論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Improving Deep Learning Efficiency Using Reservoir Computing Inspiration
著者(和文)	LOPEZ GARCIA-ARIAS ANGEL
Author(English)	Ángel López García-Arias
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第12710号, 授与年月日:2024年3月26日, 学位の種別:課程博士, 審査員:本村 真人,一色 剛,髙橋 篤司,佐々木 広,原 祐子,藤木 大地
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第12710号, Conferred date:2024/3/26, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Doctoral Dissertation

Improving Deep Learning Efficiency Using Reservoir Computing Inspiration

Department of Information and Communications Engineering
School of Engineering
Tokyo Institute of Technology

February 2024

Author
Ángel López García-Arias

Supervisor
Prof. Masato Motomura

Doctoral Dissertation

Improving Deep Learning Efficiency Using Reservoir Computing Inspiration

Ángel López García-Arias

Abstract

Forefront deep learning (DL) models involve massive computation and energy costs due to their vast number of parameters and intensive use of power-hungry arithmetic operations. Recently, AI commercial applications have experienced rapid growth in adoption, elevating this problem to a source of environmental concern. Although there have been multiple efforts to optimize DL at both the algorithmic and hardware fronts, there is an ongoing trend for exponentially larger models. Alternatively, research in various emerging technologies has explored solutions in different machine learning schemes and novel computation substrates. Among them, reservoir computing (RC) has emerged as a promising alternative by proving its high efficiency in early experiments.

This dissertation demonstrates RC's potential by proposing a low-cost image classifier based on cellular automata. However, it also analyzes the severe challenges RC faces to be upscaled to practical applications. Here, this dissertation proposes an approach to efficient DL based on borrowing the key efficiency elements of RC and applying them to current computer vision backbones for an immediate solution to the problem.

The Strong Lottery Ticket Hypothesis (SLTH) recently showed that training a deep neural network (DNN) without learning weights is possible by optimizing a sparse binary pruning mask—a supermask. Folding the DNN architecture beforehand into a recurrent structure results in much smaller models with higher or similar accuracy. Applying these two techniques to a residual neural network (ResNet) results in a DNN with strong similarities to a reservoir computer—the first RC-like DL model, the Hidden-Fold Network (HFN). On top of being RC-like, HFN is an accurate and tiny model that achieves competitive accuracies up to 71.92% on full-scale image classification and model size compression down to 2 MB, small enough for on-chip SRAM.

However, inefficient training makes SLTH models suffer in accuracy on large-scale datasets. This dissertation tackles this problem with Multicoated Supermasks (MSup), a scalar supermask that raises accuracy to match counterparts with trained weights. An SLTH model that initially only reached 68.16% accuracy on ImageNet is boosted to

74.3% by enhancing the supermask. Furthermore, MSup combines training, pruning, and quantization into a single concurrent process.

The models that result from combining HFN, MSup, and learned signs achieve 75.28% accuracy on ImageNet with a model size of only 4 MB, setting the SOTA for SLTH models. Moreover, after analyzing the SLTH and revealing that some of its models are not RC-like, but purely quantized, this dissertation proposes the Ternary Strong Lottery Ticket Hypothesis (T-SLTH). The T-SLTH extends the SLTH to operate with three types of randomness and arbitrary connectivity, opening the door to a wide variety of RC-like DL models and offering a new framework for generic quantization.

Processing the proposed RC-like DL models in a specialized processor is necessary to take advantage of their efficiency benefits. This dissertation presents *WhiteDwarf*, a holistic software-hardware co-design approach that first applies a triple model compression algorithm and then uses a novel neural inference acceleration architecture for triple unstructured sparsity exploitation. A carefully designed training schedule adapts the proposed T-SLTH models to a mixed precision format of FP8 activations, INT4 weights, and FP16 normalization with virtually no loss in accuracy. Then, the resulting models are compressed using Huffman coding, achieving off-chip compression ratios as high as 290.5×. This compression stack is also extended to a modern deep-MLP architecture. The *WhiteDwarf* architecture then exploits sparsity at the value and bit-levels of the activations and weights to reach up to 2000× on-chip model compression. Moreover, a multiplier-less arithmetic unit and a fine-grained clock-gating network harvest further power savings. The fabricated 40-nm CMOS *WhiteDwarf* chip achieves 12.24 TFLOPS/W. Additionally, it features a rich versatility in configuration aimed at research, which allows a series of ablation studies that demonstrate and quantify the efficacy of the introduced RC-like elements.

This dissertation proposes a new approach to efficient DL that can be immediately applied to real-world computer vision applications. RC-like DL models achieve competitive accuracy on full-scale image classification tasks while vastly reducing model memory size, a primary bottleneck in digital processors driving energy consumption. The results demonstrate that most of the DL computation is superfluous in multiple ways: 1) it is only necessary to learn part of the parameters, leaving the rest random; 2) architectures need not be deep if they are recurrent; 3) power-hungry multiplication arithmetic operations can be almost entirely avoided. The proposed approach is tested on ImageNet using ResNet and a deep-MLP architecture both algorithmically and on a fabricated ASIC, thus providing strong evidence of its viability and efficacy.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation: The Efficiency Problem of Deep Learning

The groundbreaking appearance of the AlexNet [79] deep neural network (DNN) in 2012, which crushed the competition by a wide margin in the ImageNet [132] image recognition challenge, is often credited as the spark that ignited the explosive *"deep learning* (DL [41, 82]) *revolution"* across several research fields. After a decade of intense DL research and development, in 2023 ChatGPT broke the record for the fastest-growing consumer application by reaching 100 million monthly active users in just two months from its public launch [62], starting the *"AI boom"*: DL has already reached the general end-user, and its applications are growing ubiquitous.

Although this impending AI-fueled era promises invaluable contributions to society in the future, such as reducing mortal traffic accidents by powering self-driving vehicles, it is immediately causing some serious negative repercussions. Major cloud computing providers are accommodating the ongoing rapid widespread of DL applications by building more and larger data centers, which drove the estimated global carbon footprint of information systems to around 2% [36] of the total greenhouse emissions in 2020, an impact as high as that of the aviation industry [131]. In the extreme case of the Republic of Ireland, data centers accounted in 2022 for 18% of the total national electricity consumption [1]. During 2023 there were multiple rumors of ongoing plans (e.g., by Microsoft and the US Department of Energy) for building small nuclear power plants with the sole purpose of keeping up with the growing demands of the cloud [3]. Additionally, data centers also consume immense amounts of water [2], and the production and update of the necessary electronics drive increases in intensive mining and electronic waste.

Apart from the growing commercial adoption of DL, an underlying technological trend contributes to this problem. Although the convolutional neural networks (CNN) leading the DL revolution had existed for decades, they only started thriving in the last decade. The

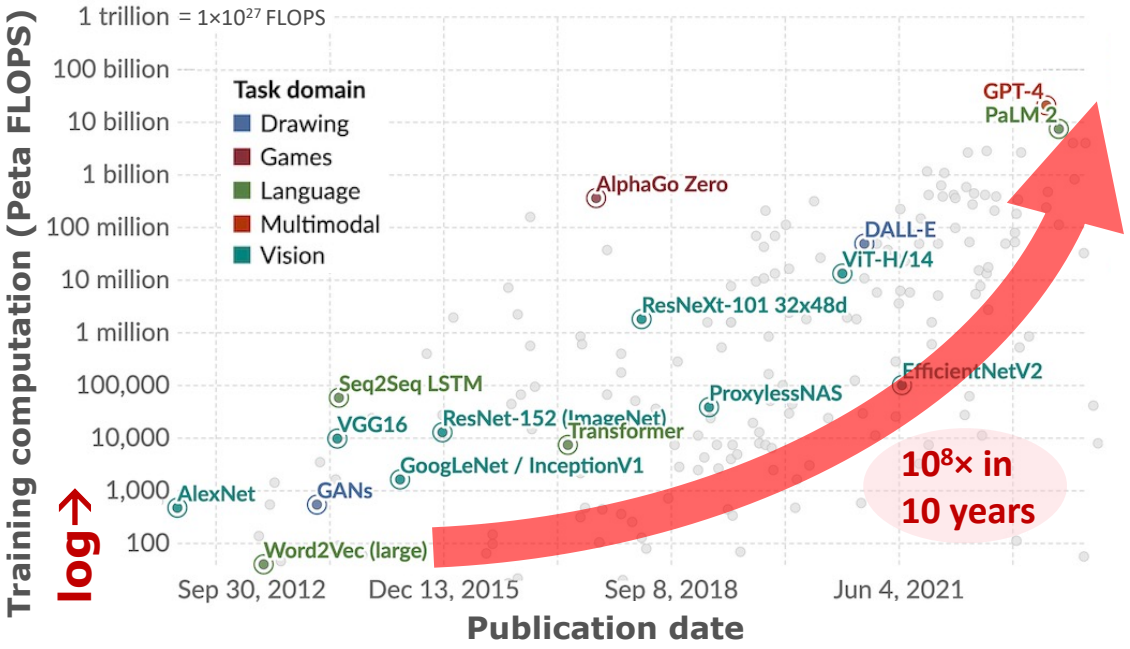


Fig. 1.1: The computational cost of training deep learning models has grown exponentially by 8 orders of magnitude in the last decade. (Figure generated with the visualization tool in [14], using data from [137].)

primary reasons for this sudden bloom were not so much critical algorithmic breakthroughs in the field as computation becoming more powerful, cheap, and accessible, and larger datasets becoming more available. Powered by Moore's Law and big data, the field of DL has grown by making models that employ more data, more parameters, and more computation. This trend, illustrated in Fig. 1.1, has only accelerated with the recent arrival of Transformer models and other immense natural language processing models [130, 144, 157]. GPT-4 (the engine of ChatGPT [120]) is estimated to have 1.76 trillion parameters (the details are not public), 30 000× more than AlexNet.

This trend is difficult to follow for resource-limited platforms. Since there is a rising demand for computer vision applications for embedded and mobile systems, with some autonomous driving cars already being released into the market (e.g., Tesla's Autopilot), and aviation safety agencies starting to work towards the certification of DNN systems for autonomous flight [18], there is an urgent demand to develop computer systems with innovative approaches to substantially reduce both computation and data movement, perhaps prioritizing processing efficiency over accuracy.

While specialized DNN accelerators have significantly improved computation efficiency,

data movement remains a primary factor driving energy consumption, costing nearly 6400× (off-chip) and 50× (on-chip) energy per access operation compared to arithmetic [58, 147]. There have been many efforts to optimize DNN, and leveraging the computational power of general-purpose processing on graphics processing units (GPGPU) is successfully enhancing the computational power of conventional systems. However, there is an intrinsic problem to this issue: digital von Neumann architectures are too different from the neural structure of the brain—which DL typically mimics with DNN—to emulate them efficiently. This issue is giving rise to innovative approaches that explore more adept architectures (e.g., neuromorphic, in-memory processing [37]), develop completely different machine learning schemes, or search for new computing substrates more powerful than modern electronics.

One of these emerging approaches is reservoir computing (RC), which has demonstrated great potential as a highly efficient machine learning paradigm due to its computational simplicity and very few learnable parameters. Most importantly, reservoir computers can be implemented by leveraging the computational power of complex phenomena present in natural substrates—i.e., exploiting the natural laws of the universe to perform the bulk of computation with barely any power consumption. When implemented on a photoelectronic substrate, this computation is also performed close to the speed of light.

Although RC has proven its potential in various small tasks, the field faces numerous challenges to graduate from being a so-called "emerging technology" and becoming a viable, practical solution for the imminent AI energy crisis. Specifically, it is still unclear how to upscale RC from proof-of-concept implementations to applications aimed at the full-size tasks for which DL models are currently being deployed, such as computer vision.

This dissertation proposes a new approach that brings the key elements of RC into the DL models currently serving as the backbone for real-world computer vision applications, aiming to solve immediately the urgent efficiency problems that DL is facing today by drawing inspiration from RC. This approach covers the conceptual, algorithmic, and hardware design fronts. It presents a family of *RC-like DL vision models* that spans over a rich accuracy-size tradeoff, catering alike to the needs of resource-limited mobile platforms and high-performance data centers. This dissertation contributes a potential solution to the problem by demonstrating that DNN model sizes can be compressed 25–50× and that more than 99% of their computation is superfluous and can be avoided on specialized hardware without a critical sacrifice in performance.

1.2 Contributions and Outline of the dissertation

This dissertation is divided into 8 chapters that present both practical and theoretical contributions to reservoir computing (RC) and deep learning (DL) and establish a new research approach at their intersection. These contributions constitute algorithmic and hardware improvements to the efficiency of practical DL, with a focus on computer vision (CV) backbones.

After this general Introduction, Chapter 2 summarizes the fundamentals of RC in both its algorithmic and physical variants, introduces the DL areas of the Strong Lottery Ticket Hypothesis (SLTH) and recurrent neural networks for CV, and covers the background of sparse neural engine architectures.

Chapter 3 demonstrates the potential of RC as a high-efficiency competitor to DL for CV tasks by proposing an enhanced image classification model based on RC and cellular automata, in addition to an FPGA accelerator design. Then, it discloses the current obstacles for upscaling RC to practical CV applications. With this reference model and analysis, it introduces the novel RC-inspired DL approach proposed by this dissertation, and specifies the contribution of each of the consecutive chapters. Fig. 3.11 then illustrates a mode detailed dissertation outline.

Chapter 4 introduces *Hidden-Fold Networks* (HFN), the first RC-like DL model, by leveraging two synergetic DL methods with strong similarities with RC: the SLTH and folding. HFN is, additionally, one of the smallest DL models for CV, fitting in just 2 MB. This tiny size, in addition to its sparsity and weight reuse, makes HFN an excellent candidate for hardware acceleration. Additionally, it presents a new theory on the effectiveness of the residual neural network (ResNet) architecture.

Chapter 5 boosts the accuracy of SLTH with *Multicoated Supermasks* (MSup), a novel method that integrates learning, pruning, and quantization in a single process. The resulting models set the SOTA in accuracy for SLTH-based models.

Chapter 6 proposes the *Ternary Strong Lottery Ticket Hypothesis*, a new interpretation of the SLTH that expands it to 3 different types of randomness and a framework for generic pruning and quantization. The presented ideas open the door for a wider variety of RC-like DL models. Furthermore, the resulting models surpass the accuracy of those proposed in Chapter 5, setting again the SLTH SOTA.

Chapter 7 presents *WhiteDwarf*, a software-hardware co-design holistic approach with a triple model compression algorithmic stack based on the ideas of the previous chapters, and a sparse neural inference accelerator architecture featuring triple unstructured sparsity exploitation. The manufactured 40-nm CMOS chip, capable of handling the wide variety

of models proposed, achieves power reduction, high throughput, and staggering on-chip model compression ratios up to $2000x$.

Finally, Chapter 8 concludes the dissertation with a summary and a discussion.

Chapter 2

Background

This chapter narrows the broader areas delineated in Chapter 1 down to the specific work leading to the original research described in this dissertation. Section 2.1 introduces the fundamentals of reservoir computing (RC), using the cellular automata algorithm as the main example of a complex system and introducing its potential as an emergent hardware technology. It serves as the background for Chapter 3, which details the RC-inspired approach followed throughout this dissertation. Section 2.2 covers the background of the Strong Lottery Ticket Hypothesis (SLTH), a central topic in this dissertation. From Chapter 4 to Chapter 7, the research presented exploits the SLTH and also contributes new elements to it. Most of this dissertation focuses on recurrent convolutional neural networks applied to computer vision. Although this is an uncommon combination, some existing work on similar approaches is summarized in Section 2.3. Finally, Section 2.4 sets the background for the hardware architecture for sparse neural inference acceleration proposed in Chapter 7.

2.1 Reservoir Computing: An Efficient Alternative to Deep Learning

Reservoir computing (RC) is a machine learning paradigm with a small number of parameters that leverages the computational power emerging from complex systems. Where deep neural networks (DNN, Fig. 2.1a) have an input layer, a number of hidden layers, and an output layer, all of whose parameters are learned, RC (Fig. 2.1b) only updates in the learning process the weights of the output layer.

Instead of hidden layers, RC performs the bulk of computation in a *reservoir*: a dynamic system that performs a non-linear mapping of the inputs into a higher dimensionality space (similarly to hyperdimensional computing). Data is input into the reservoir in a fixed

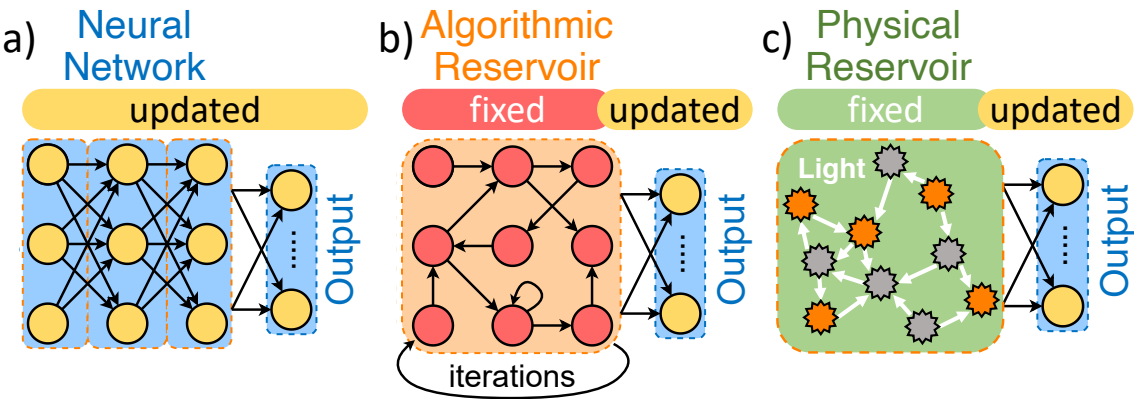


Fig. 2.1: Comparison of the neural network and the reservoir computing schemes.

way, and the learned output layer—the *readout*—consists of a simple classifier. Because of this simplicity and the small number of learned parameters RC is regarded as a low computational-cost alternative to DNN. In exchange of not being optimized, the fixed setting of the system acting as reservoir requires meticulous attention.

2.1.1 Algorithmic Reservoir Computing

RC originated from two algorithmic research trends: echo-state networks (ESN) [68] and liquid-state machines (LSM) [99]. ESN extended extreme learning machines (ELM) [65,66], which are feed-forward random neural networks, to use recurrent neural networks (RNN). Therefore, it the specific case of RC where the reservoir is formed by a randomly initialized sparse RNN. On the other hand, LSM use a reservoir of random recurrent spiking neural networks (SNN).

However, RC models are not limited to computation-hungry neural networks. In theory, any dynamical system can serve as a reservoir as long as it satisfies the following conditions [149]:

- Hyper-dimensionality: The reservoir must map the inputs into a higher dimensional space, where the simple readout can easily separate classes.
- Nonlinearity: The dynamic system must be non-linear, so that non-linearly separable data is transformed into features that can be separated easily by the linear classifier.
- Echo-state property: The system must be able to hold information through time, but this memory must fade out asymptotically.

