# AdderNet and its Minimalist Hardware Design for Energy-Efficient Artificial Intelligence

*Seminar Embedded Systems SS 2021*

Jannik Hofmann
Lehrstuhl für Technische Informatik
Wilhelm-Schickard-Institut
Eberhard Karls Universität Tübingen
Email: jannik.hofmann@student.uni-tuebingen.de

***Abstract*—This paper will highlight the drawbacks of multiplication operations in convolution layers of traditional CNNs, mainly the computational resource demands, and discuss different approaches to building networks without these multiplication operations. Next up, it presents the adder kernel, which calculates the similarity of the kernel weights with the corresponding window of input values. This kernel can be easily implemented in hardware on an FPGA using quantization and parallelization techniques. Compared to alternative kernel designs, AdderNet provides significant savings in energy consumption and logic resource utilization while producing competitive, sometimes even more accurate, results compared to traditional CNNs.**

## I. Introduction

In the past several years, artificial intelligence, and specifically machine learning, has been playing an increasingly important role in the advent of new technologies in a wide variety of sectors [1], [9]. The broad field of machine learning includes image classification, autonomous driving, cybersecurity, medical imaging, media generation, and natural language processing, among countless others [1].

### A. Motivation

Many of these applications rely on computer vision based on AI (artificial intelligence) [1]. In this discipline, a neural network analyzes input images (sometimes from a video feed), deducting higher-level semantic information from the raw pixel data. Although intuitively simple for us humans, this is a challenging task for computer algorithms. Not only in the sense of requiring vast amounts of training data until the network produces acceptable results but also in the computational resources needed for training and prediction [4].

This demand is usually not an issue for professional computers with a dedicated graphics card. However, embedded systems usually have weaker hardware without graphics cards and low power consumption requirements. So to use a traditional computer vision network on such an embedded system, the computationally intensive calculations on weaker hardware would result in much longer calculation delays and high energy consumption nonetheless (even if the calculation takes place over a longer period). Due to these requirements, embedded systems benefit from neural network architectures executable on such systems or designed explicitly for such hardware.

## II. Convolutional neural networks

Before discussing various techniques to avoid computationally expensive calculations, this paper reviews the traditional architecture of CNNs.

Most AI-based computer vision algorithms (as well as many speech recognition and signal processing solutions) utilize convolutional neural networks (CNNs), trained by supervised learning from a given dataset [4]. A CNN is a type of deep neural network (DNN) that contains convolutional layers [6]. Most standard DNNs, in turn, consist of millions or even billions of interconnected neurons, divided into various layers [10], [8]. However, specific architectures and varying approaches can result in a different composition with significantly fewer neurons [6].

### A. Dimensionality reduction

A crucial ability of convolutional layers is to reduce the dimensionality of incoming data, condensing the vast amount of information into less data with highly useful information [6]. In computer vision tasks, that means it aids in extracting meaningful information from images, producing semantic interpretations from an enormous amount of data, and reducing the number of neurons in the process [4].

Convolutional layers are responsible for low-level features like edge-detection, which facilitates a higher-level understanding of image features later in the network [4].

This dimensionality reduction is significant for image and video processing, as these data types contain many pixels of individual (usually color-) values, with high levels of redundancy. This amount of highly redundant information in the data can be demonstrated by slightly changing many pixel values or completely removing pixels from the image and letting a human assess the changed image. If the changes are not too extreme, they can still understand the image content without problems. However, if the changes are minimal, humans cannot distinguish the changed image from the original. The consequences are much different from other types of data, for example, text. Suppose words or even single letters are changed or removed. In that case, the text's whole meaning can change significantly, or in some cases, even result in the opposite interpretation compared to the original.

Therefore, reducing the input image's amount of data is essential while retaining the original version's semantic information. However, one can only reduce the dimension for the input image up to a certain point until crucial features would disappear with naïve reduction methods. Convolutional layers combat this issue by providing a method to reduce the image resolution (or dimensions) helpfully, with weights automatically adjusted by the network during the training process.

