mbientlab.

Tables 5.3.8 and 5.3.9 show the configurations of BtCNN with good performance, which performed better at testing. The time of these best performing configurations are shown in tables 5.3.14 and 5.3.15 in second. The performances of short duration are highlighted in color.

Table 5.3.14: Testing time of BtCNN with the best performance during training on the LARa mbientlab in sec

.

Mbientlab						
	Classifier	Type	Conv[1]	Conv[1-2]	Conv[1-3]	Conv[1-4]
BNN	Sigmoid	best lr	10^{-4}	10^{-4}	10^{-6}	10^{-6}
		training	1834.86	1855.00	1820.31	1807.93
		testing	25.49	25.80	25.49	25.40
	Softmax	best lr	10^{-4}	10^{-4}	10^{-4}	10^{-4}
		training	1198.69	1197.17	1174.53	1179.58
		testing	10.94	10.88	10.97	10.97

table 5.3.14 gives the duration of the BtCNN best performances on the LARa mbientlab using Sigmoid or Softmax. Unlike the four figures above, only the test and validation values of the configurations with the best performance will be shown in the table 5.3.14. In this table the most binarized layers are the fastest during both training and testing, i.e. Con[1-4] is the fastest in this table with a duration of 1807.93 sec at validation and 25.40 sec for the test.

Table 5.3.15: Testing time of BtCNN with the best performance during training on the LARa Mocap in sec

.

Mocap						
	Classifier	Type	Conv[1]	Conv[1-2]	Conv[1-3]	Conv[1-4]
BNN	Sigmoid	best lr	10^{-4}	10^{-4}	10^{-6}	10^{-4}
		training	31949.14	32035.97	31184.88	31594.1
		testing	153.99	147.77	148.974	154.72
	Softmax	best lr	10^{-4}	10^{-4}	10^{-6}	10^{-4}
		training	26540.42	26383.63	25482.93	25421.8
		testing	113.15	118.33	114.12	114.24

5.3.15 gives the duration of the BtCNN best performances on the LARa mocap using Sigmoid or Softmax. Unlike the four figures above, only the test and validation values of the configurations with the best performance will be shown in table 5.3.15. In the table, it can be seen that the most binarized layers are the fastest during both training and testing i.e., Con[1-4] is the fastest in this table with a duration of 25421.8 sec at validation and Conv[1-3] is the fastest at the test with 114.12 sec otherwise.

Observing these two tables, it can be deduced that it is not always the best performers who are faster. In other words, observing the table 5.3.14, Conv[1-3] and Conv[1-4] are rather faster during the validation and the test compared to the first two Configurations. Likewise in the table 5.3.15 the configuration Con[1-4] is faster than the others. Therefore, the speed Conv[1-3] and Con[1-4] could be a consequence of the binarization because in these two parts, all or almost all the layers are binarized Thus, reduction of memory is reduction of time. This confirms the assertion of [QGL+20] when the authors say that their work has largely reduced inference time.

The authors of [NRMR+20] did their experiments exclusively on the Mocap unlike this thesis which also performed on the mbientlab. Finally, the performance of the BtCNN on the Mocap Dataset of validation in terms of accuracy and F1-score is lower than that of the tCNN of [NRMR+20]. Precisely, of about -4% using softmax, -13% using the attribute in terms of accuracy, then in terms of F1 score, by about -1% using softmax and about -10% using attributes.

6 CONCLUSION

The aim of this thesis was to binarize neural networks in order to drastically decrease the high demand in memory of HAR. This means that neural networks use binary weights and activations at train-time in the Forward propagation, while real values are used in the backward pass when computing the parameters gradients.
For this purpose, experiments were conducted to compare the results obtained with those proposed by the authors of [NRMR+20]. To do this, a Sign function was implemented to binarize the weights and activations of the different layers during the training. At the backpropagation, the gradient had to be computed while keeping the full-precision numbers. And the training had to be done through several epochs.
The tCNN intended for the LARa dataset was utilized for this purpose. The tCNN layers have been binarized and trained for this objective. Thus, different layers were binarized independently during training. Experiments showed that the reconstructed tCNN or BtCNN in section 4.3, achieves nearly state-of-the-art results. Therefore, this experience presented a relatively HAR performance compared with the baseline or state-of-the-art despite a little margin of difference. In the experiments, the weighted F1-score recorded a slight difference of only 1% with the state-of-the-art results from [NRMR+20]. Thus, despite the loss of information, a good performance was achieved.