Most importantly, it is generally recommended to use a system with behavior on the edge between stability and chaos (this concept is explained in Section 2.1.1 using cellular automata). In conclusion, the reservoir should be a complex system that is not too chaotic.

One of the most common non-neural complex algorithms for RC, also employed in Chapter 3, are cellular automata, explained hereafter.

Elementary Cellular Automata (ECA) and Complexity

Complex systems are generally studied through experimentation with simple computer programs. The quintessential complex system used for research has been cellular automata (CA), as they show a wide range of complex behaviors while being extremely simple.

CA are simple dynamical systems with a discrete state that evolves in discrete time steps. The evolution is determined by a simple rule based on the current state of the neighboring cells state and its own. Because each cell only considers the adjacent units, and they are all updated distributedly, CA is a very local and parallel algorithm, making it very attractive for hardware acceleration.

Formally (adapted from [81]), CA form a D-dimensional lattice (or grid), with an automaton at each site of the lattice. Each automaton takes as input the states of the cells inside a neighborhood $\mathcal{N}$. A cell's $\mathcal{N}$ has a dimension of $\mathcal{N} \leq D$, and includes $N = |\mathcal{N}|$ cells in total, typically including itself. Typical neighborhoods are the von Neumann neighborhood, which includes the orthogonally adjacent cells, and the Moore neighborhood, which also includes the diagonally adjacent cells (see Fig. 2.2). These contain only the immediately adjacent cells, but other less common neighborhoods include more cells by defining a radius r.

A set of states is denoted by Σ, and there are $K = |\Sigma|$ possible states. In every iteration, a

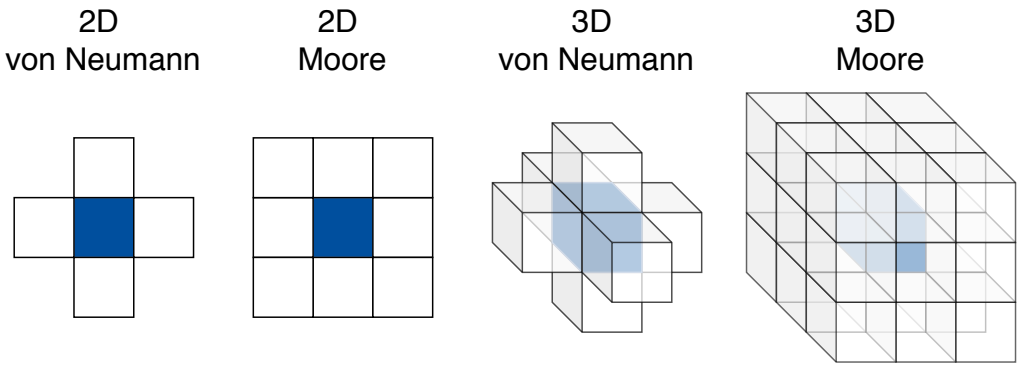


Fig. 2.2: Different types of CA neighborhood ($r = 1$). Central cell in blue.

transition function Δ—also referred to as a *rule*—performs on each automaton a mapping of the set of states of the cells inside its neighbourhood $\mathcal{N}$ at time t to the next state of its center automaton at time $t + 1$:

$$\Delta : \Sigma_t^N \mapsto \Sigma_{t+1}. \tag{2.1.1}$$

Typically, all automatons are updated synchronously using the same rule. Therefore, the size of the input alphabet α for the automata is $|\alpha| = K^N$. Since there are K possible states, there are

$$|D_N^K| = K^{(K^N)} \tag{2.1.2}$$

possible definitions of the transition function Δ (i.e., possible rules).

When considering a finite lattice, a boundary rule is needed to determine the behavior of the cells at the edges. Two common boundary rules are the fixed boundary and the wrap-around boundary. The fixed boundary gives a fixed value to states of cells outside the lattice, while the wrap-around boundary defines an adjacency between the edges, making the lattice a torus.

Elementary cellular automata (ECA) are one-dimensional binary CA (i.e.: $N = 3, K = 2$).

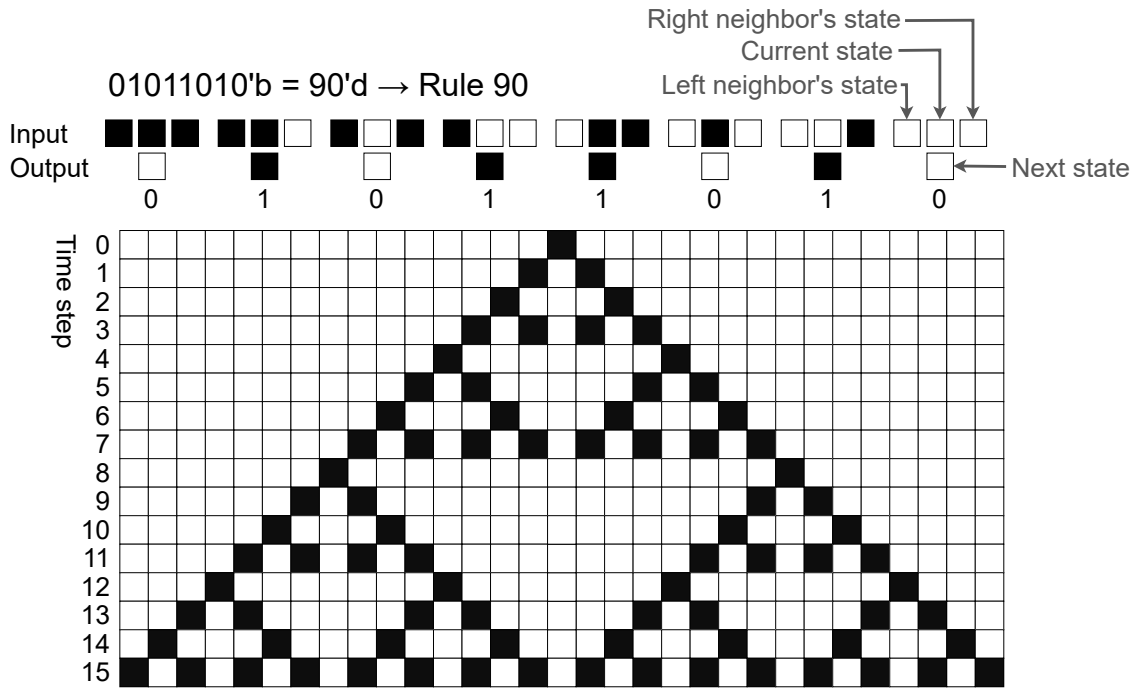


Fig. 2.3: ECA Rule 90, its Wolfram Code, and the fractal pattern it generates (the Sierpiński triangle).

Their evolution is determined by the state of the two neighboring cells and their own, and thus there are only $D_N^k = K^{(K^N)} = 2^{(2^3)} = 256$ possible rules. They are named Rule 0 to Rule 255 by a convention (known as Wolfram code [161]), illustrated in Fig. 2.3.

Depending on the rule used, CA can exhibit a wide variety of behaviors. Wolfram proposed a qualitative classification of CA (and more generally, of complex systems) in four groups based on the perceived qualitative behavior [161]:

- Group I: Almost all initial conditions lead to the same stable state.
- Group II: All possible final states are either static or cyclic.
- Group III: Initial patterns evolve into chaos. Small differences in the initial state make big differences.
- Group VI: They produce complex structures that interact with each other in long transients.

Some of the CA in Group VI have been shown to be capable of storing, processing, and transmitting information [81], therefore making them capable of computation. Indeed, it has been proven that some ECA are computationally universal [19], making them the simplest Turing-complete system known.

However, it shall be noted that the number of possible states and rules grows very quickly when considering CA of higher dimensionality (see Fig. 2.2):

- 2-dimensional binary CA with von Neumann neighborhood:

$$N = 5;\ K = 2;\ |D_N^K| = 2^{(2^5)} = 2^{32} = 4\ 294\ 967\ 296. \tag{2.1.3}$$

- 2-dimensional binary CA with Moore neighborhood:

$$N = 2;\ K = 9;\ |D_N^K| = 2^{(2^9)} = 2^{512}. \tag{2.1.4}$$

- 3-dimensional binary CA with von Neumann neighborhood:

$$N = 19;\ K = 2;\ |D_N^K| = 2^{(2^{19})} = 2^{524\ 288}. \tag{2.1.5}$$

- 2-dimensional CA with Moore neighborhood and 8-bit states:

$$N = 9;\ K = 256;\ |D_N^K| = 256^{(256^9)} = 2^{(2^{75})} \approx 2^{(3.8 \cdot 10^{22})}. \tag{2.1.6}$$

With a rule space so massive (for reference, the number of atoms in the observable universe is estimated to be $10^{80} \approx 2^{256}$), and no other definition of a system's complexity

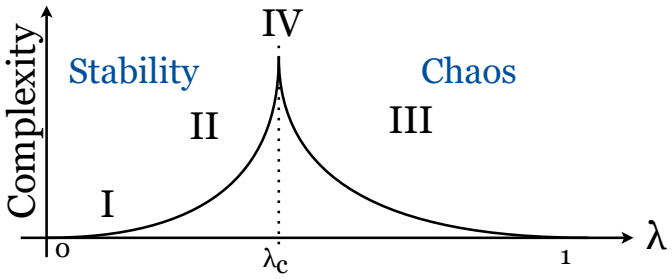


Fig. 2.4: *"The edge of chaos"*: location of Wolfram's complexity groups in Langton's λ space (reproduced from [81]). Although [81] analyzes phase transitions with several quantitative definitions of complexity, such as transient length or Shannon entropy, this representation illustrates the conclusions with a qualitative Y-axis.

other than a qualitative one based on observation, brute-force search becomes impractical very quickly for CA bigger than ECA. Moreover, the rule space is chaotic: the smallest change in a CA configuration may change its behavior completely, in an unpredictable way. This makes it a challenge to find the rules with interesting behavior.

There have been many efforts to characterize CA in a quantitative manner [19, 102, 136], including efforts to train machine learning models to classify CA [140]. which have resulted in a variety of CA classifications. One of the most popular approaches to finding a way of navigating the rule space was Christopher Langton's λ parameter [81]. λ is a measure of a CA's complexity, aiming to achieve a mapping of transition rules into a 1-dimensional space where similar behaviors are contiguous. More specifically, λ measures the homogeneity of a rule's table. λ is defined by selecting an arbitrary state $s \in \Sigma$, with n transitions to it in the transition function Δ. There are therefore $K^N - n$ transitions to other states. Then, λ is defined as

$$\lambda = \frac{K^N - n}{K^N}. \tag{2.1.7}$$

If all transitions lead to state s, $\lambda = 0$; if none of them lead to s, $\lambda = 1$; if all states are equally frequent, $\lambda = 1 - 1/K$.

Langton found that, in the same way materials experience abrupt phase transitions at critical temperatures (e.g., water changing abruptly between the solid and liquid state at 0°C), CA experience an abrupt phase transition at a critical λ_c. Most interestingly, the most complex behaviors are observed in the vicinity of this phase transition between the orderly and the chaotic behaviors. Or in Wolfram's terms: Group VI is found between Group II and Group III in the λ space, as illustrated in Fig. 2.4. This observation coined a popular expression that informally defines complexity: *"the edge between stability and chaos"*.

Although the conclusions drawn from these experiments linger as a prevailing intuition in the field, this parameter has strong limitations [105]. Finding a better way of characterizing CA and navigating the rule space is still an active investigation area. Currently, the only practical ways of finding the right CA rule for a specific application are random search or, if using ECA, brute force search on its limited 256 rule-space.

ReCA: Reservoir Based on Cellular Automata

ECA are the simplest algorithms to exhibit complex behaviour and require a very low computational cost, making them an excellent candidate for an algorithmic reservoir. RC based on CA (ReCA), first proposed in [165], combines the simplicity of ECA with the simple readout layer of RC, forming a system that requires very little computation overall.

As there are only 256 possible ECA rules, it is feasible to find by brute force search the best configuration for using in the reservoir. If using a more complex CA, finding the appropriate rule for the reservoir is nontrivial. Although there are ongoing efforts in developing a method for finding CA through evolution algorithms [115] [127], currently the only way of determining the ReCA configuration is exhaustive search.

Generally, each element of the input vector is mapped to the initial state of an automaton, and after several iterations, the evolution history of the states of the reservoir is concatenated, then fed to the readout. However, ReCA literature places particular emphasis on the ways of encoding input data into the reservoir. It is also common to map inputs into a vector of higher dimensionality in a random fixed way, often mapping the same data to multiple positions in a similar way to hyperdimensional computing. Another popular option is learning mappings with weighted summations.

As it is still a very immature area, there is a lack of consensus on many aspects, as the encoding to be used, the type of ECA boundary, or how to construct the reservoir. There have been investigations on mixing more than one ECA rule in the reservoir. [103] experimented with reservoirs using two rules sequentially, changing the reservoir configuration after a number of iterations. Alternatively, [113] constructed a reservoir shared by two ECA rules spatially instead of temporarily. Two ReCAs with different rules are set side to side, and they are let to interact through the shared boundary. Similarly to using CA of higher dimensionality, these approaches suffer from a bigger search space with no efficient method for navigating it.

2.1.2 Physical Reservoir Computing

Although reservoirs are most commonly implemented with complex algorithms, the key potential of RC is that complex systems are ubiquitous in the nature of the universe, and thus reservoirs can be implemented by leveraging a physical phenomenon instead of a man-made processor. These *physical reservoirs* (Fig. 2.1c) have been demonstrated using mechanical systems, analog electronics, photonic devices, spintronic devices, DNA, and even living organisms [149]. That is, instead of performing the distributed simulation of a complex algorithm such as CA, the reservoir can be substituted in the RC paradigm with a natural phenomena. Input data is somehow transformed into a physical stimulus to the system, and then it is only necessary to observe its complex output. The bulk of the *computation is performed "for free"* by the laws of nature (this is referred to as in-materio computing). Furthermore, in the case of a reservoir based on a photoelectronic phenomenon, such as fluorescence resonance energy transfer (FRET) discussed in Chapter 3.3, this *computation is performed close to the speed of light.*

Although physical reservoirs face the drawback of digital-analog and analog-digital overheads, the low power and high speed of these reservoirs, in addition to the computational simplicity of the learned readout, make them a promising emerging technology for rivaling the extremely power-hungry DNNs and the general-purpose GPUs generally employed to deploy them.

ReFRET: Reservoir Based on Fluorescence Resonance Energy Transfer ∗ (FRET)

FRET [116] is a photoelectronic mechanism that describes a radiationless energy transfer between two light-sensitive molecules (chromophores). When very close (a distance smaller than the wavelength of the light emitted), an excited donor chromophore n may emit a virtual photon, which is absorbed by the non-excited acceptor chromophore m. The transition rate k_{nm}^{FRET} at which this phenomenon happens decreases with the sixth power of the distance between donor and acceptor, giving it a strong non-linear nature.

A reservoir based on FRET (ReFRET) can be implemented by a 2-D network of semiconductor nanocrystals—quantum dots (QD) spread on a sheet [50, 151]. Fig. 2.5 illustrates a ReFRET implemented with QD, and the possible interactions between them under the multiple-donor model [111, 112]. The ReFRET is excited by transforming input data into light of a specific frequency, emitted by a photon source. Once the QD get excited with a probabilistic rate k_n^E (determined by the nature of the QD), a complex interaction between the QD starts taking place during time by means of three different state

∗Also known as Förster resonance energy transfer, with the same acronym.

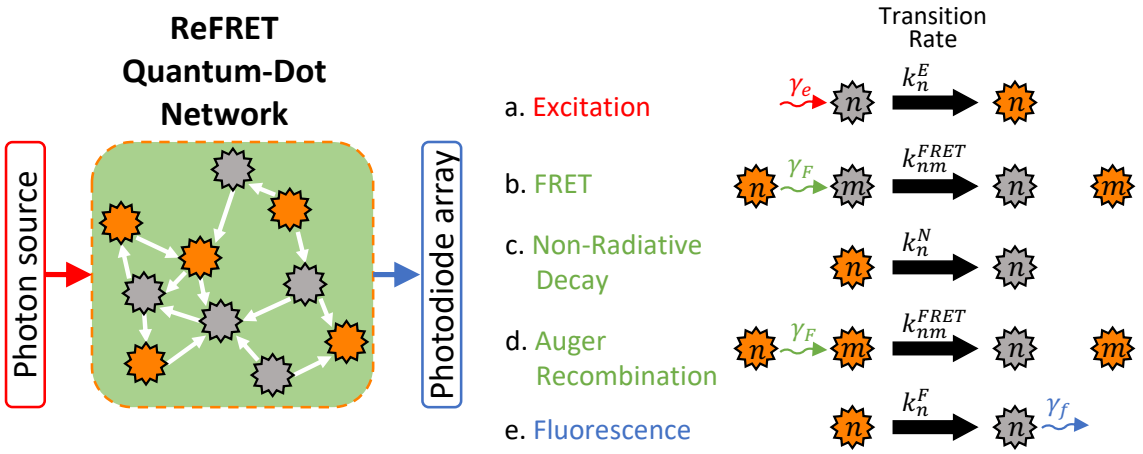


Fig. 2.5: Quantum-dot ReFRET [50] (left) and multiple-donor FRET model [111, 112] (right). The definition of each transition rate is ommitted for brevity.

transitions, whose rates k depend non-linearly on the distance between the nanocrystals. These interactions are, however, undetectable, since they are driven by virtual photons. The only observable phenomenon is the photons emitted via fluorescence at rate k_n^F (determined by the nature of the QD), captured as output with an array photodiodes. Since excitation and fluorescence involve light of different wavelength, they do not form a feedback loop, and the photodiode array can be isolated from the photon source by placing filter films between them. A study of the implementation of this system can be found in [50], in addition to a simulation algorithm for the multi-donor model [111, 112], discussed in Chapter 3.3, and a thorough study of ReFRET [146].

2.2 The Weak and Strong Lottery Ticket Hypotheses

Pruning is a common technique for compressing trained networks into much smaller models by removing unnecessary weights [11, 38, 47, 91, 179]. Applied progressively by iterating training and pruning, large portions of trained models can be removed without affecting accuracy, making it evident that the original models are overparameterized. The sparsity of the resulting network can be exploited for additional compression with entropy coding and for arithmetic optimization. Combined with weight quantization, it has resulted in very efficient model compression schemes [26, 46] and specialized hardware neural accelerators [45].

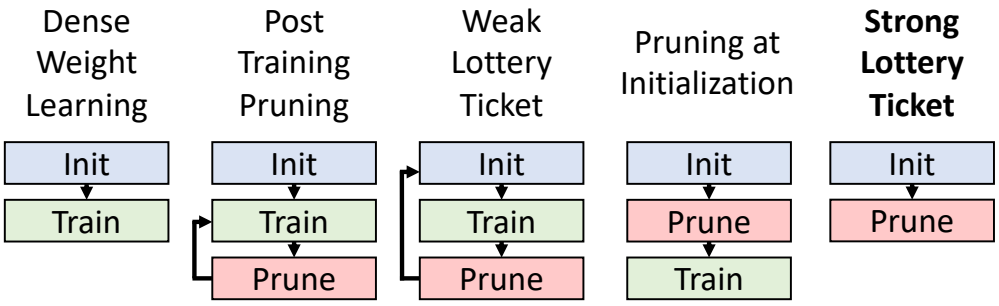


Fig. 2.6: Evolution of the training methods leading to the Strong Lottery Ticket Hypothesis.