### B. Convolution kernel

Every convolutional layer applies a convolution kernel on the layer input. Such a convolution kernel has a specific size with the same number of dimensions as the layer input. According to its size and dimensions, the kernel contains a certain number of weights. Each weight is simply a singular number (floating point or integer, which depends on the network architecture design), used as a factor for multiplication. [6]

When applying the kernel, it is laid on top of the input data, starting in the corner with indices 0. Each weight then multiplies with its corresponding input value. Finally, the algorithms adds up all of these products. This sum is the singular value of the result at that kernel position. According to the defined stride, the kernel then shifts along one dimension, and the process repeats repeatedly. When the kernel hits an edge, it goes back and shifts along the next dimension. (So left-to-right, then down by stride, left-to-right again - like one would read a text.)

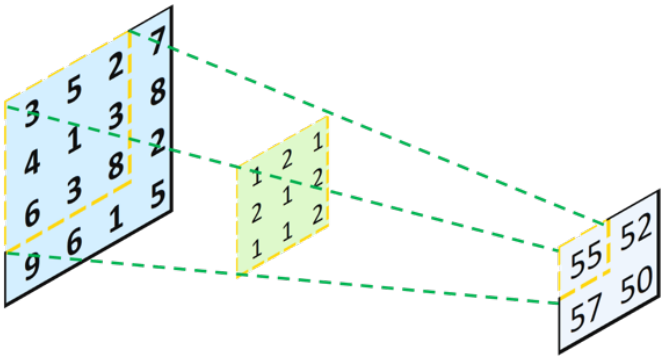


Fig. 1. Convolution kernel calculating the sum of each kernel weight with the corresponding input value [3]

A stride larger than one can significantly reduce the data's resolution. A singular kernel with the same length as the input data along one dimension could reduce the number of dimensions in the data. Usually, multiple kernels of the same size with varying weights apply to the input data. [6]

The formula to calculate the convolution $g$ of a traditional multiplication kernel $\omega$ is as follows [13]:

$$g(x,y) = \omega * f(x,y)$$
$$= \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy)$$

Here, the kernel $\omega$ has a width of $2a+1$ and a height of $2b+1$, $f$ is the input data, with the relevant input window placed at $(x,y)$.

Since this kernel uses multiplication for each singular weight with a singular input value, this paper refers to this traditional convolution kernel as a "multiplication kernel."

### C. Drawbacks

Even though this kernel design has proven itself as a good choice for neural networks, this approach has its drawbacks. Mainly, multiplication operations are computationally relatively expensive compared to other low-level processor instructions. As a usual CNN needs millions of multiplication operations per layer, these computational demands quickly add up, resulting in relatively high power consumption and powerful hardware requirements for acceptable performance – even if only for predictions with an already-trained network. As mentioned above, embedded systems usually do not have the same computational hardware used in traditional AI research and applications, like a high-performance graphics card. Furthermore, embedded systems sometimes have resource utilization targets (as they might run on battery-powered devices). Therefore, these CNNs should still run reliably on weak hardware. [13]

Considering these issues and requirements for embedded systems, there is a need for a CNN with convolution kernels that do not need any multiplication operations [13].

## III. Alternative kernel designs

This section highlights alternative kernel designs proposed in the past. As explained above, the main goal of these designs is to reduce the computational complexity of operations when calculating the convolutions.

### A. Shift-CNN

For example, the kernel shift-CNN contains signed integers $w_{exp}$ that signify a bit-shift by the number of specified bits. Furthermore, an additional sign value $w_{sign}$ (1 or -1) accompanies each weight, multiplying with that result. Given an individual input value $x$, this kernel calculates $w_{sign} 2^{w_{exp}} x$. [2]

This way, the kernel has a way to inverse activations. Hardware-wise, bit-shifts are very inexpensive, and flipping the sign is trivial and inexpensive. For circuit complexity, instead of one singular multiplier needed for the core part of a multiplication kernel, a shift kernel consists of one serial-shift-register, one multiplexer, one N-bit*1-bit multiplier for the sign. The shift kernel might contain additional shift registers and adders if the weights have a data width larger than one. [13]

