IADIS International Journal on Computer Science and Information Systems
Vol. 19, No. 1, pp. 1-14
ISSN: 1646-3692

ANALYSIS OF EMBEDDED GPU ARCHITECTURES FOR AI IN NEUROMUSCULAR APPLICATIONS

Simon Pfenning[1], Raul C. Sîmpetru[2], Niklas Pollak[2], Alessandro Del Vecchio[2], and Dietmar Fey[1]
[1]Chair of Computer Architecture, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
[2]Neuromuscular Physiology and Neural Interfacing Laboratory, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany

ABSTRACT

The advancements in deep neural network design have led to a significant increase in the possibilities and functioning of AI-assisted medical hardware. To make use of this progress in the field of mobile applications or even as wearable devices, a suitable hardware-software ecosystem must be identified to meet the high computation and memory demands of neural networks with minimal energy consumption. In this paper, we analyze an up-to-date heterogenous embedded platform employing a deep convolutional network for hand position recognition through electromyography signals. Our evaluation aimed to determine the optimization efforts required for the architecture to function as a human wearable device and identify the most suitable accelerators on the given platform for this task.

KEYWORDS

EMG, GPU, Embedded Hardware, Deep Learning, Prosthesis, AI

1. INTRODUCTION

Hand position recognition using electromyography (EMG) signals has become an increasingly popular area of research in recent years (Liu et al., 2021; Saif et al., 2022; Rahimian et al., 2022; Sîmpetru et al., 2022, 2023; Chen et al., 2023).

This technology has a variety of potential applications, including prosthetic devices, human-machine interfaces, and robotics. Deep learning algorithms have been shown to deliver reliable results in classifying EMG signals into different hand gestures (Liu et al., 2021; Rahimian et al., 2022). Compared to classical pattern recognition approaches such as support vector machines (SVM) or random forests, deep neural networks (DNN) are far less time consuming in design and achieve a better mapping of the complex and non-linear relationships between EMG signals and the hand positions. The convolutional neural network (CNN) (Goodfellow et al., 2016) which we used in this paper, is trained on regressing not only specific gestures but estimating the position of the entire hand beginning from the wrist. On a high-performance computing (HPC) system equipped with capable hardware, this network is able to deliver precise results in real-time. For the application of this algorithm in the field however, there are still some major challenges remaining, when deploying deep learning models on resource-constrained devices, such as embedded systems. Especially for the use in human wearable systems the limited power budget and size of the hardware leads to a drastic restriction of its computational capabilities. On the other hand, in particular CNNs, which are commonly used for hand position recognition, are both computationally and memory demanding to deliver accurate estimations in real-time. For this reason, embedded systems designed with executing the inference of such algorithms in mind, often come with adapted hardware architectures. Those application specific accelerators can improve the duration of the network operations most efficiently (Reuther et al., 2020). Consequently, in order to make use of this specific design, a highly optimized implementation of CNNs on embedded systems is a crucial precondition to achieve desirable results. Our objective was to find an alternative hardware solution to the current existing HPC platform, equipped with an RTX 3090 graphics card.

The device should be able to fulfill the needs of the high resource demand of the neural network, while staying very efficient at the same time. For these reasons we decided to evaluate the algorithm on NVidia's current embedded Jetson platform, which still delivers a decent amount of computational power and memory size. In order to see how the algorithm behaves when optimizing for embedded systems we investigated the real-time capability and energy consumption of the resulting implementation. The embedded device contains a small graphics processing unit (GPU) as well as two application specific deep learning accelerator (DLA) cores. The proposed system has significant potential advantages over traditional CPU focused architectures. General purpose GPUs (GPGPUs) are highly parallel architectures and have a high memory bandwidth, making them well suited for deep learning inference workloads (Kayid et al., 2018). The implemented DLA cores, which are designed specifically for accelerating common operations of CNNs, such as convolutions, activation functions, or pooling are a promising additional feature for increasing the energy efficiency of the system (Zhou et al., 2018).

To obtain reasonable results, the original network as described in (Sîmpetru et al., 2023) is modified to make use of the embedded hardware's specific architectural design. For this purpose, we describe multiple different approaches for optimization. We give insights on how the work is dispatched on the hardware and which benefits can be obtained by the conducted steps. Afterwards we then evaluate the suitability of the implemented system with regard to be mounted directly on the human body for different use cases, including orthotic and prosthetic devices. Both scenarios are extremely challenging, as there is very strict energy- and size-limitations for a comfortable application close to the human being. Finally, a proposal is given on whether the current system is sufficient, or if more optimization work needs to be done. This can be achieved by either further improving the software implementation, or by the digital design of a specific hardware architecture, for example as an embedded FPGA solution.

The remainder of the paper is structured as follows. In chapter 2 we explain the DNN that controls the joints of a hand. Chapter 3 describes the embedded GPU hardware we investigated for a future smaller and a wearable hardware solution than the original PC based one. In chapter

4 we describe in detail the different optimizations methods we used for the GPU hardware to evaluate in chapter 5 the GPU accelerator cores concerning their usefulness for an embedded AI neuromuscular controller. Finally, we conclude and summarize our paper in chapter 6.