2.2.1 The Weak Lottery Ticket Hypothesis

It was generally found that the connectivity patterns found by pruning were not disentangled from the pre-trained weights, as they could not be reinitialized and trained from scratch. However, a recent breakthrough paper [32] showed that overparameterized neural networks contain a subnetwork that can be trained in isolation to match the original model—the *Lottery Ticket Hypothesis* (LTH). These subnetworks, nicknamed winning tickets, are found by iteratively training, pruning, and resetting the remaining weights to their original value.

This paper has inspired a quickly growing body of work that has found that a network contains not one but several tickets [43], which may be connected [33]. Moreover, after it was shown that tickets can be identified early in the training process [166], some methods have succeeded in pruning before training [28, 34, 51, 84, 85, 143, 150, 159, 160, 167], even before looking at the data, in order to reduce training cost.

2.2.2 The Strong Lottery Ticket Hypothesis

In a surprising turn of events, while analyzing the LTH, [177] found that learning weights is unnecessary: an overparameterized neural network contains high-performing subnetworks at its randomly initialized state, which can be found just by pruning. Furthermore, they described an algorithm for finding these subnetworks by training a binary mask, as illustrated in Fig. 2.7. This technique was taken a step further in [128] with a training algorithm and a weight initialization scheme that delivers sparse random subnetworks of competitive performance in image classification tasks.

After a following series of papers provided theoretical ground to this phenomenon by analyzing the bounds of necessary overparametrization [100, 121, 125] in addition to showing the robustness of these subnetworks to binarization [23, 142], the existence of

subnetworks that can be obtained just by pruning has come to be named the *Strong Lottery Ticket Hypothesis* (SLTH), albeit some works refer to it as the Multi-Prize Lottery Ticket Hypothesis [23], or simply as "Hidden Networks".

Despite being a surprising finding, strong tickets have some closely related precedents. Extreme learning machines [65,66] are feedforward networks that use fixed random weights in their hidden units, only learning the output layer. Reservoir computers [149] use random recurrent architectures in an analogous manner. Similarly, [71] proposed substituting convolutional layers with fixed additive random noise and a learned linear combination. It was demonstrated that non-trivial accuracy could be achieved just by training the batch normalization parameters of a fixed random network [35]. More directly related, [101] adapted a binary neural network training method to learn binary masks that, when applied to a trained model, extracted subnetworks that performed well on untrained tasks. When using a randomly weighted backbone model, they even found subnetworks with non-trivial accuracy, closely missing the discovery of strong tickets. Furthermore, the bio-plausibility of strong tickets has been considered by linking it to the emergence of innate face-selectivity in the visual cortex [9].

The SLTH does not only merge training and pruning, but also quantization: strong lottery tickets have been found to be robust to binarization [23, 142], sparser when using ternary masks [77], and more accurate with a scalar mask [118]. Furthermore, sparse random weights and binary masks can be exploited for designing energy-efficient inference hardware [56], which can even switch the binary mask for adjusting computational cost at the edge [56] or for reusing the same random weights for a different task [162].

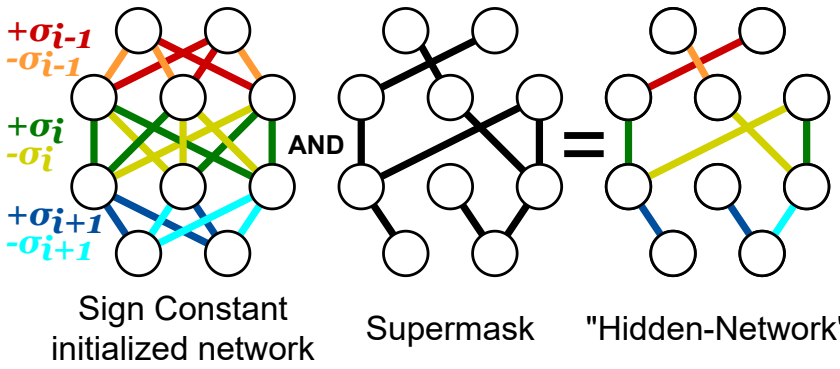


Fig. 2.7: The Strong Lottery Ticket Hypothesis: a high-performing subnetworks can be obtained from a randomly initialized model by training a binary mask—a supermask. The σs and colour pattern illustrate SC initialization (see Chapter 4.3.3).

However, current algorithms for finding strong tickets are unable to recover optimal strong tickets planted in the model artificially [31]. Training methods that use multiple random seeds, either concurrently [5] or iteratively [16], can improve performance, but at the cost of increasing model size.

2.3 Recurrent Neural Networks for Vision

Although deep neural networks for computer vision are predominantly feedforward, there have been some recurrent approaches. Recurrent convolutional neural networks (RCNN) have been used for scene parsing [126], object recognition [88], and image classification [13]. RCNN with long short-term memory (LSTM) [57] have also been used for improving weather forecasting [139]. The authors of [170] exploited recurrence for early image prediction, and found that an RCNN naturally learns taxonomic representations that are iteratively refined to finer-grain classes, improving explainability. Similar to this paper, some works have explicitly transformed ResNet into RCNN for constructing efficient models [44, 175], which can be taken a step further by using recurrent depthwise separable convolutions [72, 76].

If feedforward deep neural networks have been successful in computer vision, why use recurrent architectures, usually associated with time-series data? Here we summarize some of the compelling arguments for this paradigm shift, which are thoroughly covered in [156].

The main arguments come from the field of neuroscience. While the primate visual cortex has only a few layers deep, with a wide variety of feedback and lateral connections [73, 80], computer vision models are hundreds of layers deep and strictly feedforward. Despite this, the visual cortex has high accuracy in multiple tasks, even though it has fewer units and much lower energy consumption. Iterative computation can be exploited to implement complex functions with limited hardware, and for adjusting computational time dynamically, a behavior observed in biological brains [73].

Vision neural network models that take direct inspiration from neuroscience are accurate, small, and better predictors of the brain [80], and can effectively adjust the number of iterations dynamically for improved efficiency [86].

Furthermore, some vision capabilities can only be implemented with recurrence. Vision is an active process, therefore requiring feedback connections to control the associated organs and to integrate priors and attention into the vision tasks. Although computer vision has given overwhelming attention to static images, vision processes time-series data—"video". Feedback connectivity is necessary for processing temporal dependency, integrating stimulus history, and making predictions of the dynamic world.

Experimentally, neural network performance on vision tasks improves when models have recurrent and lateral connections [141], and they naturally learn a rich variety of feedback loops when given the freedom of choosing to share weights during training [134].

Intriguingly, ResNet, which is conjectured to approximate an RCNN, is both one of the best performing computer vision models and one of the most brain-like [135].

2.4 Sparse Neural Engines

Deep neural network (DNN) models provide high accuracy in a variety of tasks, but demand significant computational power and energy due to their large number of parameters and intensive use of multiplication operations. While specialized DNN accelerators have significantly improved computation efficiency, data movement remains a primary factor driving energy consumption, costing nearly 6400× (off-chip) and 50× (on-chip) energy per access operation compared to arithmetic [58, 147]. Since there is a trend for both bigger neural models and datasets, which recently has only accelerated with the arrival of Transformers and immense natural language processing models [130, 144, 157], there is an urgent need to develop computer systems with innovative approaches to substantially reduce both computation and data movement.

It is known that DNN models can be sparsified without much loss in performance [11, 47, 179]. Based on this, neural accelerators have exploited the inherent sparsity of general DNNs in two ways [147]: (1) reducing the footprint of data through reduced precision and compression and (2) reducing the number of MAC operations through sparse dataflow and computation.

2.4.1 Value-Level Sparse Accelerators

Prior arts have mainly focused on the value-level sparsity of DNNs, found in activations, induced by ReLU computation, and in weights, introduced by pruning. There is a rich body of work exploiting sparsity in activations [7], in weights [172], or in both [40,45,61,87,122, 171]. The primary focus of these approaches has been high performance by maximizing PE utilization. Compressed inputs contain coordinates [7,45,61,122] or bit-masks [40], which are used to detect the intersection of non-zero values. The partial sums are accumulated through a permutation network [40, 122] or using accumulation buffers [61, 87].

Many of the sparse DNN accelerators employ coordinate/payload list representations using well-known tensor representations such as compressed sparse row (CSR) [7,122,172], compressed sparse column (CSC) [45, 61], or a modified form of them (e.g., non-zero

coordinate bitmask [40]). This approach, focused on maximizing arithmetic hardware usage, requires a considerable control cost to calculate the intersection of two sparse tensors (e.g., 41% for SparTen [40] and 19% for Eyeriss v2 [61]).

2.4.2 Bit-Level Sparse Accelerators

More recently, a line of work has focused on the bit-level sparsity by decomposing multiplication operations into bit-serial shift-add operations and skipping zero-bits. Although most of these approaches have serialized activations [6, 22], some have serialized both activations and weights [87, 90, 138]. These processors, on top of being able to exploit sparsity at a finer-grained level, also benefit from a simpler control, as they are able to operate directly in an uncompressed encoding. Not needing to operate over encoded data allows the choice of a more compact code, such as Huffman coding [46], for reducing off-chip memory access.

2.4.3 SLTH Accelerators

Sparse accelerators have generally focused on accelerating pre-existing models, accommodating them with generous numerical precision and leaving sparsification to model designers. Although there has been some work on hardware-oriented model compression [46, 176], these works missed an opportunity on a recent paradigm change that moves sparsity to the central role: SLTH.

Hiddenite [56], the first SLTH accelerator, was able to leverage this opportunity to completely eliminate off-chip access for loading weights and multiplication operations by using a random number generator for on-chip on-the-fly weight generation. Although it achieved very high performance, it suffered from low accuracy, used a dense architecture, and only supports binary supermasks but not the advanced SLTH approaches proposed in this dissertation, which are critical for better accuracy and scalability.

Chapter 3

Efficient Design Inspiration From Reservoir Computing

3.1 Introduction

This dissertation aims to bring elements from reservoir computing (RC) to deep learning (DL) models for high-accuracy, high-efficiency models. This chapter details the motivation for this goal, specifying what "inspiration from RC" means by first presenting a full-RC approach to computer vision and then identifying both its limitations and key elements.

The remainder of the chapter is organized as follows. Section 3.2 reproduces a publication [97] that proposes algorithmic and hardware improvements to an RC-based image classifier that demonstrates that RC is a high-efficiency alternative to DL. However, as detailed in Section 3.3, RC faces numerous and complex challenges to be a competitive alternative in full-size tasks. This section describes some of the upscaling attempts attempted by this research. Section 3.4 proposes the new approach of this dissertation: RC-like DL. It identifies the key elements of the RC system proposed in Section 3.2 and outlines how this dissertation progressively infuses them into DL models in the following chapters.

3.2 Low-Cost Image Classification Using ReCA on FPGA

The low computational cost of reservoir computing (RC) could be a powerful alternative to CNN for energy-efficient image processing. To prove that RC is not just a compelling model for time-series data but can also be applied to image data, this section implements an image classifier based on RC for MNIST [83], an image classification dataset commonly used for

proof-of-concept implementations. Section 3.2.2 proposes algorithmic optimizations for reducing the computational cost and memory usage by using random forest (RF) with binary features as the output pattern analyzer. Also, Section 3.2.3 presents a hardware architecture targeted at FPGA that improves the reservoir's data usage and hardware mapping.

3.2.1 Reservoir Computing-Based Computer Vision

Compared to the vast amounts of power consumed by GPUs when processing CNN, RC is a very promising alternative for embedded applications, especially in the case of physical reservoirs (PR). Although most RC research focuses on time-series data, there is existing research using RC to process image data.

As most RC applications, ReCA has mostly been used for sequential data, achieving promising results in phoneme recognition [154], but there have been a few applications to image processing. A remarkable example was presented in [69], where the authors demonstrated a model based on echo-state networks (ESN) that performed both image classification and image denoising, obtaining a competitive classification error of 0.81% on the MNIST dataset [83]. However, this model transforms spatial data into sequential data by scanning the image. Although this approach is effective for data as simple as MNIST, it cannot be upscaled for complex image datasets, as the spatial information of the data is lost in the transformation.

The most notable of RC-based image processing applications are based on ReCA. Applying CA to images is not trivial. Although it might seem straightforward to use 2-D, or even 3-D CA, with a number of possible states K that matches the color's dynamic range of the input images (e.g., 8-bit states for 8-bit grayscale images), these larger CA have rule spaces for which there are no known ways to find good rules (see discussion in Section 2.1.1). With a lack of a way of navigating a larger rule space, ECA and exhaustive search are the only currently feasible candidates for ReCA.

In [75] a ReCA was used to implement a modality classification system for medical images (whether an image comes from a camera, an echography, an X-ray, etc.). Following a straightforward approach, images were fed to the reservoir as a vector of higher dimensionality. An interesting feature of this implementation was the use of a hyperdimensional classifier. This has the benefit of not needing an optimization routine, as predictions are made by measuring the Hamming distance between the vector resulting of processing the input and the prototype vectors of all the classes. A similar model applied to MNIST was implemented in FPGA in [74], but was only capable of acquiring a classification accuracy of 74.06%.

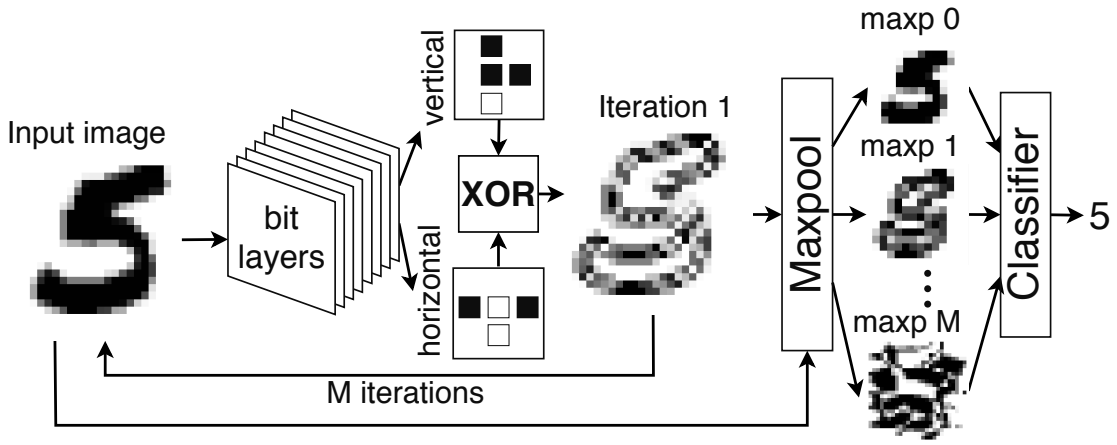


Fig. 3.1: The ReCA image classification scheme proposed in [108, 109] and improved in this section.

Using a more standard linear classifier and encoding the image data into the reservoir in a way that respects the spatial information, the ReCA image classification scheme proposed by [108, 109], illustrated in Fig. 3.1, achieved a much more competitive classification accuracy of 97.3% on MNIST. Additionally, they propose an energy-efficient specialized hardware architecture for FPGA. Although it provides a low computation cost reservoir, the inflated amount of features considerably increases computation in the used linear classifier. Additionally, the hardware design they propose, although being quite efficient, has room for improvement. Specifically, it reads the same data multiple times and is limited to work with Rule 90. The research described in this section proposes optimizations to their work.

The ReCA Image Classification Scheme

Fig. 3.1 illustrates how the ReCA image classifier processes the 28×28 8-bit grayscale images of the MNIST handwritten digit dataset [83]. First, the input image is split into eight channels: one for each of the 8 bits of the pixels. Each channel is processed independently. Each layer's data serves as the initial state of the ECA, which is applied to rows and columns separately with a fixed boundary condition. The two resulting images are combined with a bitwise `XOR` operation. This process is repeated for M iterations, which in addition to the original input, form a set of $M + 1$ images. Lastly, a pooling layer with a 2×2 window, stride 2, and no padding is applied to this set to get the feature vector for the readout.

To determine which rule to use, [108] tested all possible rules as reservoirs (except those considered useless or redundant due to symmetry) and found Rule 90 to be the

best-performing reservoir. A reproduction of this system developed as preliminary work for the research described here confirmed these results.

Experiments in [75, 113, 114, 165] also found Rule 90 standing out in different tasks. One explanation for the better performance of Rule 90 might be its self-replicating nature. From a single non-zero cell, this rule generates the fractal known as the Sierpiński triangle, shown in Fig. 2.3. When applied to multiple pixels, replicated data expands while also interacting with other replicated bits, which may explain the efficiency at feature expansion of Rule 90.

The remainder of this section describes an image classifier based on ReCA that, building on this one, achieves similar accuracy while requiring just a fraction of the computation. Additionally, it describes an even more efficient and smaller architecture optimized for FPGA.

3.2.2 Proposed Algorithmic Optimization

The implementation proposed in [108, 109] uses a linear classifier as readout, trained using a softmax regressor implemented with the limited memory BFGS (L-BFGS) optimizer provided by the Scikit-Learn [124] library for Python. The MNIST dataset is split between training data and test data at a $6:1$ ratio, and the training dataset is augmented twice using elastic distortion. A reproduction of this implementation obtained a top-1 test accuracy of 97.3% using the reported hyperparameters (a regularization of $C = 0.04$, and a maximum number of iterations of 1000).

This readout uses all the pixels of the $M + 1$ images to feed a linear classifier, thus having a feature vector of $f = \frac{(M+1)\cdot h\cdot w}{4}$ elements, where h and w are image height and width, respectively. This vector is linearly combined with a $f \times c$ weight matrix to get a score for each of the $c = 10$ classes, being the highest scoring one the classification result. Although this classifier is simple compared to multi-layer neural network classifiers, it has a high computational cost compared to the reservoir. The numbers of multiplications and additions are given by

$$\#\text{ multiplications} = f \cdot c \quad \text{and} \quad \#\text{ additions} = (f-1)\cdot c. \tag{3.2.1}$$

The number of operations needed for linear combination scales linearly as the number of features increases, making it ill-suited for a feature expansion model such as ReCA.

For solving the computational cost and scalability issues mentioned above, the ReCA classifier proposed in this thesis uses random forests with binary features extracted from a set of features augmented by ECA.