The convolution of a shift-kernel can be calculated like this [13]:

$$g(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega_{sign}(dx,dy) \cdot 2^{\omega_{exp}(dx,dy)} \cdot f(x+dx, y+dy)$$

### B. XNORnet

Another architecture of interest would be XNORnet, which only uses bit data in its network [7]. Therefore, each kernel weight can only be one of two possible values: == or != [7]. The first one leaves the input bit as is, without modifying it; the other one flips the input bit. This process uses very little energy and hardware resources, orders of magnitude less than multiplication kernels. This design is undeniably much more straightforward than either a multiplication or shift kernel considering the circuit complexity. The core part of an XNOR-kernel consists just of several AND/NAND gates to calculate the operation. [13]

For the XNOR-kernel, the convolution can be calculated in the following manner [13]:

$$g(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \mathrm{XNOR}(\omega(dx,dy), f(x+dx, y+dy))$$

### C. Variations of the multiplication kernel

It is significant to highlight that both of these designs are a subset of CNN multiplication [13]. Any calculation these two kernels do is possible with specific weights in a standard multiplication kernel. A bit-shift is just a multiplication with two to the power of the number of bits. Leaving input bits would be a multiplication with one while flipping them can be achieved by multiplying with -1. Therefore, apart from being computationally less demanding, these alternative kernel designs do not introduce new capabilities to the neural network and can hardly produce better results than a sufficiently well-trained traditional CNN.

Furthermore, the multiplication operation of a traditional CNN can also be implemented in an analog fashion using a memristor [15]. One such analog core of the memristor-CNN consists of two parallel 1-Transformer-1-Memristor modules and one differential circuit. This implementation requires dedicated hardware designed alongside the network architecture, which results in significant energy savings compared to running a typical CNN on generalized hardware, such as a graphics card. [13]

### D. AdderNet Convolution

The main kernel design analyzed in this paper is the AdderNet kernel. It measures the similarity between the filters (kernels) and the input data by summing up the negative absolute differences between input and kernel. This calculation represents the L1-norm instead of the cross-correlation that a multiplication kernel would compute. The AdderNet kernel achieves this by subtracting each weight from the corresponding input value, calculating the absolute (hence the difference between input and kernel). Finally, all these absolutes are summed up and negated to maintain a positive relationship between higher similarity and a higher output value. [13]

When applying the adder kernel, the convolution is calculated with this formula [13]:

$$g(x,y) = -\sum_{dx=-a}^{a} \sum_{dy=-b}^{b} |\omega(dx,dy) - f(x+dx, y+dy)|$$

## IV. Hardware implementation of AdderNet

Two main architecture variants can calculate the difference between each kernel weight and its corresponding input value when implementing this network in hardware [13]. However, each of these two architectures has different benefits and drawbacks.

### A. 1C1A-architecture

The first would be the 1C1A-design, which stands for 1 Comparator, 1. This implementation first uses a comparator to determine if kernel weight or input value is the larger value. Depending on the result of that comparison, a multiplexer feeds larger and smaller values into distinct inputs of an adder. Finally, the adder subtracts the smaller from the larger value, outputting the absolute difference. [13]

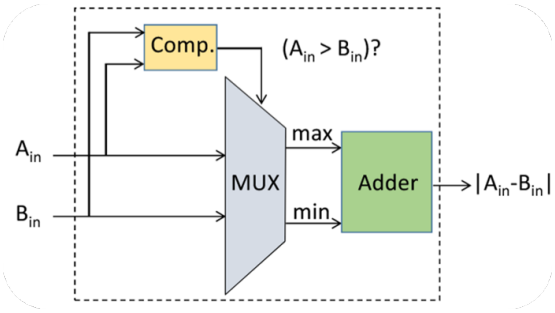


Fig. 2. 1C1A-architecture of the adder kernel hardware implementation [13]

This architecture consumes little resources due to only having one adder and a comparatively simple comparator and multiplexer. However, because the adder needs to wait for the result of the comparator before starting its calculations, this architecture has a significant gate delay. [13]

