POLITECNICO DI TORINO

Master degree course in Data Science and Engineering

Master Degree Thesis

Application of Transformers to edge-computing in ultra-low power devices

Supervisors
Doc. Daniele Jahier Pagliari
Doc. Alessio Burrello

Candidate
Francesco Bianco Morghet
matricola: 277910

Academic Year 2020-2021

Acknowledgements

Questa tesi segna il raggiungimento di una tappa importante nel percorso di formazione e crescita personale che non sarebbe stato possibile senza il sostegno fondamentale ricevuto in questi anni.

Voglio dedicare un ringraziamento speciale ai miei genitori Cristina e Michele per l'amore e sostegno incondizionati che mi hanno sempre dimostrato, per essere sempre stati al mio fianco e avermi trasmesso certezze nei momenti di difficoltà. Un grazie va anche a mia sorella Elena, perché anche se ogni tanto litighiamo, mi ha sempre sopportato e voluto bene per quello che sono.

Un ringraziamento ai miei nonni Battista e Francesco, che oggi non ci sono più, ma di cui non scorderò mai le storie e insegnamenti che mi hanno aiutato a crescere come persona. Un grazie a mia nonna Secondina per essersi sempre preoccupata per me, come le ricche cene dopo le lezioni al Politecnico, e non avermi mai fatto mancare nulla. E un grazie a mia nonna Anna, per le passeggiate in campagna, i pranzi alla domenica e tutti gli altri mille bei gesti di affetto.

Un grazie ai miei supervisori Daniele Jahier Pagliari e Alessio Burrello per la loro grande disponibilità e aiuto che si sono rivelati determinanti per lo svolgimento e compimento del lavoro di tesi.

Un grazie anche ai miei compagni di corso per avermi aiutato ad affrontare questi cinque anni, come ricordarmi le scadenze di cui mi sarei altrimenti dimenticato, o sistemare i miei bug nel codice e i paragrafi storti nelle presentazioni dei progetti di gruppo.

Grazie di cuore a tutti.

Summary

Low-power edge devices and IoT sensors are employed in many different tasks that benefit from machine learning techniques. However, the high resource requirements in terms of computing power, memory footprint and energy consumption make the deployment of Deep Learning models at the edge very challenging. In particular, an emerging class of deep learning models, the Transformers, has obtained state-of-the-art results in fields such as natural language processing (NLP) and computer vision (CV). On the other hand, typical Transformer models contain millions or billions of parameters, and perform billions of operations, which is unsuitable for execution on edge devices. The effectiveness of smaller-scale Transformers, instead, is largely unstudied.

This thesis focuses on applying transformers to hand movement classification based on surface electromyographic (sEMG) signals, a latency-sensitive application that cannot rely on cloud inference, and therefore must be executed on low-power edge devices. It is shown that Transformers can attain nearly the same accuracy of previous state-of-the-art architectures, with a complexity reduction of up to 5x in terms of memory footprint and number of multiply-accumulate operations. In particular, the number of parameters of the proposed models is in the range of 100k to 500k, which is several orders of magnitude lower than most state-of-the-art Transformers for other applications.

The thesis also explores another common practice in Transformers' literature, the use of pre-training, showing that fine-tuning a pre-trained model can improve accuracy even in the highly-compressed network architectures presented in this work. Accuracy improvements of 1-3% are observed on average.

As a last step towards optimizing Transformers for edge deployment, a Neural Architecture Search (NAS) is applied to substitute some of the self-attention layers in the network with simpler convolutions, without impairing

the final accuracy.

Contents

Chapter 1

Introduction

Machine Learning consists of a set of techniques aimed at solving a given task in an automated way by learning from past experience and data: during a so-called training phase, a learning algorithm learns how to perform a task by acquiring knowledge on past data; in the inference phase, the algorithm performs the previously learnt task on new unseen data.

Deep Learning [1], which is a branch of Machine Learning, has been gaining traction in the last few years thanks to its unmatched ability of solving complex tasks in different fields, such as Computer Vision [2, 3], Natural Language Processing [4, 5] and Time Series Analysis [6]. Traditionally, before applying any generic learning algorithm, source data were heavily transformed in a manually designed fashion in order to extract the most useful information in the most appropriate way for the given task [7]. This process, usually referred to as Feature Engineering, is task dependent and its effectiveness is limited by the domain knowledge and experiences of the humans involved in the design process. Deep Learning instead introduces the concept of learning also to this procedure by proposing a set of techniques that can learn from data which is close to its raw form: the two steps involving the extraction of a meaningful representation of data and the learning process over input data are merged into one single phase, which consists in the training of a Deep Neural Network (DNN). Prominent examples of DNNs vary according to the nature of the given task: Convolutional Neural Networks (CNNs) were developed to perform image classification and object recognition [2]; Recurrent Neural Networks (RNNs) and Transformers instead were designed to deal with sequential data, such as text, recorded speech and time series [4, 8].

Deep Learning has been made feasible for complex tasks not only thanks to several innovations in network design (such as Batch Normalization and non saturating ReLU activation), but also thanks to the availability of increasing large amounts of training data, to an easier access to parallel computing, and to the availability of application programming interfaces (APIs) that allow general-purpose computing on graphic processing units (GPUs) or other similar high-performance computing units, such as Tensor Processing Units (TPUs). In fact, the high degree of parallelism can shrink the time required for training a DNN by several orders of magnitude with respect to performing the same task on currently available CPUs [9, 10, 11].

Low-power embedded devices and IoT sensors are employed in many different tasks that would benefit from Machine Learning techniques; however the high resource requirements in terms of computing power, memory footprint and energy consumption make the usage of Deep Learning in edge devices very challenging.

A way to avoid these issues consists in adopting solutions based on cloud computing, which leverage the widespread availability of internet connectivity for sending the raw data to always-online clusters of computers that perform the required computations. However, due to the high and unpredictable latency of network connectivity, this solution can be employed only for non latency-sensitive applications. Additionally, the high bandwidth requirement for data intensive applications and large scale deployments further limit the suitability of this solution.

The opposite approach is based on edge computing, where inference is instead performed directly on embedded devices while training continues to be carried out in a centralized environment with high performing GPUs [12, 13]. In order to bring deep learning to edge inference, two solutions can be followed: on one hand the hardware can be optimized by employing hardware accelerators or application specific integrated circuits (ASICs) designed to deal with the given task in a more efficient way [14]; on the other hand, the task can be adapted by reducing its complexity in order to run on already existing constrained hardware. While the former can be suitable for certain applications, the latter is the most widely employed solution due to a number of reasons: from an economic standpoint, leveraging already-available general purpose microcontroller units presents a lower entry cost than designing and producing custom hardware, and this cost reduction is particularly appropriate for prototypes or for small and medium scale projects; furthermore a number of optimization techniques have proved to yield high complexity reductions to deep learning models with minor accuracy drops [15, 16, 17, 18, 19, 20].

Complexity reductions are based on the fact that currently available DNN architectures present a high level of information redundancy: for this reason, a number of methods borrowed from approximate computing can be applied with very promising results [21]. Instances of such techniques with proven effectiveness can be distinguished into different categories: precision reduction [18, 19], such as quantization, where floating point operations are turned into fixed point operations; cardinality reduction [15, 17, 16], such as pruning, where redundant parameters of a DNN architecture are discarded iteratively or ahead-of-time; architecture simplifications, such as the introduction of simpler operators in the network topology in place of more complex operations, as in the replacement of standard convolution with depthwise separable convolutions in CNNs [20]. The tuning and combination of these approximation techniques can also be carried out automatically using a neural architecture search (NAS), where the best performing architecture in terms of one or more target metrics is found within a constrained search space.

This work focuses on the application of Transformers to low power edge devices in the context of temporal series analysis, specifically regarding Surface Electromyography (sEMG) signals, which are bioelectrical signals related to muscle activity and whose analysis is of high interest in the field of human-machine interaction. In this work, it is empirically shown that Transformers can attain nearly the same accuracy on hand movement classification with respect to previously experimented network architectures with a substantial reduction of complexity in terms of memory footprint and number of multiply-accumulate operations (MACs). Overall the main contributions of this work include:

(a) The design and implementation of a Transformer-based deep neural network that employs $4.9\times$ less parameters and $4.8\times$ less MACs than previous state-of-the-art architectures for sEMG-based hand gesture recognition, with comparable accuracy;

(b) The implementation of a two-stage training protocol comprising a pre-training step and a finetuning step, which is shown to be beneficial for the highly compressed network architectures that are described in this work as accuracies gains in the range of 1-3% are obtained on average on the different architectures;

(c) The study of a Neural Architecture Search (NAS) over a number of inductive biases in the network design towards a hybrid convolutional and attention-based network.

The rest of this thesis follows this structure: Chapter 2 presents a brief summary of a number of DL techniques concerning supervised learning, along with a description of the aforementioned task and the dataset used in this work; Chapter 3 outlines the previous attempts at tackling sEMG-based hand gesture recognition and at bringing Transformers to edge devices; Chapters 4 and 5 describe the main contributions of this thesis, i.e., a new family of ultra-small transformers, and a further optimization knob to reduce the complexity of edge-transformers. Finally, Chapter 6 presents the results that have been achieved by the proposed methods as well as their comparison with state-of-the-art techniques.

Chapter 2

Background

2.1 Supervised Learning

Machine Learning has been successfully applied to many different fields thanks to its ability to automatically learn complex rules by extracting meaningful patterns from past experience. As a result, many different tasks, ranging from face detection in digital images to trajectory planning in autonomous vehicles, which once used to be solved via predefined rules, are now instead tackled with Machine Learning. Many datasets that are used to encode "past experience", from which an ML algorithm can learn, are labelled, meaning that for every sample in the dataset (such as a signal representing a song or an image), a so-called label that encodes the correct answer for the given task is also present. For example, ImageNet [22] can be used for learning image classification since every sample of this dataset, an image, is labelled with a noun that describes what the image contains. This form of learning is called "Supervised Learning" because the learning algorithm does not autonomously learn to correctly perform the given task but it is instead fed with a ground truth.

A more formal way to frame Supervised Learning is to introduce the concepts related to Probably Approximate Correct (PAC) Learnability theory [23]. First of all, samples are supposed to belong to a domain set X and to have labels that belong to a set Y; a dataset contains instances that have been independently and identically sampled from a distribution D over $X \times Y$. Supervised learning consists in finding a hypothesis from a finite set of hypotheses, $h \in H$, that is able to explain the relation between X and Y, i.e. a function $f : X \to Y$. The fitness of any hypothesis h is measured via a loss function $\ell : Y \times Y \to R$ that evaluates by how much the predicted label is wrong with respect to the ground truth: the simplest example of a loss is the 0-1 loss, which is 1 if the predicted label $\hat{y} = h(x)$ is different from the true label y, or 0 if $\hat{y} = y$. Therefore it is introduced the concept of true error of a hypothesis h, which is defined as $err(h) = E_{(x,y)\sim D}\ell(h(x), y)$. Under the realizability assumption that states that $\exists \hat{h} \in H$ s.t. $err(\hat{h}) = 0$, a problem is PAC learnable if, for every $\epsilon, \delta > 0$, running a learning algorithm on a dataset consisting of at least $m(\delta, \epsilon)$ samples that have been i.i.d sampled from D and labelled by f, the learning algorithm returns an hypothesis $\hat{h}$ such that $err(\hat{h}) \leq \epsilon$ with probability greater or equal to $1 - \delta$.

PAC learning is a solid framework since it states that it is possible to learn a task with a true error ϵ that is as low as possible and with a probability of failing δ that is non-zero but that can be reduced as wanted by increasing the sample complexity m, i.e. the number of training samples. However, in reality, the true error cannot be computed since only a subset of samples (x, y) are available and the distribution D is unknown: hence the expected value over all possible samples cannot be computed, and therefore learning is performed by minimizing the sample error $err_s(h) = \frac{1}{n} \sum_i \ell(h(x_i), y_i)$. The procedure of finding the best hypothesis h that has a minimal "empirical" error with respect to all the other hypotheses H is called Empirical Risk Minimization (ERM).

An important remark to do is that ERM cannot be performed on all possible learning algorithms: in fact, it is only proven that all finite hypothesis classes H are PAC-learnable. This means that a number of restrictions about the algorithms to use for learning a specific task must be made, and these restrictions are usually referred to as inductive biases.

Furthermore, it has been proven that a universal learner that works for any problem does not exist. Specifically, for any learner A that outputs an hypothesis $A(S)$, there exists a distribution D and a function $f : X \to Y$ such that, with probability of at least $\delta > 0$ over the generalization of the training set S of size m, the learner outputs an hypothesis $A(S)$ with a true error $err(A(S)) \geq \epsilon$. If ϵ is fixed to be $\frac{1}{2}$, basically this theorem states that any learner cannot be better than a random guess for certain problems, or in other words, that it will fail to learn certain classes of problems.

All of the above motivates the experimentations that have been carried out in this work: since it is impossible to have a universal learner and since a learner for a given task cannot be searched among all possible hypotheses, it is important to always research a suitable class of learners for the studied task and to improve the existing learners, which corresponds to refining the aforementioned H set.

Deep Learning has made this process easier since it is more immediate to design a network topology of a DNN that is able to process training data in its raw form. The main turning point of DL is that it allows "Representation Learning", which means that the learning algorithm not only learns a suitable labelling function for the training data points, but also finds an optimal way of turning the raw input data into data points (also referred to as embeddings). The structure of a DNN is modular and in fact, it is often modelled as an oriented graph: every block of a DNN performs a specific parametrized mathematical operation, and the learning process, i.e. training, results in finding the most suitable parameters of all blocks of a DNN for the given dataset.

This approach is the opposite of classical Machine Learning, where instead a hand-crafted step of feature engineering is involved in order to turn the raw training data into data points with a meaningful representation and on which a learning algorithm with a well-defined set of hypotheses set H is applied. In the context of DL instead, these two steps are merged together; however, it is often unclear what are the inductive biases that make a DL architecture successful at a given task: that is why, after a DL architecture is empirically proven to be competitive at tackling a given task, very often many attempts of applying that very same architecture on other tasks are performed after applying the appropriate set of modifications and adaptations.

2.2 Introduction to DNNs

2.2.1 Multi Layer perceptron

The basic unit of a DNN is the neuron, which is a mathematical operation that is loosely inspired by how neurons in the human brain are thought to be working. An artificial neuron consists of n inputs x_i and one output y, and the value of the output is given by the weighted sum of all the inputs; then a non-linear function is applied to this weighted sum, which is usually referred to as activation function. The explicit formula of a neuron is:

$$y = f(\sum_i x_i \cdot w_i + b) \tag{2.1}$$

where f is the activation function and b is the bias term. Each input contributes to the overall "activation" of a neuron, which is the value of the output; the training process consists in learning the optimal weights w_i and b, which are the parameters of the neuron.

Many neurons that have all same inputs but in general different learnt parameters are logically grouped together in what is called a layer, specifically a fully connected layer. A fully connected layer takes a number of inputs and produces a number of outputs, where each output value is given by the activation value of a different neuron, which depends on all inputs. A fully connected layer is called input layer if its inputs correspond to the features of the input data points or output layer if its output value is used to perform the given task. An important note is that the neurons in the output layer may lack the activation function. A hidden layer is instead any intermediate layer between an input layer and an output layer, and it is called this way because its output values are not inspected by any outside observer but instead, they are fed to another hidden layer or to the output layer. A Multi Layer Perceptron (MLP) consists of one input layer, one output layer and at least one hidden layer.

It is important for the activation function to be non-linear, otherwise stacking multiple fully connected layers without any non-linearity is equivalent to having one single layer. Common activation functions are represented in figure 2.1 and are:

- The Sigmoid, which is an s-shaped function and has an output that ranges in the $(0, 1)$ interval.

  $$f(x) = \frac{e^x}{e^x + 1} \tag{2.2}$$

  This activation function is particularly useful when used in an output layer where it is desired to estimate the probability of a certain event to be true.

- The Hyperbolic Tangent Function, which is again an s-shaped function but its output values range the $(-1, +1)$ interval.

  $$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.3}$$

  One main difference with respect to the sigmoid is that tanh has a higher slope for the values around 0, which is an important property that is useful during training of a DNN.

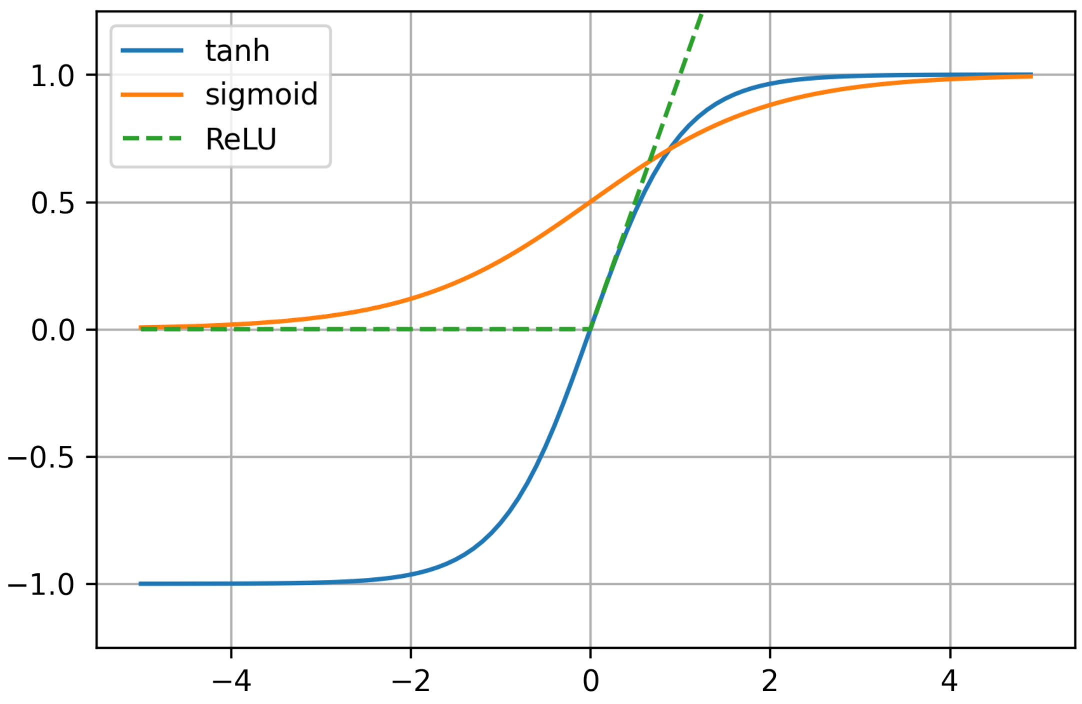

Figure 2.1: Activation functions

- The Rectified Linear Unit (ReLU), which is given by the following function:
$$f(x) = max(0, x) \tag{2.4}$$
First of all, this is a non-saturating function, which means that large input values will not present a gradient tending to 0. However this function is not zero centred, and since it is equal to 0 for negative inputs, it may produce "dead" neurons, which are neurons whose output values are always 0. As a workaround, the Leaky ReLU has been introduced, which is a variant that is non-zero even for negative numbers and can be implemented, for example, as:
$$f(x) = \begin{cases} x & x > 0 \\ Cx & x \leq 0 \end{cases} \tag{2.5}$$
where $C \in R$ is constant, and it is commonly set to a low value, such as 0.01.