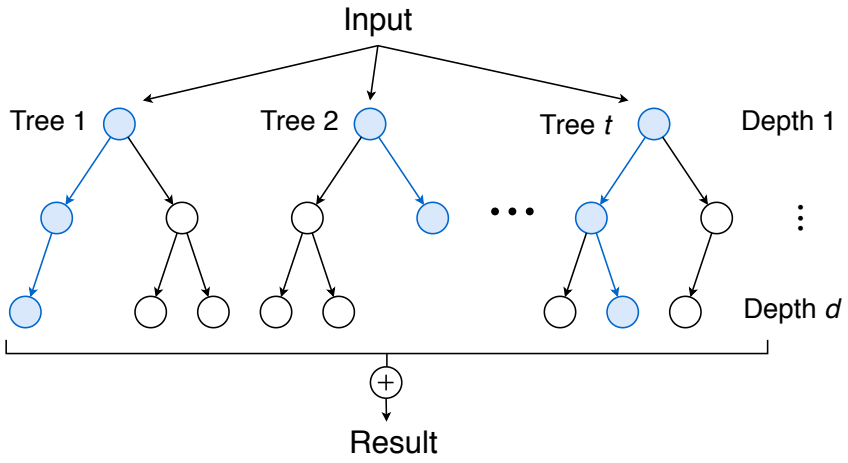


Fig. 3.2: Random forest (RF) classifier: an ensemble of t decision trees of a maximum depth of d nodes, trained with bootstrapping and bagging.

Random Forest Classifier

Aiming to reduce the computational cost of the ReCA readout, this section proposes to use a random forest (RF) classifier instead of a linear model. RF consists of an ensemble of decision trees (see Fig. 3.2), trained using bootstrap aggregation and feature bagging. Bootstrap aggregation (also called bagging) trains each decision tree using a random subset (with replacement) of the training data, whereas feature bagging trains each of them using a random subset (with replacement) of the features. These methods help build an ensemble that corrects the tendency of decision trees to overfitting and also serves as implicit cross-validation.

The decision trees are traversed by comparing a weight and a feature value at each node until reaching a leaf node, which determines the predicted class. Once all trees have been fully traversed, the prediction of the ensemble is determined through voting.

Thus, random forests use comparators as their fundamental operation. Since comparison is performed similarly to addition in standard processors, they can be considered as equivalent operations. Then, the number of operations in the worst case for RF is given by

$$\#\text{ multiplications} = 0 \quad \text{and} \quad \#\text{ additions} = t \cdot d, \tag{3.2.2}$$

where t is the number of decision trees, and d is their maximum depth. RF entirely eliminates multiplication operations, which are the most computationally expensive. Furthermore, its complexity is solely determined by the number and size of the trees, independently of the number of features.

This system was implemented with Scikit-Learn's function `RandomForestClassifier`, and a custon NumPy [48] implementation in C++ of ECA accelerated with the Boost libraries [12]. Using $M = 16$ ReCA iterations, an RF classifier with $t = 400$, $d = 20$, and entropy as criterion for splitting nodes, achieves a test accuracy of 97.0%, showing only a slight change in accuracy despite the large computation reduction.

Single-Bit Features

Since RF uses no multiplication operations, it is unnecessary to merge back the bits coming out of the ReCA to interpret them as 8-bit unsigned integers. Treating each output pixel as an independent feature provides the readout with 8× more features without increasing the number of ECA iterations. While this growth of features does not increase RF's complexity, as explained previously, it gives the RF a finer granularity to choose more effective parts of the reservoir.

More importantly, it reduces the number of operations required for classification. The number of comparisons needed in the worst case is still determined by $t \cdot d$, but with single-bit features, these comparators are single-bit. Since an 8-bit comparator is built with eight single-bit comparators (full adders), in custom hardware the number of operations can be reduced by a factor of 8× if using single-bit features (see Fig. 3.3). Making the computation at a bit level is straightforward in a specialized hardware implementation and also simplifies the maxpool layer by avoiding the 8-bit unsigned reinterpretation.

The RF classifier described achieved a test accuracy of 96.0% when trained with single-bit features.

Iteration Pruning

Unlike data augmentation, which only adds computation to the training phase, feature augmentation may increase the model's complexity both in training and inference. Although

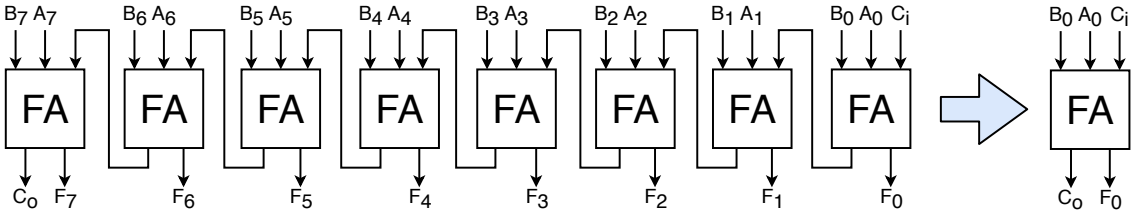


Fig. 3.3: An 8-bit ripple-carry adder is made of 8 full-adders. For single-bit computations, only one full-adder is needed.

this is not a concern with an RF readout, extra features also require more memory and data transfers. Therefore, it is essential to have a mechanism that discerns the relevant augmented features and prunes the rest. Since RF does this by design, being able to rank the features by importance, a straightforward approach is to prune the features regarded by the RF as less relevant. Nonetheless, this research suggests a more aggressive approach based on experimental observation.

Not all ECA iterations contribute equally to the readout. Training independent linear classifiers for each iteration of Rule 90 and comparing their classification error suggests that iterations 0 (the original data), 8, and 16 make a significantly bigger contribution than the rest (Fig. 3.4). This pruning method is easier to implement in hardware, as it only requires ignoring the output of some iterations.

Training the RF classifier only on iterations 0, 8, and 16, thus pruning 82% of features, achieves a test accuracy of 96.7%, only slightly affecting classification accuracy. This suggests that the RF was not selecting these features in the first place, considering them less important. Pruning features can vastly reduce the memory usage in hardware, improving speed and power efficiency.

It is interesting to notice how the classification error per iteration loosely follows the tendency of self-similar Gould's sequence, which counts the number of active cells at each layer of Sierpiński's triangle. The intuition behind this is that valleys of this sequence (found at powers of two) correspond to the iterations when the reservoir is less chaotic. This observation could be used to predict the iterations with a higher contribution to accuracy,

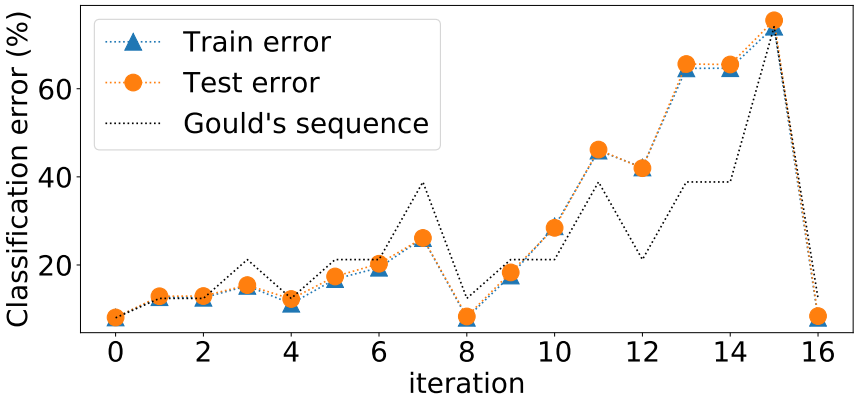


Fig. 3.4: Classification error for each of the single-iteration trained classifiers, and Gould's sequence scaled for comparison.

pruning the rest to keep the feature vector size from exploding when increasing the number of ECA iterations in the reservoir.

Algorithmic Results

Table 3.1 compares the accuracy, number of operations, and size of the feature vector for all the classifiers described in this section. The shown number of additions corresponds to the number of single-bit add operations. After the algorithmic improvements introduced, the ReCA classifier uses 100% fewer multiplications (i.e., no multiplications), 97% fewer additions, and 82% fewer features, with a slight accuracy drop of 0.6%. These reductions directly translate into cuts in computation and memory usage, guaranteeing a faster and more energy-efficient hardware implementation.

TABLE 3.1: Accuracy, number of operations, and memory required for features for each classifier.

Classifier	Top-1 Acc. (%)	# Mults.	# 1-bit Adds.	Features (Bytes)
Linear Combination	97.3	33 320	266 480	3332
Random Forest (RF)	97.0	0	64 000	3332
RF, binary features	96.0	0	8000	3332
RF, pruned bin. features	96.7	0	8000	588

3.2.3 Proposed Architecture for Reservoir Layer

This section describes the hardware architecture for ReCA feature generation, which targets FPGA implementation.

The ReCA classifier hardware proposed in [108, 109] has three main inefficiencies. First, although it can process an input in a single iteration, this requires loading the whole image on the hardware and a larger area. This is especially problematic when upscaling to images bigger than MNIST. This research proposes an architecture that, while being smaller, does not require additional data transfers.

The reference work implements Rule 90 with XOR gates, which perform an equivalent operation. This does not optimize FPGA's resources and limits the described hardware to compute using this rule. This research tackles both problems with an optimized processing

unit.

Finally, the reference implementation used 40 DSP blocks for the MAC operations of the linear classifier. These are rendered unnecessary just by choosing an RF classifier instead. It shall be noted that this work does not describe any hardware implementation for RF, as its implementation is entirely independent of the rest of the work described. Since this research proposes using a standard RF classifier, any existing or future optimized implementation of the algorithm is appropriate.

Architectural Principles

In order to achieve a fast and efficient implementation, this architecture is designed with the following three considerations:

- **Level of Parallelism:** The ReCA model can be highly parallelized, since all the ECA are independent of each other and channels are processed independently. Ideally, a matrix of 28×28 ECA processing units could be employed for each layer, performing an iteration in a single cycle. However, that circuit would have a large area. The proposed architecture parallelizes at layer and column levels by using an array of 8 processing elements (Fig. 3.5 and Fig. 3.6), which process the 8 channels of one line in parallel.

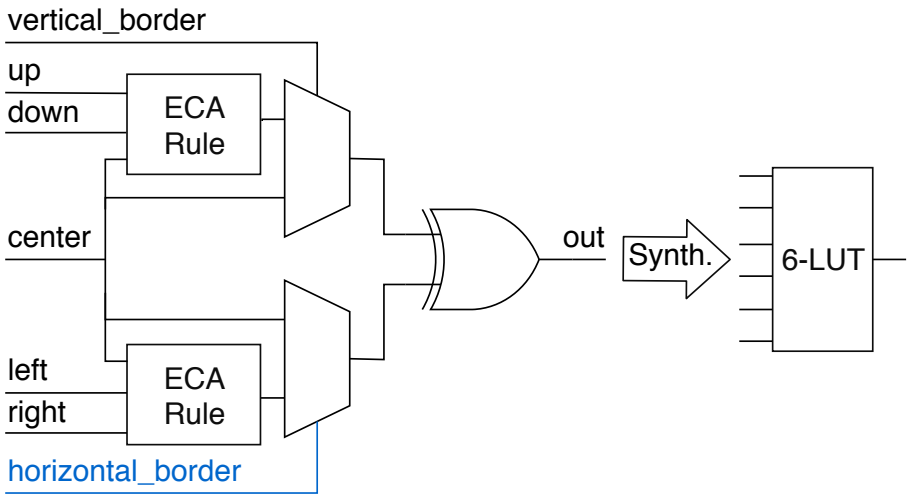


Fig. 3.5: ECA processing unit (ECA-PU). Since the horizontal border is fixed, this circuit is synthesized as a 6-input LUT.

- **FPGA Mapping Optimization:** To reduce FPGA resource utilization, the ECA processing unit has been designed to make maximal use of FPGAs' fundamental building block: the logic block (LB). Specifically, it targets FPGAs whose LB has a 6-input look-up table (6-LUT). Since the design parallelizes at the line level, the horizontal boundary is statically determined by the ECA processing unit's horizontal position. By setting the horizontal boundary condition as a fixed input, each ECA processing unit is synthesized as a 6-input, 1-output circuit, and thus is efficiently mapped to a single 6-LUT (Fig. 3.5). It shall be noted that this is independent of the chosen ECA rule. Additionally, this functionality could be beneficial for accelerating experiments that use reservoirs that apply a combination of different rules, like those discussed in Chapter 2.1.1.

- **Memory Transfer Optimization:** Memory read and write operations constitute the main bottleneck for hardware acceleration and also are the most energy-hungry. This issue is worse in the case of off-chip memory reads. For this reason, it is crucial to reduce to a minimum the number of times that the accelerator loads data. The proposed architecture optimizes the datapath by processing each iteration in a single pass, loading each pixel only once.

  To achieve this, a FIFO buffers three lines of the input (Fig. 3.6), which is all the data needed to output one line. This data is fed to an array of 28 ECA-PUs, which calculate the next state for each pixel of the central line in a single cycle, after which the next line is loaded. By the time a pixel leaves the line buffer, all its neighboring ECAs have been updated, and thus is never loaded again.

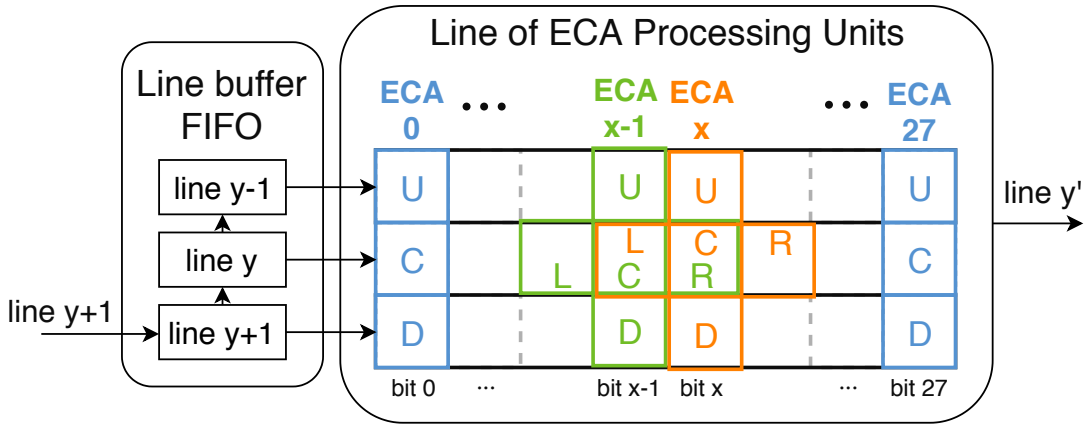


Fig. 3.6: The Processing Element (PE) is formed by a 3-line buffer FIFO and a line of 28 ECA-PUs. U, C, D, L, and R stand for up, center, down, left, and right, respectively.

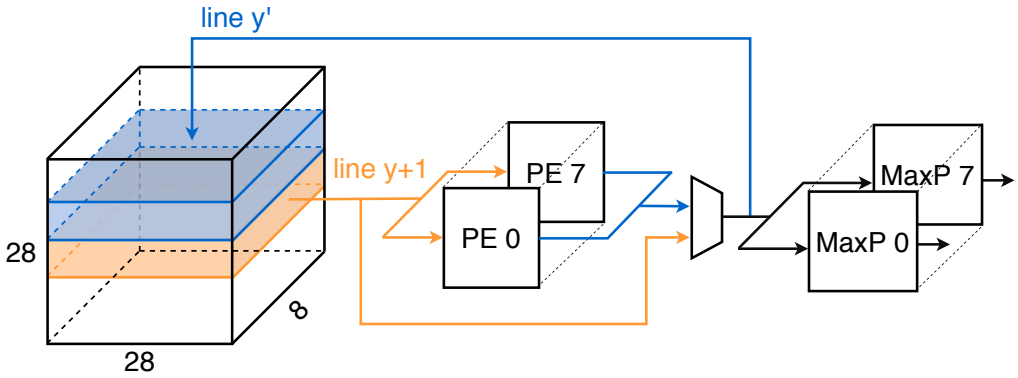


Fig. 3.7: Diagram of the ReCA accelerator architecture.

ReCA Architecture Design

As illustrated in Fig. 3.7, the proposed hardware processes each of the eight layers parallelly. The pipeline starts at a double-port BRAM, which contains the input image. Every clock cycle, the PE fetches and processes one image line. The processed line is written back to the BRAM for the next iteration and also fed to the maxpooling unit. Since the maxpool window has a height of 2 and no overlap, the maxpool unit outputs feature data every two cycles.

This design takes 28 cycles to process each iteration, totaling $17 \times 28 = 476$ cycles per input. The first line of the original input is fed in parallel to the PE and the maxpooling unit to avoid an extra latency cycle. This design was implemented in Verilog, verified using ModelSim, and synthesized in Xilinx's Vivado. Verification software was designed using Python and Verilog, checking that both software and hardware implementations produced the same output for the same input images.

Table 3.2 compares resource utilization for the synthesis of the proposed design and a reproduction of the architecture proposed in [108, 109]. By combining vertical and horizontal ECA processing in a single unit and optimizing it for 6-LUT, this design reduces almost 60% of LUT usage.

TABLE 3.2: FPGA resource utilization comparison.

Implementation	# CLB LUTs	# CLB Registers	# BRAM
Reproduction of [108, 109]	971	822	8
This work	392	822	8

3.2.4 Results and Discussion

This section proposes algorithmic optimizations that completely eliminate multiplications and most of the rest of arithmetic operations and vastly reduce the memory required for the ReCA classifier's weights, with a minimal accuracy tradeoff. Additionally, it describes an architecture that optimizes dataflow and FPGA mapping. Fig. 3.8 summarizes the improvements of the proposed system. The resulting system gets a competitive classification accuracy of 96.7% on MNIST while requiring very small computation, minimal data transfers, and exploits the parallelism of the ReCA. It is fair to say that this proves that RC is not just a promising model for sequential data but also potentially a good alternative to CNN for image processing applications.

Future work should investigate the parameters that have not been considered in this work. Different ways of mapping images into the ReCA, such as the scanning used by [69], should be examined, as well as different ways of constructing this reservoir. Functions other than `XOR` for combining the vertical and horizontal ECA might work better, as well as a different kind of boundary. Reservoir based on other kinds of CA or that use more than one ECA rule should also be investigated for image processing applications. Furthermore, a full system including the RF readout should be deployed in FPGA for more thorough energy efficiency evaluation.

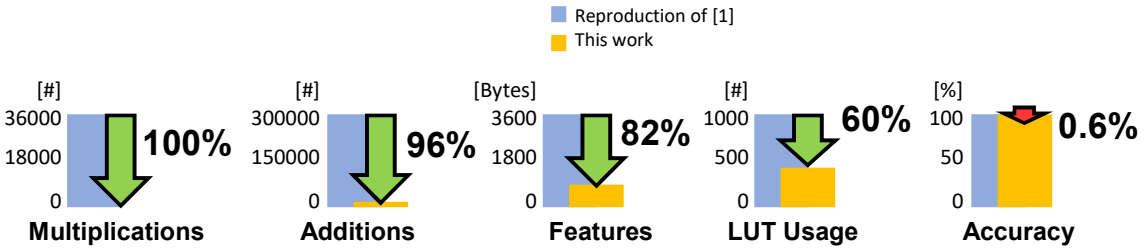


Fig. 3.8: Summary of the results obtained in [97]. [1] in this figure refers to [108, 109]

Impact

The work in this section was presented as a conference paper [97] at the *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Online, October 2020.

This research was supported by Japan Science and Technology Agency (JST) CREST Grant Number JPMJCR18K2, Japan.