2. DEEP CNN FOR EMG REGRESSION

2.1 Experimental Protocol

We previously collected surface EMG (sEMG) data from nine subjects performing three hand gestures (pointing, peace sign, and rock sign) as well as a multitude of dynamic movements (flexing and extending of the fingers and opening and closing of the fist) in sync with their hand kinematics (Sîmpetru et al., 2023). The sEMG was acquired from 5 electrode grids (8 rows x 8 columns, 10-mm interelectrode distance; OT Bioelettronica, Turin, Italy). Figure 1 shows the positioning of the electrodes. Three high-density EMG grids were placed around the thickest part of the forearm and two around the wrist, proximal to the ulnar head. Before placing the electrode grids, the skin was shaved and cleaned with an alcoholic solution. We recorded a total of 320 monopolar sEMG signals using a multichannel amplifier (EMG-Quattrocento, A/D converted to 16 bits; OT Bioelettronica, Turin, Italy), amplified (×150) and band-pass filtered (0.7–500 Hz) at source. The signals were sampled at 2048 Hz and captured during the movement execution using a custom script written in Python.

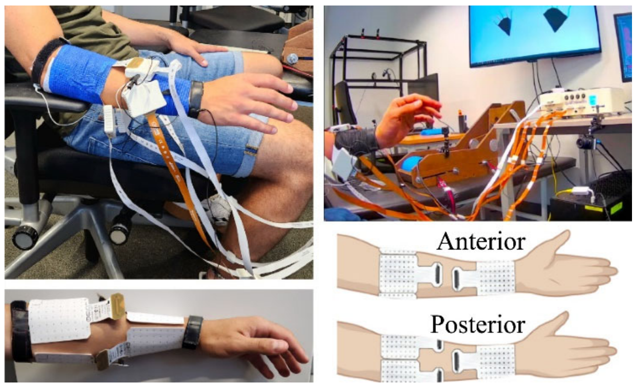


Figure 1. Experimental setup taken from Sîmpetru et al., 2023. The subjects have 5 electrode grids attached to their forearm. Each electrode grid has 64 electrodes resulting in a total of 320 EMG signals (reused with permission from Sîmpetru et al.)

We recorded 30 seconds per movement, resulting in a total of 12 minutes of data for training a previously published network (Sîmpetru et al., 2022) from zero that we have updated for real-time capabilities (Sîmpetru et al., 2023).

2.2 EMG Preprocessing

In order to provide an embodied feeling for the subjects during real-time testing we set the EMG multichannel amplifier to provide the sEMG signals as 32 non-overlapping segments per second. This allowed our system to make a prediction every 31.25ms resulting in smooth movement intent prediction. Ideally, prediction works only based on the neurophysiological information contained in a single sEMG segment, but we have found empirically that the temporal information contained in a single segment is not sufficient. To collect enough temporal information, we create a queue containing 3 such segments, which gives 93.75ms of information. The queue is always shifted by one segment to ensure a smooth prediction of movement.

We further augment the 93.75ms sEMG with its 20Hz low-passed version to provide a less variable view of the neurophysiological information. The low-pass filter was a fourth order Butterworth filter, which we used to filter forward and backward to achieve phase-free filtering.

2.3 Model Architecture

The model (Figure 2) for motion intention detection has already been described in detail in Sîmpetru et al. (Sîmpetru et al., 2023) thus we only provide information necessary for understanding the computational necessity of the individual layers. Detailed theoretical justifications for the layers can be found in the above-mentioned paper.

The network consists of two parts: a CNN, which maps the sEMG signals into a high-dimensional latent space, and a subsequent multilayer perceptron (MLP) (Goodfellow et al., 2016), which transforms the latent space into Cartesian space.

The input of the network is a 3D tensor that has the following shape:

- height: 320 electrodes,
- width: 93.75ms or 192 samples temporal information,
- depth: two used to index the low-pass filtered and unfiltered EMG signals.

In total one input tensor consists of 61442 floating point values. The first step of the network is to reshape the input by dividing the 320 electrodes into the five grids they correspond to, creating a 4D tensor of the form 2 x 5 x 64 x 192. This reshaped tensor is then standardized grid-wise.

The CNN operations described in chronological order are:

- The first 3D CNN layer contained 128 channels and ran a time-wise kernel of size 32 with a stride of 8.
- Then, the output of the first layer is circularly padded by 2 in the grid dimension (from 5 to 9) and 16 in the electrode dimension (from 64 to 96). On the padded tensor we ran a 5 x 32 x 18 kernel that integrates the electrode wise only information into cross-grid patterns. To reduce computational complexity, the kernel is dilated in the electrode dimension by a factor of two. The layer had 32 channels.
- The last layer also had 32 channels and was used to further extract information from the previously generated patterns by applying a 5 x 9 x 1 kernel.

