Deep Learning Application Mapping on Ultra Low Power SoC

Vedansh Bansal
School of Computer Science and Engineering

Dr.Mohamed M. Sabry (Asst Prof)
School of Computer Science and Engineering

Abstract **-** The emergence of artificial intelligence has brought about a profound transformation in our technological landscape. The widespread integration of AI has generated an urgent demand for improved hardware and software solutions to effectively deploy AI applications. Conventional architectures have encountered limitations in terms of power and memory resources, hindering the efficient execution of resource-intensive deep learning applications. To address this challenge, we present an inventive approach involving the implementation of an ultra-low power System-on-Chip (SoC) architecture, leveraging the compact NVIDIA Deep Learning Accelerator (NVDLA) in conjunction with a RISC-V core. This innovative solution aims to optimize the performance and energy efficiency of AI applications, offering a promising hardware framework for AI deployment.

The paper explores the optimization process of mapping and efficiently execute AI applications while minimizing energy consumption. The study presents an ultra-low power SoC architecture that integrates with sensors through serial communication protocols, showcasing real-world applications. By leveraging Python-based deep learning libraries and the C based environment, the research demonstrates the successful mapping strategy, leading to improved energy efficiency and extended battery lifespan for AI-integrated devices. This research contributes significantly to the field of ultra-low power AI systems, setting a benchmark for enhanced performance and prolonged battery life.

Keywords – CNN, RISC-V, NVDLA, Efficiency, AI

1 INTRODUCTION

Convolutional Neural Networks (CNNs) have emerged as a prominent category of deep learning models, renowned for their remarkable efficacy. Several highly successful CNN models have emerged, such as *GoogLeNet*, which have demonstrated exceptional performance in classification tasks, surpassing previously developed models by achieving an accuracy rate of nearly 95% [1]. These advancements signify a significant improvement in the field of image classification, indicating the effectiveness of CNN architectures in achieving state-of-the-art results.

While achieving high performance, it is worth noting that these advanced CNN models often demand extensive computational resources and significant memory storage compared to traditional models, involving operations on the order of 10^9 per image. Nevertheless, the need for power-efficient implementations of CNNs cannot be overstated, particularly in the context of embedded systems that necessitate real-time, low-latency, and high-accuracy capabilities. This requirement is especially critical for applications such as robots and autonomous vehicles. Therefore, developing power-efficient implementations of CNNs ensuring the feasibility and effectiveness of deploying these models in resource-constrained environments is of paramount importance. The primary objective of this paper is to provide a comprehensive overview of CNNs, elucidating their architecture, training methodologies and principles. Furthermore, we delve into deploying these CNNs on a System-on-a-Chip (SoC), with an emphasis on attaining optimal accuracy while ensuring minimal utilization of both space and power.The remainder of this paper is organized as follows: Section 2 Literature Review, Section 3 Introduction to the CNN architecture, Section 4 proposed hardware solution, Section 5 Methodology, Section 6 Experimental results, Section 7 Conclusion.

2 LITERATURE REVIEW

Efforts have been made to explore the deployment of CNN inference accelerators on compact and cost-optimized devices. Various techniques have been investigated, including implementing CNNs on FPGA and RRAM platforms, along with the application of dynamic quantization methods. These approaches aim to reduce the size of the deployed models while preserving their accuracy. By leveraging FPGA and RRAM technologies, which offer reconfigurability and efficient memory capabilities, researchers have sought to develop efficient and compact CNN implementations suitable for resource-constrained environments [3].

3 CNN ARCHITECTURE

Our human brain perceives the surrounding in a pictorial fashion and is absolutely phenomenal at recognizing different features, since our aim is to mimic the brain therefore, we use CNN which is

engineered to extract and identify features in an image. CNNs basic architecture is identical to any other Neural Network, it has various layers which are all build of perceptron and are connected