### B. 2A-architecture

The second architecture for the difference calculation hardware of the adder kernel is the 2A-design, which consists of 2 Adders. These two adders calculate the difference in two ways at the same time. That means one of the adders subtracts the kernel weight from the input value, while the other adder calculates it the other way around. Therefore, for differing values, one of the results will be positive and the other one negative. As the network requires the absolute difference, a multiplexer then selects and outputs the positive of the two values. [13]

Due to needing two adders, which are comparatively more complex, the circuit area of this architecture is higher. On the other hand, the parallel calculations without waiting on a comparator make this architecture faster and provides higher performance than the 1C1A-design. [13]

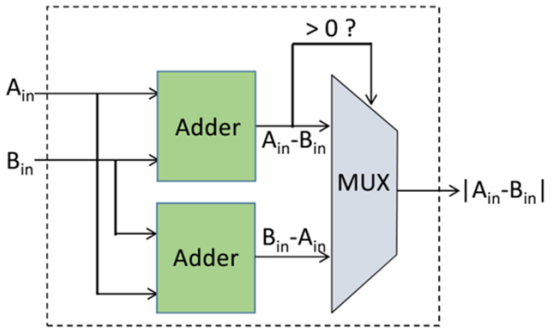


Fig. 3. 2A-architecture of the adder kernel hardware implementation [13]

This paper exclusively focuses on the 2A-implementation for analysis due to its increased performance.

In contrast to a multiplication kernel, the whole AdderNet kernel (with 2A-design) needs two adders and one multiplexer in place of every multiplicator [13].

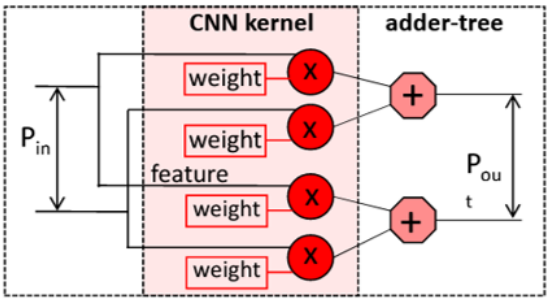


Fig. 4. Hardware implementation of a traditional multiplication convolution kernel [13]

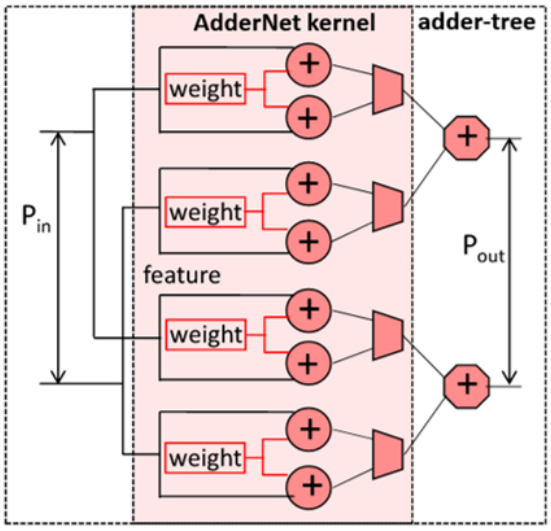


Fig. 5. 2A-design hardware implementation of the adder kernel [13]

These requirements might appear to result in more circuit area and computational performance requirements. However, since a multiplicator is much more complex and requires significantly more power, this design is more straightforward and energy-efficient [13], as evident in this paper's quantitative analysis section.