3.3 The Scalability Problem of Reservoir Computing

3.3.1 Upscaling ReCA

Image classification on MNIST is frequently used as a proof-of-concept experiment for novel methods. With accuracy scores of around 97%, it is fair to claim that the ReCA image classifier discussed in the previous section has passed this test and may be a viable alternative for image processing. However, for practical use on computer vision applications it is necessary to upscale the simple single-layer ReCA architecture. This section covers some attempts of upscaling ReCA into a full-scale computer vision model, represented in Fig. 3.9.

Using a Bigger Reservoir

Unlike MNIST, which has a single channel, typical image data has three channels (e.g., RGB), but as explained in Chapter 2.1.1, a CA of higher dimensionality has a search space too big to practically search for an optimal reservoir configuration. One way of adapting the ReCA classifier to use three channels is to add another level of parallelism, having a separate reservoir for each channel. This prevents the interaction of information in

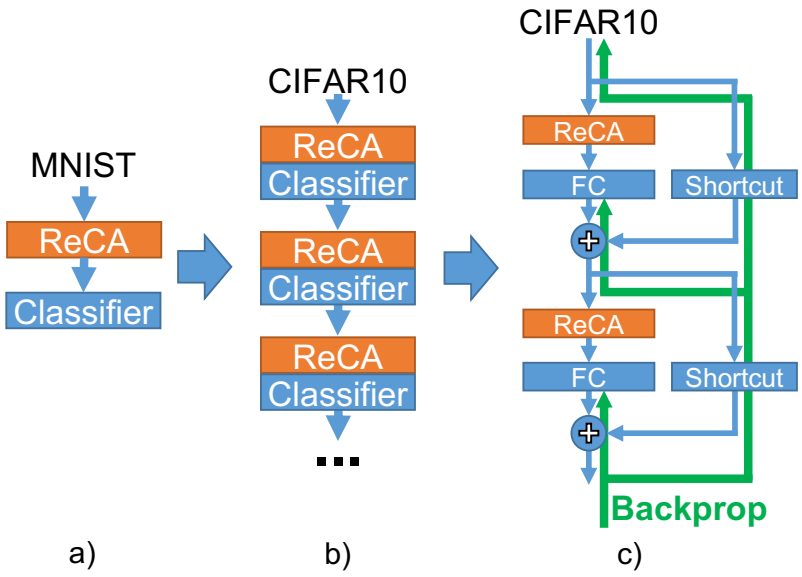


Fig. 3.9: Attempts of upscaling of ReCA image classifier to CIFAR-10. (a) Original single-layer ReCA; (b) Deep ReCA; (c) Proposed deep residual ReCA.

different layers, entirely leaving this to the classifier. A different approach is to increase the reservoir's size—i.e., increasing the CA lattice—which could also be a way of scaling for larger inputs. However, the complexity of the patterns in a larger reservoir's output grows non-linearly, quickly becoming an impossible task for the simple linear classifier at the readout.

Deep Reservoir With Per-Layer Training

Deep learning (DL) models have found great success in upscaling via stacking multiple layers. Following, there have been some attempts at implementing multilayer RC. For instance, [154] stacks multiple RC for phoneme recognition, and [114] stacks multiple ReCA. Most notably, [69] stacks multiple ESN layers to obtain a classification error of 0.81% on MNIST, much better than the ReCA image classifier, and close to state-of-the-art performance.

However, all of these models suffer from the same problem. DNN are fully traversed by backpropagation, optimizing each layer to intermediate representations that as a whole form a hierarchy of features. By contrast, all layers of existing multilayer RC are trained towards the same target. In other words, the input of an additional layer is the predicted label scores of the previous layer, so it only learns how to adjust for errors in this prediction. Consequently, none of the multilayer proposals referenced above measure accuracy improvements past the third additional layer. Stacking RC in error-correction mode does not form a hierarchy of features and thus does not scale RC to deep RC (DRC)—it just improves accuracy slightly.

Deep Reservoir With Backpropagation

Although for a model based in neural networks, such as ESN, it is plausible to propagate error gradients through a multilayer structure (even theoretically for ReCA, as CA can be computed as an artificial neural network (ANN) [127]), this may be difficult for some algorithmic reservoirs, and impossible for physical reservoirs. To successfully upscale RC into DRC it is necessary to investigate how to build a multilayer structure that forms hierarchical features with the limitation of having no gradients being propagated through the reservoirs.

Deep Residual Reservoir With Backpropagation

A solution for using backpropagation on a multi-layer reservoir without propagating gradients through the reservoirs is to employ residual connections, such as the ones used

by ResNets [53]. These shortcuts provide a path for the backpropagation algorithm to propagate gradients as deep as necessary, hopefully forming a feature hierarchy.

This idea, depicted in Fig. 3.9c, was implemented using the same ReCA as the classifier in the previous section adapted to the PyTorch [123] deep learning library, fully connected (FC) layers as each layer's readout, and the CIFAR-10 [78] image classification dataset as input. Even for a single layer this model required over 6 hours per epoch of training on a GPU, an impractical high computational cost. Most of this computation is spent at the ReCA: even with the highly-optimized C++ implementation used, the intrinsic distributed nature of the cellular automata algorithm makes the computational cost rise very quickly when growing the lattice size to accommodate for the larger inputs.

To train a ResNet-18-like DRC architecture trained for at least 100 epochs on CIFAR-10 in a practical time of around 100 hours, it can then be estimated that to make a practical deep residual ReCA simulation it would be necessary a 100× speedup of the reservoir computation, a task that exceeds the scope of this dissertation.

3.3.2 Upscaling ReFRET

On the other hand, physical reservoirs still have to be upscaled to the proof-of-concept task of MNIST. This work attempted to upscale the ReFRET introduced in Chapter 2.1.2 by using the FRET Monte Carlo simulation proposed in [111, 112] to substitute the reservoir in the ReCA image classifier with a simulated physical reservoir: a ReFRET image classifier, illustrated in Fig. 3.10.

After modifying the original algorithm for enhanced computational performance [111], an

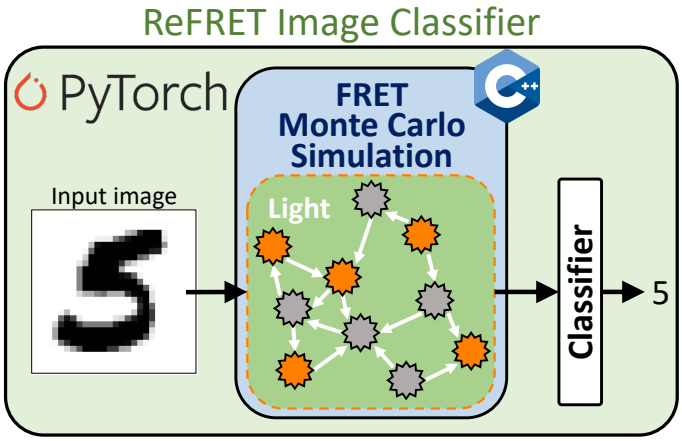


Fig. 3.10: Outline of the ReFRET MNIST image classifier implemented.

implementation of the photoelectronic simulator of [111, 112], upgraded with a photodiode array simulation, was sped up with the multi-threading library OpenMP [21]. It was also adapted as a PyTorch library in hopes of upscaling it to a deep learning architecture, but on the light of the failure of upscaling ReCA, a simple readout was implemented using classifiers from the Scikit-Learn [124] library. In line with the proposed ReCA improvements, a random forest classifier resulted in the best results (compared to ridge regression and L-BFGS classifiers), but this implementation was still incapable of passing the proof-of-concept test.

Faster simulations (with fewer Monte Carlo trials) failed to provide FRET simulation results accurate enough for reservoir computing—which is generally considered to require a deterministic system—, and simulations with the required high accuracy demanded an increase in computation time of several orders of magnitude. Despite the proven reliability of the underlying mathematical modeling of the physical phenomenon, and the numerous software optimizations, it was estimated that to make a practical ReFRET proof-of-concept simulation it would be necessary a 1000× speedup of the physics simulation, a task that exceeds the scope of this dissertation.

3.4 A New Approach to Efficient Deep Learning

The previous sections demonstrate that reservoir computing (RC) is a viable and efficient alternative to deep neural networks (DNN) for computer vision (CV). However, they also show that there are currently many obstacles that prevent the promotion of RC from fundamental research to practical applications.

This dissertation proposes an approach to take immediate advantage of the benefits offered by RC by mimicking its key elements in the design of DL models, which are currently seeing an explosion in practical commercial applications. This RC-inspired approach to DL is developed progressively in the following chapters, with a focus on CV. Specifically, this research is mainly presented using the ResNet [53] neural architecture applied to image classification tasks [78, 132], as this combination forms the core of most CV applications.

Fig. 3.11 illustrates how each of the remaining chapters introduces RC elements to DL, making a parallelism with the ReCA classifier introduced in this chapter. These key elements are:

1. **Random Fixed Weights:**
   This dissertation largely capitalizes on the Strong Lottery Ticket Hypothesis (SLTH), introduced in Chapter 2.2. Like reservoir computers, SLTH-based models employ

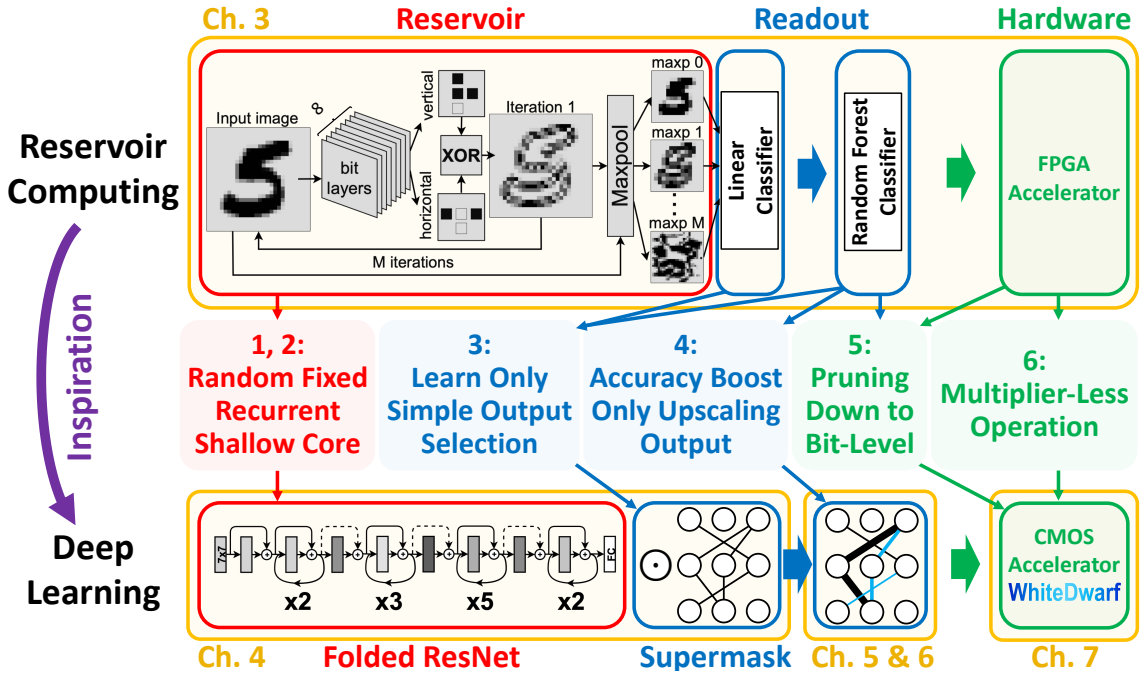


Fig. 3.11: Outline of the RC-like elements brought into DL models in the following chapters of this dissertation.

random parameters that are never updated. Instead, only the network connectivity is learned. Chapter 4 introduces the first RC-like model by taking advantage of this similarity. Furthermore, Chapter 6 expands the SLTH to three types of neural randomness, opening the door for a wide variety of RC-like DL models.

2. **Shallow Recurrent Architecture:**
   Deep neural networks employ tens or even hundreds of layers, the source of their immense model size and computational cost. On the other hand, the visual cortex of the primate brain is thought to be formed by only 4–6 layers. Furthermore, where DNNs for CV are typically strictly feed-forward, its biological counterpart is thought to have a rich variety of recurrent connections. Chapter 4 takes advantage of Folding, a neuro-inspired DL method, to transform DNNs into recurrent architectures. The resulting models are shallow, like reservoir computers, and their computational layers are dynamical systems, like the complex systems in reservoirs.

3. **Learning Based on Simple Selection:**
   The only parameters trained in a RC are the ones of its simple output classifier.

Generally, this readout layer is formed by a linear classifier, but this chapter proposes using a random forest classifier (RF). The RF is capable of bearing the whole learning load of the system just by learning to choose the right elements coming out from the complexity of the reservoir. In a similar manner, the SLTH-models are trained by just training a binary selection mask—a supermask. Chapter 4 takes advantage of this similarity with RC, and leverages its synergy with the RC-like recurrent architecture to boost the SLTH.

4. **Model Boosted Only by Upscaling Output Mechanism:**
   The algorithmic improvements of the ReCA classifier presented in this chapter improve its accuracy and efficiency by just enhancing its readout (i.e., the change to the RF classifier, the single-bit features, and the proposed pruning). Similarly, the presented RC-like DL models based on the SLTH can be boosted by just enhancing their supermasks. Chapter 5 proposes a new type of scalar supermask that raises the accuracy of the proposed models to match standard DL models on large-scale image classification, setting the SOTA for SLTH models. Chapter 6 builds further in this direction by proposing a system of three elemental supermasks, unlocking a wider variety of DL-inspired DNNs and surpassing the SOTA results set by Chapter 4 and Chapter 5.

5. **Bit-Level Fine-Grained Pruning:**
   The RF classifier with single-bit features proposed in this chapter allows to vastly reduce the memory size and computation requirements of the ReCA classifier by exploiting the sparsity of binary representations and bit-level pruning. These benefits are then harvested on the proposed FPGA accelerator. Chapter 7 takes inspiration on this key element and presents *WhiteDwarf*, a CMOS neural engine for the RC-like models presented in this dissertation that capitalizes on fine-grained bit sparsity for inference acceleration and energy reduction. Like the proposed ReCA, *WhiteDwarf* removes from the computation unnecessary elements at the fine-grained bit-level.

6. **Multiplier-Less Hardware:**
   The biggest contribution of the improvements proposed in this chapter for the ReCA algorithm is the 100% reduction of multiplication operations: this expensive arithmetic operation is completely unnecessary in RC, whether at the reservoir or the readout. The *WhiteDwarf* hardware architecture proposed in Chapter 7 also infuses this RC-like element into neural acceleration by adopting a multiplier-less processing element that exploits the binary nature of supermasks to bear the neural computation with only binary-shift operations.

In addition to proposing and refining RC-inspired DL, adopting these elements uncovers multiple contributions to pure DNN models along this dissertation. The ideas proposed in Chapter 4, Chapter 5, and Chapter 6 make important conceptual contributions to the SLTH, set the SOTA in image classification accuracy for these type of models, and present some of the most competitive tiny DNN models, with a memory size of just around 2 MB. Most remarkably, these ideas culminate in Chapter 6 in a novel framework for concurrent training, pruning, and weight quantization of generic DNNs, then exploited for hardware acceleration and tested on a manufactured CMOS chip in Chapter 7.

Chapter 4

Hidden-Fold Networks: Random Recurrent Residual Networks Contain Stronger Lottery Tickets

This chapter presents the cornerstone of this dissertation: a novel method for transforming a deep residual neural network—the leading architecture for computer vision application backbones—into a reservoir computing (RC)-like architecture. Namely, it brings the following RC-like elements into deep learning (DL):

1. Shallow architecture
2. Recurrent dynamics
3. Random fixed parameters
4. Training based on simple selection

The resulting models achieve similar accuracy to the original models, while achieving model memory size compression ratios close to 50×.

4.1 Introduction

Accurate neural networks can be found just by pruning a randomly initialized overparameterized model, leaving out the need for any weight optimization. The resulting subnetworks are small, sparse, and ternary, making excellent candidates for efficient hardware implementation. However, finding optimal connectivity patterns is an open challenge. Based on the evidence that residual networks may be approximating unrolled shallow recurrent neural networks, this chapter conjectures that they contain better candidate subnetworks at inference time when explicitly transformed into recurrent architectures. This hypothesis is put to the test on image classification tasks, where the method proposed in this chapter finds

subnetworks within the recurrent models that are more accurate and parameter-efficient than both the ones found within feedforward models and than the full models with learned weights. Furthermore, random recurrent subnetworks are tiny: under a simple compression scheme, ResNet-50 is compressed without a drastic loss in performance to 48.55× less memory size, fitting in under 2 MB.

The Strong Lottery Ticket Hypothesis (SLTH) [128, 177] states that high-performing neural networks can be obtained just by pruning an overparameterized dense model, as they are already available hidden inside of the randomly initialized parent network. This claim supersedes the Lottery Ticket Hypothesis [32], as weight training is entirely unnecessary. These sparse, random, and tiny subnetworks achieve competitive performance in vision tasks and can be exploited for very efficient inference hardware implementation [56].

However, current training methods have trouble finding the optimal connectivity patterns [31]. It has been found that better-performing random subnetworks can be discovered by constraining the weight precision to ternary via the combined effect of random binary weight initialization and a learned binary mask [128].

This chapter shows that even higher-performing subnetworks can be obtained by imposing two additional constraints: one in the hypothesis space and one in the search space. Both constraints are imposed simultaneously by transforming a residual network into a recurrent architecture, restraining the hypothesis space to iterative functions to be searched within fewer random weights. This chapter makes the observation that residual networks are architecturally inclined to learn ensembles of all the possible unrollings of a shallow recurrent neural network, based on which this restriction on the hypothesis space actually increases the number of candidate subnetworks of interest available at initialization time.

The resulting subnetworks are parameter-efficient and have a competitive performance on image classification tasks, supporting the arguments for embracing feedback connections in computer vision. Furthermore, they can be compressed to tiny memory sizes and have a high degree of parameter reusability, making them even better candidates for inference acceleration.

The remainder of the chapter is organized as follows. Section 4.2 reviews the theory surrounding residual networks and proposes a novel view that explains the effectiveness of the proposed method. After Section 4.3 presents a method for obtaining high performing, sparse, random, recurrent residual subnetworks, Section 4.4 explores it in detail. Finally, after a discussion, Section 4.5 concludes the chapter.