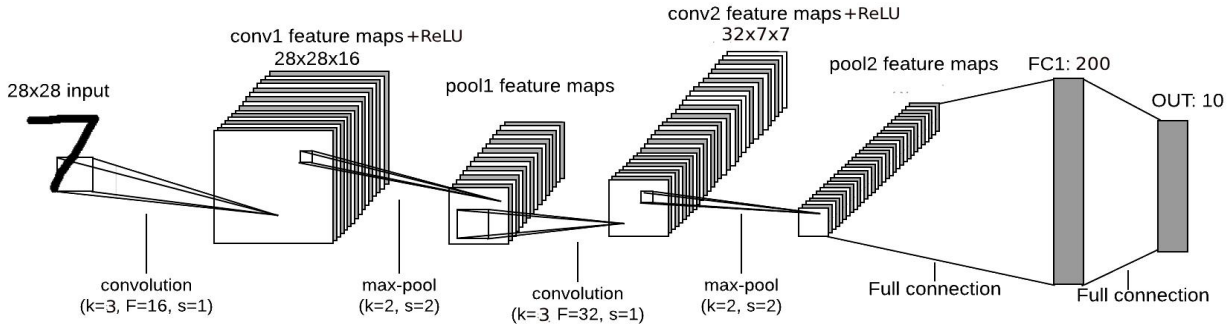


Figure 1. CNN Architecture we implemented

together in a web fashion. The differentiating factor is here for CNNs is are the functions that these different layers perform. A CNN has following 5 basic components:

3.1 Convolutional Layer

Convolution at its very basic is just a mathematical operation which extracts important features from a provided image. The image is converted to a matrix before being fed to the input layer. The convolutional layer has variety of kernels and filters for extracting different kind of features like for blurring an image or detecting edges of the

image. Majority of featuring extraction is just repetitive matrix multiplication. A kernel is a matrix of weights which are multiplied with the input matrix extracting all the significant features so that the model focuses on important and necessary information only. Often the terms 'kernel' and 'filters' are used interchangeably but there is clear difference between them. Kernel as we already discussed are matrix of weights whereas filter is a concentration of multiple kernels. Suppose there is an image which has 3 channels i.e., Red, Green and Blue (RGB) therefore a separate kernel is assigned to all three channels and therefore the filters are always one dimension extra than the kernels.

Now how do we decide that which kernel size bet suits our model? Generally, the kernels can be categorized in two sizes smaller and larger ones. Large kernel size would be any kernel dimensions 5x5 and above and the small kernels consists of 1x1, 2x2, 3x3, and 4x4. Since there have been some popular CNNs like *AlexNet* which used large kernel sizes but because of their extremely long training times (more than 2 weeks in some cases) we no longer use large sizes [3]. Using small size kernels is not only efficient in terms of computational costs but space capacity too.

Among the four choices 3x3 kernel size is the most used universally because of the fact that odd-sized filters divide the previous layer pixels around the output pixel symmetrically. 1x1 is also an odd sized kernel but it is only used for dimensionality reduction and provides very fine graining which is not required in most cases.

3.2 Activation Functions

Images in generally are represented in pixels which could be considered as linear data. This layer introduces non-linearity to the CNN model which enables it to learn complex relationships between the features. Without an activation function, the weights and bias in a neural network would only perform a linear transformation. While linear equations are relatively straightforward to solve, they have inherent limitations when it comes to addressing complex problems and lacks the ability to learn complex functional mappings from the data. *A neural network without an activation function is just a linear regression model.* [4]

The activation functions are divided into 2 types: Linear activation functions & Non-Linear activation functions. Linear activation functions have two major drawbacks. First one being that back-propagation is not possible and secondly it collapses all the layers into one, and that's why almost every modern neural network model uses non-linear activations. Following are some of the commonly used activation functions:

Functions	Range	Equations
Sigmoid	(0,1)	$f(x) = s = 1/(1+e^{-x})$
Tan-h	(-1,1)	$f(x) = a = \tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})$
ReLU	(0, +INF)	$f(x) = a = \max(0, x)$
SoftMax	(0,1)	$f(x) = e^{x_i} / (\Sigma_{j=0} e^{x_i})$

Sigmoid: It is generally works better for classification problems. Sometimes it is avoided due to its vanishing gradient problem. This occurs when the gradients calculated during backpropagation diminish as they propagate backward hindering effective learning.

Tan-h: It is not frequently used not just due to its vanishing gradient problem but also because its output range. The output of tanh can become highly centred around zero, leading to slower convergence and reduced numerical stability.

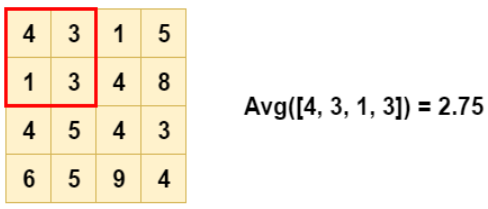
ReLU: This is the most used function since it brings a drastic change when there is a difference in feature. This is generally used for the hidden layers only but it is prone to dead neuron problem. This occurs when the input to a neuron becomes negative, the ReLU function outputs zero, effectively “killing” or deactivating the neuron.