Depending on the kernel size, the CNN kernel module contains one of the 2A-modules for each kernel weight in order to be able to calculate all differences simultaneously. The calculated absolute differences are then summed up by an adder tree in the same manner as is done in traditional multiplication CNN hardware implementations. [13]

### C. FPGA

The developers of AdderNet implemented this circuit on FPGA hardware [13]. An FPGA (field-programmable gate array) is an integrated circuit (complete circuit on a small semiconductor), which can be mass-produced and is cheap, tiny, fast, and reliable, compared to designing and producing the dedicated hardware chip for this implementation oneself. The FPGA is reconfigurable with a hardware description language. Programmable logic blocks can be easily interconnected, resulting in the ability to realize the desired hardware implementation. This way, developers can simulate a desired chip design's hardware performance without manufacturing expensive custom chips. [12]

### D. Universal AdderNet accelerator

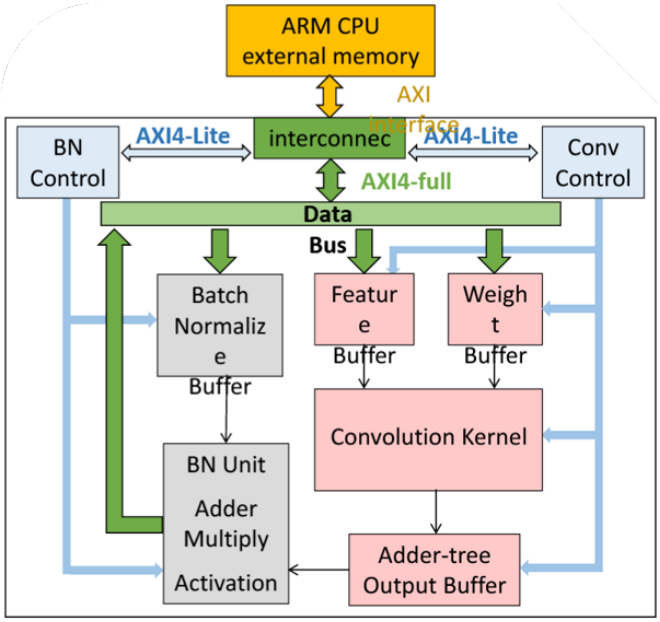


Fig. 6. Architecture of the universal AdderNet accelerator [13]

The universal AdderNet accelerator, as implemented on FPGA hardware, consists of several key components which communicate with each other of the interconnection protocol AXI (advanced extensible interface). Over this AXI interface, the accelerator receives information from a linked ARM CPU and external memory about which calculations to perform (usually the input data for a requested CNN prediction or weight data of a pre-trained network). This interconnection then unpacks this data and relays it via a bus to three different

buffers: the batch normalize buffer, the feature buffer, and the weight buffer. [13]

A convolution controller and batch normalization controller are responsible for the correct order of operations. During calculations, the convolution controller releases the significant feature and weight data from their respective buffers into the convolution kernel module, which calculates the differences between individual kernel weights and corresponding input values (using one 2A-module for each kernel weight, as described above). These differences are then summed up and negated in an adder-tree, which combines the individual results to calculate the similarity of the whole kernel with the relevant input window and stores the result of that single kernel application in an output buffer. [13]

The batch normalization controller releases the data of the batch normalize buffer into a batch normalization unit, which also receives the convolution result from the output buffer. The result of these activations is then relayed back over the bus to the data cache until the final result of the calculation can be returned via AXI to the external memory. [13]

### E. Quantization

It is crucial for performance to consider the data type used to quantize the network's weight and feature data. AdderNet utilizes integers for its values and weights. Usually, more integer bits result in higher computational cost but also in higher result accuracy, which facilitates a more accurate performance. [13]

In this case, an analysis of feature and weight data used in network training and inference shows that at least 96% of the data lies within an 8-bit region. So if these values are normalized to fall into the 8-bit integer region, one can quantize the values with that 8-bit integer without a significant loss of information. Furthermore, after training the network on 8-bit integers, it becomes evident that the CNN quantized this way has zero accuracy loss (as the network itself adapts to this value range in the training process). Therefore, to have the best possible combination of best accuracy and little computational cost, an 8-bit range for the values in AdderNet is recommendable. [13]

### F. Parallel computation

This section analyzes the logic resource utilization of AdderNet synthesized in an FPGA with varying parallelism levels. Logic resource utilization describes how many hardware resources of the computer chip an algorithm needs for its calculations. Evidently, the convolution kernel occupies more logic resources with increasing parallelism level. The simultaneous execution of more convolution operations results in a higher logic resource utilization for these calculations to satisfy their computational demand. [13]