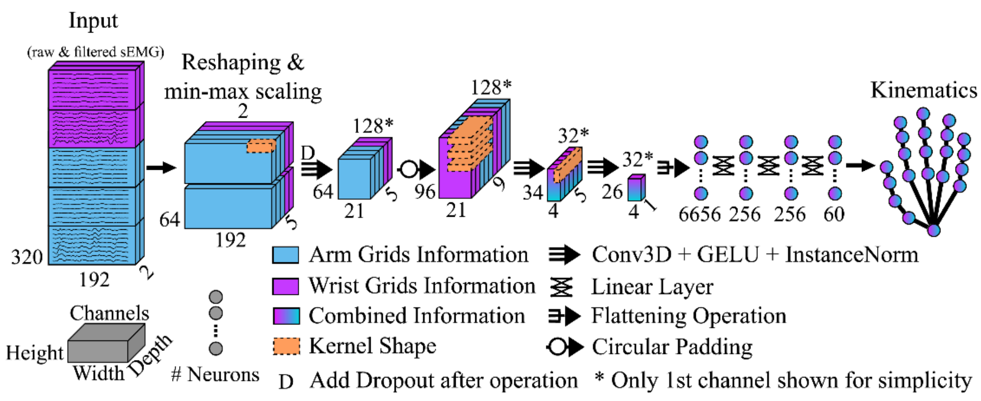


Figure 2. Model architecture overview taken from Sîmpetru et al., 2023. The hand movement intention detection model takes 320 EMG electrode signals and outputs the hand position in Cartesian space as 60D vector corresponding to the x, y, and z values of 20 joints. The network consists of two parts: a CNN and a multilayer perceptron (MLP) (Goodfellow et al., 2016). The CNN maps the EMG signals into a high-dimensional latent space, which is then converted into kinematics using the MLP (reused with permission from Sîmpetru et al.)

Each layer mentioned is followed by an Instance Normalization layer (Ulyanov et al., 2017) and the activation function GELU (Hendrycks & Gimpel, 2020). The output of the CNN is flattened and passed through a three layered MLP with sizes of 6656, 128, and 128, respectively. The last layer of the network outputs a 60D vector that represents the twenty hand joint coordinates in Cartesian space (x, y, and z).

3. HARDWARE OVERVIEW

For the implementation of the embedded inference system, we decided to use two platforms from Nvidia, the Xavier AGX and Xavier NX. Both being built upon the same Tegra Xavier architecture, but with a different configuration of the integrated CPU- and GPU-cores as well as the usable memory bandwidth. Xavier AGX has 33 % more GPU shaders and a memory interface double the size. So, in theory it should be able to deliver more computational performance but at the cost of a higher power consumption of 30 Watt instead of 15 Watt. Even though these numbers sound quite high for an embedded system, it is already a significant improvement to the previous system using an RTX 3090 with more than 300W.

The two DLA cores are identical and directly attached to the main memory interface. In the evaluation section we will investigate if the differences between both hardware configurations makes for a better sweet spot regarding the entire efficiency of the system. In the next two subsections we will give a brief overview on the respective architectures of the accelerator cores as well as where their strengths reside.

3.1 GPU Architecture

In recent years GPUs have proven themselves repeatedly to be well suited for the usage in Deep Learning (Kayid et al., 2018). Their properties as streaming processors make them an ideal choice for processing graph-based algorithms such as neural networks. Both Xavier SoCs contain GPUs, which are based on NVidias Volta design. This architecture introduced a couple of new features specifically designed for the improved execution of neural network algorithms in both training and inference.

1) Quantization: A decisive technology that had an incredibly significant impact on the efficiency of executing deep neural networks on embedded architectures is the improved acceleration of values quantized in a lower bit width, for example FP16 or INT8, instead of FP32 (Choukroun et al., 2019). The use of lower precision data types for both activations and weights has had a twofold positive impact. First, the same compute units can perform double or quadruple the number of calculations at the same time and with the same necessary energy. Second, the memory footprint of the network can be shrunk to an excessive degree, which makes up for the smaller amount of memory available in embedded devices. Even though this leads to a certain drop in the accuracy of the inferred results, many deep learning algorithms have proven to be very robust against more aggressive quantization due to the under determined character. We will examine the influence of quantization on the accuracy of our network in the evaluation section.

2) Tensor Cores: The most prominent improvement of NVidia's Volta GPU family is the introduction of so-called Tensor Cores (Choquette et al., 2019). Tensor Cores are vector processing units, based on the design principles of systolic arrays (Chowdhury et al., 2020). They are able to conduct a 4x4 mixed-precision matrix-matrix multiplication and addition per clock-cycle. For each 64 scalar compute units there are eight Tensor Cores, effectively increasing the number of peak calculations per cycle by a factor of five. This special type of compute units is greatly beneficial for the deployment of neural networks, especially CNNs. The most common components for these types of networks are convolutional and fully connected layers, which can be easily mapped onto matrix-matrix multiply accumulate (MAC) operations and thus being very efficiently computed by the tensor cores.

3.2 DLA Cores

Both platforms also offer two NVDLA accelerators, which are designed specifically for improving the inference run of convolutional networks. Every DLA core contains five optimized blocks for conducting certain types of convolutions, activation functions, pooling, normalization and reshape (Zhou et al., 2018). During runtime it is possible to use each of the functional units separately, such that independent parts of the network can be executed in parallel. Or a pipelined approach can be adopted where small chunks of consecutive network-layers are mapped onto the respective functional units. The advantage of this procedure consists in the more effective data transfers between consecutive layers of the network, which can now be passed from one execution unit to the next by register transfers.