SoftMax: This is generally used only for the output layer. The combination of SoftMax activation in the final layer and the categorical cross-entropy loss function allows the network to optimize the probabilities of different classes directly, leading to efficient training and accurate predictions

3.3 Pooling Layer

Pooling layer aims to reduce the spatial dimensions of the extracted feature maps in order to enhance computational efficiency and reduce the network complexity. This layer ensures that the layer identify features even in the case the features of the image are a little bit distorted or not consistent throughout. Max pooling and average pooling are some of the commonly used techniques.

i. Average Pooling: Average Pooling differs from Max Pooling in that it preserves a significant amount of information about the "less important" elements within a block or pool. Unlike Max Pooling, which discards them by selecting the maximum value, Average Pooling incorporates these elements through blending. This characteristic proves beneficial in various scenarios where such information holds value.



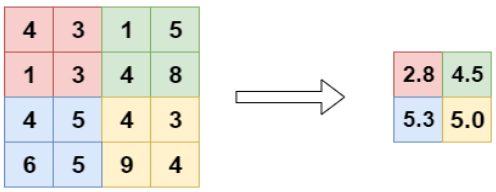


Figure 1. Average Pooling

ii. Max Pooling: Max pooling is a straightforward technique that involves selecting the highest value within a specific region. This approach is beneficial for highlighting the most significant features in an image. By selecting the maximum values, max pooling effectively identifies the brighter pixels. This technique proves particularly useful in scenarios where the image's background is dark, and the focus lies solely on the lighter pixels.

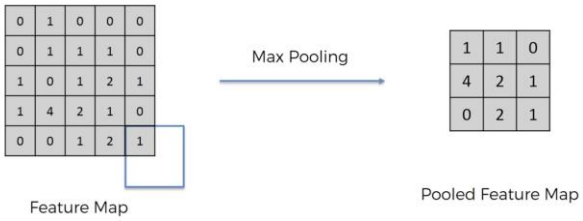


Figure 3. Max Pooling [3]

3.4 Full Connection

Fully Connected layer could be intuitively understood as a simple feed forward neural network. The output from the first phase is fed into this layer, then we perform forward feed and back-propagation repetitively adjusting the hyper parameters and the weights and biases. This process is generally what is referred as training the model. We train the model until the error function is minimized.

4 SYSTEM ON A CHIP(SOC)

Using a RISC-V core in a System-on-Chip (SoC) for deploying AI applications on embedded systems alongside the NVDLA accelerator brings several important benefits, particularly in terms of low-space and low power consumption requirements. Firstly, RISC-V's open-source nature allows for customization and optimization of the core specifically for the target embedded system, enabling efficient resource utilization. This customization ensures that the core is tailored to the specific needs of the AI application, maximizing performance while minimizing power consumption. Additionally, RISC-V's modular

design and scalability provide flexibility in choosing the level of complexity needed for the embedded system, ensuring an optimal balance between functionality and resource usage. The lightweight and compact nature of RISC-V cores also aligns well with the goal of low-space chips, enabling efficient integration with the NVDLA accelerator. Overall, utilizing a RISC-V core in conjunction with the NVDLA accelerator empowers developers to create energy-efficient and space-conscious AI solutions for embedded systems.

4.1 RISC-V

RISC-V core has become surprisingly prevalent in the computing space and especially in power efficient computing due to its advantages over CISC based processors.

RISC-V is based on RISC (Reduced Instruction Set Computing) which follows a minimalistic design philosophy which emphasizes on reduced complexity and simplicity. It promotes a streamlined architecture that inherently supports low-power operations compared to the other processors like x86(Intel and AMD processors) which follow CISC (Complex Instruction Set Computing) [6]. RISC-V even offers power aware extensions, such as the 'P' extension (e.g.: RV32IMAC) which provides the user with extra features like dynamic voltage and frequency scaling for fine grain power management.

RISC-V, as an open-standard Instruction Set Architecture (ISA), provides developers with unrestricted access to modify and customize the ISA to suit their precise low-power demands. This inherent flexibility has not only facilitated tailored optimizations to enhance power efficiency but has also garnered considerable support from the industry, leading to the development of a resilient and flourishing ecosystem. The openness of RISC-V has proven pivotal in substantially reducing costs and has fostered a realm of innovative possibilities and therefore we making RISC-V the most appealing choice for low-powered and space constrained computing.