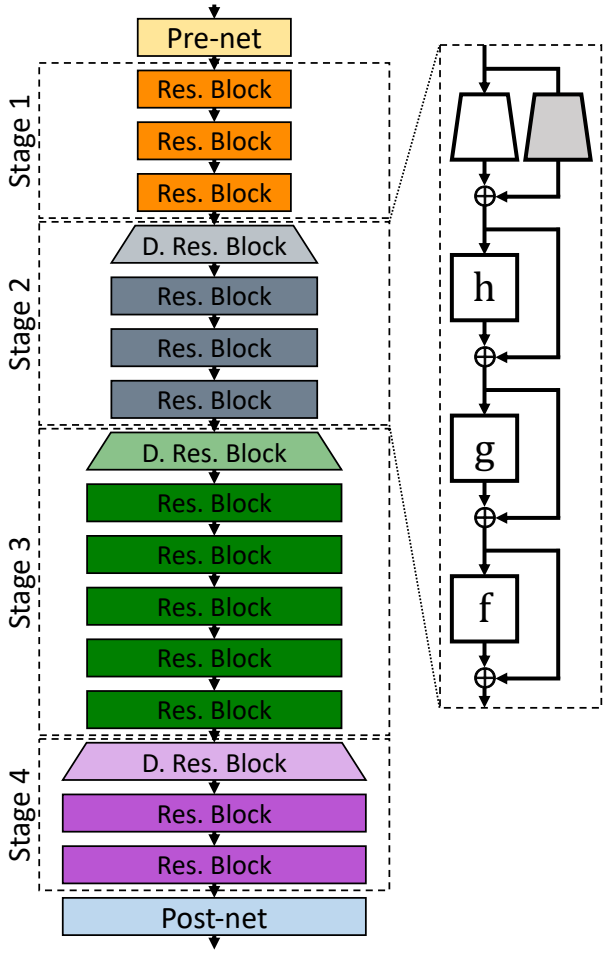


Fig. 4.1: The residual network architecture (ResNet). Specifically ResNet-50, which has 3, 4, 6, and 3 blocks in each stage, from input to output.

4.2 Residual Neural Networks: A New View

Among the ever-growing variety of neural network architectures, the residual neural network architecture (ResNet) [53], depicted in Fig. 4.1, remains the backbone of many SOTA models. ResNet is a pyramidal feedforward deep convolutional neural network formed by a convolutional pre-net followed by four stages of residual blocks and finished with a fully-connected post-net classifier. Each of the four stages uses a different representation space size, adjusted by the first residual block by downsampling the feature map and doubling the number of channels. The remaining blocks within a stage, of identical shape and size, keep the same representation space size.

Residual blocks are formed by a concatenation of BatchNorm, ReLU, and convolutional layers. Specifically, this research uses the bottleneck block [53], composed of three convolutional layers with kernel sizes 1×1, 3×3, and 1×1, in order. Each residual block has a skip-connection: an identity function in parallel to the block that adds its input to its output. In order to adjust the different representation space sizes, downsampling blocks have a learnable layer in the skip-connection—a projection shortcut.

4.2.1 The Representation View

The original intuition behind this architecture was that skip-connections provide a clear path for backpropagation for reaching a layer directly, thus solving the vanishing gradient problem. Based on the *representation view*, which directly associates layer depth with level of representation, a deeper network is able to form a deeper feature hierarchy and thus can solve more complex problems. Indeed, skip connections allowed to successfully train ResNets of thousands of layers [53]. Furthermore, a continuation of the original work [54] updated the original architecture with a full pre-activation structure, relocating all activation layers into the residual blocks to leave a clear path of identity shortcuts. In the forward pass, this path could be seen as an implementation of all the feedforward lateral connections observed in the visual cortex [73].

However, this is not the only way to interpret ResNet. Lesion studies showed that removing or shuffling the order of residual blocks does not have an acute effect on its performance [158]. Instead, there is a proportional relationship between the introduced corruption and the performance loss. This phenomenon is not observed in other feedforward models, where similar lesions are critical. From these results stem two alternative views of ResNet: some authors have defended that ResNet is an ensemble of shallow networks, while others have argued that it may be an approximation of an unrolled shallow recurrent neural network. This section attempts to reconcile these two views into a single one.

4.2.2 The Ensemble View

Each residual block can be viewed as a path bifurcation. Then, ResNet can be interpreted as an ensemble of all the possible paths within it [158], i.e., 2^n paths for a model with n residual blocks.

If considering a chain of three full pre-activation residual blocks approximating functions

h, g, and f, their composition can be written as

$$\begin{aligned}
&(f+I)\circ(g+I)\circ(h+I)\\
=\ &(f+I)\circ(g\circ(h+I)+(h+I))\\
=\ &f\circ(g\circ(h+I)+(h+I))\\
&\quad+g\circ(h+I)\\
&\quad+(h+I),
\end{aligned} \tag{4.2.1}$$

where $\circ$ denotes function composition, and I is the identity function with the same domain and codomain as h, g, and f. Residual blocks perform a transformation on the previous output, but this transformation is not compositional, as they contribute their output additively—as a new element of an ensemble. Indeed, if approximating h, g, and f as linear functions, this chain of residual blocks can be directly interpreted as the ensemble of all possible paths:

$$\begin{aligned}
&(f+I)\circ(g+I)\circ(h+I)\approx\\
&(f\circ g\circ h)+(f\circ g)+(f\circ h)+(g\circ h)+f+g+h+I.
\end{aligned} \tag{4.2.2}$$

A downsampling block also contributes additively to the ensemble, but its projection shortcut transforms all the previously ensembled outputs:

$$\begin{aligned}
&(f+s)\circ(g+I)\circ(h+I)\\
=\ &(f+s)\circ(g\circ(h+I)+(h+I))\\
=\ &f\circ(g\circ(h+I)+(h+I))\\
&\quad+s\circ(g\circ(h+I)+(h+I)),
\end{aligned} \tag{4.2.3}$$

where s is the shortcut function of downsampling block f.

A parallel analysis could be done for any other feedforward neural network, but in the case of ResNet, paths are not the same length and do not go through the same layers, i.e., ResNet is an ensemble of networks of various depths. Then, it makes sense that removing a block only has a mild effect on accuracy, as it only removes a subset of the ensemble. The effect of shuffling can also be understood through the linear approximation.

Of course, residual blocks are non-linear. However, residual blocks' nonlinearity can be considered weak, as their outputs have a small magnitude centered around zero that contributes little compared to the identity function [42, 53, 70].

Several ResNet improvements have been proposed based on this view. Removing random subsets of blocks during training makes the ensembled networks shallower and acts as regularization [64]. Using multiple skip-connections per block increases the amount of

ensembled paths [4], boosting performance. Network depth can be reduced by increasing width, allowing to train bigger models to higher performance in lower time [169].

4.2.3 The Unrolled Iterative Estimation View

Another possible explanation for the lesion and shuffling results is that all the blocks within a stage—which have the same shape and receive partially the same inputs and gradients through the identity shortcuts—are approximating the same function. That is, that ResNet naturally converges to the approximation of an unrolled shallow recurrent neural network.

Proponents of this view have argued that each of ResNet's stages corresponds to a different hierarchical level of representation, composed by the downsampling block, whereas the rest of the blocks perform iterative refinement of features [42].

ResNet's iterative behavior was observed by visualizing its activations [17], and it has also been analytically and empirically demonstrated that residual blocks perform iterative refinement of features by approximating a gradient descent step during inference [70]. Furthermore, ResNet can be transformed explicitly into a recurrent architecture without a critical loss in performance [89], a transformation exploited in this chapter.

4.2.4 The Ensemble of Unrollings View

Although they have been presented as opposing views, it can be argued that these two views are compatible: ResNet can be interpreted as both an ensemble and an unrolled shallow recurrent neural network at the same time.

If $h \approx g \approx f$, (4.2.1) is rewritten as

$$\begin{aligned} & (f+I) \circ (g+I) \circ (h+I) \\ \approx\ & (f+I) \circ (f+I) \circ (f+I) \\ =\ & (f+I)^3 \\ =\ & f \circ (f \circ (f+I) + (f+I)) \\ & + (f \circ (f+I) \\ & + (f+I), \end{aligned} \tag{4.2.4}$$

which can be seen as an ensemble of three iterations, where each iteration performs an additional recursion of f over the previous ensemble. Using the same linear approximation as above,

$$(f+I)^3 \approx f^3 + 3f^2 + 3f + I. \tag{4.2.5}$$

More generally, a chain of n identical linear residual blocks can be seen as *a weighted ensemble of all the possible unrollings of a linear recurrent residual block* up to n iterations:

$$(f + I)^n \approx f^n + \sum_{i=1}^{n-1} n f^i + I. \tag{4.2.6}$$

Strictly, the repetitive application of an iterative function would compound to

$$f^n, \tag{4.2.7}$$

so the ensemble of unrollings view is a closer approximation than the strict unrolled iteration view.

This ensemble of unrollings provides a rich search space for finding strong tickets within a model with few parameters and suggests that stronger tickets will be present in a ResNet if it contains more recurrent approximations at its randomly initialized state.

4.3 Hidden-Fold Networks

This section describes a method, first proposed in [93] and then expanded in [96], that first folds a residual network to then find a strong lottery ticket within it. These folded subnetworks, which overperform the strong tickets hidden in feedforward ResNets, are referred to as *Hidden-Fold Networks (HFN)*.

Section 4.3.1 defines the recurrent ResNet architecture, and the training method employed to discover the strong ticket is described in Section 4.3.2. The importance of the weight initialization choice for constraining the search space is detailed in Section 4.3.3, after which Section 4.3.4 explains the normalization strategies considered. Finally, Section 4.3.5 clarifies the memory size compression scheme considered, which makes HFNs some of the tiniest ResNets.

4.3.1 Folded ResNet Architecture

According to the unrolled iteration view (see Section 4.2.3), the chains of residual blocks of identical shape within each stage are approximating an iterative function. *Folding* [89]∗ is a method that transforms these chains into recurrent blocks, via weight sharing. That is, $h \approx g \approx f$ in Eq. 4.2.4 is transformed explicitly to $h = g = f$, and since applying identical

∗Not to be confused with the method that compresses trained networks by removing unnecessary activation layers [25], or with the method that adds longer skip-connections [29], both also named folding.

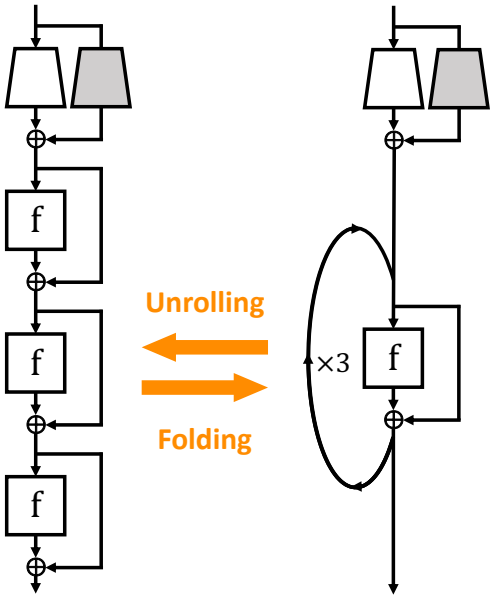


Fig. 4.2: A ResNet stage folded into a recurrent residual stage through the opposite process of time unrolling. The downsampling block is left untouched, whereas the rest of the stage is folded into a single recurrent block. That is, a stage of n blocks is folded into 2 blocks, the second of which is applied for $n-1$ iterations.

functions in succession is equivalent to applying one instance of them repeatedly, the feedforward chain can be transformed into a single recurrent block in a process opposite to time unrolling.

Since the downsampling block has a different shape than the rest of the blocks, it cannot be folded with the rest. This block could be removed by transforming ResNet into an isotropic architecture or by substituting it with a simpler block, strategies that were explored in [70, 89]. Nevertheless, based on the view that different stages correspond to different hierarchical levels of features composed by downsampling blocks [42], this paper leaves them untouched. The rest of the stage is folded into a single recurrent residual block that is iterated the same number of times as the number of original blocks, as illustrated in Fig. 4.2. Fig. 4.3 shows the structure resulting from folding all stages of a ResNet-50, but Section 4.4.3 experiments with the number of folded stages.

Folding has a double effect on the search space: it limits the set of functions that the model can learn—the hypothesis space—to iterative functions, and reduces the number of parameters. This research argues that, despite this trim in parameters means an exponential reduction of the available subnetwork candidates, folded residual networks contain better

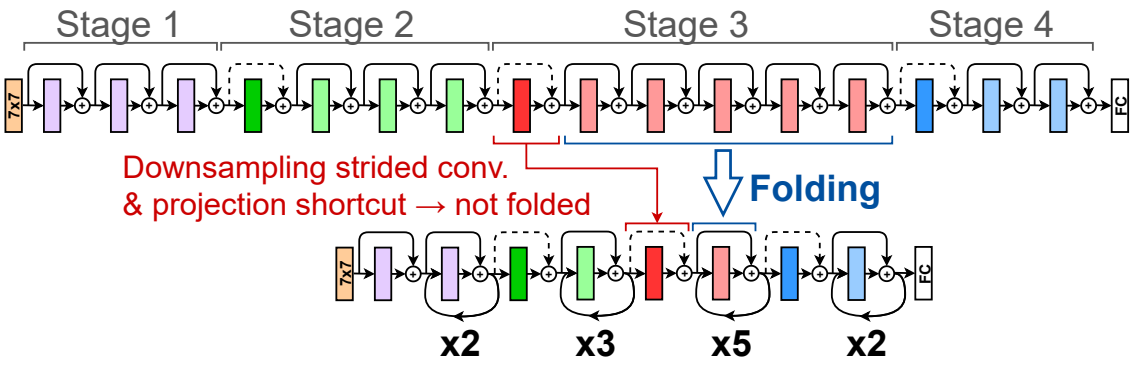


Fig. 4.3: Architecture of HFN applied to ResNet-50. ResNets are formed by an input layer, four main stages, and an output layer. Repeated identical instances of a structure are equivalent to a recurrent structure, so they can be folded. In this example, identically shaped bottleneck blocks in all stages of ResNet-50 are folded.

performing strong tickets than their feedforward counterparts. If training ResNet's weights naturally converges to approximations of unrolled iterative functions, likely, the strong tickets contained within it are also approximations of recurrent networks. This limits the number of subnetworks of interest to only those that include similar random weights in consecutive blocks, a very small subset. Folding ResNet forces *all* candidate subnetworks to have a recurrent form. Therefore, the number of candidate subnetworks of interest is larger and likely to contain stronger tickets.

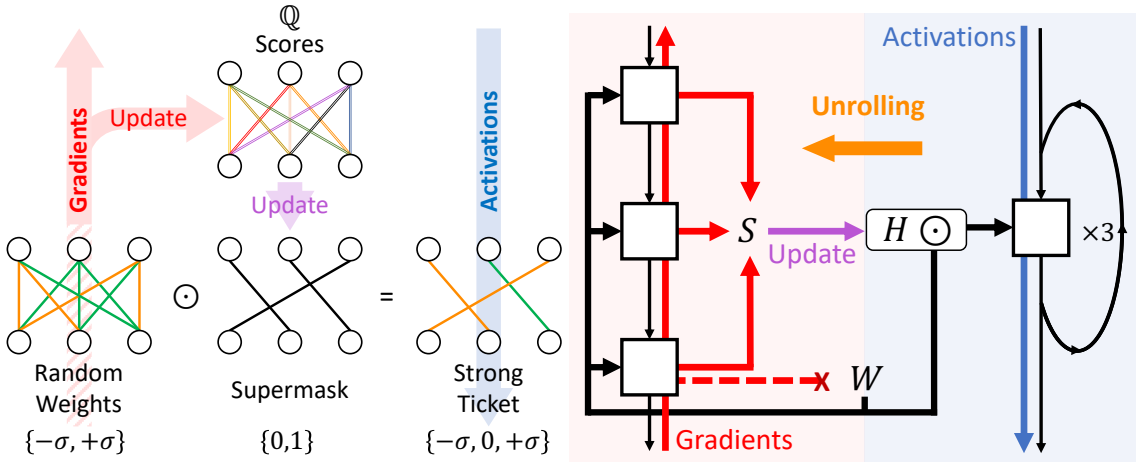
Furthermore, the parameter reduction happens not only at inference time but also at train time. At train time, the search space for strong tickets is largely reduced, making these folded tickets also easier to find. At inference time, the found subnetworks are even smaller and exploit parameter reusability, making them excellent candidates for efficient hardware implementation.

4.3.2 Supermask Training

Instead of optimizing the model's weights, the model is pruned to find a high-performing subnetwork hidden within the randomly initialized folded model—an HFN.

This connectivity pattern is learned by training a *supermask* [128, 177], a pruning mask that contains a binary element for each weight. The ticket is unearthed at inference by applying an element-wise product of the random weight tensor and the trained supermask.

The proposed method uses the `edge-popup` algorithm [128] for training the supermasks, illustrated in Fig. 4.4a. `Edge-popup` defines a score for each weight, which is updated in the



(a) Supermask training with `edge-popup` [128]

(b) Folded models are unrolled during training, so scores receive a gradient from each iteration.

Fig. 4.4: Training an HFN with a supermask. The supermask (H) includes the random weights (W) with the top-k% scores (S), updated via backpropagation. $\odot$ is the Hadamard product.

backward pass via backpropagation using straight-through estimation [10] for the supermask (i.e., the supermask is not applied in the backward pass). The scores are then sorted by absolute value, and the supermask is updated to include the weights with the top-k% highest ranking scores and prune the rest. Top-k%, thus, is a hyperparameter that sets the supermask's density a priori. Although top-k% is set globally, the sparsity is enforced at the layer level (in Chapter 7 this variant will be referred to as *Local* `edge-popup`).

Folding does not affect this process: supermasks and scores are shared in the same way as their corresponding weights, and backpropagation flows through the unrolled model in the same way it would travel a feedforward model. Therefore, folded parameters receive a separate gradient from each iteration, as depicted in Fig. 4.4b and Fig. 4.5.

This training method takes more computation time than standard dense weight learning (about 1.75× in the presented experiments), as it uses three parameters for each node, and scores must be sorted every training step. However, the resulting models are smaller, sparse, and formed by random weights, allowing for efficient inference implementations.

This algorithm is improved on Chapter 5 with scalar supermasks, and alternative versions of `edge-popup` are explored in Chapter 7.

4.3.3 Random Sign Constant Weight Initialization

Following [128, 177], this research considers two weight initialization strategies: Kaiming normal initialization (KN) [52], and signed Kaiming constant initialization (SC).

KN initializes weights of layer l by sampling from

$$\mathcal{N}_l(0, \sigma_l^2), \tag{4.3.1}$$

where $\mathcal{N}$ denotes the normal distribution, and the standard deviation is scaled by the density k [128, 177]:

$$\sigma_l = \sqrt{\tfrac{2}{n_l k}}, \tag{4.3.2}$$

where n_l is the number of input dimensions of layer l.

On the other hand, SC initializes weights to the same modulus and a random sign by sampling from

$$\mathcal{D}_l\{-\sigma_l, +\sigma_l\}, \tag{4.3.3}$$

where $\mathcal{D}$ denotes the discrete uniform distribution.

Since the same modulus σ_l is shared by all the random weights in layer l, it can be absorbed by the normalization layer's scaling factor γ. Then, all weights in the model belong to $\{-1, +1\}$, and after applying the binary supermask in inference, the effective weights belong to $\{-1, 0, +1\}$. In other words, in combination with the binary supermask, SC initialization constrains the search space to *balanced ternary neural networks*. Although this chapter argues that this strict constraint is the reason for its positive effect on accuracy, Chapter 5 proposes a better method based on finding a flexible middle ground between the continuous weights and the ternary weights—quantization.