In the present case, the implementation is realized as a headless core. This implies that the configuration and control of a DLA core is handled by the CPU via a kernel driver. The reason for this kind of design is power saving because an additional control unit and an SRAM memory can be saved. On the other hand, the DLA's function units now need to directly access the slow main memory, because the cores do not offer their own L2 cache. This has the potential of a huge negative impact in cases where a pipelined execution of the network is not easily feasible, because every unit has to access the slower main memory for obtaining its data separately. In Chapter 5 we give an evaluation of how well the execution of the given network was improved by these cores.

4. OPTIMIZATION METHODS

In order to get the most out of the hardware we decided to firstly conduct the following optimization steps, to obtain an overview of the achievable results and to see which way turns out to be most promising.

4.1 Network Structure

The first steps towards achieving a better runtime are hardware agnostic methods that optimize the generic mathematical operations in the network. Traditionally the computation of neural networks is executed layer-by-layer, requiring intermediate results to be stored in memory and transferred between layers. This process incurs additional memory accesses and computational overhead, resulting in increased latency and reduced overall efficiency. As a countermeasure we use NVidia's model-compiler TensorRT, to modify the network structure. In the process unnecessary operations are eliminated from the compute graph, for example dropout layers, which are only relevant during training. Furthermore, it fuses the operations of sequential or adjacent layers into a single kernel. This fusion eliminates the need for intermediate memory transfers, redundant computations, and the number of kernel calls. For our network model the predominant fused layer types are convolutions and activations.

4.2 Quantization

The second measure is already more hardware oriented, since it exploits the ability of the embedded GPU to compute on smaller data types in a vectorized way (Choukroun et al., 2019). In this manner, it is possible for the regular scalar processing unit to achieve twice as high a computing throughput. Another particularly key factor to note here are the aforementioned Tensor Cores, which are implemented within NVidia's Volta architecture and only allow for a data width of 16bit for matrix-matrix multiplication. This is the reason why we chose 16bit as a first step for quantizing the weights and activations of the neural network. Because on the one hand we lose a minimal amount of information, but on the other hand we do not only gain twice the computational performance, but a factor of more than seven. This is because the scalar units on Xavier NX delivering a peak single precision computational power of roughly 850GFLOPS and the 48 Tensor Cores having a total of 6.1 TFLOPS. For the quantization procedure there are two possible implementations, either during the training phase, or afterwards. The major benefit of so-called quantization-aware training is a reduced loss in precision of the estimation results of the network. For our case we decided to stick with post training quantization, because our goal was to optimize an already trained network onto embedded hardware without the necessity of retraining. In the evaluation, we will address the impact of this decision when considering the loss of accuracy compared to the original model.

4.3 DLA

The final stage of our optimization pipeline is represented by the integration of the dedicated deep learning accelerator cores. As mentioned before the DLA implementation on the Jetson boards is headless, which means they do not have a dedicated cache and all memory transfers need to be conducted directly to the slow main memory. For this reason, we decided to opt for the pipelined programming option of the cores as extensively as possible to keep the number of accesses to the main memory reasonably low. Memory transfers already are a burden since the DLA cores alone are not able to solve all the network's layers. This leads to an additional continuous data traffic between DLA cores and GPU, which needs to be handled via the main memory. Because of this, our primary intention of making use of the accelerator cores was not necessarily an improvement in execution time, but rather to achieve a significant advance in energy efficient computing.

5. EVALUATION

For the overall evaluation we compare the three optimized variants (A to C in chapter 4) on both Jetson boards with each other and against the original unoptimized network runtime, as well as the formerly used desktop GPU RTX 3090. To be able to classify the results correctly, it must first be determined which threshold value the inference time must meet to be considered real-time capable. It is not easy to establish an adequate threshold for latency since a prosthetic hand that can be considered a complete replacement has not yet been implemented. As the current frequency of input data from the sensors is a maximum of 32Hz, we assume that inference within this time frame can be tentatively assumed to be real-time capable. If in the future this threshold should decrease, due to improvements in signal transmission, the estimation of real-time capability in this paper might need to be reconsidered.

5.1 Inference Latency

Before we get more into detail, we first want to provide an overview of the inference latency (Figure 3), to obtain a first impression of which implementation yields a good value. As expected, the desktop GPU delivers the best results by far and is always able to stay below the real-time threshold, but at the cost of a twenty-fold increase in power consumption (360W).