Activation functions typically do not involve learnable parameters; however, in some cases, they might do, such as in parametric ReLU, where the slope of the line in the negative quadrant is not fixed ahead of time but is learned during training.

2.2.2 Training a DNN

Training a DNN such as an MLP is carried out by performing empirical risk minimization over training data. Therefore two main elements are needed:

- a minimization method for finding the optimal parameters of the layers of the considered DNN that minimize the empirical risk (or sample error);
- a loss function ℓ that is plugged in the formula that computes the empirical risk.

It can be noted that ERM is actually performed on a limited set of hypotheses: the architecture of the neural network is fixed; furthermore, the values of the parameters are not infinite but are discretized in floating-point numbers. Therefore training is consistent with the PAC Learnability framework, thus a solution with manageable error is guaranteed to be found with a reasonable probability.

However, ERM does not necessarily lead to the optimal solution in the sense of true risk minimization: the hypotheses set H may contain a so-called bad hypothesis $\tilde{h}$ such that $error_s(\tilde{h}) < error_s(h)$ where h is an alternative hypothesis and $error_s$ is the sample error, but $error(\tilde{h}) > error(h)$ where $error$ is the true error. This is because the sample error underestimates the true error due to the limited size of the training set. This situation is commonly referred to as overfitting because the learner has overfitted on the limited training data and has failed somehow to generalize over all cases. The ways to avoid overfitting include gathering more training data (i.e. increasing the sample complexity thus reducing the underestimation of the true error) and limiting the size of the considered neural network (i.e. limiting the set H and the possibility of encountering a bad hypothesis). However also the opposite has to be avoided, which is underfitting: in this case, the learner fails to accurately learn the task because the considered hypotheses set is too constrained. Usually, in DL this happens when the considered DNN has a too low number of layers or the training procedure has not been properly performed.

Loss function

In classification tasks, usually, the last layer of a DNN consists of N_c output neurons where N_c is the total number of classes which the classifier is able to divide every sample into. The last step of the classification process consists of assigning the class with the highest estimated probability. In this case, output neurons lack the activation function, so in order to turn their unscaled values into a probability mass function, the softmax operation is employed:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}} \tag{2.6}$$

where softmax(x_i) is the probability of each different individual output class. One of the most used losses in classification tasks is the Categorical Cross-Entropy Loss, which computes the cross-entropy between the expected probability distribution and the predicted one, and it is given by:

$$\ell(\hat{y}, y) = -\sum_{i}^{N_c} y_c \log(\hat{y}_c) \tag{2.7}$$

where $\hat{y}$ is the estimated probability mass function and $\hat{y}_c$ is the predicted probability for class c. The true probability mass function y is always set to 1 for the correct class and 0 to all the others in order to condition the model to predict the correct label with probability of 1, and to highly penalise it if all predicted probabilities are very close to each other or if the wrong class is predicted with a high probability.

Loss minimization

In supervised learning, training is performed by finding the values of weights w of the considered deep neural network DNN that minimize the loss over all N training samples (x_i, y_i), i.e.

$$\underset{w}{argmin} \frac{1}{N} \sum_i \ell(\text{DNN}(x_i; w), y_i) \tag{2.8}$$

The main issue is that this problem cannot be solved in a closed-form. So instead a heuristic is performed: the weights are initialized to some random values, then iteratively the average loss is computed, the direction towards its minimum is calculated and then the weights of the network are updated by a proportional entity assuming a linear approximation of the loss curve in the near region. This is why all methods employed in DL training involve a learning rate ν, which is the size of the step on this linear plane that approximates the loss curve. Updating the network weights is based on the chain rule of derivatives and on the fact that every module in a DNN is characterized by a relatively simple formula, hence it is easy to compute its outputs and the gradient with respect to its inputs. Therefore, during the so-called forward pass, the computations are run through the network graph from start to end by computing the outputs of each module; then in the backward pass, the network graph is traversed backward from end to start, and at each module, gradients are computed and weights are updated. There are different policies for updating network weights: the simplest one is Stochastic Gradient Descent (SGD), which consists in updating weights at every forward pass of each sample according to the current gradient, i.e.

$$w_{new} = w_{old} - \nu \nabla \ell(w_{old}) \tag{2.9}$$

Usually, in DL, Minibatch Gradient Descent is preferred, which simply consists in performing B forward passes, where B is referred to as minibatch size, and then averaging the loss value and performing one single error backpropagation. Further improvements over vanilla SGD have been proposed in order to increase the speed of convergence: many are based on the concept of momentum that takes into consideration the values of the weight update of the previous iteration, such as the Adaptive Moment Estimation (Adam) optimizer [24].

Finally, it is important to note that this minimization problem is in general non-convex, thus the convergence to the absolute minimum is not guaranteed in general.

For more details on DNN structure, training, loss, and optimizers, refer to the book of Ian Goodfellow and Yoshua Bengio [25].

2.3 Transformer

The Transformer is a Deep Learning architecture introduced for the first time in [8] and is primarily based on the attention mechanism. This architecture has been designed to deal with sequences or in general data that can be divided into tokens whose order is meaningful. As such, transformers have been initially applied in the field of natural language processing before extending their use to other contexts, such as computer vision.

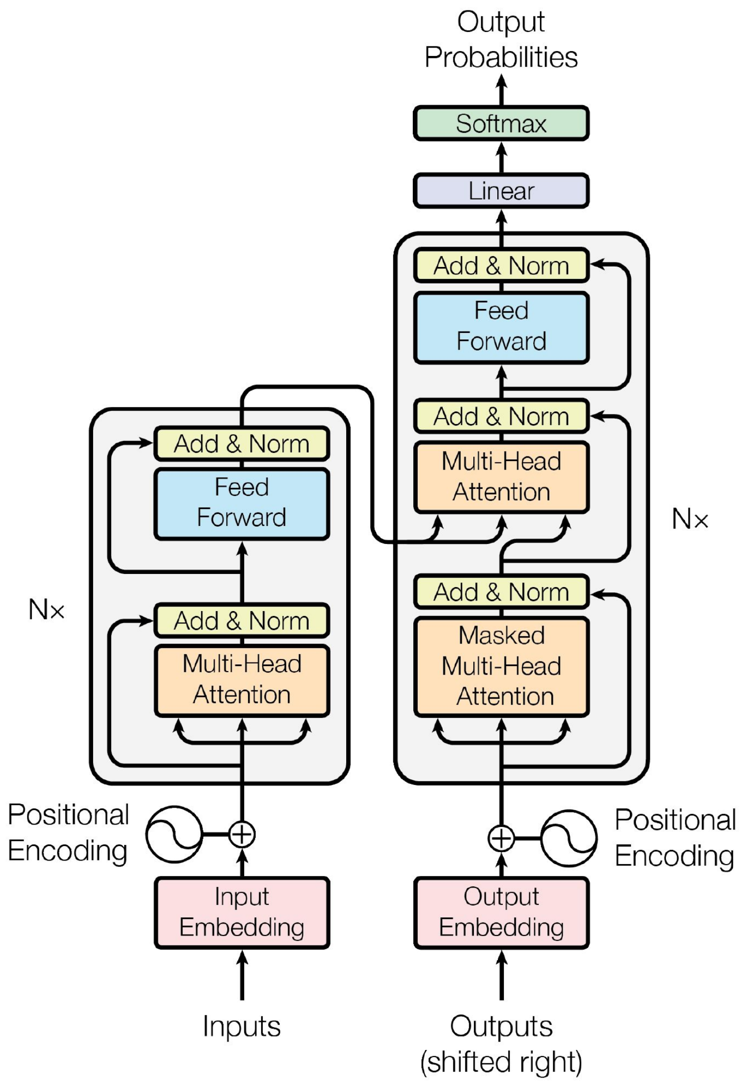

Figure 2.2: The Transformer model architecture [8]

The complete structure of a Transformer architecture shown in figure 2.2 is a consequence of the fact that its first application was in the field of Machine Translation: it is composed of two main parts, an encoder and a decoder. While the former tries to find a latent representation of a source sentence, the latter tries to reconstruct a new equivalent sentence from these embeddings, hence automatic translating from a language to another is performed. However, it is to be noted that, given the great ability of this architecture to extract a powerful representation from its input, in several works [26, 27] only the Encoder element is employed, which is treated as a feature extractor; then the given task, usually classification, is performed with a MLP head that processes the extracted features. This approach will be also followed in this work.

Transformers are seeing high consideration thanks to the many advantages brought by the attention mechanism:

- **Higher parallelization of computations** compared to RNNs. One of the main bottlenecks of RNNs is their internal stateful representation that is reached in a number of sequential operations: in a basic recurrent layer, tokens are usually processed one after the other, meaning that the n-th token cannot be processed until all the $n - 1$ previous tokens have been sequentially processed. In other words, given an input sequence of n tokens, the number of required sequential operations is $O(n)$. Instead, computing attention for each token can be performed independently from the others: this does not mean that the result of the attention computed for a given token is not related to any other input token, as indeed it is. However, the result of the attention does not directly depend on any state variable nor any previous result computed from the other tokens: therefore there is no ordering constraint for performing computations, which means that their parallelization is possible.

- **Longer range dependencies**. The limited receptive field of CNNs and vanishing gradient issues of standard RNNs hinder the ability to model long-range dependencies. While these problems are at least partially mitigated by some other innovations, such as skip connections of LSTM cells in RNNs and a higher dilation value in convolutional layers, it has been found [8] that attention can model dependencies with unprecedented effectiveness in even longer sequences.

- **Transfer Learning**. So far, recurrent layers have not been very successful in the context of transfer learning, meaning that performing finetuning of a pre-trained RNN has always been difficult and has not ever yielded significantly better results than training from scratch. Attention modules and Transformers, in general, have been empirically shown to be easier to finetune, which can be beneficial for more effectively learning many tasks [28].

- **More interpretable models**. Since computing a linear combination of all tokens is at the core of the attention mechanism (as it will be more deeply detailed later), it is easier to inspect and interpret partial results, as it has been shown in [8] where the results of attention appear to be related to the syntactic and semantic structure of the input sentences.

2.3.1 Encoder

The encoder of a Transformer consists of a variable number of L stacked layers, where the output encodings of each layer are fed as input to the next layer. Every layer consists of two main elements:

- a Multihead Self-Attention module (MSA) that produces a new set of encodings by applying the self-attention mechanism on each input encoding;
- an MLP head, which is a feed-forward neural network that further embeds each encoding.

In order to improve the convergence and performance of the overall architecture, skip connections and Layer Normalization (LN) are employed. The overall layer equations are:

$$\begin{aligned} \tilde{X}_{l+1} &= \mathrm{MSA}(\mathrm{LN}(X_l)) + X_l \\ X_{l+1} &= \mathrm{MLP}(\mathrm{LN}(\tilde{X}_{l+1})) + \tilde{X}_{l+1} \end{aligned} \tag{2.10}$$

One of the limits of standard MSA is its permutation equivariance: MSA does not take into account the positional information of its input encodings, which is an obvious limit for structured data and sequences where the order of its tokens carries meaningful information. To overcome this drawback, every input token is enriched with positional information before being fed to the Transformer Encoder.

Furthermore, an additional learnable class token can be prepended to the input sequence: it is treated as any other token of the sequence by the Encoder, and its final encoded value depends on all other tokens thanks to the MSA mechanism that is applied in every layer of the Encoder. However, the class token is particularly useful when the Encoder is used as a feature extractor to perform categorical classification of input sequences since only the last layer encoding corresponding to the class token is fed to the final classification head that is placed after the Encoder. The classification head usually consists of a multilayer perceptron and its input can be the class token as described above, or alternatively the mean of all output encodings.

2.3.2 Decoder

The structure of the decoder is similar to the one of the Encoder since each block consists of a self-attention layer, an MLP head, and an extra attention layer that processes the encodings coming from the Encoder in order to perform meaningful sequence transduction from input encodings to output encodings.

In the architecture proposed in [8], an additional final block which consists of a fully connected layer and a softmax layer is added in order to produce the probability distribution over the vocabulary for every output token.

2.3.3 Positional Embeddings

As described above, input tokens need to be enriched with positional information since no other module of the Transformer would be able to directly extract this information. Positional embeddings usually consist in mapping each position of the input sequence to a different vector that in turn gets summed to the vector that represents the respective input token. There are different ways for computing positional information, including sinusoidal positional encoding and learnable positional embeddings.

In [8], fixed non-trainable sinusoidal positional encodings are considered. Instead of concatenating positional features to the features of the token, positional information is actually summed: in this way, the embedding dimension of each token is retained to its initial dimension C. The values for each position are computed as follows:

$$\begin{aligned} PE_{pos,2i} &= sin(pos/10000^{2i/C}) \\ PE_{pos,2i+1} &= cos(pos/10000^{2i/C}) \end{aligned} \tag{2.11}$$

where C is the input dimension of each token, i is the index of considered C-th embedding dimension of the token and *pos* is the position of the token in the input sequence. This positional encoding is represented in figure 2.3, where it is shown the case of a sequence of 30 tokens whose embedding dimension is $C = 64$.

On the other hand, [26, 27] employ a learnable positional embedding that consists of $(C \times N)$ trainable parameters that are summed to the input tokens. This solution does not seem to really provide an edge over the other solution, and the accuracy gain may be negligible as stated in [8]. One main drawback is the fact that the input sequence at inference time must be

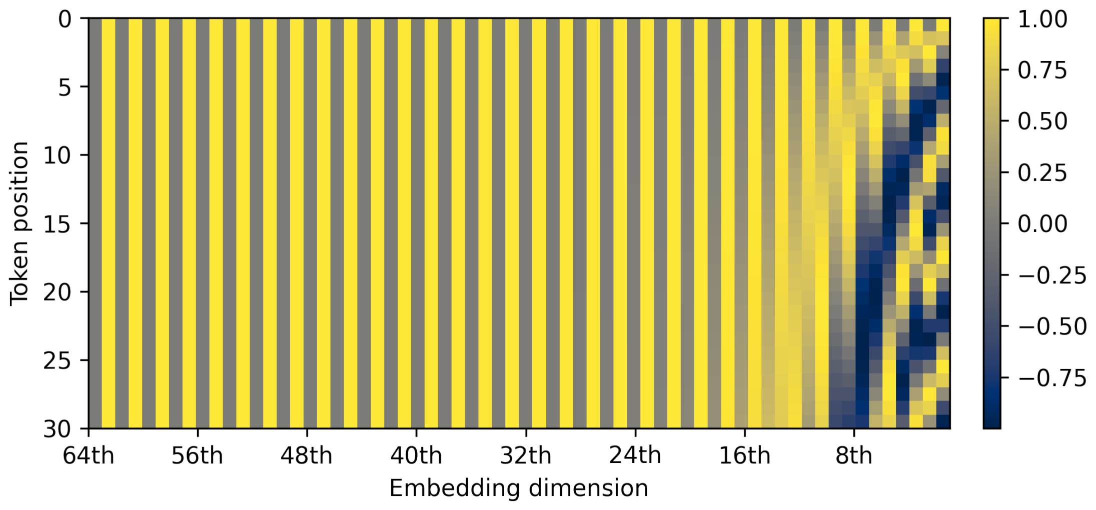

Figure 2.3: Illustration of the positional encodings: on the x-axis, the positional encoding values for tokens of embedding dimension $C = 64$; along the y-axis, each token in a sequence $N = 30$ long is shown. Hence, in the top row, the value that will be added to the first token is shown; while in the bottom row, the value to be added to the last token is shown. Note how these positional values resemble a sort of "binary notation" of 1s and -1s, but of course it is made of continuous values; also note that only the least significant part (the values on the right) varies while the most significant part is always the same, since the length of the sequence is lower than the embedding dimension of each token, i.e. $N < C$

at most N-long. Variable-length sequences may be a key characteristic in certain fields, such as NLP, but it is not in several other applications, such as image recognition, where input images are all of a predefined size, thus the input sequence length is supposed to be fixed. Therefore using this trainable positional embedding may provide a slight accuracy gain with respect to the sinusoidal positional embedding.

2.3.4 Multihead Self Attention (MSA)

The Attention mechanism is a technique inspired by cognitive attention and it has been first introduced in [29] to enhance state-of-the-art neural networks applied to NLP. Generally speaking, the aim of attention is to learn what are the most important parts of the input data for the given target task: by weighting each part of the input according to their learned relative importance, the network is supposed to "pay attention" to smaller but more important parts of the input, thus the performances achieved on the target task should be enhanced.