4.1 NVIDIA Deep Learning Accelerator (NVDLA)

The NVDLA (NVIDIA Deep Learning Accelerator) stands out as an exceptional choice for AI applications due to its impressive features and technical capabilities. With a focus on high performance, the NVDLA leverages a specialized hardware architecture and optimized design to deliver exceptional speed and efficiency in executing complex neural network computations. Its modular architecture allows for flexibility and scalability, accommodating a wide range of AI application requirements. In terms of energy efficiency, the NVDLA incorporates advanced power management techniques and power-optimized design principles, ensuring optimal resource utilization and minimized energy consumption.

Furthermore, the NVDLA demonstrates seamless integration with widely adopted processor architectures, including the popular RISC-V. This integration greatly facilitates the efficient co-design and co-optimization of both hardware and software components, enabling enhanced synergy and performance. Moreover, NVDLA's hardware capabilities are complemented by its extensive software support. It encompasses a comprehensive suite of tools and drivers that flawlessly integrate with renowned deep learning frameworks like Caffe, TensorFlow, and Pytorch. This compatibility simplifies the development and deployment process, empowering developers to harness the full potential of NVDLA's hardware acceleration efficiently and effectively.

5 METHODOLOGY

To accomplish the objective of this research, a systematic methodology was followed for the development and evaluation of the image classification model. Generally, neural networks are implemented using Python frameworks, primarily due to the extensive ecosystem of open-source libraries and frameworks specifically tailored for machine learning and deep learning tasks. The iterative nature of neural network design often necessitates frequent adjustments, and Python's dynamic characteristics enable rapid prototyping. We divided our methodology in 4 stages.

5.1 Model Design and Creation

In this study, an image classification model was initially developed in Pytorch to gain a comprehensive understanding of its functioning and to establish familiarity with the model. Our Pytorch model consisted of five layers: three Linear layers, two ReLU layers and one LogSoftMax layer for optimizing and normalization of the output layer predictions.

```
Sequential(
  (0): Linear(in_features=784, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=10, bias=True)
  (5): LogSoftmax(dim=1)
)
```

Figure 5. Pytorch CNN implementation

The training and evaluation process employed the widely used MNIST dataset, which comprised of comprehensive collection of handwritten digit images. To ensure reliable model assessment the dataset was split into training, validation and test subsets in the following ratios 80:20:20 respectively. The dataset consists of 60,000 gray-scale images of handwritten digits, owing to the large size of the dataset we used a NVIDIA GTX 1660Ti mobile GPU, enabling us to accelerate the training process resulting in exceptional accuracy.

5.2 Model Enhancement

Despite designing the CNN architecture with careful consideration of our specific requirements and the constraints imposed by the low-power SoC, further optimization was necessary. Hence, upon obtaining the trained model, we proceeded with implementing weight pruning and quantization techniques to enhance its efficiency. Weight pruning facilitated the removal of redundant weights, while quantization reduced the model's size, effectively minimizing memory consumption and computational costs by decreasing the numerical precision of the weights. This optimization process played a pivotal role in enabling the model to effectively operate within the constraints of limited computational power and the designated memory footprint.

5.3 Model Conversion

Once the Pytorch model reached a satisfactory accuracy level then the next step was to converting the model to C language. The Pytorch model was subsequently converted to C language in order to investigate the viability of deploying the model on a C-based environment, specifically a RISC-V core in this particular scenario.

We have 5 layers in our model out of which 2 are convolutional layers with ReLU implemented already in the forward fee and one fully connected and last one being tanh optimization layer for the output layer predictions. The C model was different in terms of implementation, unlike Pytorch C does not have an inbuilt library for developing CNNs and therefore we made several user defined functions for creating layer architecture, neuron architecture, initializing_weights, forward_feed and back propagation etc. This conversion process enabled the model to be compatible with our core and opened up possibilities for further experimentation and deployment.

```
/*Creating CNN Architecture*/
/* Initialize layers. */
/* Input layer - 1x28x28. */
Layer* linput = Layer_create_input(1, 28, 28);
/* Conv1 layer - 16x14x14, 3x3 conv, padding=1, stride=2. */
/* + ReLU     */
Layer* lconv1 = Layer_create_conv(linput, 16, 14, 14, 3, 1, 2, 0.1);
/* Conv2 layer - 32x7x7, 3x3 conv, padding=1, stride=2. */
/* + ReLU */
Layer* lconv2 = Layer_create_conv(lconv1, 32, 7, 7, 3, 1, 2, 0.1);
/* FC1 layer - 200 nodes. + tanH for output layer*/
Layer* lfull1 = Layer_create_full(lconv2, 200, 0.1);
/* Output layer - 10 nodes. */
Layer* loutput = Layer_create_full(lfull1, 10, 0.1);
```