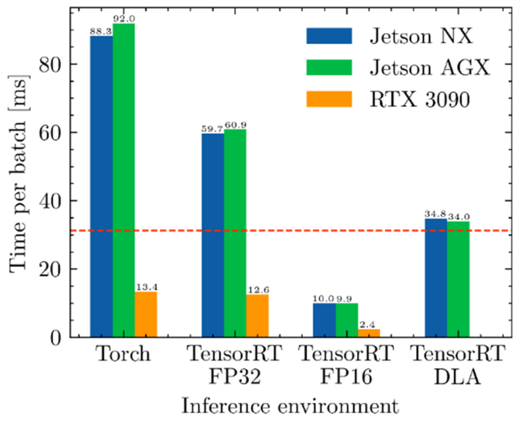


Figure 3. We observe 33% performance increase by the adaptation of the network structure and another 6x increase due to quantization with FP16 precision. However, the DLA implementation which suffers from memory transfers is substantially slower, so that it cannot keep up with the signal frequency from the sensors

From the original Torch-model we can already obtain a 33% lower latency by reducing the operations of the networks compute graph with the help of a TensorRT engine. This is without losing any information. When applying the quantization to FP16 the GPU's Tensor Cores come into play. This can be observed in the graph as a decrease in the latency by a factor of six, which is just the amount in peak computational performance delivered by the Tensor Cores. With this boost, the quantized inference time is easily below the real-time threshold. The DLA implementation, however, suffers a lot from the additional memory transfers and is not able to keep up even though it also uses weights and activations of FP16 precision. As we will see later on, the problem arises from the layer structure of the network, which is also responsible for the indistinguishable performance between the two Jetson boards, despite their different hardware capabilities.

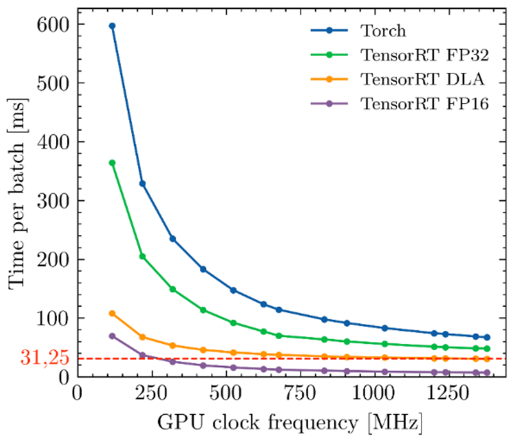


Figure 4. The sweet spot can be found for FP16 at 300MHz. The DLA implementation converges against the real-time threshold but does not reach it

In order to find the sweet spot for real-time execution of the inference, the frequency of the processing units is now varied for every iteration. (Figure 4). The only optimized model available to reach well below the real-time threshold is the FP16 GPU implementation. For this a GPU core frequency of 300 MHz is sufficient. The DLA implementation, even though it also works with 16bit quantized parameters, cannot achieve the same performance. We will see later on why this is the case.

5.2 Quantization Error

With the improvements in runtime due to quantization, a certain error is likewise introduced into the estimation of finger joint positions. Before we start talking about the effects of the quantization induced error, we first need to define how to measure it. For this we calculate the Euclidean distance between the estimated 3D-coordinate of the *i* joints of the quantized network to the original one.

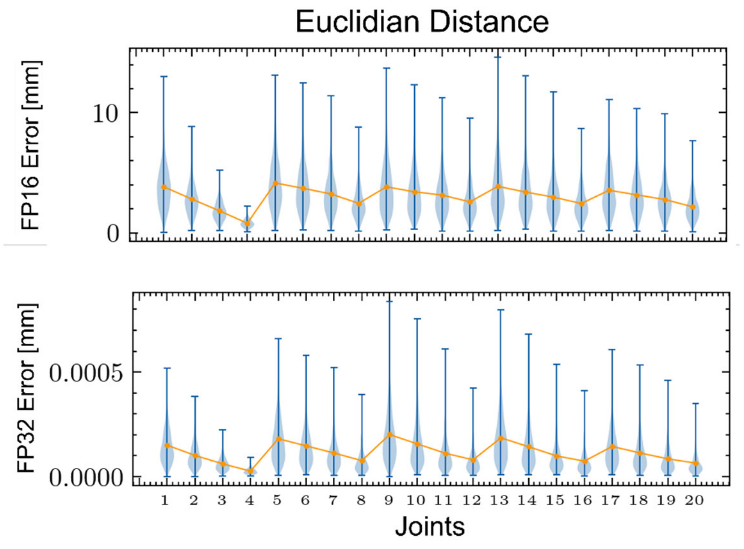


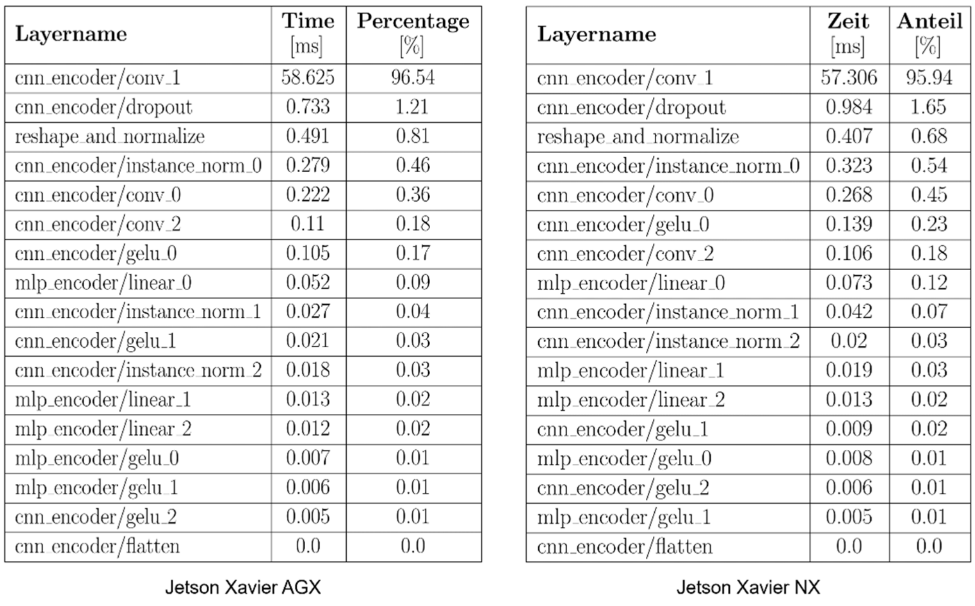
Figure 5. The quantization introduced error is significant but still acceptable, 2mm average per finger joint