At its core, Attention consists in computing a linear combination of values $c = \sum_i \alpha_i x_i$ for each input value, where $\sum_i \alpha_i = 1$ and x_i are embedded input values. In the first formulation, each weight of this linear combination, which is also usually referred to as attention score, is calculated by combining the value of a RNN state variable and the encoded input token.

Self-attention (SA) has been introduced in [30] and it removes any recurrency of previously introduced attention mechanism by computing the attention scores as the result of a similarity measure between the embedded input tokens themselves.

The input sequence can be defined as a matrix $X \in R^{N \times C}$: N is the number of tokens of the input sequence and each token is represented as a vector of embedding dimension C. The aim of self-attention is to produce a new set of encoded tokens $Z \in R^{N \times D}$ in a way such that each token embeds its interactions with itself and the other tokens of the sequences. While the embedding size D of every encoded token can be different from its initial size C, it is often set to the same value. To produce this new set of encoded tokens, three separate learnable projection matrices are taken into consideration, and each of those produces a different projection of the input: queries Q, keys K and values V, which can be expressed as $Q = XW_q; K = XW_k; V = XW_v$. Then, the attention matrix is computed, which is given by the scaled dot product of Q and K, where the scaling factor is $\sqrt{D}$. This matrix represents the attention scores for any given token with respect to all the tokens of the sequence. A scaling factor is present in order to stabilize the gradient flow during the backward pass. Furthermore, in order to turn these scores into a discrete probability distribution, the softmax operation is applied. Finally, self-attention is obtained by computing the weighted sum over V for every token and using the previously computed scores as weights. Hence, self-attention is given by

$$SA(X) = Z = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right) V \tag{2.12}$$

This means that every new encoded token Z_i, where $i \in [1, \dots, N]$, is a linear combination of the projected tokens of the sequence, represented by V.

Multihead self-attention consists in applying self-attention multiple times with different learned projection matrices. Every instance of self-attention is also called attention head and each attention head is supposed to learn different kinds of relationships and dependencies between input tokens. Therefore, every output encoding results from the concatenation of the corresponding encodings of the different attention heads. Given that each output encoding would have an embedding dimension $H \cdot D$, where H is the number of heads, a further projection matrix $W_{out} \in R^{(H \cdot D) \times C}$ is applied in order to restore each token to their original embedding size C:

$$MSA(X) = [SA_1(X), \ldots, SA_H(X)]W_{out} \tag{2.13}$$

2.3.5 MLP module

As its name suggests, the MLP module is a feed-forward module that consists of two fully connected layers with a non-linear activation function between the two and optional dropout at the output of each fully connected layer.

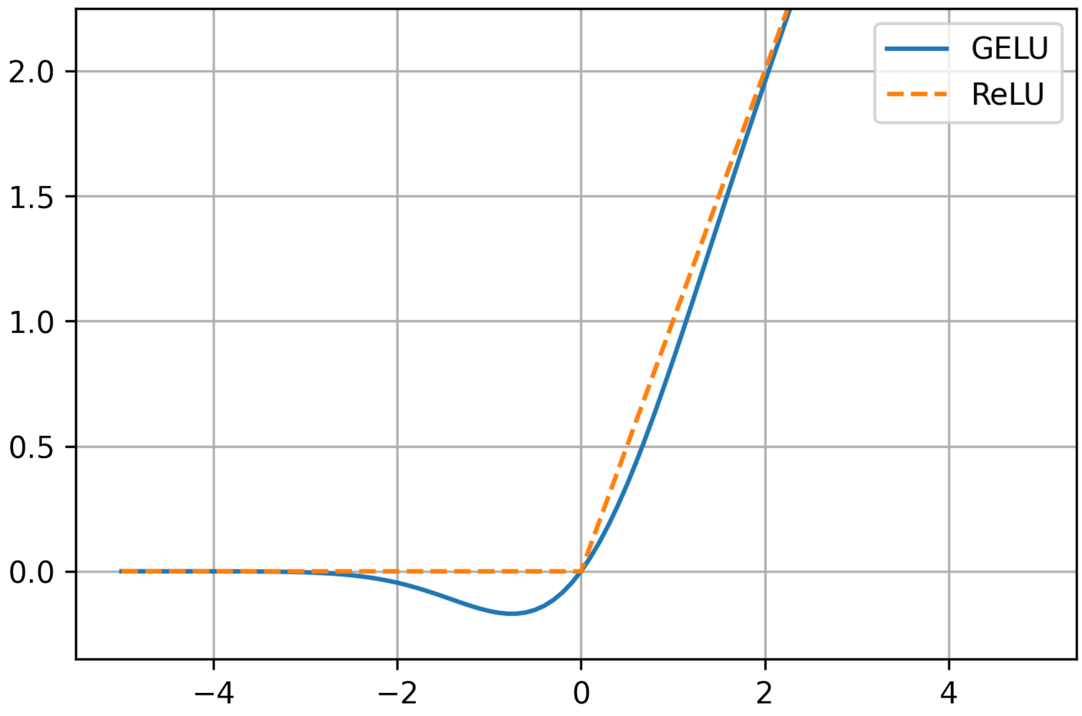

Figure 2.4: GELU activation function compared to ReLU function

The activation function in many applications of Transformers is the

Gaussian Error Linear Unit (GELU) function, whose shape is roughly similar to the ReLU function with the notable differences of being differentiable in 0 and having a non-zero value for a small range of negative inputs as shown in Figure 2.4. The formal definition of GELU is:

$$GELU(x) = x \cdot \Phi(x) = \frac{1}{2}x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) \tag{2.14}$$

where $\Phi(x)$ is the Cumulative Distribution Function for the gaussian distribution and $\text{erf} = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$

The two fully connected layers can be represented as two projection matrices $W_1 \in R^{C \times H}$ and $W_2 \in R^{H \times C}$, where C is the token input size and H is the hidden embedding size, which is usually set to a multiple of the input size, such as $1\times$, $2\times$ or $4\times$.

2.3.6 Patch Embedding in ViT

In [26], a pure Transformer architecture called ViT is applied to image recognition. The main obstacle is that, at first glance, an image is a single entity, while instead self-attention in transformers deals with sequences of tokens. To overcome this issue, the authors propose to divide the input 2D image into a 1D sequence of non-overlapping patches. Specifically, the input image $x \in R^{H \times W \times K}$ (H is the image height, W is the image width and K is the number of channels, which is 3 for an image in the RGB colour space) is reshaped to $x \in R^{N \times (P^2 \cdot K)}$, where N is the number of patches, given by $N = HW/P^2$, and P is the height and width of each patch. In this formulation, patches are square and the image height and width are supposed to be a multiple of the patch size, but it is of course possible to generalize to the case of non-square patches or to input 1D signals. Finally, a linear embedding can be applied to each input flattened patch, optionally with the intent of projecting each patch to a lower-dimensional space. Hence the resulting input of the Transformer can be described as $x \in R^{N \times C}$, where C is the size of each linearly embedded token, which is set to $P^2 \cdot K$ in the original ViT.

2.3.7 Positional self-attention

Self-attention does not explicitly take into account positional information during the computation of self-attention scores; this is why, one of the initial layers of the transformer encoder is responsible for adding positional information to each token. However, this information might be lost as the transformer grows in depth.

To overcome this issue, positional self-attention has been introduced in [31], which consists in computing self-attention considering also the relative position of each patch:

$$PSA(X) = \text{softmax}\left(QK^T + v_{pos}^T r_{ij}\right) V \tag{2.15}$$

where $v_{pos} \in R^{D_{pos}}$ is a learnable embedding and $r_{ij} \in R^{D_{pos}}$ is a relative positional encodings that encodes relative distances between token i and token j.

As proved in [32], if the relative positional encoding r_{ij} is a vector with size $D_{pos} \geq 2$ and is in the form $r_{ij} = [\delta^2, \delta, 0, \ldots, 0]$, where δ is the relative distance between tokens i and j, then an attention layer with H heads can express a pure convolutional filter if the following conditions hold:

$$\begin{aligned} v_{pos}^h = -\alpha^h[1, -2\Delta^h, 0, \ldots, 0] \\ W_q = W_k = 0 \\ W_v = I \end{aligned} \tag{2.16}$$

In fact, $\Delta^h \in R$ is the center of the attention for head h, and represents the position at which the attention-head h pays most attention to. Note that, since W_q and W_k are set to 0, the QK^T term in 2.15 disappears; also, since $W_v = I$ is the identity matrix, the pure linear combination of attention scores is considered; furthermore, if α is very high, the result of the softmax will be in the form $[0, \ldots, 1, 0, \ldots, 0]$, where the position of the 1 is expressed by the center of attention of the head Δ^h. This means that, if for example a given multi-head PSA layer is characterized by 3 heads, and the learned centers of attention are $\Delta^1 = -1, \Delta^1 = 0, \Delta^1 = 1$, then the concatenation of encoded token in the multi-head self attention for position i of the input sequence will be actually equal to the tokens at relative position -1, 0 (i.e. the token itself) and +1:

$$MSA(X) = \text{concat}(SA_H(X))W_{out} = [X_{-1}, X_0, X_{+1}]W_{out}$$

where $W_{out} \in R^{3D \times C}$. Since MSA is performed for all positions in the sequence, the output sequence will be given by

$$[[0, X_0, X_{+1}]W_{out}, [X_0, X_{1,X_2}]W_{out}, \ldots, [X_{N-1}, X_N, 0]W_{out}]$$

whose output is equivalent to a 0-padded 1D convolution with kernel size of 3, number of output channels equal to C and a stride of 1.

On the other hand, as the factor α of the learned positional embedding v_{pos} decreases for every head in the PSA, the center of attention of each attention head will be less narrow: this means that each attention head will take into account a number of neighbour tokens for computing the attention scores, and the resulting behaviour will be similar to a trucated self-attention, where SA is computed within a fixed sized window as described in [33]. Finally, if QK^T is non-zero and its magnitude is greater than the positional term $v_{pos}^T r_{ij}$, the resulting behaviour will be very close to standard SA.

2.3.8 Gated Positional Self Attention (GPSA) in ConViT

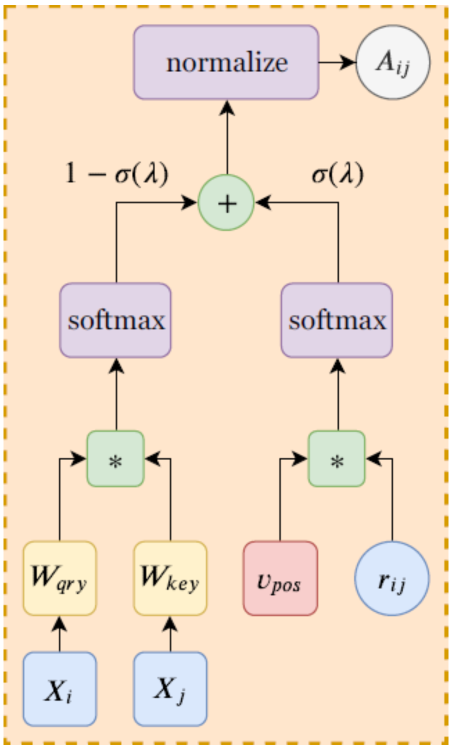

Figure 2.5: Scheme of the Gated Positional Self Attention (GPSA) [27]

One of the shortcomings of PSA, as formulated in 2.15, is that, if the two addenda have different magnitude, the softmax ignores the smallest of the two. This is not a secondary problem since, if a "convolutional" weight initialization is performed, i.e. the locality strength α is set to a high value in order to focus the centre of attention of each head on a low number of neighbour tokens, then the attention mechanism tends to ignore the content information and learns to pay attention to the relative positional information only. To avoid this problem, [27] introduces the Gated Positional Self Attention (GPSA), which is represented in figure 2.5 and consists in:

- summing the content information and positional information only after each one has been normalized with softmax;
- introducing a relative importance parameter, λ, which is also called gating parameters and controls which of the two information terms is most relevant.

The gated positional self-attention mechanism for head h is therefore defined as:

$$\begin{aligned} \mathrm{A}_{ij}^{h} &= (1-\sigma(\lambda)) \cdot \mathrm{softmax}(Q_i^h K_j^{hT}) + \sigma(\lambda) \cdot \mathrm{softmax}(v_{pos}^{hT} r_{ij}) \\ \mathrm{GPSA}^h &= \mathrm{normalize}(A^h) V^h \end{aligned} \tag{2.17}$$

where $\mathrm{normalize}(A^h)_{ij} = A_{ij}/\sum_k A_{ik}$ and σ is the sigmoid function. Note that, for every attention head, a different λ is learned, and if its value is high, i.e. $\sigma(\lambda) \approx 1$, it means that GPSA focuses only on the positional information. Additionally, similarly to standard multi-head self-attention, the multi-head GPSA results from the concatenation of all the outputs from the different heads, which is then linearly embedded with a linear projection W_{out}, b_{out}:

$$MGPSA(X) = \mathrm{concat}[GPSA_h(X)]W_{out} + b_{out} \tag{2.18}$$

Finally, if $\sigma(\lambda) \approx 1$ holds true for all heads of the MGPSA layer and if all learned v_{pos} are in the same form as 2.16, then the layer behaves as a convolutional layer with a receptive field of h, where h is the number of heads of the layer.

2.4 Integer quantization

Several methods have been introduced in order to perform quantization, which is a precision reduction technique aimed at relying on only integer arithmetic and reducing the bit-width of the involved operands. In fact,

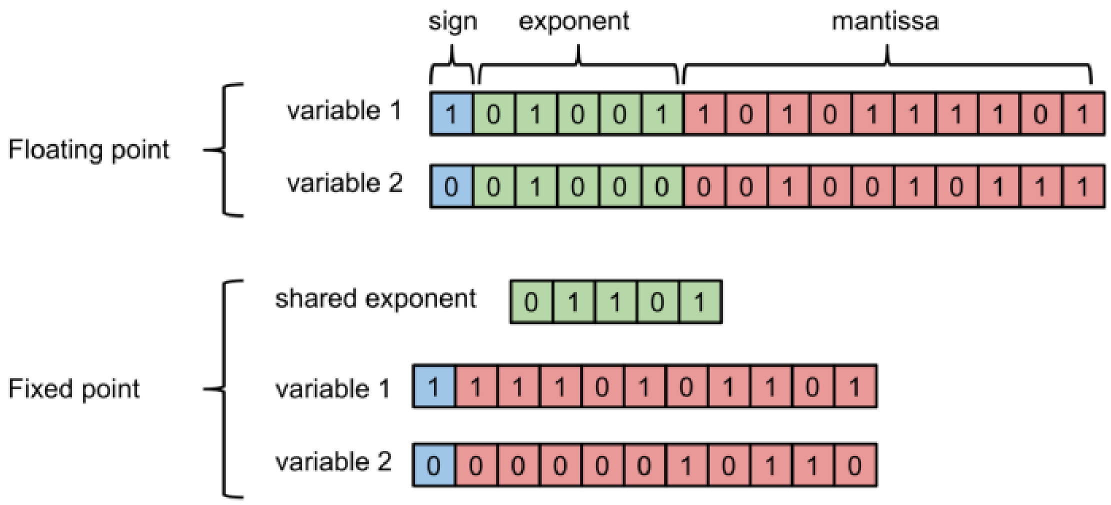

Figure 2.6: Fixed point quantization scheme [34]

many MCUs in embedded devices lack a floating-point unit (FPU), hence the computation of floating-point arithmetic must be performed in software by combining several integer-only arithmetical operations computed with the arithmetic logic unit (ALU), with a consequently higher computation burden and a detrimental effect on latency. Furthermore, operations that involve FPUs are usually more energy intensive than those involving ALUs: an FPU is usually less efficient than an ALU in terms of energy consumption and processing speed due to the intrisic higher complexity of the former; furthermore, an higher bit-width for the involved operands is more energy demanding also due to a higher energy consumption in memory transfers. Hence quantization is advisable for extending battery life and reducing latency also for those MCUs that are equipped with a FPU.

As shown in figure 2.6, floating-point numbers and fixed-point numbers are a discretization of real numbers in the form $(-1)^{\text{sign}} \times \text{base}^{\text{exponent}} \times \text{mantissa}$. In a floating-point notation, both the mantissa and the exponent of a predefined base are encoded in the number, along with its sign: this allows the representation of a large interval of values, and furthermore, it allows a high precision for representing numbers around the 0, which may also be beneficial for neural networks; on the other hand, in order to efficiently compute operations involving floating-point numbers, a floating-point unit is needed. On the other hand, in fixed-point notation, not only the base but also the exponent are predetermined and only the mantissa and sign are encoded in the number: this means that the same circuitry used for performing integer computations can be used to perform operations with real numbers at the cost of a lowered precision.

With the uniform symmetric quantization scheme, a real floating-point number is mapped to an integer value $q \in [-2^{b-1}, 2^{b-1} - 1]$, where b is the precision of the quantization, which is usually 8. This operation converts any real floating-point number belonging to a certain interval $[-\alpha, \alpha]$ to a fixed point notation. In particular, the definition (as stated in [35]), is

$$q = \text{Round}(\frac{clip(x, -\alpha, \alpha)}{\Delta}) \tag{2.19}$$

where $\Delta = \alpha/(2^{b-1} - 1)$

The reverse, i.e. the dequantization operation, consists in computing the following

$$\tilde{x} = q \cdot \Delta \tag{2.20}$$

This representation is particularly convenient for quantizing the weights of a DNN since

- the range of values $[-\alpha, \alpha]$ is known a priori;
- the 0-value has an exact representation, which represents an important value in DNN.

However, more often than not, activations are not symmetric around the origin, so an extra z offset is computed in what is referred to as asymmetric quantization. Furthermore, the activation ranges depend on the inputs and are not predetermined: in order to allow static quantization where both Δ and z are predetermined, a representative subset of samples from the dataset are used to compute the activation values and capture their distribution, and consequently to compute Δ and z

2.4.1 Quantization-aware training

Quantization is performed to reduce the representation size of each weight from 32 bits to 8 bits (thus the size of the model is reduced by $4\times$), and to allow 8-bit integer arithmetic on a devices that lacks a FPU or if its use would incur a higher energy consumption.