Moreover, during this experiment four binary configurations were created and explained in the section 5.3.2. In the experiments, different proportions of the layers were binarized. The first layers learned binary features that are activated under different activities. However, the deeper the layers, the more complex the features become, and binarization ends up deleting vital information of these features. This subsequently influences the performance for the deeps layers. However, the more the layers were binarized the faster the training was. In summary, even though the number of bits has been drastically reduced, binarization was achieved for the first and second layers without affecting the performance.
Finally, future work could improve the work while considering data storage. Because the program saves only the memory occupied by all layers. That is, the convolutional part and the MLP part (FC and Classifier) combined. So, the memory is stored in float and not in binary because of the MLPs. A solution will be to separate the networks into two parts. Precisely, one part for the convolution layers (Binary), and another part

for the MLPs. Thus, the data will be stored in binary in the convolutional part and then in float in the MLP part (if the FC layers are not yet binarized). Then the memory occupied during training at each configuration for BtCNN will be obtained. It will be useful in the future to have two storages: one for convolutional networks and the other for MLPs. Another idea would be to binarize the entire network, i.e.,including the FC layers.

LIST OF FIGURES

BIBLIOGRAPHY

[Aka20] Akagündüz, Erdem: *Binary Neural Networks*. https://ee545.cankaya.edu.tr/uploads/files/EE545-W10.zip, 2020. – Accessed: 2022-07-13

[Bas20] Basta, Nikola: *The Differences between Sigmoid and Softmax Activation Functions*. https://medium.com/arteos-ai/the-differences-between-sigmoid-and-softmax-activation-function-12adee8cf322, 2020. – Accessed: 2022-07-25

[BBS13] Bulling, Andreas ; Blanke, Ulf ; Schiele, Bernt: A Tutorial on Human Activity Recognition Using Body-Worn Inertial Sensors. In: *ACM Computing Surveys* 46 (2013), 01. http://dx.doi.org/10.1145/2499621. – DOI 10.1145/2499621

[Bru20] Bruin, Barry de: *Deep Neural Network optimization: Binary Neural Networks*. https://slideplayer.com/slide/17585887/, 2020. – Accessed: 2022-06-25

[Bus18] Bushaev, Vitaly: *Understanding RMSprop — faster neural network learning*. https://t.co/lovpLkfVYE, 2018. – Accessed: 2022-05-30

[CB16] Courbariaux, Matthieu ; Bengio, Yoshua: BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. In: *CoRR* abs/1602.02830 (2016). http://arxiv.org/abs/1602.02830

[CHS+16] Courbariaux, Matthieu ; Hubara, Itay ; Soudry, Daniel ; El-Yaniv, Ran ; Bengio, Yoshua: *Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1*. http://dx.doi.org/10.48550/ARXIV.1602.02830. Version: 2016

[Dat20] Datta, Leonid: A Survey on Activation Functions and their relation with Xavier and He Normal Initialization. In: *CoRR* abs/2004.06632 (2020). https://arxiv.org/abs/2004.06632

[Dil19] Dilmegani, Cem: *Machine Learning Accuracy: True vs. False Positive/Negative*. https://cs231n.github.io/convolutional-networks/, 2019. – Accessed: 2022-07-08

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – http://www.deeplearningbook.org

[GLR+17] Grzeszick, René ; Lenk, Jan M. ; Rueda, Fernando M. ; Fink, Gernot A. ; Feldhorst, Sascha ; Hompel, Michael ten: Deep Neural Network based Human Activity Recognition for the Order Picking Process. In: *Proceedings of the 4th international Workshop on Sensor-based Activity Recognition and Interaction* (2017)

[Gom18] Gomez, Raul: *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. https://gombru.github.io/2018/05/23/cross_entropy_loss/, 2018. – Accessed: 2022-07-20

[HBFS01] Hochreiter, S. ; Bengio, Y. ; Frasconi, P. ; Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: Kremer, S. C. (Hrsg.) ; Kolen, J. F. (Hrsg.): *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001

[HCS+16] Hubara, Itay ; Courbariaux, Matthieu ; Soudry, Daniel ; El-Yaniv, Ran ; Bengio, Yoshua: Binarized Neural Networks. In: Lee, D. (Hrsg.) ; Sugiyama, M. (Hrsg.) ; Luxburg, U. (Hrsg.) ; Guyon, I. (Hrsg.) ; Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 29, Curran Associates, Inc., 2016

[Hoc98] Hochreiter, Sepp: The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (1998), 04, S. 107–116. http://dx.doi.org/10.1142/S0218488598000094. – DOI 10.1142/S0218488598000094

[HSS12] Hinton, Geoffrey ; Srivastava, Nitish ; Swersky, Kevin: Overview of mini-batch gradient descent. In: *Neural Networks for Machine Learning* 575 (2012), Nr. 8