This error already has relatively high quantity going from an average of 0.001mm to 2mm in the FP16 model (Figure 5). Nevertheless, it is quite acceptable for the early stage of development, but for the future a solution based on quantization-aware training should be aimed at. Especially considering another reduction in parameter size to 8bit for further improving on the inference efficiency.

$$Euclidean\ Distance = \sqrt{\sum_{i=1}^{n}(quantized_i - original_i)^2}$$

5.3 Layer-wise Latency Analysis

In Figure 3 we detected that Xavier AGX, even though having the substantially more powerful GPU, could not deliver up to this promise compared to its smaller counterpart Xavier NX. Since this seems very counter-intuitive at first sight, we measured the execution time for each layer of the network, to identify any bottlenecks. It appears that layer conv_1 of size 5 × 32 × 18 has by far the most impact, since it amounts to over 90% of an entire inference run (Figure 6). The dimensionality of the convolutional layer leads to the same mapping of operations onto the processing units of the GPUs on both platforms. For this reason, Xavier AGX cannot gain any profit from its additional two GPU compute clusters, because they do not get used in this situation. Another mapping of the operations onto more compute clusters is possible but will also lead to a more inefficient implementation inside a single cluster and eventually lead to an even worse result. For a better utilization of the larger GPU, a redesign of the layer proportions or a split into two consecutive layers are the most promising alternatives to get rid of the bottleneck.



Layername	Time [ms]	Percentage [%]
cnn_encoder/conv_1	58.625	96.54
cnn_encoder/dropout	0.733	1.21
reshape_and_normalize	0.491	0.81
cnn_encoder/instance_norm_0	0.279	0.46
cnn_encoder/conv_0	0.222	0.36
cnn_encoder/conv_2	0.11	0.18
cnn_encoder/gelu_0	0.105	0.17
mlp_encoder/linear_0	0.052	0.09
cnn_encoder/instance_norm_1	0.027	0.04
cnn_encoder/gelu_1	0.021	0.03
cnn_encoder/instance_norm_2	0.018	0.03
mlp_encoder/linear_1	0.013	0.02
mlp_encoder/linear_2	0.012	0.02
mlp_encoder/gelu_0	0.007	0.01
mlp_encoder/gelu_1	0.006	0.01
cnn_encoder/gelu_2	0.005	0.01
cnn_encoder/flatten	0.0	0.0