This operation can be performed after training has completed, by computing the Δ for each layer in order to quantize its weights and the Δ

and z to quantize its activation as described in section 2.4. This technique is usually referred to as Post Training Quantization, since quantization is performed as a discrete step after training. This procedure usually results in non negligible accuracies drops since the network weights are suboptimally updated to their quantized versions.

An alternative operation that tries to reduce accuracy drops consists in taking into account weight and activation quantization already during training, in a procedure that is often referred to as Quantization-aware Training (QAT).

Quantization-aware training consists in adding the discretization step-wise function already during training; however the main issue is that this function makes gradient flow during back-propagation very difficult. Hence a Straight-Through Estimator is introduced [36]: it consists in adopting the quantization function only during the forward pass and replacing it with the identity function during the backward pass, thus allowing the gradient to "flow" as-is from output to input. In this way, network training is still performed in 32 bit floating point precision, but operations in the forward pass are computed as if quantization was in place, hence weights are somehow optimized taking into consideration that their final values will be discretized with a more coarse discretization step, which usually leads to better final accuracy then plain PTQ.

2.5 Surface electromyography

Surface electromyography (sEMG) is a non-invasive technique that consists in acquiring bioelectrical signals related to muscle activity: an sEMG sensor consists of a pair of electrodes that can measure ionic currents on the surface of the skin caused by the activity of the underlying muscle fibres. The characterization of movements of arm muscles associated with hand gestures is of high interest in the field of human-machine interaction, for instance for the development of myoelectric-controlled prostheses, which are artificial limbs that mimic human anatomy and are aimed at being controlled by amputee subjects via the aforementioned bioelectrical signals generated naturally by their own muscles.

Usually, in this research context, signals are recorded from multiple sensors that are placed on the forearms of amputee and non-amputee subjects who are asked to perform a given set of gestures as shown in figure 2.7. Sensors

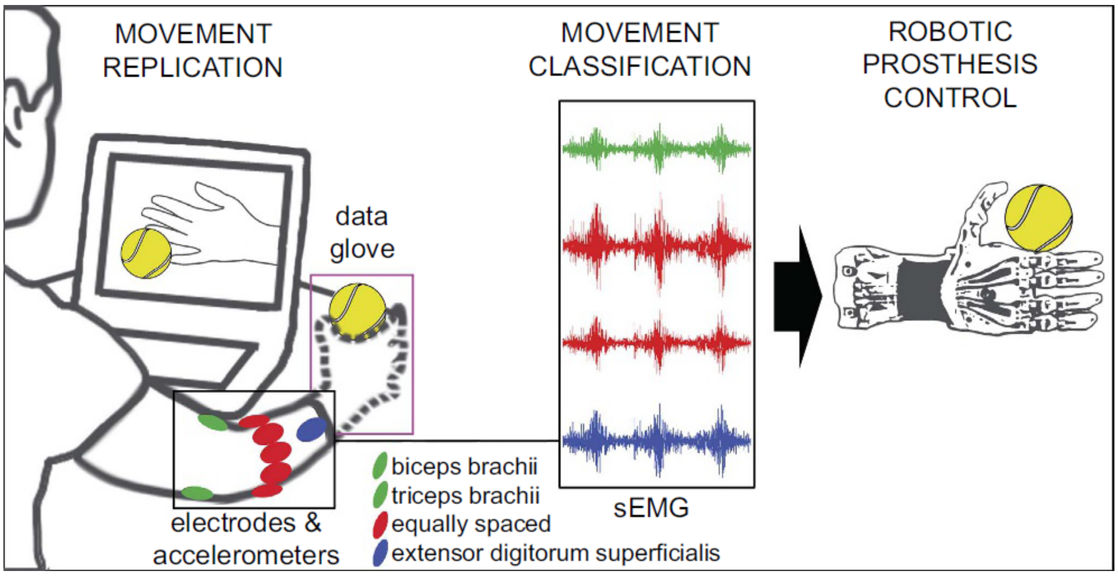

Figure 2.7: The illustrated acquisition protocol of sEMG signals in the context of hand gestures [37]

can employ active or passive electrodes: in active electrodes, the signal is amplified very close to its source thanks to an appropriate amplifier placed on the sensor itself, which is therefore self-contained, and it typically wirelessly transmits the recorded signal to an external digital data acquisition system; in passive electrodes, bioelectrical signal amplification is instead performed by an external sEMG amplifier, which is connected to each sensor via appropriate wiring, and, as a result, individual sensors are cheaper than their active counterparts but are more susceptible to noise interferences. Additionally, the interface between skin and electrodes can be dry or wet: the moisturization of the skin with the application of gel is a way of reducing the impedance of the skin-electrode interface, which is advisable especially when passive electrodes are employed; instead, active electrodes can deal with higher interface impedance, hence they are more suitable for prolonged use since skin preparation and periodic gel reapplications are not required.

2.5.1 Non-Invasive Adaptive Hand Prosthetics Dataset

One of the biggest challenges for hand movement classification via sEMG signals is the great variability that occurs even when acquisitions are performed in carefully controlled environments. Sources of variations include not only the slight differences in shape and size of body parts of different subjects, but also several factors of temporal nature, such as the consistency by which the involved patients repeat the specified movements across the same and the different days of experimentations, and shifts of the placement of sensors across different sessions. That is why multi-day multi-session datasets are currently the most favoured benchmarking tool in the context of hand movement recognition via sEMG signals.

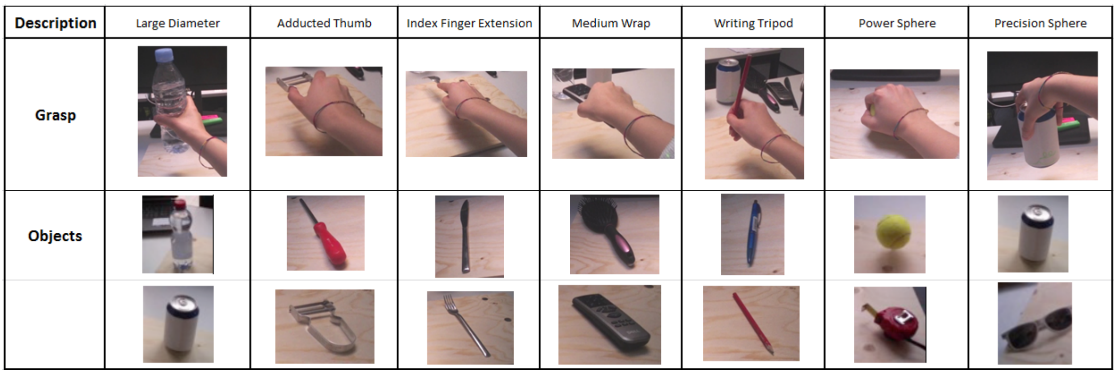

Figure 2.8: The list of grasps performed in NinaPro DB6 along with the corresponding objects [38]

In this work, the NinaPro DB6 dataset [38] is used for evaluating the different analysis approaches. It consists of 7 possible hand grasps that are repeated 12 times and are interleaved with the resting position, totalling for 8 possible hand positions. The acquisitions were performed twice a day for 5 days, resulting in a total of 10 sessions, and they involved 10 intact subjects (3 females, 7 males, average age 27 $\pm$ 6 years). A total of 14 active sensors were arranged on dry skin in two adjacent rows just below the radio-humeral joint of the forearm of the involved subject, as shown in Figure 2.9, resulting in a 14-channel digital signal with a sampling frequency of 2 kHz derived from the amplified and synchronized sEMG signals.

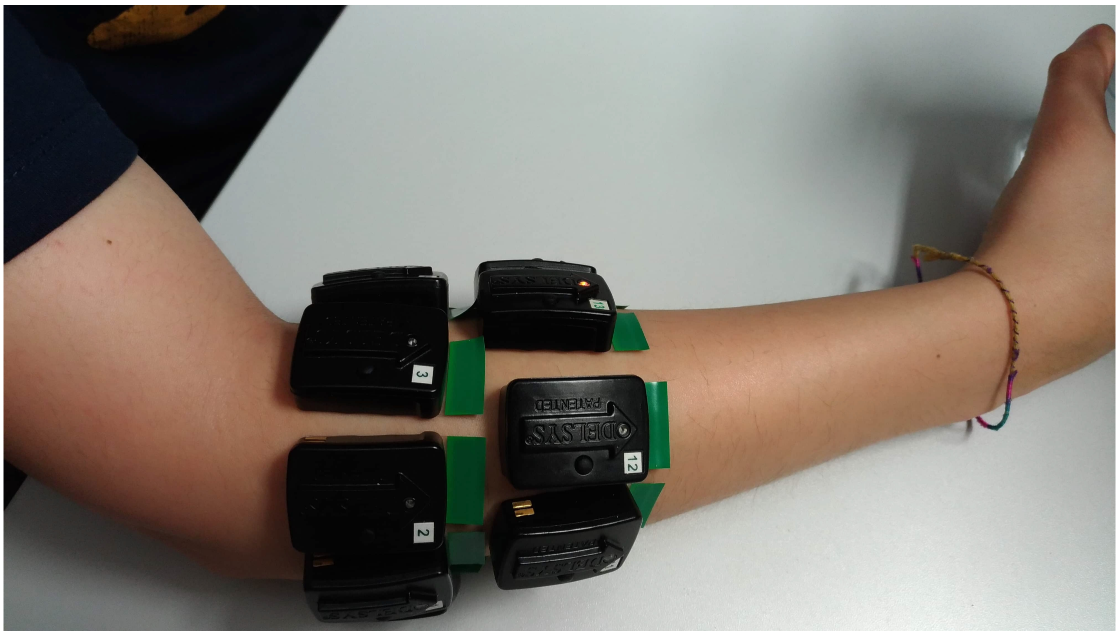

Figure 2.9: Example of final sensor placement on a subject involved in the acquisition of NinaPro DB6 [38]

Chapter 3

Related works

3.1 Surface Electromyography

Hand gesture recognition via sEMG signal analysis has been gaining traction in the last few years thanks to its promised improvements in Human-Computer Interaction. Most techniques proposed both in academia and in commercial applications for performing live gesture recognition are based on a three step pipeline: i) the multi-channel analog signal is acquired by a number of electrodes placed on the subject's arm; ii) the continuous signal is divided into sliding windows on which further data preprocessing and feature extraction may be carried out; iii) classification is performed on the extracted samples.

Initial approaches were based on the application of classical machine learning techniques and manual feature engineering. For instance, in the context of movement recognition among 4 possible hand gestures, [39] demonstrates the application of shallow artificial neural networks in the classification of time-windows whose raw digital samples were summarised in 5 time-domain hand-crafted features that leverage domain-specific knowledge, namely Mean Absolute Value (MAV), Mean Absolute Value Slope (MAVS), number of Slope Sign Changes (SSC), number of zero crossings (ZC), and Waveform Length (WL). Regarding the recognition of up to 50 different gestures that were collected in the Non-Invasive Adaptive hand Prosthetics Databases 1, 2, and 3 (NinaPro DB1, DB2, and DB3), [37] compares the promising classification accuracy achieved by different learners, including K-Nearest Neighbours, Support Vector Machine, Random Forests and Linear Discriminant Analysis. Furthermore, for each classification algorithm, five different data preprocessing techniques are compared against each other: the 5 time-domain features described above, frequency-domain features extracted via marginal Discrete Wavelet Transform (mDWT) with a db7 wavelet and three levels, Root-Mean-Square (RMS), Histogram (HIST) binning with 20 bins along a 3σ threshold, and a normalized combination of all of the above features.

One main drawback of the aforementioned works is the limitation to single-session acquisitions in the considered datasets: the collection of hand movements and their repetitions were performed all together in a single session for any involved subject. As it is argued in [40, 41, 42], this acquisition setup is a big limitation for real-world applications because several factors of variability are present in multi-session acquisitions, not only due to user adaptation and consequently different motion artifacts and variable arm postures, but also simply due to the re-positioning of each sensors at every use. This is why these works propose a number of solutions for increasing the robustness of the previously presented methods. These improvements include the extension of the training datasets or the modification of the acquisition setup such as increasing the electrodes count. However, one of the main and most significant innovations aimed at making hand-gesture recognition more resilient against temporal variability is multi-session training, which has been made possible by the release of several multi-session sEMG datasets, including the Non-Invasive Adaptive hand Prosthetics Database 6 (NinaPro DB6) proposed in [38] and used in this thesis as a benchmark. The authors who made this dataset available also attempted classification with a Random Forests classifier on input time-windows from which Mean Absolute Value (MAV) and Waveform Length (WL) were computed, where the underwhelming classification accuracies highlight the limits posed by the techniques attempted that far.

In fact, all the aforementioned works concerning the application of classical machine learning techniques show one of their main limitations, which is the strong reliance on the effectiveness of the employed feature engineering step that inevitably limits the generalization capability of the proposed methods. Therefore, more recent works have investigated Deep Learning for sEMG-based gesture recognition.

The first DL architecture for performing sEMG analysis was proposed in [44], where a Convolutional Neural Network (CNN) was applied on NinaPro DB1 and the performances achieved by the proposed CNN outperformed SVM. Following attempts proved Temporal Convolutional Networks (TCNs)

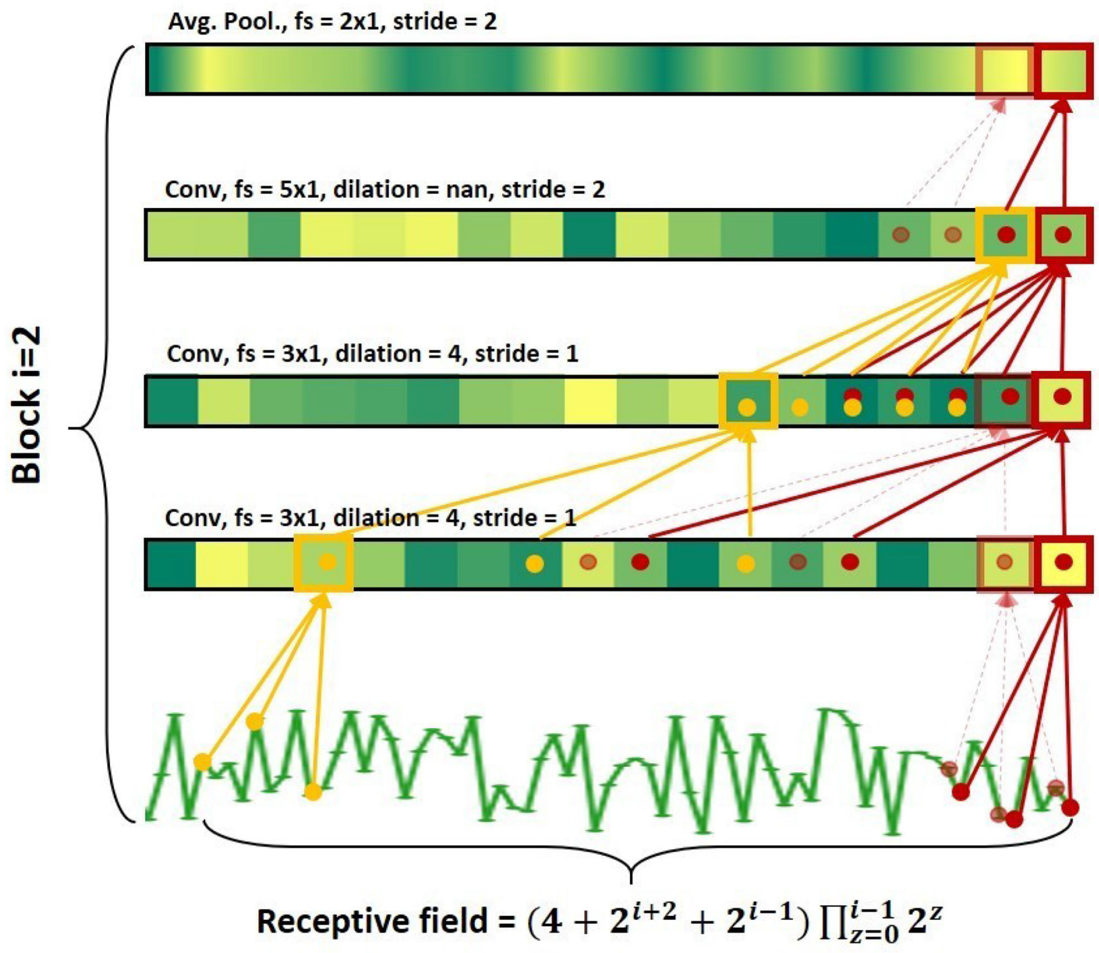
Avg. Pool., fs = 2x1, stride = 2
Conv, fs = 5x1, dilation = nan, stride = 2
Conv, fs = 3x1, dilation = 4, stride = 1
Conv, fs = 3x1, dilation = 4, stride = 1
Block i=2
Receptive field = $(4 + 2^{i+2} + 2^{i-1}) \prod_{z=0}^{i-1} 2^z$

Figure 3.1: Structure and functioning of a convolutional block that involves dilation in TEMPONet [43]

to be more effective than recurrent neural networks that employ long-short term memory (LSTM) cells. TCNs are based on convolutional layers like CNNs, however they introduce the dilation technique in order to increase the otherwise narrow receptive field of a standard convolution without increasing the number of parameters. The resulting operation is called d-dilated convolution, where d is a hyper-parameter that controls the number of steps between each pixel (or time-sample) in the input feature map. The authors of [43] propose TEMPONet, which is a TCN aimed at performing real-time classification on embedded devices. To obtain these results, the authors propose to train on 300ms sliding windows with a 15ms overlap in order to keep the input-to-output latency under the human perception threshold; furthermore, the feature extraction task is completely left to the neural network itself by stacking multitple convolutional blocks whose illustration is shown in Figure 3.1, thus eliminating the computation burden that would be required by the execution of any additional preprocessing; finally, the proposed network architecture is highly compressed and approximate computing techniques, such as weight and activation quantization, are employed, thus making TEMPONet deployable to constrained devices with low memory availability and lack of floating point units.