Using AdderNet instead of a multiplication kernel results in about 80%-savings of logic resource utilization within the convolutional part and 67.6% resource savings for the whole system, when either network uses 16-bit quantization. However, when both networks switch to 8-bit data types,

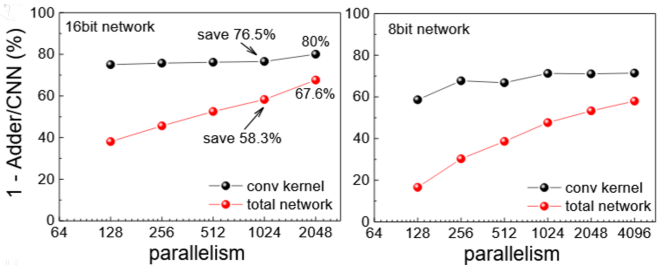


Fig. 7. Comparison of logic resource savings at different parallelism levels and bit-width [13]

AdderNet only saves about 70% of logic resources in the convolutional part and about 58% for the total network. [13]

Also, AdderNet runs on a higher frequency due to its calculations requiring fewer processor cycles. It executes its calculations at 250 MHz, while the usage of a traditional multiplication kernel results in 214 MHz. Furthermore, the intrinsic power of AdderNet convolutions lies at 1.24 W instead of 2.57 W for multiplication convolutions. [13]

### G. All on-chip design

Instead of using the previously described universal AdderNet accelerator with an external ARM processor and extra memory, a whole network can fit on a singular chip. The figure below shows an FPGA-based CNN accelerator without needing off-chip data storage or external input/output connections. This CNN accelerator contains the LeNet-5 network, which was developed in 1989 to recognize images of digits. Data storage and calculations all happen on-board the same FPGA chip. [13]

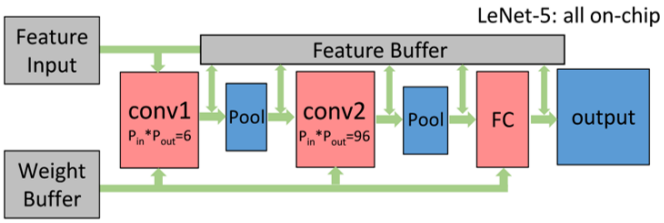


Fig. 8. Architecture of the LeNet-5 all on-chip design [13]

## V. Comparison to other kernels

### A. Performance / prediction accuracy

When comparing the results on several benchmark datasets (various ResNet implementations and a VGG-16 architecture), AdderNet and the traditional multiplication CNN seem to be on par with their intrinsic performance. In most cases, the multiplication CNN provides more accurate predictions, sometimes AdderNet. Compared to either performance, it becomes evident that the shift-CNN, Shift-Adder, binary neural network, and especially the memristor network make significantly less accurate predictions. [13]

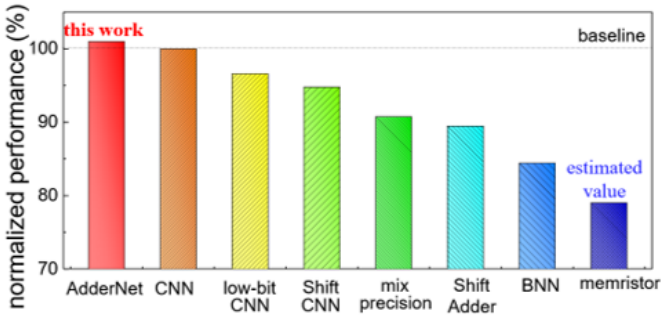


Fig. 9. Comparison of normalized performance of various kernel architectures [13]