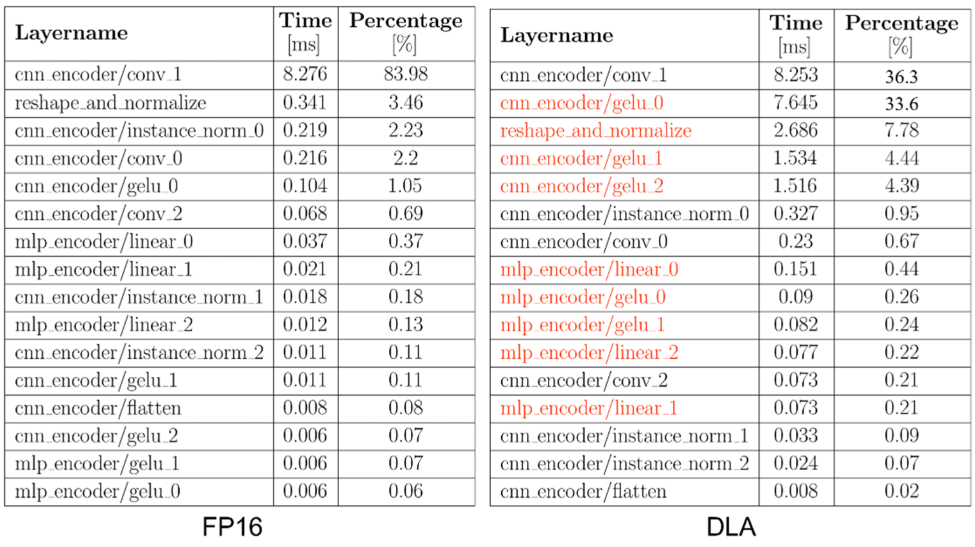
Jetson Xavier AGX

Layername	Zeit [ms]	Anteil [%]
cnn_encoder/conv_1	57.306	95.94
cnn_encoder/dropout	0.984	1.65
reshape_and_normalize	0.407	0.68
cnn_encoder/instance_norm_0	0.323	0.54
cnn_encoder/conv_0	0.268	0.45
cnn_encoder/gelu_0	0.139	0.23
cnn_encoder/conv_2	0.106	0.18
mlp_encoder/linear_0	0.073	0.12
cnn_encoder/instance_norm_1	0.042	0.07
cnn_encoder/instance_norm_2	0.02	0.03
mlp_encoder/linear_1	0.019	0.03
mlp_encoder/linear_2	0.013	0.02
cnn_encoder/gelu_1	0.009	0.02
mlp_encoder/gelu_0	0.008	0.01
cnn_encoder/gelu_2	0.006	0.01
mlp_encoder/gelu_1	0.005	0.01
cnn_encoder/flatten	0.0	0.0

Jetson Xavier NX

Figure 6. The network layers execution time for TensorRT FP32 configuration, sorted by their impact on the execution time. Conv_1 has the highest impact on the inference runtime overall

For the FP16 and DLA implementation we can observe a similar behavior. However, since NVidia's DLA compute units do not yet support the 3D convolutional layers, they are not offloaded to them but stay on the GPU instead. The consecutively following layers can be processed by the deep learning accelerators, but at the cost of a higher number of main memory transfers due to the pipelined principle of the architecture not in use. Even though these results are not too bad, in the future we will investigate how well the 3D convolutions can be replaced by their 2D counterparts in order to make better use of the accelerator cores.



Layername	Time [ms]	Percentage [%]
cnn_encoder/conv_1	8.276	83.98
reshape_and_normalize	0.341	3.46
cnn_encoder/instance_norm_0	0.219	2.23
cnn_encoder/conv_0	0.216	2.2
cnn_encoder/gelu_0	0.104	1.05
cnn_encoder/conv_2	0.068	0.69
mlp_encoder/linear_0	0.037	0.37
mlp_encoder/linear_1	0.021	0.21
cnn_encoder/instance_norm_1	0.018	0.18
mlp_encoder/linear_2	0.012	0.13
cnn_encoder/instance_norm_2	0.011	0.11
cnn_encoder/gelu_1	0.011	0.11
cnn_encoder/flatten	0.008	0.08
cnn_encoder/gelu_2	0.006	0.07
mlp_encoder/gelu_1	0.006	0.07
mlp_encoder/gelu_0	0.006	0.06

FP16

Layername	Time [ms]	Percentage [%]
cnn_encoder/conv_1	8.253	36.3
cnn_encoder/gelu_0	7.645	33.6
reshape_and_normalize	2.686	7.78
cnn_encoder/gelu_1	1.534	4.44
cnn_encoder/gelu_2	1.516	4.39
cnn_encoder/instance_norm_0	0.327	0.95
cnn_encoder/conv_0	0.23	0.67
mlp_encoder/linear_0	0.151	0.44
mlp_encoder/gelu_0	0.09	0.26
mlp_encoder/gelu_1	0.082	0.24
mlp_encoder/linear_2	0.077	0.22
cnn_encoder/conv_2	0.073	0.21
mlp_encoder/linear_1	0.073	0.21
cnn_encoder/instance_norm_1	0.033	0.09
cnn_encoder/instance_norm_2	0.024	0.07
cnn_encoder/flatten	0.008	0.02

DLA

Figure 7. The individual layers of the network for the GPU FP16 and DLA configurations, sorted by their respective impact on the execution time. The layers marked in red are executed by the DLA cores. However, the computation-heavy 3D convolutional layers need to be assigned to the GPU. This also leads to a lot of memory transfers because of the DLAs not working in pipelined mode.

5.4 Power Consumption

Power efficiency is one of the major points of interest, when it comes to designing embedded systems. For this part we neglected the differentiation between the Jetson boards, since both of them behave remarkably similar when running our network. At the sweet spot of 300MHz the quantized GPU-only implementation has a power consumption of roughly 7W. Especially for lower clock speeds the GPU appears to be the more efficient choice, at least regarding our FP16 model. When it comes to higher frequencies, the additionally used DLA cores start to pay off and the system gets a lot more power efficient.

So, chances are good that with a few more adaptations, such as the exchange from 3D to 2D convolutions and further 8bit quantization-aware training, the already achieved results will be significantly surpassed in the near future.