Scores are always initialized with Kaiming uniform initialization [52] by sampling from

$$\mathcal{U}_l(-\sqrt{\tfrac{6}{n_l}}, \sqrt{\tfrac{6}{n_l}}), \tag{4.3.4}$$

where $\mathcal{U}$ denotes the uniform distribution.

4.3.4 Unshared Batch Normalization

Following [128], supermask training uses non-affine batch normalization (BatchNorm) [67], i.e., BatchNorm whose learnable parameters are fixed (bias $\beta = 0$ and scale $\gamma = 1$). However, it is well known that the choice of normalization strategy is crucial for folded architectures: folded ResNets suffer from overfitting and exploding layer activations. Specifically, BatchNorm with independent affine parameters for each iteration—which has

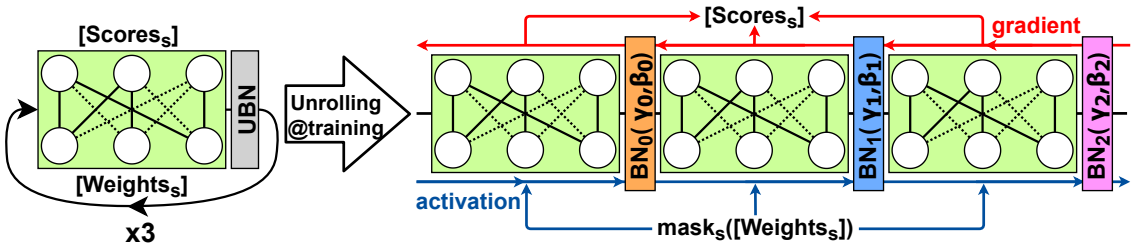


Fig. 4.5: How a folded block with UBN is unrolled during training time. In all iterations the same mask and weights are used, and gradients update the same scores. However, an independent affine BatchNorm layer is learned for each iteration.

been rediscovered numerous times under different names [44, 70, 76, 89, 175]—has been reported to be especially effective. The method proposed in this chapter also employs this strategy and refers to it as *unshared batch normalization* (UBN), following [70]. For clarity, Fig. 4.5 details the unrolled structure of a HFN block with shared weights but unshared BatchNorm parameters.

In addition to UBN, this work experiments with folding using shared affine BatchNorm parameters (SBN), and folding using non-affine BatchNorm (NA-BN), If the affine parameters are unshared, technically each iteration of a folded block applies a different function. Although this may have a significant impact on a recursive block's expressive power, in practice, it requires a very small number of additional parameters, and may even be bio-plausible [89].

Other normalization layers that do not normalize along the mini-batch dimension, which have been reported to be well suited for recurrent architectures, are also considered for folded blocks. Namely: Layer Normalization [8], which computes statistics along the channel dimension instead of the batch dimension; Instance Normalization [155], which

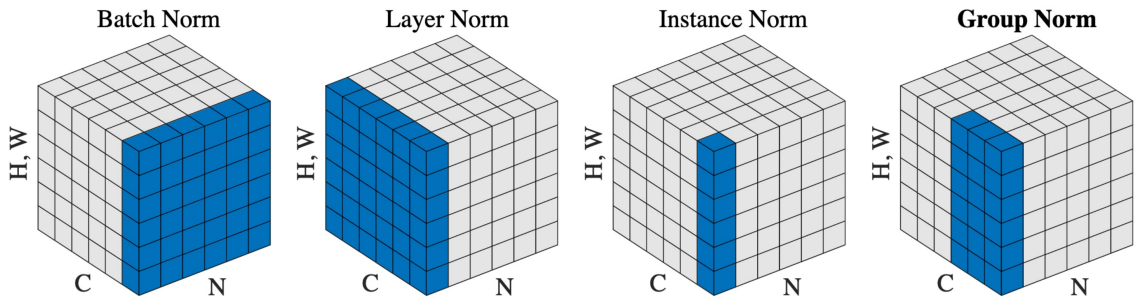


Fig. 4.6: Comparison of normalization methods (replicated from [163]).

uses a single channel; and Group Normalization [163], which generalizes the previous two by considering an arbitrary number of channels. Fig. 4.6 depicts a comparison of BatchNorm with these normalization methods. Furthermore, the experiments in this chapter also consider Local Response Normalization [79], a non-affine normalization scheme inspired by the lateral inhibition of biological neurons that has been successful on recurrent convolutional neural networks [88].

Previous work on supermask training of feedforward networks only considered either non-affine BatchNorm [128] or affine BatchNorm [118]. This chapter explores both options also for the feedforward sections of the HFN model, as well as for the baselines.

4.3.5 Model Compression Scheme

This chapter reports model memory size considering a specific compression scheme oriented to specialized hardware.

All weights and biases are counted as occupying 32 bit. However, it is not necessary to store weights in the case of supermask training, since they can be generated on the fly from the original seed with a random number generator [56, 177]. Furthermore, this seed can be substituted with a hash of other model parameters [56], so it is not necessary to store it either. Therefore, the model size of models trained with supermask training is only the size of the supermask—one bit per weight—in addition to the memory size of the affine normalization parameters, if any. When used, affine parameters constitute a very small part of the total size: UBN models only occupy 5% to 15% more memory than their SBN or BN counterparts.

Although supermasks can be compressed further by exploiting their sparsity with entropy coding [56, 118], these techniques are not considered in this chapter for simplicity. Counterintuitively, this means that sparsity has no impact on the model sizes reported in this chapter. However, Chapter 5 and Chapter 7 discuss additional compression based on entropy coding from both the algorithmic and hardware standpoints.

HFNs can fit in under 2 MB under this scheme, as will be demonstrated later in Section 4.4.4, small enough for specialized hardware's on-chip SRAM memory resources. This provides a significant advantage, as off-chip DRAM memory access consumes orders of magnitude more energy and time than on-chip DRAM access or arithmetic operations [58]. Moreover, the weight's balanced ternary precision and sparsity can be leveraged to reduce the arithmetic cost.

It shall be noted that in conventional hardware, weights must be uncompressed to full precision and processed densely. In that case, the proposed models provide no computational

cost advantage compared to their feedforward dense learned weight counterparts. For this reason, and because the arithmetic cost on specialized hardware is heavily implementation-dependent, this chapter does not report FLOP counts and instead focuses on model memory size gains under the described compression scheme.

4.4 Experiments and Results

This section analyzes how to better combine supermask training with a recurrent residual network and compares the results with the baseline methods, summarized in Table 4.1.

TABLE 4.1: Summary of the four methods compared on ResNet in this section.

Method	Architecture	Training
Standard ("Vanilla") [53]	Feedforward	Dense weight learning
Folding [89]	**Recurrent**	Dense weight learning
Hidden Networks (HNN) [128]	Feedforward	**Supermasks**
Hidden-Fold Networks (HFN)	**Recurrent**	**Supermasks**

After describing the methodology in Section 4.4.1, Section 4.4.2 examines the compatibility of both methods under diverse strategies of weight initialization, normalization, and bias learning. Section 4.4.3 explores different model size options, including supermask sparsity, the number of stages folded, and model depth and width. Now with the optimal settings, the four methods are compared on CIFAR-100 and ImageNet on Section 4.4.4, revealing HFN's strengths and weaknesses. Section 4.4.5 investigates the iterative generality of the learned recurrent blocks by modifying the number of iterations after training, and Section 4.4.6 concludes this section.

4.4.1 Experimental Settings

All experiments described were implemented on PyTorch [123] based on the original code of [128], which is publicly available [129]. The baseline model for all experiments is ResNet-50, as described in [53], and variations of it. Since folding only applies to deep networks, basic experiments were performed on the rather complex CIFAR-100 [78] dataset, whereas ImageNet [132] was used for large-scale dataset evaluation.

Except when explicitly stated otherwise, experiments were carried out using the following

methodology. The 60 000 images of CIFAR-100 were split into 45 000 for training, 5000 for validation, and 10 000 for the test set. Image pre-processing and augmentation were done in the same way as [128]. Training on CIFAR-100 is performed using stochastic gradient descent (SGD) with weight decay 0.0005, momentum 0.9, and batch size 128 during 200 epochs, with an additional 100 epochs for models deeper than 100 layers or double width. The learning rate is reduced using cosine annealing starting from 0.1, with no warmup. Experiments using ImageNet were performed for 100 epochs, using the hyperparameters recommended in [117]. Reported accuracy on CIFAR-100 is the average of three runs of top-1 test accuracy scores, measured at the highest scoring validation epoch, whereas on ImageNet they correspond to single-run top-1 validation accuracy. Standard deviation is indicated with lines on bar graphs and with shaded areas on plots.

Training a vanilla ResNet-50 on CIFAR-100 on an NVIDIA GeForce RTX 3090 takes 2.4 h, whereas its folded, HNN, and HFN versions take 2.2 h, 4.4 h, and 4.2 h, respectively.

An early version of the implementation of this method and its experiments can be found publicly available at [94].

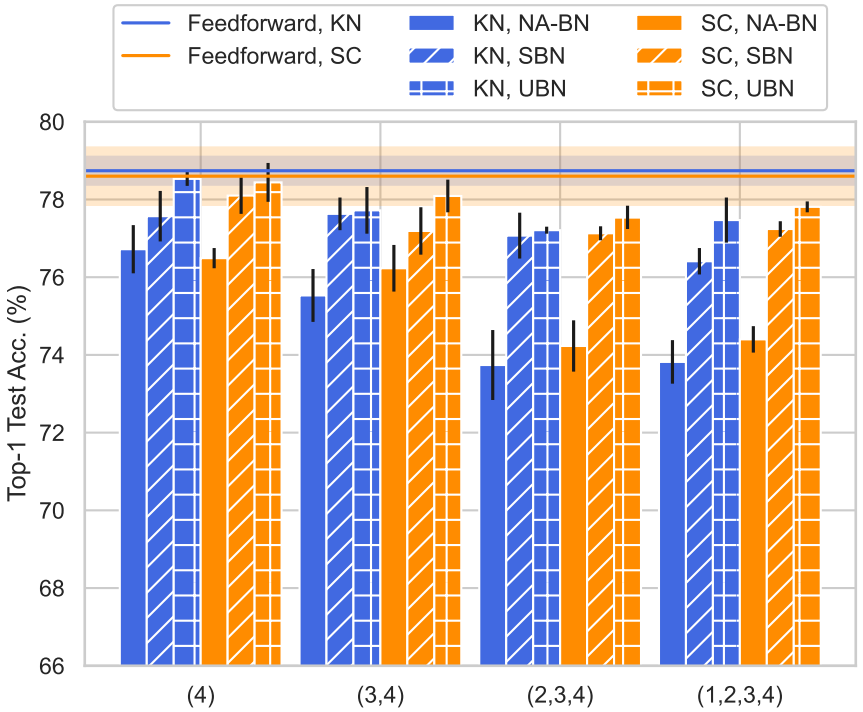
4.4.2 Compatibility of Supermasks and Folding

This section takes a preliminary look at how well supermask training and folding mix under different weight initialization and normalization strategies. Fig. 4.7 compares feedforward ResNet-50 to models with increasingly more stages folded, trained with dense weight learning in Fig. 4.7a and top-50% supermask training in Fig. 4.7b.

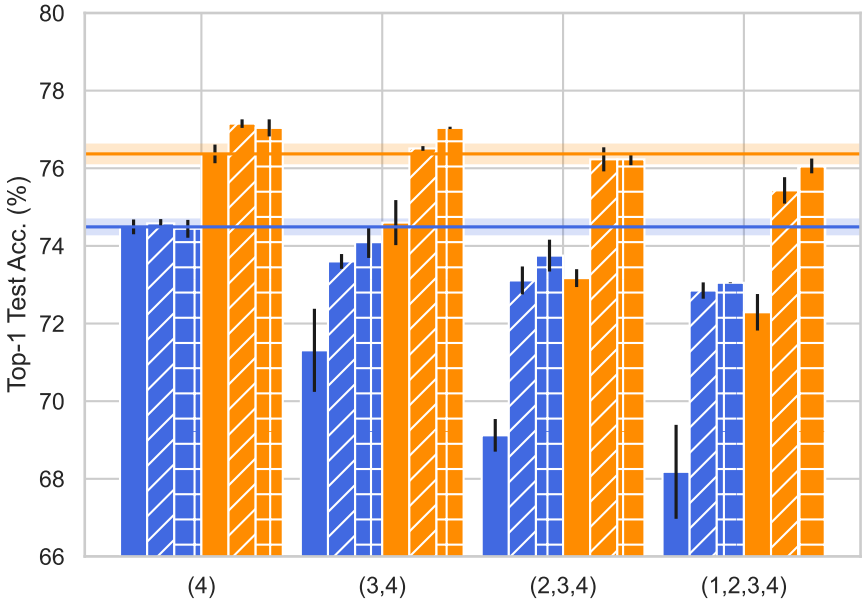
The impact on accuracy of transforming ResNet into a recurrent model is slightly more negative the more stages are folded, but very small in any case. In the case of supermask training, folding even results in a slight improvement in accuracy, as predicted. Both weight initialization choices produce similar results for weight learning, but for supermask training, signed Kaiming constant (SC) initialized models reach notably higher accuracy, as reported in [128]. SC is used for all methods hereafter, except for dense feedforward models, for which Kaiming normal (KN) is used.

Folding the Normalization Layer

Fig. 4.7 compares three BatchNorm strategies for folded blocks: non-affine (NA-BN), affine with shared learned parameters (SBN), and affine with independent learned parameters for each iteration (UBN). Feedforward blocks use non-affine BatchNorm, except when learning dense weights, when affine BatchNorm is used. Accuracy tends to be higher the more affine parameters are learned, although the difference between SBN and UBN is



(a) Dense weight learning



(b) Supermask training (top-50%)

Fig. 4.7: Impact of folding ResNet-50 when training (a) dense weights (b) a supermask for different combinations of weight initialization and BatchNorm strategies. Numbers in parentheses indicate folded stages.

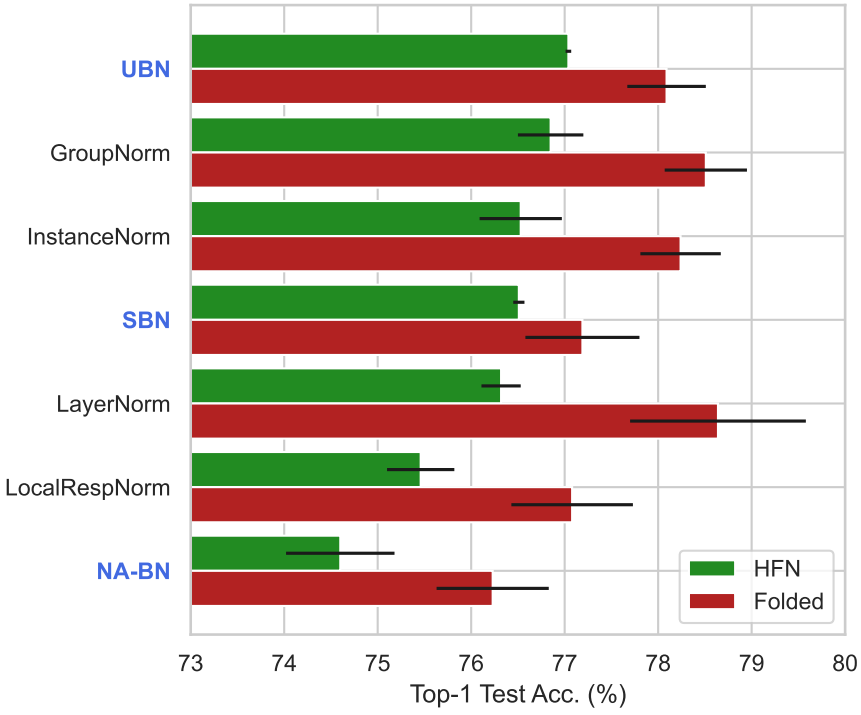


Fig. 4.8: Impact of normalization layer choice for folded blocks. Only folding stages 3 and 4 of ResNet-50.

smaller than previously reported.

This analysis is expanded in Fig. 4.8 to also compare Layer Norm (LN) [8], Instance Norm (IN) [155], Group Norm (GN) [163], and Local Response Normalization (LRN) [79] (see Chapter 4.3.4). Their affine parameters are shared in all cases except for LRN, which is intrinsically non-affine. GN uses 32 groups, and LRN uses size 5, additive factor 2, multiplicative factor 0.0001, and exponent 0.75.

As reported in previous work, additional affine parameters have a significant impact when folding with learned weights. However, normalizing along the channel dimension provides an even more significant advantage—bigger the more channels are considered. Unlike weight learning, HFN just works better when more affine parameters are learned. LRN overperforms NA-BN, but is surpassed by all the affine methods. Although closely followed by GN, UBN provides the highest accuracy for HFN. All experiments hereafter use UBN for recurrent blocks.

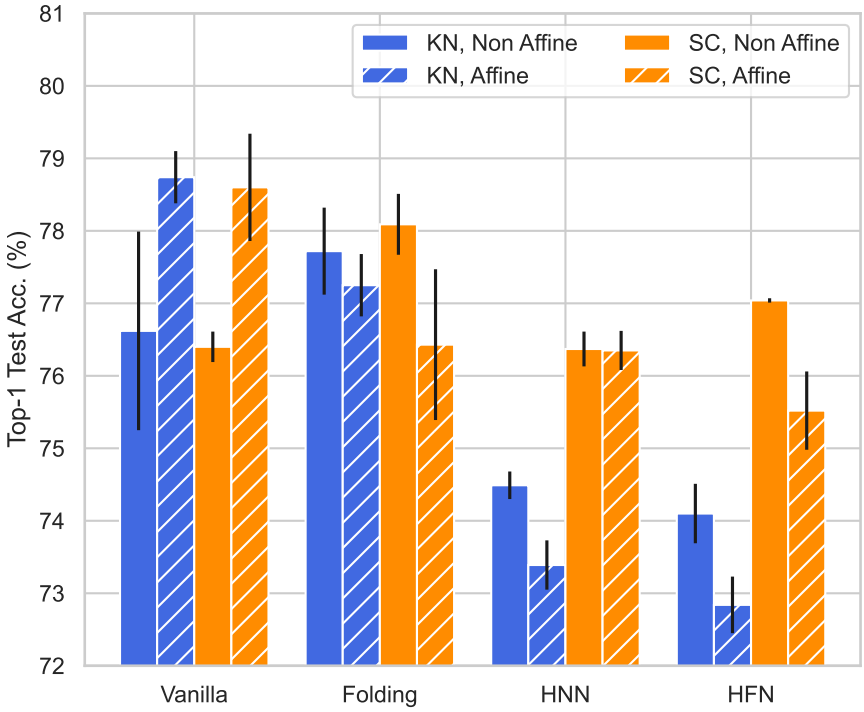


Fig. 4.9: Affine BatchNorm on non-folded blocks is only beneficial for Vanilla. Folded stages use UBN, and supermasks use top-50%.