### B. Energy consumption and logic circuit area utilization

It becomes evident that a higher data width corresponds to higher energy demands for the convolution kernel. The binary neural network and memristor implementation have the lowest kernel energy demands due to only working on bits and efficient, direct hardware implementation, respectively. However, they were also among the worst performers when it comes to prediction accuracy. Among the network designs with a higher bit-width, the traditional CNN and shift-CNN have by far the highest energy consumption, with shift-CNN being higher for small bit-width and the traditional CNN using more energy for larger bit-widths. AdderNet, while still being one of the networks with a larger data width and competitive results with a traditional CNN, uses relatively less energy. [13]

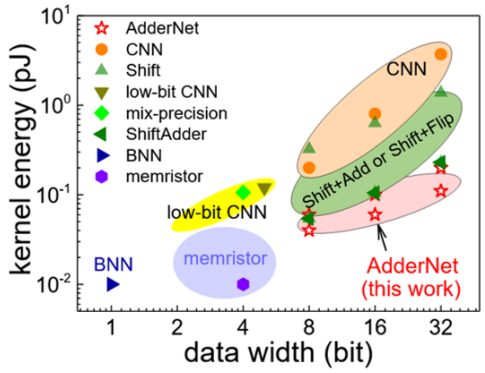


Fig. 10. Comparison of energy consumption of various kernel architectures in relation to their data width [13]

When looking at the 16-bit LeNet-5 all on-chip implementation, the whole AdderNet network consumes 77.9% less energy and 71.4% less logic circuit area compared to a traditional CNN. When utilizing 8-bit data types, AdderNet achieves 56.6% energy savings and 61.9% less logic circuit area usage. [13]

## VI. Further Research

### A. DNN Training without Multiplications

When trying to multiply two floating-point numbers, it is possible to approximate the product simply by interpreting these numbers as unsigned integers and using an integer addition operation on these bit values. The result can then be interpreted as a floating-point number again. Due to the composition of the floating data type of significand and exponent, this relatively basic operation approximates the product of the two numbers with a maximum error of 12.5% in the worst case. [5]

When training and testing the CNN ResNet-50 with this operation, the trained network provides a competitive or even equal performance, compared to a traditional ResNet-50, trained and operated with float multiplications [5].

### B. ShiftAddNet

ShiftAddNet uses bit-shift and addition operations to approximate a multiplication operation. This approximation provides high performance due to hardware implementations, resulting in 80% energy-cost saving compared to traditional multiplication kernels while producing comparable or even better results after training the network. [16]

### C. Adder Super Resolution

The Adder Super Resolution network uses the AdderNet described in this work with shortcuts as enhancements to produce a super-resolution network. The resulting network achieves similar performance in its results as traditional CNNs with a multiplication kernel while at the same time reaping the performance, logic resource utilization, and energy consumption benefits that AdderNets provide. [11]

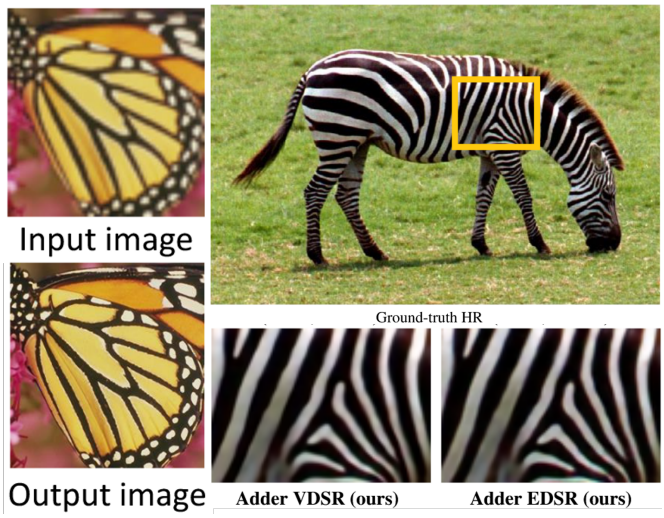


Fig. 11. Examples of the results of the Adder Super Resolution network [11]

### D. Kernel based progressive distillation for AdderNNs

It is also possible to train a typical CNN with multiplication operations as a teacher network, which in turn guides the learning of an AdderNet of the same architecture. The AdderNet is then used for inference and provides better performance and more accurate results than directly trained without a teacher network. [14]