Figure 5. C-code CNN implementation

5.4 System-level Integration

Due to the chip's limited memory capacity, we conducted the model training process separately and focused on deploying it for testing purposes on the chip. As the core lacks support for file input/output instructions, we utilized the UART interface to transmit the data intended for testing. This involved storing the image pixels into an array and subsequently feeding this array as the input layer to the convolutional neural network (CNN) through UART. This approach facilitated real-time AI processing directly on the chip, despite the memory constraints.

6 EXPERIMENTAL RESULTS

We conducted a comprehensive evaluation of the AI application on our System-on-Chip (SoC) to assess its performance. Our evaluation focused on measuring the power consumption during the execution of the AI application, which allowed us to evaluate the energy efficiency of our SoC architecture. Additionally, we measured the execution time to gauge the processing speed of our design and juxtapose ensuring high accuracy. The experiment aimed to analyze the energy efficiency and performance of our ultra-low power SoC, specifically designed for AI applications.

```
Number Of Images Tested = 10000

Model Accuracy = 0.9704
```

Figure 6. Accuracy of the model

6.1 Power Consumption

Our research revealed a substantial decrease in power consumption through the implementation of our System-on-Chip (SoC) design. By integrating the NVIDIA Deep Learning Accelerator (NVDLA) model, we observed a significant reduction in power usage during the execution of the CNN model, surpassing the power consumption of a

baseline SoC without NVDLA. These findings affirm our initial hypothesis that utilizing a specialized AI hardware accelerator can lead to remarkable energy efficiency gains.

6.2 Computational Speed

Furthermore, our investigation demonstrated noteworthy enhancements in processing speed. Leveraging the dedicated AI acceleration capabilities of the NVDLA, we achieved acceleration in executing AI workloads compared to conventional SoCs. This notable improvement can be attributed to the modular architecture of the NVDLA, which enabled us to selectively utilize the required components, effectively optimizing resource allocation and mitigating system latency.

7 CONCLUSIONS

In conclusion, our experimental findings validate the effectiveness of our ultra-low power System-on-Chip (SoC) design for Deep Learning applications. The notable enhancements in energy efficiency and processing speed, coupled with the successful real-time interpretation of sensor data, highlight the potential of our approach. These results indicate that our SoC design has the potential to revolutionize AI-integrated devices, especially in environments with limited resources where extended battery life and improved performance are critical factor.

ACKNOWLEDGMENT

I would like to acknowledge my project supervisor, Dr. Mohamed M. Sabry for his continuous support and guidance throughout the project.

I would like to acknowledge the funding support from Nanyang Technological University – URECA Undergraduate Research Programme for this research project.

REFERENCES

[1] Low power convolutional neural networks on a chip - researchgate. (n.d.). https://www.researchgate.net/publication/309613178_Low_power_Convolutional_Neural_Networks_on_a_chip

[2] Mishra, P. (2019, February 4). *Convolutional Neural Networks (CNN)*. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/convolutional-neural-networks/

[3] Pandey, S. (2020, July 1). *How to choose the size of the convolution filter or kernel size for CNN?*. Medium. https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15

[4] Gharat, S. (2019, April 14). *What, why and which?? activation functions*. Medium. https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441

[5] Low power convolutional neural networks on a chip - researchgate. (n.d.). https://www.researchgate.net/publication/309613178_Low_power_Convolutional_Neural_Networks_on_a_chip

[6] Maning, J. (2022, November 14). *RISC vs. RISC-V vs. ARM: What is the difference?*. MUO. https://www.makeuseof.com/risc-vs-arm-what-is-the-difference/

[7] *NVDLA primer*¶. NVDLA Primer - NVDLA Documentation. (n.d.). http://nvdla.org/primer.html