[HW59] Hubel, D. H. ; Wiesel, T. N.: Receptive fields of single neurones in the cat's striate cortex. In: *The Journal of Physiology* 148 (1959), Nr. 3, 574-591. http://dx.doi.org/https://doi.org/10.1113/jphysiol.1959.sp006308. – DOI https://doi.org/10.1113/jphysiol.1959.sp006308

[HWG$^+$19] HELWEGEN, Koen ; WIDDICOMBE, James ; GEIGER, Lukas ; LIU, Zechun ; CHENG, Kwang-Ting ; NUSSELDER, Roeland: *Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization*. http://dx.doi.org/10.48550/ARXIV.1906.02107. Version: 2019

[Jon17] JONES, M. T.: *A neural networks deep dive*. https://developer.ibm.com/articles/cc-cognitive-neural-networks-deep-dive/, 2017. – Accessed: 2022-05-30

[KA21] KOPPERT-ANISIMOVA, Inara: *Understanding RMSprop — faster neural network learning*. https://medium.com/unpackai/cross-entropy-loss-in-ml-d9f22fc11fe0, 2021. – Accessed: 2022-06-30

[LC11] LEE, Young Seol ; CHO, Sung Bae: Activity recognition using hierarchical hidden markov models on a smartphone with 3D accelerometer. In: *Hybrid Artificial Intelligent Systems - 6th International Conference, HAIS 2011, Proceedings*. Germany : Springer Verlag, 2011 (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) PART 1). – ISBN 9783642212185, S. 460–467. – Funding Information: Acknowledgments. This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (No. 2009-0083838).

[Lin17] LIN, Fang: *XNOR Neural Networks on FPGA*. http://cs231n.stanford.edu/reports/2017/pdfs/118.pdf, 2017. – Accessed: 2022-06-25

[LKBS20] LI, Fei-Fei ; KIML, Moo J. ; BANSAL, Dhruva ; SHEN, William: *Convolutional Neural Networks (CNNs / ConvNets)*. https://cs231n.github.io/convolutional-networks/, 2020. – Accessed: 2022-07-25

[LMRA$^+$21] LÜDTKE, Stefan ; MOYA RUEDA, Fernando ; AHMED, Waqas ; FINK, Gernot ; KIRSTE, Thomas: Human Activity Recognition using Attribute-Based Neural Networks and Context Information, 2021

[MP43] MCCULLOCH, Warren ; PITTS, Walter: A Logical Calculus of Ideas Immanent in Nervous Activity. In: *Bulletin of Mathematical Biophysics* 5 (1943), S. 127–147

[MRGF+18] Moya Rueda, Fernando ; Grzeszick, René ; Fink, Gernot A. ; Feldhorst, Sascha ; Ten Hompel, Michael: Convolutional Neural Networks for Human Activity Recognition Using Body-Worn Sensors. In: *Informatics* 5 (2018), Nr. 2. http://dx.doi.org/10.3390/informatics5020026. – DOI 10.3390/informatics5020026. – ISSN 2227–9709

[Nat18] Natsu: *Paper Explanation: Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or -1*. https://mohitjain.me/2018/07/14/bnn/, 2018. – Accessed: 2022-07-03

[NGLK18] Neves, Ana C. ; Gonzalez, Ignacio ; Leander, John ; Karoumi, Raid: A New Approach to Damage Detection in Bridges Using Machine Learning, 2018. – ISBN 978–3–319–67442–1, S. 73–84

[NRMR+20] Niemann, Friedrich ; Reining, Christopher ; Moya Rueda, Fernando ; Nair, Nilah R. ; Steffens, Janine A. ; Fink, Gernot A. ; Hompel, Michael ten: LARa: Creating a Dataset for Human Activity Recognition in Logistics Using Semantic Attributes. In: *Sensors* 20 (2020), Nr. 15. http://dx.doi.org/10.3390/s20154083. – DOI 10.3390/s20154083. – ISSN 1424–8220

[NZ21] Neta Zmora, Jay R. Hao Wu W. Hao Wu: *Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training with NVIDIA TensorRT*. https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training 2021. – Accessed: 2022-07-03

[OR16] Ordóñez, Francisco J. ; Roggen, Daniel: Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. In: *Sensors* 16 (2016), Nr. 1. http://dx.doi.org/10.3390/s16010115. – DOI 10.3390/s16010115. – ISSN 1424–8220

[OZCR20] Otness, Alfredo Canzianiand K. ; Zhang, Xiaoyi ; Chandrakaladharan, Shreyas ; Raach, Chady: *Overtraining and regulation*. https://atcold.github.io/pytorch-Deep-Learning/en/week14/14-3/, 2020. – Accessed: 2022-05-20

[PG17] Patterson, J. ; Gibson, A.: *Deep Learning: A Practitioner's Approach*. O'Reilly Media, 2017 https://books.google.de/books?id=qrcuDwAAQBAJ. – ISBN 9781491914236