## VII. Summary

This paper mainly presented the adder kernel design as an alternative to the traditional multiplication convolution kernel. In conclusion, it is possible to create CNNs without multiplication operations, which results in significantly lower resource demands. The resulting network is 16% faster, consumes between 47.85% and 77.9% less power, and utilizes 67.4% to 71.4% fewer logic resources – while still providing competitive results compared to a normal CNN [13]. Furthermore, this architecture can be implemented efficiently on an FPGA chip without the need for custom manufacturing. [13]

This performance is especially useful for AI in embedded systems that commonly have resource utilization targets, where devices might be battery-powered and usually have minimal hardware instead of a dedicated high-performance graphics card [13].

Further research could take place into how accurately this kernel alternative performs on other network types. In the main paper that introduced this technology, the developers of AdderNet only analyzed it with image classification networks. So whether or not this design is competitive or sometimes even better might vary significantly with the different types of CNN applications.

Furthermore, the type of recommended convolution kernel depends highly on the desired use case and on the hardware requirements associated with that implementation. Therefore, an adder kernel might not always be the better choice. Instead, critical analysis and comparison between different kernel designs should take place before choosing a specific architecture.

Finally, this paper presents various other techniques that exist to avoid multiplication operations in CNNs. Researchers are developing more of these architectures and alternative approaches. This active field becomes increasingly important with the advent of the internet of things and the growing importance of embedded systems.

## Supervision

Presentation and seminar paper have been supervised by Adrian Frischknecht.

## References

[1] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., ... & Asari, V. K. (2019). A state-of-the-art survey on deep learning theory and architectures. Electronics, 8(3), 292.

[2] Elhoushi, M., Chen, Z., Shafiq, F., Tian, Y. H., & Li, J. Y. (2021). Deepshift: Towards multiplication-less neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2359-2368).

[3] Greenholt, V. (2020). Understand transposed convolutions. Retrieved July 16, 2021, from https://morioh.com/p/6349bbe063b2

[4] Khan, S., Rahmani, H., Shah, S. A. A., & Bennamoun, M. (2018). A guide to convolutional neural networks for computer vision. Synthesis Lectures on Computer Vision, 8(1), 1-207.

[5] Mogami, T. (2020). Deep Neural Network Training without Multiplications. arXiv preprint arXiv:2012.03458.

[6] O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.

[7] Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016, October). Xnor-net: Imagenet classification using binary convolutional neural networks. In European conference on computer vision (pp. 525-542). Springer, Cham.

[8] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538.

[9] Shinde, P. P., & Shah, S. (2018, August). A review of machine learning and deep learning applications. In 2018 Fourth international conference on computing communication control and automation (ICCUBEA) (pp. 1-6). IEEE.

[10] Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. In In Workshop at International Conference on Learning Representations.

[11] Song, D., Wang, Y., Chen, H., Xu, C., Xu, C., & Tao, D. (2021). Addersr: Towards energy efficient image super-resolution. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 15648-15657).

[12] Trimberger, S. M. (Ed.). (2012). Field-programmable gate array technology. Springer Science & Business Media.

[13] Wang, Y., Huang, M., Han, K., Chen, H., Zhang, W., Xu, C., & Tao, D. (2021). AdderNet and its Minimalist Hardware Design for Energy-Efficient Artificial Intelligence. arXiv preprint arXiv:2101.10015.

[14] Xu, Y., Xu, C., Chen, X., Zhang, W., Xu, C., & Wang, Y. (2020). Kernel based progressive distillation for adder neural networks. arXiv preprint arXiv:2009.13044.

[15] Yao, P., Wu, H., Gao, B., Tang, J., Zhang, Q., Zhang, W., ... & Qian, H. (2020). Fully hardware-implemented memristor convolutional neural network. Nature, 577(7792), 641-646.

[16] You, H., Chen, X., Zhang, Y., Li, C., Li, S., Liu, Z., ... & Lin, Y. (2020). Shiftaddnet: A hardware-inspired deep network. arXiv preprint arXiv:2010.12785.