3.2 Transformers

In [8], it is presented a novel network architecture which is entirely based on self-attention and does not make use of convolutional layers nor LSTM cells: the Transformer. This model has been proposed in order to improve existing sequence transduction models that are mainly based on recurrent neural networks. Given this premise, transformers have been first employed for automated natural language translation: on the English-to-German translation task, this architecture not only outperformed all previous models with respect to all target metrics, including the bilingual evaluation understudy (BLEU) metric, but it also significantly reduced the training cost compared to its competitive alternatives. Given the powerful feature extraction capability of Transformers, their use has been extended also as feature extractors for performing classification by employing only a portion of their architecture, the Encoder. In [26], the Vision Transformer is introduced, which is an architecture that turns an image into a sequence of patches for performing image classification.

All of these architectures are characterized by a relatively expensive training and inference cost compared to the methods employed in the context of sEMG-based movement recognition, mainly due to the high number of parameters and MAC operations involved. For instance, the original transformer architecture was trained continuously for 3.5 days on eight GPUs: the base variant with subpar performances consists of 65M parameters, while the variant with SoTA performance is characterized by 213M parameters. The high computing requirements for these first use cases of this architecture are not only due to the complex nature of the tasks at stake, but also because their deployment is supposed to take place on the cloud, where less constrained hardware is employed, be it dedicated GPUs or specifically designed ASICs for inference in datacenter premises, such as Tensor Processing Units (TPUs) [14].

Given the high interest of applying this architecture in several hardware-constrained contexts, there exist several works in literature concerning tuning of Transformers computational complexity, both with respect to network topology, as illustrated in [45], and to the approximate computing approaches, as illustrated in [35]. The combination of these approaches allow transformers to be deployed on general purpose constrained MCUs and hardware aware static or dynamic optimization methods have been proposed in different works [46, 47] for distinctively leveraging the high information redundancy of the Transformer architecture. In particular, in these works one big SuperTransformer is trained and then SubTransformers with weight sharing are optimally constructed by combining only the minimum number of blocks of the SuperTransformer, thus highly reducing the final computation complexity of the best architectures and allowing them to be deployed on embedded devices for edge inference.

Starting from the aforementioned works, this thesis focuses on the application of Transformers on embedded MCUs for tackling hand movement recognition via sEMG signals.

Chapter 4

Transformers for sEMG-based Gesture Recognition

4.1 Objective

The main objective is to design a Transformer-based network architecture that can be deployed to an edge device to perform sEMG-based hand movement recognition. The aim is to perform real-time inference, meaning that, once the model has been deployed on the target microcontroller, the signal is acquired, pre-processed and then classified immediately and continuously. This is done, for instance, to allow the creation of a myoelectric-controlled hand prostheses that is able to perform the intended gesture by reacting with a predictable and low response time to the acquired sEMG signal. To effectively perform this operation, a number of optimizations and design choices have to be made. First, a proper preprocessing of the input signal must be applied in order to frame the classification task in a real-time scenario: for instance, if a classification method takes into account the activity of the previous 2 seconds, then that technique is not suitable for this scenario since the latency from input to output would not be acceptable. Furthermore, a proper architectural design of the involved Deep Neural Network has to be considered in order to allow its deployment to constrained and low-resource hardware. These issues are tackled in the following sections of this chapter.

4.2 Signal preprocessing

The gestures and their repetitions are performed by the subject consecutively one after the other and they are stored as a continuous 14-channel signal into a file. Every subject is in front of a screen that shows what movement should be performed and when. However, since the involved subject may not react promptly to the indications on the screen, the labelling of each gesture is manually re-synchronized in the given dataset. The labelling consists of two extra channels with the same sampling rate of the raw signal in order to assign a label to every sample, and the first channel represents what is the involved movement, while the second channel represents the repetition number.

The raw signal is divided into fixed-length sliding windows: once the DL model is deployed on the embedded device, classification is performed in an X amount of time considering the samples collected in the preceding Y-long time interval. So, while labelling is done on a per-sample basis, the task consists instead in classifying each overlapping window.

In order to perform a "real-time" or "live" classification of signal windows, the sliding time of each window must be greater than or equal to the time required to perform the classification by the embedded device, otherwise the backlog of windows yet to be processed would grow over time. In this work, the same parameters of [43] are employed, meaning that the input signal is divided into windows of 300ms and with a slide of 15ms. The windows size is a compromise between the amount of input that can be processed in order to give a precise classification and the desired delay between user input and its reflection on the output: given that 300ms is the approximate human response time, this value is chosen because it is the maximum time under which the user should not appreciate any delay between input and output. On the other hand, the sliding value is limited, as stated above, by the processing power of the employed device: in [43], the involved microcontroller can process the input window in less than 12.8ms, hence 15ms is chosen also in this work in order to make fair comparisons between different methods.

Training is instead performed "offline", meaning not only that it is entirely performed ahead of time, but also that certain windows are discarded. To be more specific, those windows that overlap the resting position and the beginning (or the end) of a gesture are discarded; additionally, with the exception of the rest gesture, the first and the last 1.5s of each gesture are removed. When inference is performed on a signal that has been pre-processed in this way, this process is referred to as “steady movement classification”, since shaky and unstable bits of each movement have been removed from the dataset.

In addition to the considerations above, min-max normalization is applied to the raw signal, which is performed in order to have the same full-scale of $[-1; +1]$ for every session.

So the overall preprocessing pipeline of the training dataset for each subject consists in:

- Applying min-max normalization individually to each channel of the input signal;
- Removing 1.5s transients for every gesture;
- Dividing the resulting signal into sliding windows.

4.3 Target evaluation metric

First and foremost, it has to be noted that, due to the great variability between one subject and another, according to the state of the art, in this work we train a different learner to perform hand gesture classification on each specific subject: hence at least 10 different learners are required to perform the classification task on the 10 different subjects of the considered dataset. Note that this does not limit the generality of the approach, since each algorithm is tuned per patient in a first phase, but it can be then re-utilized forever for that specific patient.

Given that the NinaPro DB6 dataset is a multisession dataset, the first consecutive $N \in [1 \dots 5]$ sessions are used for training with 2-folds cross validation, while the remaining $10 - N$ sessions are used for testing. Specifically, the training data is divided into two folds, where in one fold only the windows belonging to even-numbered repetitions are included, while the remaining windows are included in the other fold. Therefore, for every subject, two classifiers are trained, each one on a different training fold: in the rest of this work, the average value of the top-1 classification accuracies on the validation folds is referred to as intra-session validation accuracy, while the average value of the top-1 classification accuracies on the test split is referred to as inter-session validation accuracy. Therefore, in order to fairly compare the effectiveness of different classifiers, these two metrics are computed for every fold of every subject and then averaged over the 10 involved subjects, meaning that 20 different trainings are performed for each model.

4.4 Network architecture

The network architecture employed to perform sEMG-based gesture recognition is closely inspired by ViT as proposed in [26] and described in chapter 2.3.6. The architecture is detailed in table 4.1 and consists of three main building blocks:

- a patch embedding block for turning the input signal window into a sequence of embedded tokens;
- a Transformer encoder that further encodes the input sequence;
- a classification head for classifying the input signal window.

Block	Layers	
Patch Embedding	1D Convolution	Turn input signal into N embedded tokens
	Class Token	Add learnable token to the input sequence
	Positional Embedding	Add positional information to each token
Transformer Encoder	Attention	MSA Layer
	Feed Forward	FC Layer GELU FC Layer
Classification Head	MLP head	Layer Normalization FC Layer

Table 4.1: Description of the proposed architecture

4.4.1 Patch Embedding Block

The input signal can be referred to as $x \in R^{14\times300}$ since every input example consists of a 14-channel signal window of 300 time samples. The main novelty is the adoption of a 1D convolution with the aim of turning the input signal into non-overlapping patches and linearly embedding each each patch into a lower dimensional space. To perform this operation, the input signal is convolved with a 1D filter $w \in 14 \times P \times C$ with a stride of P and no padding; P is the patch-size (i.e. the number of time-samples of each patch), and C is the number of output channels of the 1D convolution, which can also be thought as the embedding size of the linear projection that is applied to each patch. In fact, this operation is equivalent to linearizing the 14 channels of the input signal, reshaping it into $N = 300/P$ non-overlapping patches, i.e. $x' \in R^{N\times 14\cdot P}$, and applying to each patch a linear embedding whose projection matrix is $w' \in R^{14\cdot P\times C}$. The hyper-parameter P regulates the size of each patch, and consequently the length N of the input sequence: if P was set to 1, then the input sequence would consist of $N = 300$ tokens, where each token would be 1 time sample of the input signal; on the other hand, if P was set to 300, then the input sequence would degenerate into a $N = 1$ token sequence. As described in Chapter 6, the aim is to find the optimal patch size P, keeping in mind that the computing complexity of the MSA layer in terms of MACs is $O(N^2)$.

After encoding the input signal into a sequence of $N = 300/P$ tokens, a learnable classification token $cls \in R^C$ is prepended to the input sequence and a learnable positional embedding $w_{pe} \in R^{N+1\times C}$ as described in section 2.3.3 is summed to each token.

4.4.2 Transformer Encoder

The encoder is divided into repeating blocks of equal structure, each consisting of a self-attention layer with internal hidden embedding (also referred to as head dimension) of $D \leq C$ and a Feed Forward unit with hidden embedding dimension H being a multiple of C. As described in equation 2.10, Layer Normalization is applied on the input before applying Attention and before applying Feed Forward on it, and residual sum is performed at the output of Attention and Feed Forward.

Self-attention, as described in section 2.3.4, projects the input tokens into matrices Q,K,V, whose embedding dimension is referred to as $D \leq C$ in this work. Since this operation is repeated multiple times in parallel, i.e. multiple attention heads are applied to the same input sequence, the embedding dimension of each final token is $D \cdot N_H$, where N_H is the number of attention heads: therefore, each token is projected back to its original embedding dimension C with a fully connected layer.

The first Fully Connected layer of the Feed Forward unit projects each input token into a space of hidden dimension $H = mC, m \in N$; on the other hand, the second layer projects each token back to its initial embedding dimension C.

4.4.3 Classification head

Classification is performed with a fully connected layer. It is important to note that both Transformer Encoder and Patch Embedding block deal with a sequence of tokens for each input example, and a way of combining all tokens to perform classification of the whole sequence must be identified. As proposed in [26], classification is performed by feeding a MLP head with the processed class token.

4.5 Network training

A DNN is deployed to a low-power device when:

- The considered device has been loaded with the proper instructions to perform all the mathematical operations which the DNN consists of. This is usually carried out by loading a compiled software that implements these operations onto the MCU, or alternatively, by transferring a "recipe" file that describes all the involved operations and that is interpreted by a software already persisting on the embedded device;

- The optimal parameters of the DNN have been transferred to the embedded device and are plugged into the proper mathematical operations.

As it can be guessed, edge inference does not ultimately pose any restrictions on how the optimal parameters are found: this means that a DNN training, which is several orders of magnitude more complex than inference in terms of computational complexity, can be performed on another hardware. Therefore, in this work training is performed with dedicated GPUs since their highly parallelized architecture is ideal for computing the results of many independent mathematical operations, which is what is done during the forward and backward passes of a training step.

4.5.1 Weight initialization

Since DNN training is ultimately an optimization heuristic, an initial solution to be optimized has to be set, which is not a trivial operation. This task is usually referred to as weight initialization, and it is carried out in order to:

- improve convergence speed since having a bad initial solution may lead to slow and inefficient training;
- avoid gradient vanishing or gradient explosion due to the increase or reduction of the variances of the activations as depth grows.

This is why several works in literature, including [48, 49], are focused on weight initialization.

Specifically, the procedure outlined in [49] has been thought with non-linear activation functions in mind, which is the case of the employed architecture since it makes use of several operation of that kind, including Softmax and GELU. In order to prevent variance of activations from increasing or reducing as layer depth grows, Kaiming et al. propose to randomly initialize the layer weights according to some random distribution whose parameters depend on the input size of the layer. In practise, weights are independently drawn from a linear distribution:

$$\begin{gathered}\mathcal{U}(-\text{bound}, \text{bound}) \\ \text{bound} = \text{gain} \cdot \sqrt{\frac{3}{n}}\end{gathered} \tag{4.1}$$

where n is the input size.

In this work, gain was set to $1/\sqrt{3}$, which is commonly employed in many DL frameworks as it was empirically found to produce slight performance increases in terms of goal metrics [50].

Kaiming initialization is performed for all linear projection matrices. In the projection matrix of the positional embedding, weights are drawn from a normal distribution $N(0, 0.02)$ not only because it is done in the proposing article [26], but also because this is the approximate distribution that was reached at convergence for this model architecture trained on DB6 when weights were initialized using either a normal distribution or a uniform distribution. Therefore, this particular initialization is performed since initializing the weights with the same approximate distribution that emerges at convergence is supposed to increase convergence speed itself. Finally, the class token's weights are initialized to zero since weight updates Δw of class token are due to the gradient flow of all other tokens in the input sequence, so weight update is still possible with a 0s initialization, and furthermore, a class token consisting of 0 does not negative influence self-attention at the start of training.

4.5.2 Warmup of the learning rate

Training a Transformer with the traditional approach of using a fixed learning rate that is optionally decayed by a certain factor everytime a number of training steps are performed may result in suboptimal results: using a too low learning rate may slow the convergence, while a too high learning may lead to divergence in the first few steps. It is therefore suggested in literature [28] to vary the learning rate according to a more sophisticated policy, as in the warmup policy. The warmup of the learning rate consists in setting the initial learning rate to a very low value close to 0, such as 10^{-7}, and then to vary the learning rate according to a certain policy. The simplest one consists in linearly increasing the learning rate for a number of epochs until a target fixed learning rate is reached. A slight variation of it consists in also decaying the learning rate up to the end of the training: Figure 4.1 graphically shows the trend of the learning rate at each epoch with a linear warmup of 150 epochs from approximately 0 and 0.001 and a subsequent linear decay of 50 epochs.

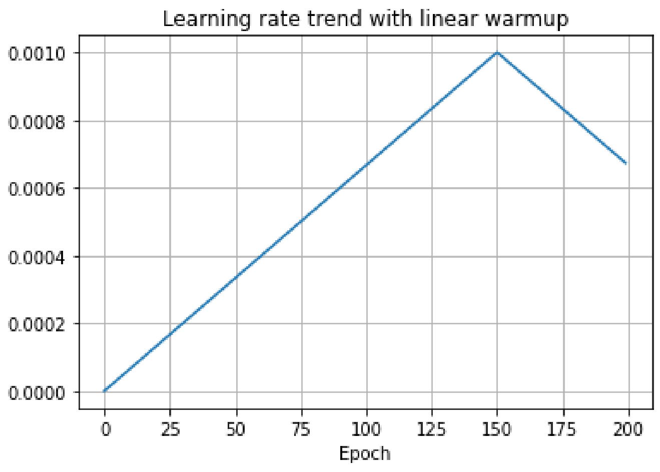

Figure 4.1: Example of a linear warmup policy with linear decay after warmup

The linear warmup of the learning rate seems to provide not only marginally better accuracies at convergence, but also a more stable training: 10 different training curves regarding intra-session and inter-session accuracy on a single subject of DB6 are reported in Figure 4.2 and each curve is resulted from training the same model from scratch on the same subject and with the same hyper-parameters. In the right panel, the Adam optimizer adopts a fixed learning rate of 0.001 that is halved every 50 epoch, while instead a linear warmup for 100 epochs with the same final learning rate of 0.001 is used in the panel on the left. From the different training curves it can be seen that, with linear warmup of the learning rate, the intra-session accuracy of 65% is reached after (14 ± 3) epochs, while instead the fixed learning strategy exhibits a larger variance, since the same value of accuracy is reached only by 9 trainings under 100 epochs and specifically after (19 ± 8) epochs.

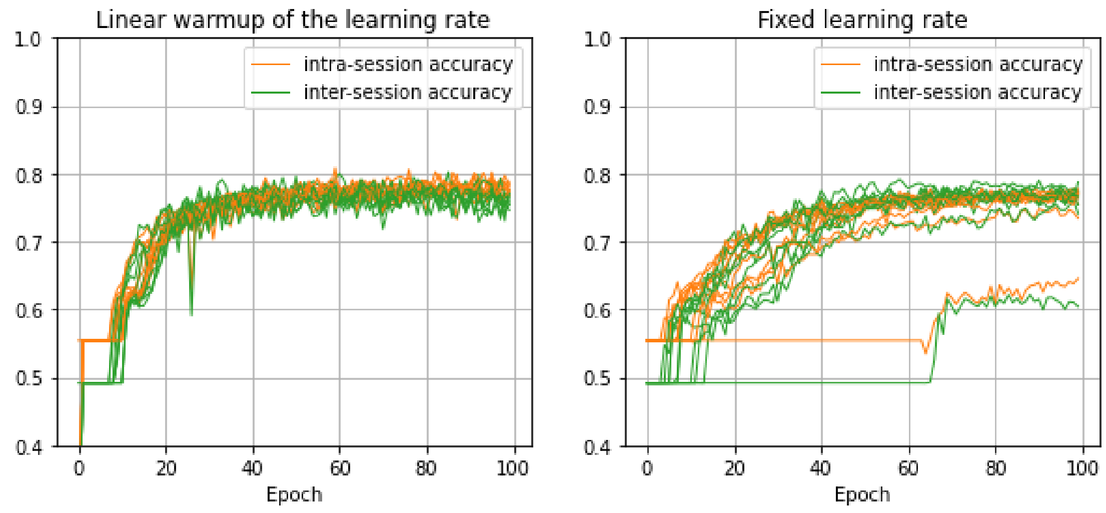

Figure 4.2: Comparison of training curves concerning different learning rate schedule policies

4.5.3 Two-stage training protocol

As discussed above, the standard training procedure for sEMG-based gesture recognition concerns only a specific subject, meaning that the considered training samples belong to one subject only; therefore the trained model is subject specific. This procedure is followed because there is significant variance from one subject to another in terms of recorded signals due to different muscle movements and contractions among different subjects. However, single-subject data is scarce and training on a wider dataset may allow the model to extract more meaningful features. Transformers have already been successfully trained with fine-tuning in mind: such instances include [51], where a model for natural language processing is pre-trained on large corpus in order to allow the straightforward creation of state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task specific architecture modifications but with the application of transfer learning techniques. In this work we try to adopt a similar protocol also in the context of sEMG-based gesture recognition. In particular, instead of training from scratch the final subject-specific model on a random weight initialization, for every subject it is performed fine-tuning of a model that was pre-trained on all the other subjects of the same benchmark dataset. This implies a two-stage procedure for training a model for a given subject: in the first step, the model parameters are randomly initialized and pre-training is performed considering all the other subjects samples; in the second step, the samples of the selected subject are normalized with the same min-max ranges, and the parameters of all the layers of the previously trained model are fine-tuned. As shown by our results, this protocol is beneficial for final accuracies since a greater generalization capability of the trained models is achieved: this is because, despite every recorded signal being subject-dependant, pre-training on a wider dataset allow the resulting model to extract more meaningful features, since those sEMG signal features that are useful for gesture classification are supposed to share many similarities among the different subjects.