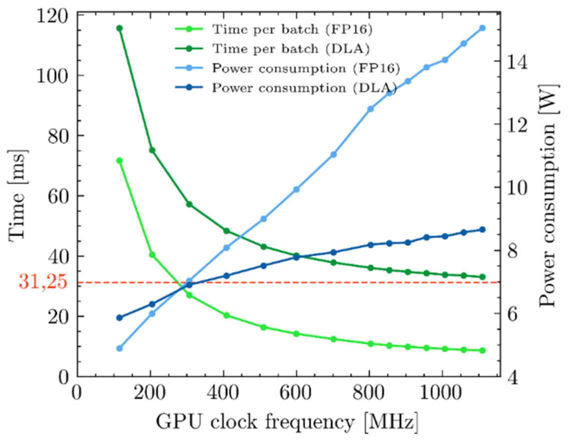


Figure 8. The GPU implementation at sweet spot of 300MHz consumes roughly 7W. The DLA core implementation shines at higher clock speeds with a better energy efficiency below 9Watt

6. CONCLUSION

With our rapid prototyped GPU/DLA implementation, we were already able to obtain decent results. A power consumption of below 10 Watt is a great achievement, compared to the desktop GPU with more than the twenty-fold energy expense. However, to be of reasonable use in a portable device, it is still far too high. Therefore, our future efforts from now on will be in two directions. Further tuning of the existing GPU/DLA design for even higher efficiency, to quickly examine how robust the neural network behaves on more severe measures and further optimization techniques. For the development of a future portable device, we are also striving for a custom digital design around an FPGA. We anticipate that this will allow for a reduction of the power consumption for the entire system to less than 1 Watt. Which in turn would allow us to dispense with the bulky cooling system, as well as significantly extend battery operation. To ease our efforts on the digital circuit design, we can draw on preliminary work already available at our institute. For an effective framework on implementing systolic DNN accelerators, we rely on the work achieved by (Knödtel et al., 2020). The mapping of the entire network on hardware, will be based on the process described in (Pfenning et al., 2021). It uses a high-level Python implementation of a Neural Network in TensorFlow and executes the inference tasks according to the principles described in (Holzinger & Reichenbach, 2021). In this way, we expect to be able to quickly achieve decent results on low-power FPGA SoCs, such as Xilinx Zynq Ultrascale+ family.

REFERENCES

Chen, C. et al. (Jun. 2023). Real-Time Hand Gesture Recognition by Decoding Motor Unit Discharges Across Multiple Motor Tasks from Surface Electromyography. *IEEE Transactions on Biomedical Engineering, pp. 1-11*

Choquette, J. et al. (Apr. 2018). Volta: Performance and programmability. IEEE Micro, vol. 38, no. 2. pp. 42–52

Choukroun, Y. et al. (Oct. 2019). Low-bit quantization of neural networks for efficient inference. pp. 3009–3018.

Chowdhury, R. et al. (Jul. 2020). A computational model for tensor core units. 32nd ACM Symposium on Parallelism in Algorithms and Architectures. p. 519–521. Available: https://doi.org/10.1145/3350755.3400252

Goodfellow, I. et al. (2016). Deep Learning. MIT Press. Available: https://www.deeplearningbook.org/

Hendrycks, D., & Gimpel, K. (Jul. 2020). Gaussian Error Linear Units (GELUs). arXiv:1606.08415. Available: http://arxiv.org/abs/1606.08415

Holzinger, P. , & Reichenbach, M. (Oct. 2021). The HERA methodology: Reconfigurable logic in general-purpose computing. IEEE Access, vol. 9. pp.147 212–147 236

Kayid, A. et al. (May 2018). Performance of cpus/gpus for deep learning workloads.

Knödtel, J. et al. (Sep. 2020). A model-to-circuit compiler for evaluation of DNN accelerators based on systolic arrays and multibit emerging memories. 9th International Conference on Modern Circuits and Systems Technologies (MOCAST). pp. 1–6.

Liu Y. et al. (Apr. 2021). NeuroPose: 3D Hand Pose Tracking using EMG Wearables. Web Conference, ser. WWW ’21. New York, NY, USA: Association for Computing Machinery. Available: https://doi.org/10.1145/3442381.3449890

Pfenning, S. et al. (Jan. 2021). Transparent FPGA Acceleration with TensorFlow. DATE Friday Workshop on Systemlevel design Methods for Deep Learning on Heterogeneous Architectures (SLOHA 2021)

Rahimian, E. et al. (May 2022). Hand Gesture Recognition Using Temporal Convolutions and Attention Mechanism. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1196–1200.

Reuther, A. et al. (Sep. 2020). Survey of machine learning accelerators. *IEEE High Performance Extreme Computing Conference (HPEC). pp. 1-12. doi:* 10.1109/HPEC43674.2020.9286149.

Saif, R. et al. (Sep. 2022). Multi-Channel EMG Signal analysis for Italian Sign Language Interpretation. International Conference on Emerging Trends in Smart Technologies (ICETST). pp. 1–5.

Sîmpetru, R. C. et al. (Jul. 2022). Accurate Continuous Prediction of 14 Degrees of Freedom of the Hand from Myoelectrical Signals through Convolutive Deep Learning. 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC). pp. 702–706.

Sîmpetru, R. C. et al. (Jul. 2023). Proportional and Simultaneous Real-Time Control of the Full Human Hand from High-Density Electromyography. *IEEE Transactions on Neural Systems and Rehabilitation Engineering (*TNSRE). doi: 10.1109/TNSRE.2023.3295060

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (Nov. 2017). Instance Normalization: The Missing Ingredient for Fast Stylization. arXiv:1607.08022 [cs]. Available: http://arxiv.org/abs/1607.08022

Zhou, G. et al. (Nov. 2018). Research on NVidia deep learning accelerator. 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID). pp. 192–195.