[QGL+20] Qin, Haotong ; Gong, Ruihao ; Liu, Xianglong ; Bai, Xiao ; Song, Jingkuan ; Sebe, Nicu: Binary Neural Networks: A Survey. In: *Pattern Recognition* 105 (2020), 02, S. 107281. http://dx.doi.org/10.1016/j.patcog.2020.107281. – DOI 10.1016/j.patcog.2020.107281

[Rad20] Radečić, Dario: *Softmax Activation Function Explained*. https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60, 2020. – Accessed: 2022-07-21

[RC15] Ronao, Charissa Ann ; Cho, Sung Bae: Deep convolutional neural networks for human activity recognition with smartphone sensors. In: Arik, Sabri (Hrsg.) ; Huang, Tingwen (Hrsg.) ; Lai, Weng Kin (Hrsg.) ; Liu, Qingshan (Hrsg.): *Neural Information Processing - 22nd International Conference, ICONIP 2015, Proceedings*. Germany : Springer Verlag, 2015 (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)). – ISBN 9783319265605, S. 46–53. – Funding Information: This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2015-R0992-15-1011) supervised by the IITP (Institute for Information & communications Technology Promotion). Publisher Copyright: © Springer International Publishing Switzerland 2015.; 22nd International Conference on Neural Information Processing, ICONIP 2015 ; Conference date: 09-11-2015 Through 12-11-2015

[RNMR+19] Reining, Christopher ; Niemann, Friedrich ; Moya Rueda, Fernando ; Fink, Gernot A. ; Hompel, Michael ten: Human Activity Recognition for Production and Logistics—A Systematic Literature Review. In: *Information* 10 (2019), Nr. 8. http://dx.doi.org/10.3390/info10080245. – DOI 10.3390/info10080245. – ISSN 2078–2489

[Ros58] Rosenblatt, Frank: The perceptron: a probabilistic model for information storage and organization in the brain. In: *Psychological review* 65 6 (1958), S. 386–408

[Rud16] Ruder, Sebastian: An overview of gradient descent optimization algorithms. In: *CoRR* abs/1609.04747 (2016). http://arxiv.org/abs/1609.04747

[RZL17] Ramachandran, Prajit ; Zoph, Barret ; Le, Quoc V.: Searching for Activation Functions. In: *CoRR* abs/1710.05941 (2017). http://arxiv.org/abs/1710.05941

[Sar14] Sarkar, A. M. J.: Hidden Markov Mined Activity Model for Human Activity Recognition. In: *International Journal of Distributed Sensor Networks* 10 (2014), Nr. 3, 949175. http://dx.doi.org/10.1155/2014/949175. – DOI 10.1155/2014/949175

[SP22] Santos, Claudio Filipi G. ; Papa, Jo

[Spa22] Sparrow, Jack: *Intuition d'Adam Optimizer*. https://fr.acervolima.com/intuition-d-adam-optimizer/, 2022. – Accessed: 2022-08-19

[Var21] Varman, Rahul: *Binary Neural Networks — Future of low-cost neural networks?* https://slideplayer.com/slide/17585887/, 2021. – Accessed: 2022-07-03

[WJZ+20] Wu, Hao ; Judd, Patrick ; Zhang, Xiaojie ; Isaev, Mikhail ; Micikevicius, Paulius: Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. In: *CoRR* abs/2004.09602 (2020). https://arxiv.org/abs/2004.09602

[YNS+15] Yang, Jianbo ; Nguyen, Minh N. ; San, Phyo P. ; Li, Xiaoli ; Krishnaswamy, Shonali: Deep Convolutional Neural Networks on Multichannel Time Series for Human Activity Recognition. In: *IJCAI*, 2015

[ZLL+14] Zeng, Daojian ; Liu, Kang ; Lai, Siwei ; Zhou, Guangyou ; Zhao, Jun: Relation classification via convolutional deep neural network. In: *Proceedings of COLING 2014, the 25th international conference on computational linguistics: technical papers*, 2014, S. 2335–2344

Eidesstattliche Versicherung

(Affidavit)

Kembou Nguefack, Emmanuel	189965
Name, Vorname (surname, first name)	Matrikelnummer (student ID number)

[✓ Bachelorarbeit (Bachelor's thesis)] ☐ Masterarbeit (Master's thesis)

Titel (Title)

Binary Temporal Convolutional Networks for the Logistic Activity Recognition Challenge (LARa) dataset.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 21.08.2022	[illegible]
Ort, Datum (place, date)	Unterschrift (signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 21.08.2022	[illegible]
Ort, Datum (place, date)	Unterschrift (signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**