4.5.4 Transformer-specific quantizations

One of the challenges for performing integer-only quantization of Transformers is their strong reliance on nonlinear operations. The authors of [35] propose a framework to approximate the nonlinear layers of Transformers, namely GELU and Softmax, with polynomial functions that can be computed with integer-only arithmetic. This method is effective because computing polynomials consists of only additions and multiplications, which can be performed with integer arithmetic. These approximations are carried out in order to avoid quantization and dequantization steps inside the network architecture graph, where operations in certain layers are performed in integer arithmetic while in other layers operations are performed with floating-point arithmetic.

GELU

The integer-only approximation to GELU can be defined as follows:

$$iGELU(x) = x \cdot \frac{1}{2}\left(1 + L\left(\frac{x}{\sqrt{2}}\right)\right) \tag{4.2}$$

where $L(x)$ is defined by solving this optimization problem:

$$\begin{aligned} \min_{a,b} \frac{1}{2}||GELU(x) - iGELU(x)||_2^2 \\ s.t. L(x) = sgn(x)(a(clip(|x|; max = -b) + b)^2 + 1) \end{aligned} \tag{4.3}$$

Softmax

The softmax operation involves the use of the exponential operation, which is the nonlinear term of this operation. A polynomial approximation of the exponential that can be calculated with integer arithmetic can be defined as follows:

$$\begin{aligned} iexp(\tilde{x}) &:= L(p) >> z \\ L(p) &= 0.3585(p + 1.353)^2 + 0.344 \\ p &= \tilde{x} + z \cdot \log 2 \\ z &= \lfloor -\tilde{x}/\log 2 \rfloor \end{aligned} \tag{4.4}$$

where $>>$ represents the bit shifting operation.

4.5.5 Overall training procedure

Training an instance of the proposed Transformer-based architecture to perform hand gesture recognition of a certain subject in the benchmark dataset can be summarized in the following:

1. Perform a pre-training step of a randomly initialized model on the training data of the remaining 9 subjects of the dataset;
2. Perform a quantization-aware finetuning step of the whole architecture from the previous checkpoint with the training data of the target subject.

Chapter 5

Automatically Generated Hybrid CNN/Transformer Architectures

5.1 Objective

In the context of sEMG analysis, or Time Series Analysis in general, after having successfully applied the self-attention mechanism as described in the previous chapter, and the convolutional layers as done in previously introduced TCN architectures, such as TEMPONet [43], the most obvious next step is to introduce a hybrid architecture that is able to leverage both the long-range modelling capability of the self-attention mechanism and the shorter-range summarization capability and lower memory occupation of convolutional layers. The work of [27] already introduces an innovation in the MSA mechanism by introducing a relative positional embedding, which can bias the attention mechanism to be narrower and focused on the neighbour tokens, thus behaving similarly to a convolutional layer. The objective of our work is to start from that formulation to realize a sort of NAS where each layer of a hybrid transformer is either pure attention-based or pure convolutional based.

5.2 Application of GPSA to sEMG-based hand gesture recognition

One of the main advantages of GPSA when applied in the context of sEMG-based hand gesture recognition is that the resulting transformer encoder is less prone to overfitting as the size of the network architecture grows. Hence, a higher number of layers can be stacked without hurting the target metric performances. This is particularly convenient because it enables the inspection of the GPSA behaviour at an increasing number of layers. One way to interpret the behaviour of a GPSA-based layer consists in averaging the gating parameters $\sigma(\lambda_h)$ for every head: if the result is close to 1, then all the attention heads pay more attention to position information and the whole layer is closer to a convolutional layer; if instead, the average value is lower or close to 0, this means that the various attention heads are paying attention to the content information and the layer is closer to a vanilla multi-head self-attention layer.

Figures 5.1 and 5.2 show the trend of the gating parameters of each layer (gray lines) and their average value (black thicker line) in two different transformers that employ GPSA and that were trained on 5 sessions of all 10 subjects of DB6 for 150 epochs. First of all, it can be seen that the first layers of each Transformer exhibit a behaviour closer to a convolutional layer at convergence since the average gating parameter is higher, while instead deeper layers are closer to classical MSA, which is compatible with the findings of the proposing paper [27] applied to image classification. It is important to note also that, for the variant with 2 attention heads, the gating parameter is higher in the first layers, while instead in the variant of 9 heads, the average value of the gating parameter tends to decay during training even for the first layers.

These insights suggest that, in order to achieve a closer convolutional behavior in the GPSA mechanism, a small kernel size for the equivalent convolution may be appropriate, hence a lower number of attention heads may be sufficient.

5.3 GPSA binarization via a pretraining step

GPSA binarization consists in identifying a threshold for the average value of the gating parameters of a certain MGPSA layer above which all values

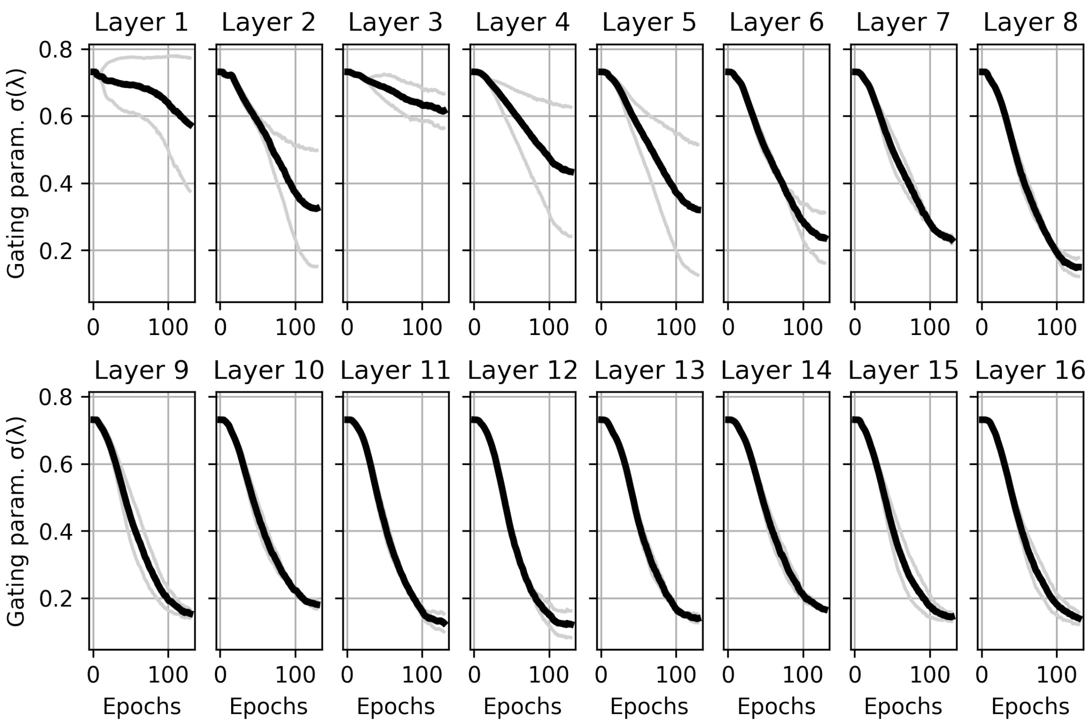

Figure 5.1: Gating parameter of a 16 layer transformer with 2 heads in each layer

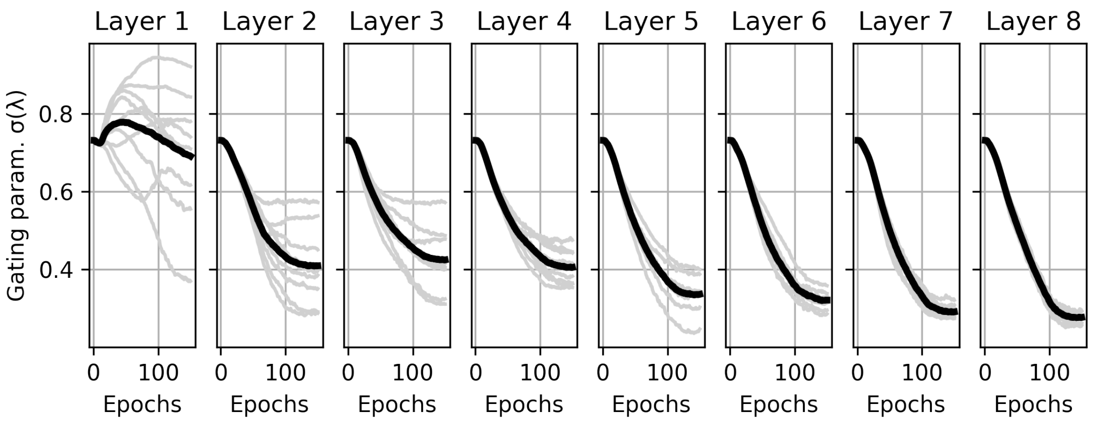

Figure 5.2: Gating parameter of a 8 layer transformer with 9 heads in each layer

of the gating parameters are rounded to 1 and below which all values of the gating parameters are rounded to 0. This is performed in order to force a

MGPSA layer to strictly pay attention either to positional information or to content information. While the former exhibits a behaviour closer to a convolution, the latter is totally equivalent to vanilla MSA.

To perform this operation, in this setting a training step is performed on all subjects in order to gather a snapshot of the final values of each gating parameters. In fact, after the completition of this training step, the values of the gating parameter of each layer in the trained model are inspected: if the average gating parameter of a given MGPSA layer is above a certain threshold, the layer is turned into a pure convolutional layer, otherwise the layer is turned into a pure MSA layer. Therefore, after having empirically determined the new configuration of each layer of the architecture for the given task, the model is re-initialized and then trained in single-subject trainings, optionally with pre-training as described in the chapter before.

Identifying the correct threshold t_h of the average gating parameter can be difficult. In this work, we set t_h to the central value between the minimum and maximum mean gating parameter over all layers, i.e. $t_h = \sum_j \sum_i \sigma(\lambda_{ij})/N_H, i \in [1, \ldots, N_H], j \in [1, \ldots, L]$, where L is the depth of the transformer encoder, N_H is the number of heads in each layer, λ_{ij} is the unscaled gating parameter of head i in layer j and σ is the sigmoid function.

The objective of this search is to find a hybrid network that takes full advantage of the strong embedding capabilities of the attention mechanism, and fallbacks to a less memory-intensive convolution operation when the use of pure self-attention may be redundant or even detrimental for the target metrics. In fact, considering an embedding dimension $D = C$ for each head, then the number of parameters of an attention layer is $O(4N_HC^2)$ where N_H is the number of heads, while instead the number of parameters of the equivalent convolutional layer is N_HC^2, which leads to a reduction of $4\times$.

5.4 GPSA binarization via Straight-Through Estimator

The problem of the previous solution is that a further extra training step is required, which can be a significant burden. In this section, a dynamic binarization that is performed during training is described.

This binarization operation is described in the equation below.

$$\tilde{\sigma}(\lambda_i) = \begin{cases} 1 & \sum_i \sigma(\lambda_i)/N_H > t_h \\ 0 & \sum_i \sigma(\lambda_i)/N_H \leq t_h \end{cases} \tag{5.1}$$

where $\sigma(\lambda_i)$ is the gating parameter of head i and t_h is a threshold value.

However, training with error backpropagation cannot be directly performed since $\tilde{\sigma}(\lambda_i)$ is not differentiable in t_h and most importantly its derivative is 0 for all other values. Hence a Straight-Through Estimator is employed, which consists in evalutating to $\tilde{\sigma}(\lambda_i)$ during the forward pass and in removing the binarization during the backward pass, thus allowing gradient flow during the backward pass.

In this way, at convergence time the MGPSA layer will behave either as pure MSA layer or as an embedding layer of the relative position information of the input patches, which is not exactly the intended result of having an alternative between pure MSA and pure convolution. This is because relative position embedding can result in a layer being equivalent to a convolutional layer, but only if the learned embedding is in a certain form as mentioned in section 2.3.8.

To solve this issue, the learnable relative positional embedding is replaced with a fixed value. Therefore the new altered gated attention $\tilde{A^h_{ij}}$ that measures interaction between token i w.r.t to token j in head h is given by

$$\tilde{A^h_{ij}} = [1 - \sigma(\lambda)\text{softmax}(Q_i K_j^T)] + \sigma(\lambda)[0, \dots, 1_{i_h}, \dots, 0] \tag{5.2}$$

where $i_h \in [i - (N_h - 1)/2, \dots, i + (N_h - 1)/2]$ where N_h is the number of heads of the layer.

This way, if $\sigma(\lambda) \leq t_h$, then the layer is equivalent to plain MSA in the forward pass due to GPSA binarization. If $\sigma(\lambda) > t_h$, then multi head attention for token i in a 3 head layer will be equivalent to:

$$\begin{aligned}
A_i^{h=1} &= \begin{bmatrix} 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ \vdots & & 1 & 0 & 0 & & \vdots \\ 0 & \dots & 1 & 0 & 0 & \dots & 0 \end{bmatrix} XW_{val} = X_{i-1}W_{val} \\
A_i^{h=2} &= \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ \vdots & & 0 & 1 & 0 & & \vdots \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix} XW_{val} = X_{i}W_{val} \\
A_i^{h=3} &= \begin{bmatrix} 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & & 0 & 0 & 1 & & \vdots \\ 0 & \dots & 0 & 0 & 1 & \dots & 0 \end{bmatrix} XW_{val} = X_{i+1}W_{val}
\end{aligned} \tag{5.3}$$

Thus, when attentions from all 3 heads get concatenated, the result of self-attention is given by:

$$\begin{bmatrix} 0 \cdot W_{val} & X_1 W_{val} & X_2 W_{val} \\ X_1 W_{val} & X_2 W_{val} & X_3 W_{val} \\ \vdots & \vdots & \vdots \\ X_{N-2} W_{val} & X_{N-1} W_{val} & X_N W_{val} \\ X_{N-1} W_{val} & X_N W_{val} & 0 \cdot W_{val} \end{bmatrix} W_{out} + b_{out} \tag{5.4}$$

which is equivalent to a 1D convolution with a kernel size of h (in this case 3) with a stride of 1 and 0 padding. In this case, projections $X_i W_{val}$ instead of pure X_is are being multiplicated by the moving kernel W_{out}, but this "double" linear projection is equivalent to the single linear projection $W_{val} W_{out}$.

Chapter 6

Experimental results

6.1 Setup

Deep Learning has been gaining popularity also thanks to the availability of many open-source frameworks that allow fast development cycles since all major hurdles, ranging from low-level programming of GPU hardware to high-level topic-specific operations, such as convolutions, have already been implemented. In this work the Pytorch framework [52] has been adopted and training has been performed on a single NVIDIA Tesla V100 SXM2.

The target deployment MCU is the Green-Waves Technologies GAP8 SoC [53], which is based on the RISC-V architecture and on the PULP (Parallel Ultra-Low-Power Processing Platform) open-source platform. This MCU allows a ultra-low-power operation thanks to an optimized set of I/O ports and to a suitable architectural organization that consists of one computing core for control, communications and security functions, which is called Fabric Controller, and a cluster of 8 cores that are designed for the execution of vectorized and parallelized operations. The memory organization consists of three levels: level 1 consists of a 64KB memory shared by all eight cores and a 8KB memory for the Fabric Controller; level 2 memory consists of a 512KB memory that is shared by all cores; level 3 memory, or RAM, is given by an off-chip low-power DRAM, such as an 8MB "HyperRAM" which is used on the GAPuino development board [54].

Given the target architecture, the energy consumption for performing inference on a single sample can be estimate as:

$$E = \frac{c}{f} \cdot P \tag{6.1}$$

while the latency in ms can be estimated as:

$$E = \frac{c}{f} \cdot 1000 \tag{6.2}$$

where P is the power consumption, which is 51mW, f is the clock frequency, which is 100MHz, and c is the overall number of clock cycles that are required to perform a single forward-pass.

6.2 Pure transformer architectures

The pre-processing of the benchmark dataset is based on [43] in order to allow fair comparisons between the transformer-based architectures and TEMPONet, a state-of-the-art TCN-based technique for handling sEMG signals. Consequently, only training and architecture hyper-parameters of the Transformer architectures are tuned to find the best configuration among a predetermined search space that leads to the highest performances in terms of the target metrics.

6.2.1 Number of parameters and MAC operations

The computation of the number of parameters of the involved transformers results from the sum of the different contributions of the various layers, which is reported below:

- Patch embedding (input 1D convolutional filter): $14 \cdot P \cdot C + C$, where P is the receptive field of the non-overlapping convolution, C is embedding dimension of each encoded token, 14 is the number of channels of the input signal;
- Learnable class token: C;
- Learnable positional embedding: $C \cdot (N + 1)$, where $N = 300/P$ is the number of tokens (note that it is considered $N + 1$ due to the class token);
- Transformer encoder:
  - Attention Layer:
    - Layer Normalization: $2C$;

    * QKV projection matrices: $C \cdot D \cdot N_H \cdot 3$, where N_H is the number of heads and D is the head size (note that no bias term is present in this projection);
    * Projection: $D \cdot N_H \cdot C + C$
  - MLP block:
    * Layer Normalization: $2C$;
    * Fully connected layer 1: $C \cdot H + H$;
    * Fully connected layer 2: $H \cdot C + C$;

- Classification head
  - Layer Normalization: $2C$;
  - Output fully connected layer: $C \cdot N_c + N_c$, where N_c is the number of classes

Regarding the MAC operations, their computations is summarized below:

- $MACs(encoder) = L \cdot [MACs(attention) + MACs(feedforward)]$, where L is the number of layers of the transformer encoder;

- $MACs(attention) = N_H \cdot MACs(attentionhead) + N \cdot (D \cdot N_H)$ where N_H is the number of heads, D is the embedding dimension of each head and N is the sequence length (including the eventual class token);

- $MACs(attentionhead) = 3 \cdot N \cdot C \cdot D + N \cdot D \cdot N + N \cdot N \cdot D$ where N is the sequence length, C is the embedding dimension of each token and D is the embedding dimension of the head; the first addendum concerns the QKV matrix projection, the second terms refers to the similarity dot product QK^T while the last term concerns the multiplication of the similarity scores computed by QK^T by V;

- $MACs(feedforward) = 2 \cdot N \cdot C \cdot H$, where H is the hidden embedding dimension of the MLP head;

- $MACs(transformer) = N \cdot (14 \times P) \cdot C + MACs(encoder) + C \cdot N_c$ where P is the patch size of 14 channel input signal, and N_c is the number of classes

6.2.2 Training hyper-parameters

For determining the proper configuration of training hyper-parameters, a number of attempts have been made with different architectures. In particular, different training policies were tested against different variants of the Transformer architecture in terms of its size: for each training policy, the tested depth, i.e. number of layers of the transformer encoder, were $[1, 2, 4, 8]$, while the number of attention heads in each self-attention layers were either 2, 4 or 8.

Training has been performed with Adam optimizer with a batch size of 64 and no weight decay on 5 sessions. Table 6.1 shows different hyper-parameter configurations for the two-stage training: all pre-training stages have been performed by adopting the warmup of the learning rate with a 5e-6 increase at each epoch; finetuning has been performed with a fixed learning rate of 1e-4 with a reduction of 10× after half of total number of epochs were past. This table shows that adopting a too short pre-training stage may actually not be sufficient for achieving optimal performances in terms of accuracy, since both attempts with 20 pre-training epochs are outperformed by the configuration with 100 epochs; on the other hand, finetuning for 40 epochs does not seem to provide an edge over 20 epochs, and may actually lead to a slight overfit of the involved networks.

Hence, the final configurations consist of the following:

- In case of two-step training, for the pre-training step, linear warmup of the learning rate from 1e-7 to 5e-4 is employed for 100 epochs; for the fine-tuning step, a fixed learning rate of 1e-4 is used, with a reduction of 10× after 10 epochs;
- In case of single training from scratch (no pre-training), then a linear warmup of the learning rate from 1e-7 to 1e-4 is employed for 100 epochs, which is then followed by 50 epochs of linear decay up to 5e-4.

6.2.3 Architecture hyper-parameters

A grid search was performed over a number of hyper-parameters, including:

- patch size, i.e. the size of the receptive field of the 1D convolution at the input of the transformer architecture;
- projection size, i.e. embedding dimension of each token;

Pre-tr. epochs	Ft. epochs	Intra-sess. acc.	Inter-sess. acc.
20	20	63.1%	59.2 %
20	40	64.3%	59.1 %
100	20	68.4%	62.1 %

Table 6.1: Averages of inter-session and intra-session accuracies over all 10 subjects and all 12 architectures for every configuration of the two-stage training protocol, i.e. a total of 240 instances for every configuration

- number of attention heads in the self-attention layers;
- size of each attention head, i.e embedding dimension of Q,K,V;
- depth L, i.e. number of layers of the transformer encoder;
- MLP size H, i.e. the embedding dimension of the MLP block.

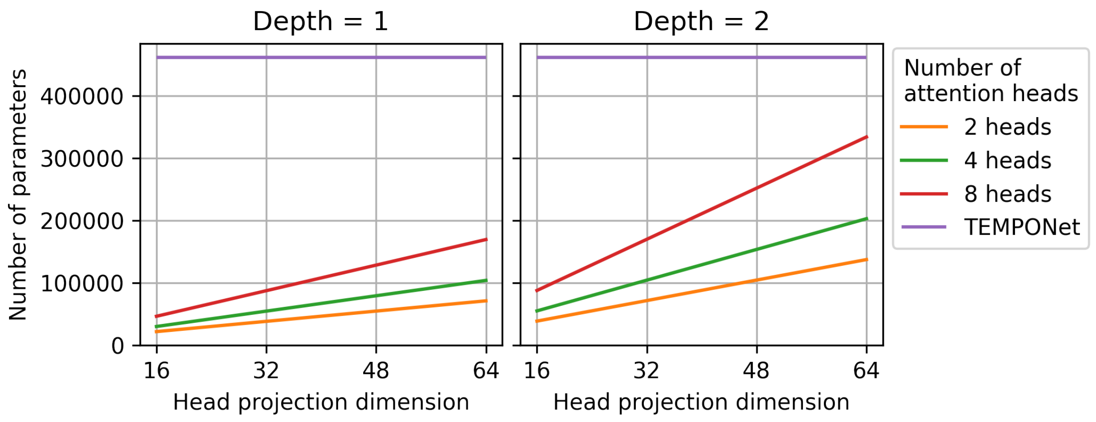

Figure 6.1: Number of parameters with respect to head projection dimension

The overall number of parameters of the architecture are mostly determined by the attention layers in the transformer encoder, whose complexity is $O(C \cdot D \cdot N_H \cdot L)$: hence this are the most important hyper-parameters to constraint in order to design a final architecture whose final memory footprint is compatible with an edge deployment. Figure 6.1 shows how the total number of parameters varies in the architecture with increasing depth L of the transformer encoder, embedding dimension size D

of the self-attention mechanism and number of attention heads N_H, after having set a projection size $C = 64$; patch size P and MLP size H do not heavily affect the total number of parameters, and in this instance they are set to 10 and 128 respectively.

	Intra-session acc.			Inter-session acc		
	min	max	mean	min	max	mean
D						
16	0.628670	0.689759	0.676824	0.592398	0.637153	0.622521
32	0.631735	0.699998	0.679062	0.593883	0.642666	0.624112
64	0.629874	0.707415	0.680656	0.594235	0.639759	0.625326

Table 6.2: Aggregate values of inter-session and intra-session accuracies in 5-session training with depths $L = [1, 2]$, number of heads $N_H = [2, 4, 8]$ and head size $D = [16, 32, 64]$.

In order to find a good set of hyper-parameters that represent a good compromise between goal metrics and computing complexity, different architectures were trained with the aforementioned two-stage training protocol in 5-session trainings with different depths, number of heads and head size. Looking at the aggregate values of intra-session and inter-session accuracies of table 6.2, it seems that setting a projection $D = 16$ leads to slight underfit given the lowest maximal intra-session accuracy; on the other hand, setting a projection $D = 64$ seems to lead to slight overfit, since the maximal inter-session accuracy is lower than $D = 32$, and most importantly, the accuracy gains seem negligible compared to the increased memory footprint.

As a result, two architectures are chosen and they share most of their hyper-parameters: MLP size H is set to 128, projection size C to 64, head size D to 32 and patch size P to 10. The two architectures differ in the number of heads and depth: the first model is characterized by 1 layer with 8 heads, while the second model by 2 layers with 2 heads each. The variant with 8 heads was chosen since it reaches the maximum performances in terms of goal metrics, while instead the variant with 2 heads was chosen as a less computing-intensive alternative to compare with the first one.

6.2.4 Results without quantization

Training protocol comparison

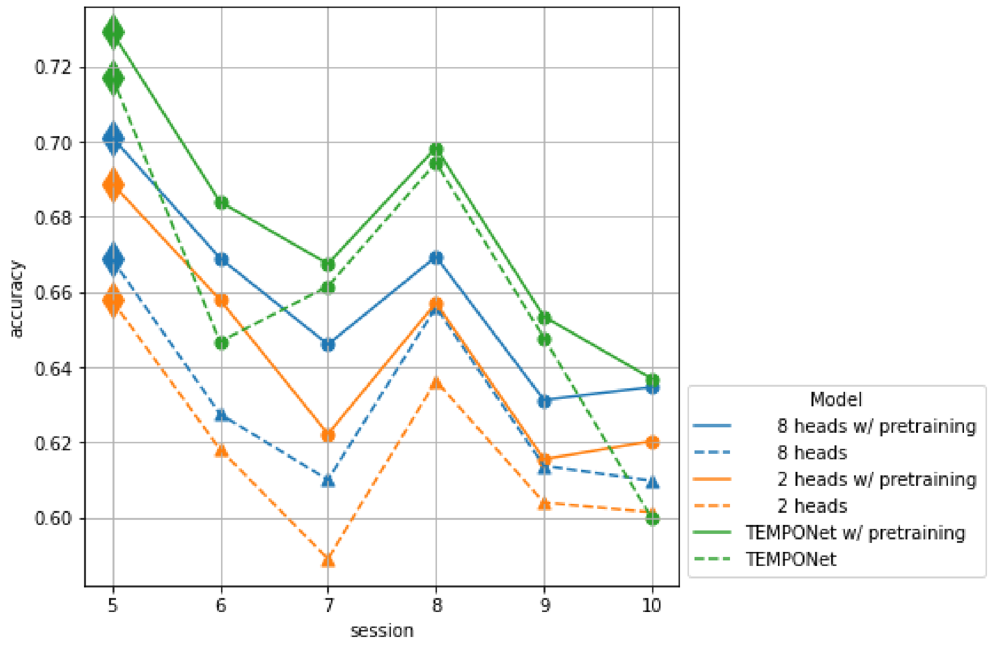

Figure 6.2: Intra-session accuracy (5) and inter-session accuracies (6-10) of the considered models

Model	Pre-tr.	Inter-sess. acc.
Tr1	No	62.3%
Tr1	Yes	65.7%
Tr2	No	60.9%
Tr2	Yes	63.4%
TEMPONet	No	65.0%
TEMPONet	Yes	66.8%

Table 6.3: Mean intra-session and inter-sessions accuracies. Tr1 corresponds to Transformer with 1 layer and 8 heads, Tr2 to Transformer with 2 layers and 2 heads.

The results achieved by transformer applied on 5-sessions trainings are compared with TEMPONet, which is trained following the same training hyper-parameters of [43]. First it is to be noted that, while TEMPONet employs 460k parameters and 16M MACs, the computation complexity of the considered transformers is notably lower, characterized by 94k and 78k parameters and 3.3M and 2.5M MACs respectively for the 1 layer and 2 layers variants. Applying the pre-training is beneficial both for transformers and TEMPONet, resulting in appreciable accuracy gains for both architectures. In particular, in figure 6.2 and in table 6.3 it is showed how the intra-session (5) and inter-session accuracies (6-10) vary: both for the considered TCN and transformers, the two-stage training procedure does provide an edge over training from scratch since both accuracy metrics are higher when pre-training is applied; it can be observed an average gain of 3.39%, 2.48%, and 1.80% for Transformer with depth 1, Transformer with depth 2, and TEMPONet, respectively. This confirms the recent identified trends that see Transformers being able to achieve higher accuracy gains with respect to convolutional-based networks when finetuning is employed; in this particular task however, Transformers do not provide better accuracies than state-of-the-art when the matching training protocols are employed.

Per-subject accuracy comparison

In figure 6.3, it can be seen the per-subject accuracy variations between 2-stage training protocol and training from scratch when applied to the 1-layer Transformer: while most subjects see a slight to considerable increase in terms of inter-session accuracy, there are few instances (subject 2 and 6) where performing a pre-training does not lead to any appreciable increase or it actually worsens the final accuracy.

Therefore, while the extracted features during the pre-training step prove to be beneficial for final accuracy in most cases, given the high variability between different subjects, it is very well possible for a particular subject with strong differences from the "mean" case to performs worse with a pre-training step. This insight may suggest that performing pre-training step on a subset of similar subjects instead of the whole corpus may provide better results for this particular instances.

Ablation study on the patch-size

In figure 6.4, it is showed how the receptive field of the input 1D convolutional filter impacts on the achieved inter-session accuracy.

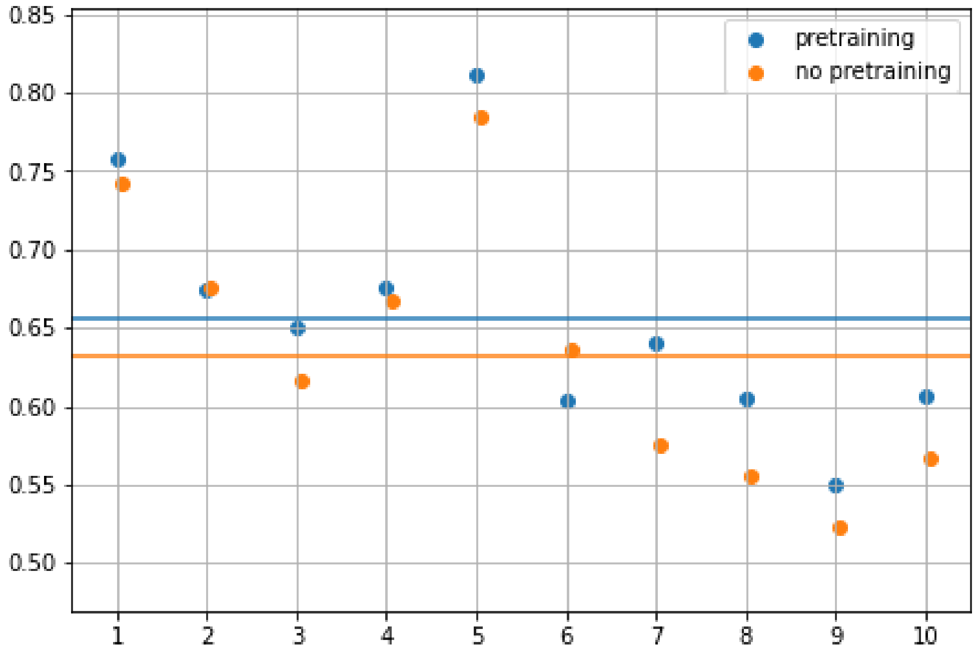

Figure 6.3: 1-stage vs 2-stage training results for each subject

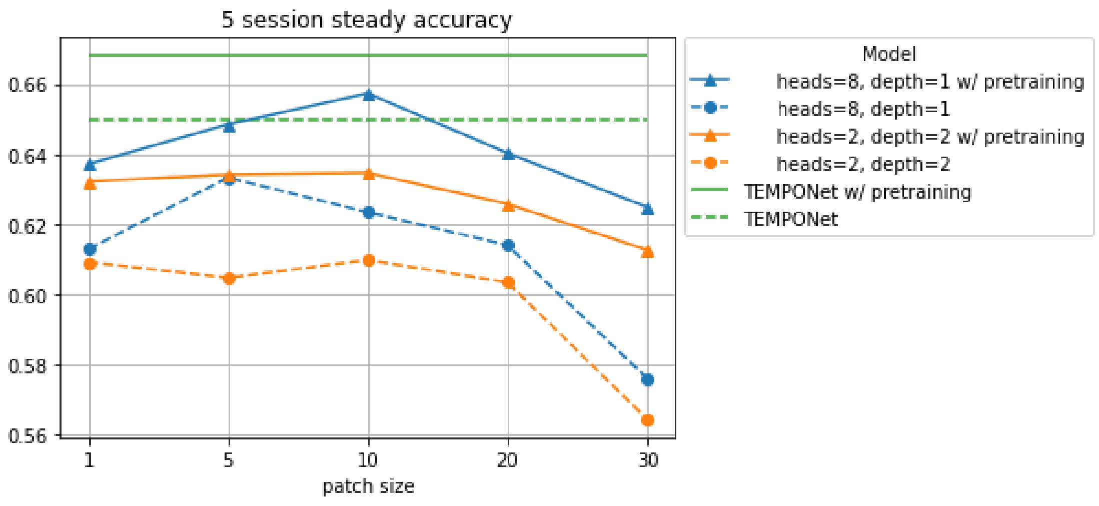

Figure 6.4: Input 1D convolutional filter dimension

It is to be noted that, since the stride of the convolution is the same as the its receptive field, the embedded tokens are obtained from non-overlapping segments of the input. The patch size P strongly influences the computing complexity of the resulting architecture since the length N of the resulting sequences is given by $N = L/P$, where $L = 300$ is the length of the input window. The sweetspot seems to be around a value of 10 for the patch size: the lower accuracies for the other architectures may be justified by stating that a higher patch size determines a too short sequence, while a too long sequence requires a too much small input token. Also note, as shown in figure 6.5 how the patch size influences the complexity of the resulting architecture: while the number of parameters does not heavily depends on the resulting input sequence length (only the positional embedding and the 1D patch embedding are influenced), it is instead evident that the number of MACs increases with the patch size, mainly due to the higher number of operations that take place in the self-attention layers.

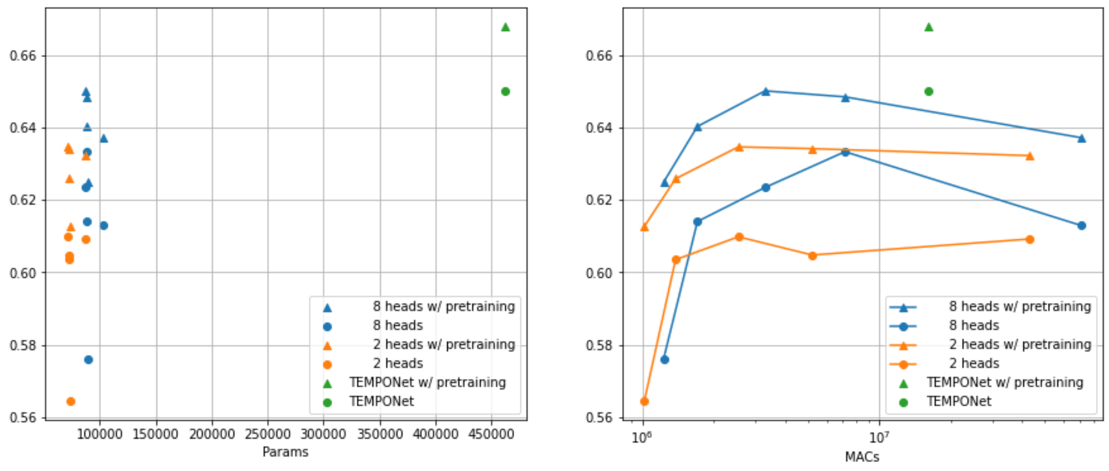

Figure 6.5: Number of parameters and MAC operations of TEMPONet and the two considered transformer variants with various input 1D convolutional filter dimensions

6.2.5 Results with quantization

The results of quantizing some of the aforementioned architectures are shown in Table 6.4. The accuracy of the reported models is the one obtained after the quantization-aware fine-tuning, which is the second step of the two-stage training that has been introduced in chapter 4. After quantization, the most accurate model achieves 64.69% inter-session accuracy. Energy consumption and latency are computed as shown in section 6.1

Table 6.4: Performance of the quantized architectures.
Abbreviations: Lat.: latency, Q.Acc.: quantized accuracy.

Network	Memory	MMAC	Lat.[ms]	E.[mJ]	Q. Acc.
MCU: GAP8, 100 MHz @ 1V, 51 mW					
Tr1, patch=30	110.8 kB	1.2	1.03	0.052	61.09%
Tr1, patch=20	102.1 kB	1.7	1.37	0.070	63.14%
Tr1, patch=10	94.2 kB	3.3	2.72	0.139	64.69%
Tr2, patch=30	92.2 kB	1.0	1.55	0.079	60.19%
Tr2, patch=10	78.3 kB	2.5	4.82	0.246	62.43%
TEMPONet [43]	461 kB	16.0	21.82	1.11	61.00%

6.3 GPSA-based architecture

Training is performed with AdamW optimizer in a two-step training with a batch size of 64 and a weight decay of 1e-2. For the pre-training step, linear warmup of the learning rate from 1e-7 to 1e-3 is employed for 50 epochs followed by 25 epochs of linear decay of the learning rate to 5e-4; for the fine-tuning step, a fixed learning rate of 1e-4 is used, with a reduction of 10× after 10 epochs. All experimented network architectures consists of an embedding size of patch-size of 10 with a linear projection dimension of 32, hidden dimension of the MLP block of 128, and 3 heads per layer with embedding size of 32 each.

First, the network architecture search was performed with the additional pre-training step as described in section 5.3, which was performed on the first 5 sessions of all subjects of DB6, and then single-subject trainings was performed with the two-stage fashion: table 6.5 shows that, as the number of employed layers increases, also the inter-session accuracy does, with the exception of the largest model with 8 layers where a small accuracy drop is registered. The higher generalization capability of the GPSA-based transformer is probably due to the narrower receptive field of the layer when it is biased to be a convolution, since it takes into account only the closest tokens and is not constrained to model farther time-dependencies.

Finally, when GPSA binarization was performed with a Straight-Trough Estimator, an average inter-session accuracy of 59.3% has been achieved with single-subject training without pretraining with a linear warmup of the learning rate from 1e-7 to 1e-4 for 50 epochs of a 8 layer architecture.

Table 6.5: Performance of the GPSA-based transformer architecture where binarization has been performed via a pre training step.
Abbreviations: Acc.: inter-session accuracy

Depth	Conv. Layers	Attention Layers	Acc.	Params	MMAC
2	1: (0)	1: (1)	61.3 %	166k	1.3
3	2: (0, 1)	1: (2)	63.6 %	241k	1.6
4	3: (0, 1, 2)	1: (3)	64.3 %	315k	2.0
8	3: (0, 1, 2)	5: (3, 4, 5, 6, 7)	63.0 %	613k	5.2

By comparing the results achieved by the models that feature binarized GPSA layers with transformers that are based on pure self-attention, it can be seen that the pure transformers require an higher amount of MAC operations: the pure transformer with 2 layers and 2 heads per layer requires 2.5M MACs and achieves an inter-session accuracy of 63.4%, while instead the hybrid 4-layer transformer with 3 convolutional layers, despite having 2 more layers in total, scores an higher accuracy of 64.3% while requiring only 2.0M MACs, and the 3-layer hybrid model, which achieves the same accuracy of the pure transformer, only requires 1.6M MACs. These results are obtained thanks to the reduction of the number of attention layers to 1 and to the introduction of convolutional layers that are more lightweight in terms of MACs.

Chapter 7

Conclusions and future works

This work showed that Transformers can achieve comparable state-of-the-art performances even in highly-compressed architectures applied to embedded tasks such as sEMG-based hand movement recognition.

Despite the many challenges that make it difficult to bring this architecture to edge inference, several previous works introduced a number of techniques that allow this operation to be performed without incurring significant accuracy penalties.

The application of a two-stage training procedure led to better performances, and is compatible with previous findings that sees Transformers being able to achieve higher accuracy gains with respect to convolutional-based networks when finetuning is employed. In this particular task, however, Transformers do not provide better accuracies than state-of-the-art when matching training protocols are employed also for competitive architectures.

One of the issues faced when carrying out this work is the propensity of the employed models to overfit as the number of layers of the encoder increases. This issue is not prominent when GPSA-based layers are employed, but still, the overall accuracy metrics do not surpass previously introduced convolutional architectures.

Future works include the extension of NAS approaches applied in GPSA-based networks in order to further experiment with hybrid architectures consisting of convolutional and self-attention layers.

Bibliography

[1] Geoffrey Hinton. «Boltzmann Machines». In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 132–136. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_83. URL: https://doi.org/10.1007/978-0-387-30164-8_83 (cit. on p. 1).

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. on p. 1).

[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. «Backpropagation Applied to Handwritten Zip Code Recognition». In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541 (cit. on p. 1).

[4] Alex Graves and Jürgen Schmidhuber. «Framewise phoneme classification with bidirectional LSTM and other neural network architectures». In: *Neural Networks* 18.5 (2005). IJCNN 2005, pp. 602–610. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2005.06.042. URL: https://www.sciencedirect.com/science/article/pii/S0893608005001206 (cit. on p. 1).

[5] Ruhi Sarikaya, Geoffrey E. Hinton, and Anoop Deoras. «Application of Deep Belief Networks for Natural Language Understanding». In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.4 (2014), pp. 778–784. DOI: 10.1109/TASLP.2014.2303296 (cit. on p. 1).

[6] Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton. «Acoustic Modeling Using Deep Belief Networks». In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.1 (2012), pp. 14–22. DOI: 10.1109/TASL.2011.2109382 (cit. on p. 1).

[7] M.D. Levine. «Feature extraction: A survey». In: *Proceedings of the IEEE* 57.8 (1969), pp. 1391–1407. DOI: 10.1109/PROC.1969.7277 (cit. on p. 1).

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need.* 2017. arXiv: 1706.03762 [cs.CL] (cit. on pp. 1, 12–16, 34).

[9] Dave Steinkrau, Patrice Y. Simard, and Ian Buck. «Using GPUs for Machine Learning Algorithms». In: *Proceedings of the Eighth International Conference on Document Analysis and Recognition.* ICDAR '05. USA: IEEE Computer Society, 2005, pp. 1115–1119. ISBN: 0769524206. DOI: 10.1109/ICDAR.2005.251. URL: https://doi.org/10.1109/ICDAR.2005.251 (cit. on p. 2).

[10] Rajat Raina, Anand Madhavan, and Andrew Ng. «Large-scale deep unsupervised learning using graphics processors». In: vol. 382. Jan. 2009, p. 110. DOI: 10.1145/1553374.1553486 (cit. on p. 2).

[11] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. *Multi-column Deep Neural Networks for Image Classification.* 2012. arXiv: 1202.2745 [cs.CV] (cit. on p. 2).

[12] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. «Edge Computing: Vision and Challenges». In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198 (cit. on p. 2).

[13] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. «Edge Computing: Vision and Challenges». In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: 10.1109/JIOT.2016.2579198 (cit. on p. 2).

[14] Norman P. Jouppi et al. *In-Datacenter Performance Analysis of a Tensor Processing Unit.* 2017. arXiv: 1704.04760 [cs.AR] (cit. on pp. 2, 35).

[15] B. Hassibi, D.G. Stork, and G.J. Wolff. «Optimal Brain Surgeon and general network pruning». In: *IEEE International Conference on Neural Networks*. 1993, 293–299 vol.1. DOI: 10.1109/ICNN.1993.298572 (cit. on p. 3).

[16] Sheng Xu, Anran Huang, Lei Chen, and Baochang Zhang. «Convolutional Neural Network Pruning: A Survey». In: *2020 39th Chinese Control Conference (CCC)*. 2020, pp. 7458–7463. DOI: 10.23919/CCC50068.2020.9189610 (cit. on p. 3).

[17] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. *Pruning and Quantization for Deep Neural Network Acceleration: A Survey*. 2021. arXiv: 2101.09671 [cs.CV] (cit. on p. 3).

[18] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. *Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations*. 2016. arXiv: 1609.07061 [cs.NE] (cit. on p. 3).

[19] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. *A White Paper on Neural Network Quantization*. 2021. arXiv: 2106.08295 [cs.LG] (cit. on p. 3).

[20] Peter Mølgaard Sørensen, Bastian Epp, and Tobias May. «A depthwise separable convolutional neural network for keyword spotting on an embedded system». In: 2020.1 (June 2020). DOI: 10.1186/s13636-020-00176-2. URL: https://doi.org/10.1186/s13636-020-00176-2 (cit. on p. 3).

[21] Georgios Zervakis, Hassaan Saadat, Hussam Amrouch, Andreas Gerstlauer, Sri Parameswaran, and Jörg Henkel. «Approximate Computing for ML: State-of-the-Art, Challenges and Visions». In: *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. ASPDAC '21. Tokyo, Japan: Association for Computing Machinery, 2021, pp. 189–196. ISBN: 9781450379991. DOI: 10.1145/3394885.3431632. URL: https://doi.org/10.1145/3394885.3431632 (cit. on p. 3).

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «ImageNet: A large-scale hierarchical image database». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on p. 5).

[23] David Haussler. «Probably Approximately Correct Learning». In: *Proceedings of the Eighth National Conference on Artificial Intelligence.* AAAI Press, 1990, pp. 1101–1108 (cit. on p. 5).

[24] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: `1412.6980 [cs.LG]` (cit. on p. 12).

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* `http://www.deeplearningbook.org`. MIT Press, 2016 (cit. on p. 12).

[26] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.* 2021. arXiv: `2010.11929 [cs.CV]` (cit. on pp. 14, 16, 20, 34, 40, 42, 43).

[27] Stéphane d'Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. *ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases.* 2021. arXiv: `2103.10697 [cs.CV]` (cit. on pp. 14, 16, 22, 23, 49, 50).

[28] Martin Popel and Ondřej Bojar. «Training Tips for the Transformer Model». In: *The Prague Bulletin of Mathematical Linguistics* 110.1 (Apr. 2018), pp. 43–70. ISSN: 1804-0462. DOI: `10.2478/pralin-2018-0002`. URL: `http://dx.doi.org/10.2478/pralin-2018-0002` (cit. on pp. 14, 44).

[29] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate.* 2016. arXiv: `1409.0473 [cs.CL]` (cit. on p. 17).

[30] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate.* 2016. arXiv: `1409.0473 [cs.CL]` (cit. on p. 18).

[31] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. *Stand-Alone Self-Attention in Vision Models.* 2019. arXiv: `1906.05909 [cs.CV]` (cit. on p. 21).

[32] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. *On the Relationship between Self-Attention and Convolutional Layers.* 2020. arXiv: `1911.03584 [cs.LG]` (cit. on p. 21).

[33] Daniel Povey, Hossein Hadian, Pegah Ghahremani, Ke Li, and Sanjeev Khudanpur. «A Time-Restricted Self-Attention Layer for ASR». In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5874–5878. DOI: `10.1109/ICASSP.2018.8462497` (cit. on p. 22).

[34] `https://heartbeat.comet.ml/8-bit-quantization-and-tensorflow-lite-speeding-up-mobile-inference-with-low-precision-a882dfcafbbd`. [Online; accessed 5-Nov-2021]. 2021 (cit. on p. 24).

[35] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. *I-BERT: Integer-only BERT Quantization*. 2021. arXiv: `2101.01321 [cs.CL]` (cit. on pp. 25, 35, 46).

[36] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation*. 2013. arXiv: `1308.3432 [cs.LG]` (cit. on p. 26).

[37] Manfredo Atzori, Arjan Gijsberts, Claudio Castellini, Barbara Caputo, Anne-Gabrielle Mittaz Hager, Simone Elsig, Giorgio Giatsidis, Franco Bassetto, and Henning Müller. «Electromyography data for non-invasive naturally-controlled robotic hand prostheses». In: 1.1 (Dec. 2014). DOI: `10.1038/sdata.2014.53`. URL: `https://doi.org/10.1038/sdata.2014.53` (cit. on pp. 27, 31).

[38] Francesca Palermo, Matteo Cognolato, Arjan Gijsberts, Henning Müller, Barbara Caputo, and Manfredo Atzori. «Repeatability of grasp recognition for robotic hand prosthesis control based on sEMG data». In: *2017 International Conference on Rehabilitation Robotics (ICORR)*. 2017, pp. 1154–1159. DOI: `10.1109/ICORR.2017.8009405` (cit. on pp. 28, 29, 32).

[39] B. Hudgins, P. Parker, and R.N. Scott. «A new strategy for multifunction myoelectric control». In: *IEEE Transactions on Biomedical Engineering* 40.1 (1993), pp. 82–94. DOI: `10.1109/10.204774` (cit. on p. 31).

[40] Bojan Milosevic, Elisabetta Farella, and Simone Benatti. «Exploring Arm Posture and Temporal Variability in Myoelectric Hand Gesture Recognition». In: *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*. 2018, pp. 1032–1037. DOI: `10.1109/BIOROB.2018.8487838` (cit. on p. 32).

[41] Simone Benatti, Elisabetta Farella, Emanuele Gruppioni, and Luca Benini. «Analysis of Robust Implementation of an EMG Pattern Recognition Based Control». In: Mar. 2014 (cit. on p. 32).

[42] P Kaufmann, K Englehart, and M Platzner. «Fluctuating emg signals: Investigating long-term effects of pattern matching algorithms». In: IEEE, Aug. 2010. DOI: `10.1109/iembs.2010.5627288`. URL: `https://doi.org/10.1109/iembs.2010.5627288` (cit. on p. 32).

[43] Marcello Zanghieri, Simone Benatti, Alessio Burrello, Victor Kartsch, Francesco Conti, and Luca Benini. «Robust Real-Time Embedded EMG Recognition Framework Using Temporal Convolutional Networks on a Multicore IoT Processor». In: *IEEE Transactions on Biomedical Circuits and Systems* 14.2 (2020), pp. 244–256. DOI: `10.1109/TBCAS.2019.2959160` (cit. on pp. 33, 38, 49, 56, 62, 65).

[44] Ki-Hee Park and Seong-Whan Lee. «Movement intention decoding based on deep learning for multiuser myoelectric interfaces». In: *2016 4th International Winter Conference on Brain-Computer Interface (BCI)*. 2016, pp. 1–2. DOI: `10.1109/IWW-BCI.2016.7457459` (cit. on p. 32).

[45] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. *Efficient Transformers: A Survey*. 2020. arXiv: `2009.06732 [cs.LG]` (cit. on p. 35).

[46] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. «HAT: Hardware-Aware Transformers for Efficient Natural Language Processing». In: (2020). arXiv: `2005.14187 [cs.CL]` (cit. on p. 35).

[47] Hishan Parry, Lei Xun, Amin Sabet, Jia Bi, Jonathon Hare, and Geoff V. Merrett. «Dynamic Transformer for Efficient Machine Translation on Embedded Devices». In: (2021). arXiv: `2107.08199 [cs.CL]` (cit. on p. 35).

[48] Xavier Glorot and Yoshua Bengio. «Understanding the difficulty of training deep feedforward neural networks». In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 249–256. URL: `https://proceedings.mlr.press/v9/glorot10a.html` (cit. on p. 43).

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.* 2015. arXiv: `1502.01852 [cs.CV]` (cit. on p. 43).

[50] *Giustification for gain factor in Kaiming initialization.* `https://github.com/pytorch/pytorch/issues/57109#issuecomment-828847575`. [Online; accessed 10-Nov-2021]. 2021 (cit. on p. 43).

[51] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2019. arXiv: `1810.04805 [cs.CL]` (cit. on p. 46).

[52] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* 2019. arXiv: `1912.01703 [cs.LG]` (cit. on p. 55).

[53] `https://greenwaves-technologies.com/manuals/BUILD/HOME/html/index.html`. [Online; accessed 5-Nov-2021]. 2021 (cit. on p. 55).

[54] `https://greenwaves-technologies.com/product/gapuino/`. [Online; accessed 5-Nov-2021]. 2021 (cit. on p. 55).

Acknowledgment: Computational resources were provided by HPC@POLITO, a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (`http://www.hpc.polito.it`).