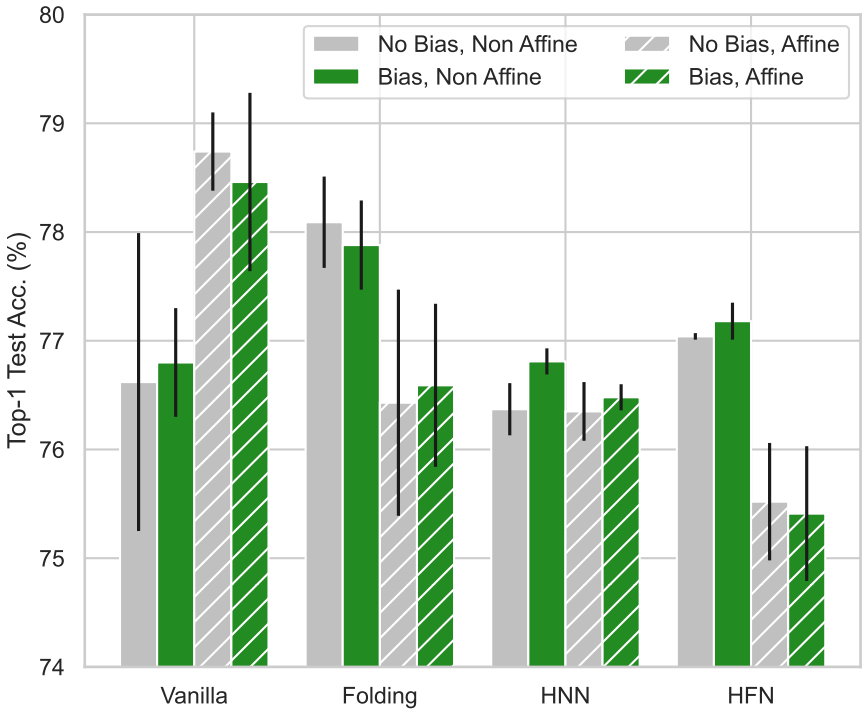


Fig. 4.10: Learned biases have a negligible effect.

Learning Additional Parameters

Since learning more affine parameters results in better performance, it could seem obvious that they should be learned in all layers. However, Fig. 4.9 shows that independently of the weight initialization used, using affine BatchNorm on non-folded blocks is harmful if some part of the architecture is folded or using supermask training. Therefore, affine BatchNorm is only used for dense feedforward models and folded blocks (as UBN).

Previous work [93, 128] reported results without learning biases of convolutional and fully connected layers. This is standard practice when using BatchNorm, as the role of biases is absorbed by the shift parameter β in BatchNorm's affine transformation, but these works used non-affine BatchNorm. The impact of learning these additional parameters is explored in Fig. 4.10, which shows that it has an insignificant influence. Therefore, biases are not learned hereafter.

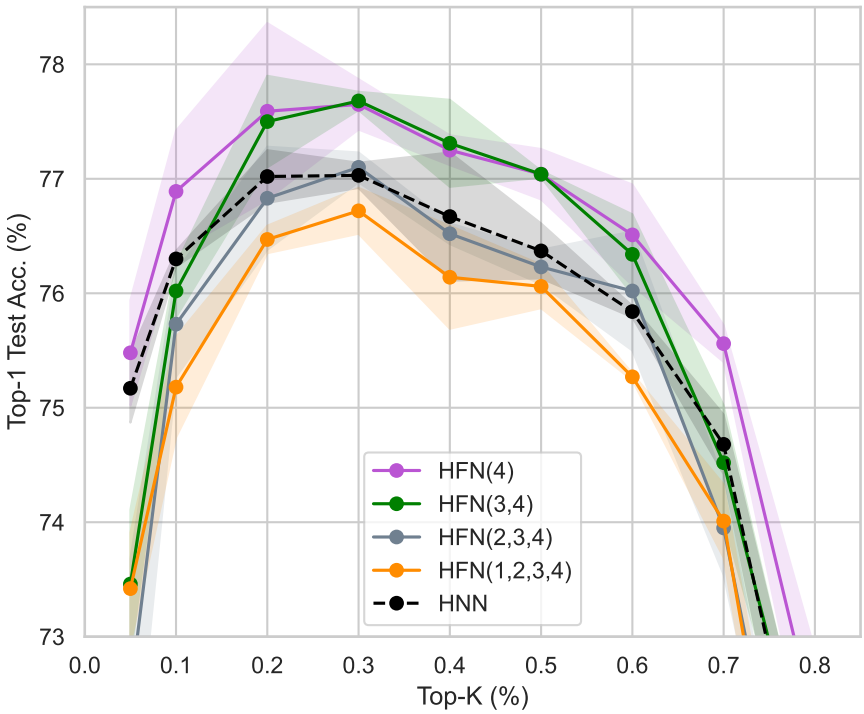
4.4.3 Tuning HFN's Size

This Subsection analyzes HFN's tradeoff between size and accuracy by exploring its different size hyperparameters: the sparsity of the supermask, the number of stages folded into recurrent blocks, and the architectures' depth and width. Evaluating HFN's scalability is especially relevant since the objective of this dissertation is to bring the benefits of reservoir computing to deep learning models without also bringing its drawbacks, i.e., scalability problems.

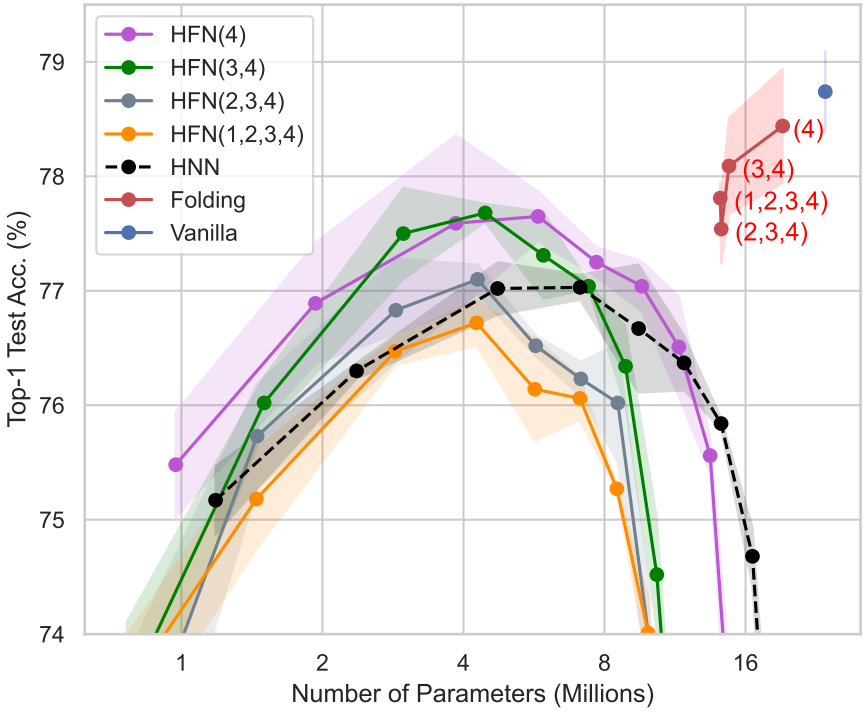
Supermask Density

Fig. 4.11a explores a range of top-k% values for different numbers of folded stages. The number of folded stages is increased from the deeper to the shallower stages, based on the evidence that the later stages of the visual ventral stream are more likely to exploit iterative computation [73].

The optimal density range is between 20% and 40% in all cases, with a peak around top-30%. Interestingly, it is the same for the feedforward and folded models, suggesting that it may be optimal for the learning algorithm used (`edge-popup`) but not necessarily the density of the optimal strong ticket. Fig. 4.11b shows that HFNs are more parameter-efficient than feedforward strong tickets for the whole optimal range. Since density has no impact on model size in the proposed compression scheme, all experiments hereafter adopt the optimal density value of top-30%. This may not be the optimal density for the deeper and wider models introduced later. However, due to `edge-popup`'s drawback of sparsity having to be set a priori, tuning it for each configuration is costly.



(a) Impact of supermask density (top-k%) on feedforward and folded ResNet-50.



(b) Parameter count of the compared methods.

Fig. 4.11: Accuracy to parameter count tradeoff for different supermask densities and number of folded stages. Points in (b) correspond to the densities in (a). Numbers in parentheses indicate folded stages.

Number of Folded Stages

Fig. 4.11 shows that, despite the smaller size, folding even a single stage of ResNet results in higher accuracy and only suffers a drop when folding all stages. That means that the recurrent version of ResNet either contains strong tickets that are better performing or at least are easier to find for `edge-popup`. The best results are achieved when only folding stages 3 and 4, which contain most of the parameters of ResNet. Folding additional stages has a very small effect on model size and negatively impacts accuracy. For weight learning, accuracy drops when more stages are folded, but folding only stages 3 and 4 provides the best size-accuracy tradeoff. Therefore, this is the folding strategy used for both methods in all experiments hereafter.

Architecture Depth

Deep learning's success in general, and ResNet's in specific, has been commonly attributed to network depth. However, folding collapses a stage's depth into a single block, transforming it into number of iterations. This subsection looks at the effect of this transformation and examines whether there is a benefit to extra depth when it does not imply additional parameters.

Fig. 4.12 depicts the effect of increasing the depth of a single stage of an HFN version of ResNet-50. Although increasing depth in stage 3 has a monotonic positive impact, it is milder than what would be expected for deep neural networks. Additional iterations in stage 4 have a destructive effect, presumably because of its much larger size (ResNets rarely have more than 3 blocks in this stage).

Additional iterations should not have a different impact on gradients than additional depth: folding preserves the identity mappings of the skip-connections, and after unrolling, gradients flow through the same number of layers as in the corresponding feedforward model. The reduced amount of parameters could explain the impaired effect of extra depth, but Fig. 4.13 shows that increasing iterations in folded models with learned weights has a much stronger effect. Furthermore, extra depth also has a more substantial effect on supermask training on feedforward models. However, when also increasing depth in stages that are not folded, HFN scales better than when not folded at all, although with increasingly bigger variance. Deeper recurrent ResNets also contain stronger tickets, supporting our conjecture on hypothesis space: extra depth makes subnetworks randomly initialized close to a recurrent approximation even scarcer in feedforward ResNets of finite width.

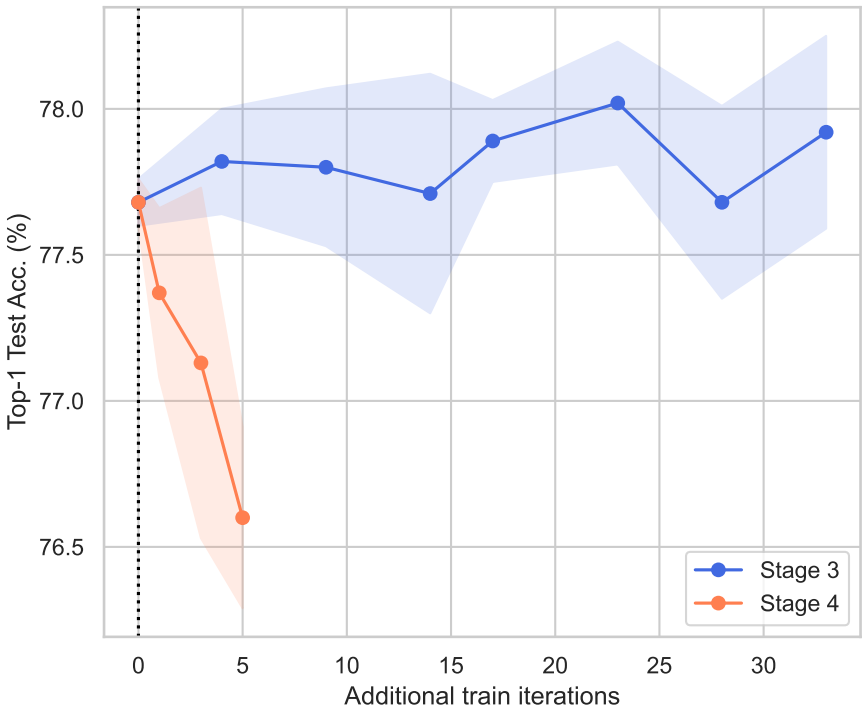


Fig. 4.12: Adding extra iterations to individual folded stages of ResNet-50. ResNet-101 is equivalent to ResNet-50 with 17 additional blocks in stage 3

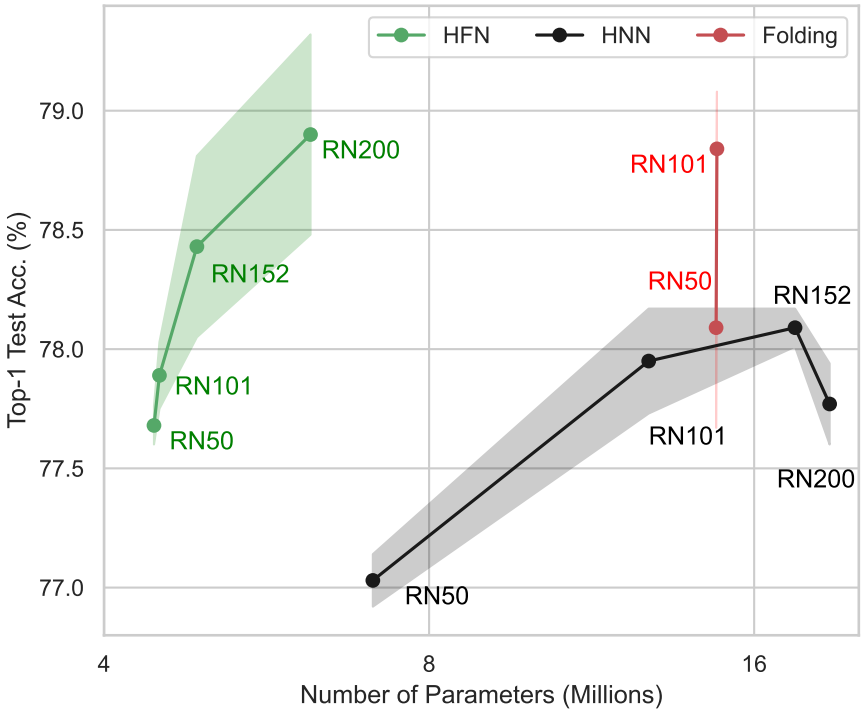
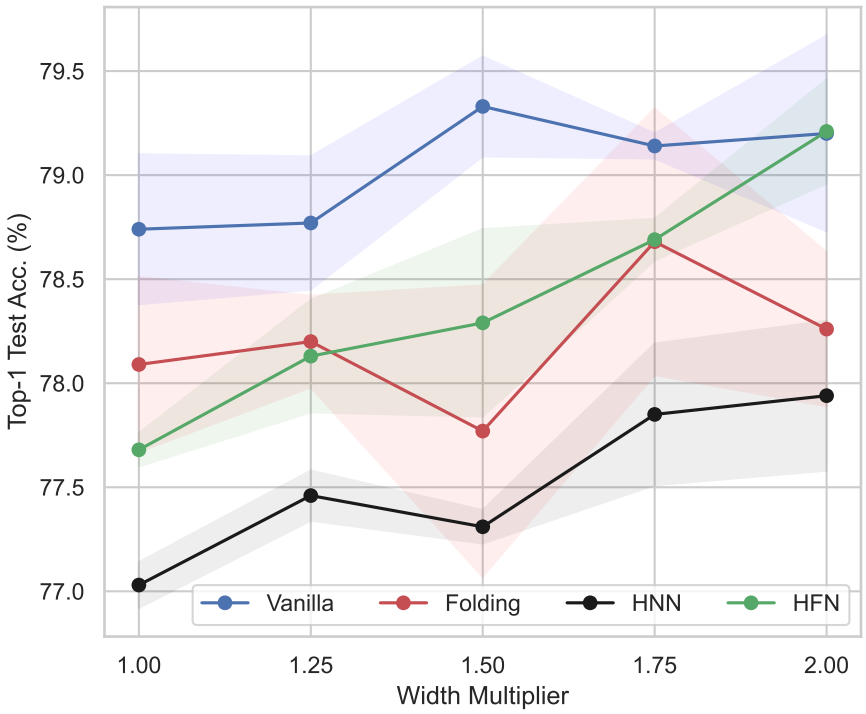
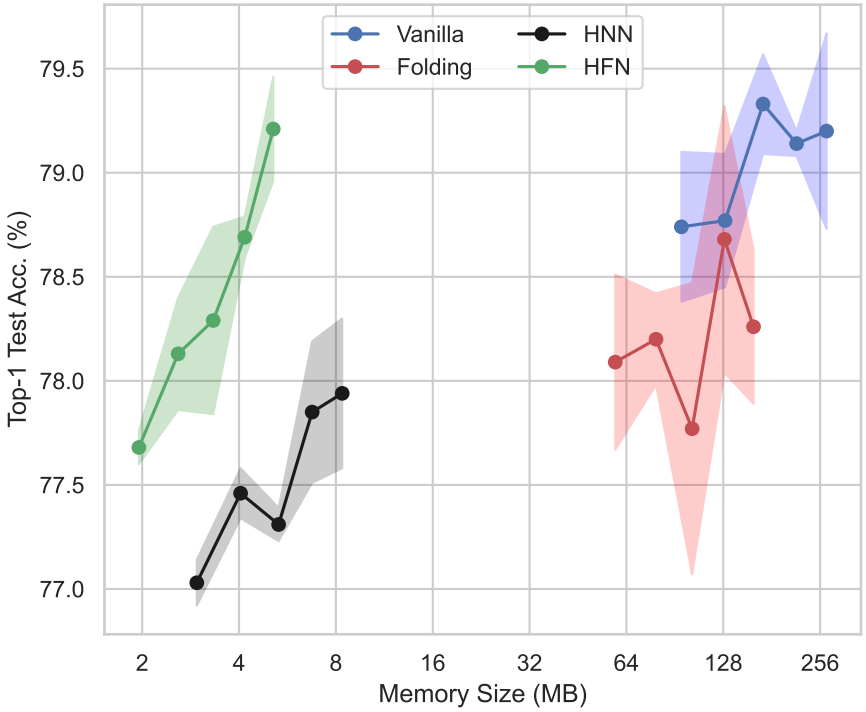


Fig. 4.13: Depth scalability of folding and supermask training. ResNet-101 is only deeper than ResNet-50 in stage 3, but ResNet-152 and ResNet-200 are also deeper in stage 2.



(a) Testing a variety of widths on ResNet-50.



(b) Impact of width on model size.

Fig. 4.14: Width scalability of the different methods on ResNet-50. Points in (b) correspond to the width multipliers in (a). All models are trained for 200 epochs.

Architecture Width

Although incrementing depth in a folded block only increases the number of iterations, deeper non-folded stages are not the only way of upscaling HFN. The search space for HFNs can be broadened by expanding ResNet's width [169].

Fig. 4.14 compares HFN's width scalability to that of the other methods by multiplying the number of channels of all layers by a width multiplier. Although all methods benefit from extra channels, HFN profits the most, overperforming both feedforward supermasks and dense folded models, with smaller variance. More remarkably, HFN even matches the wider dense feedforward architecture, despite being 52.47x smaller.

4.4.4 Method Comparison on Image Classification

CIFAR-100

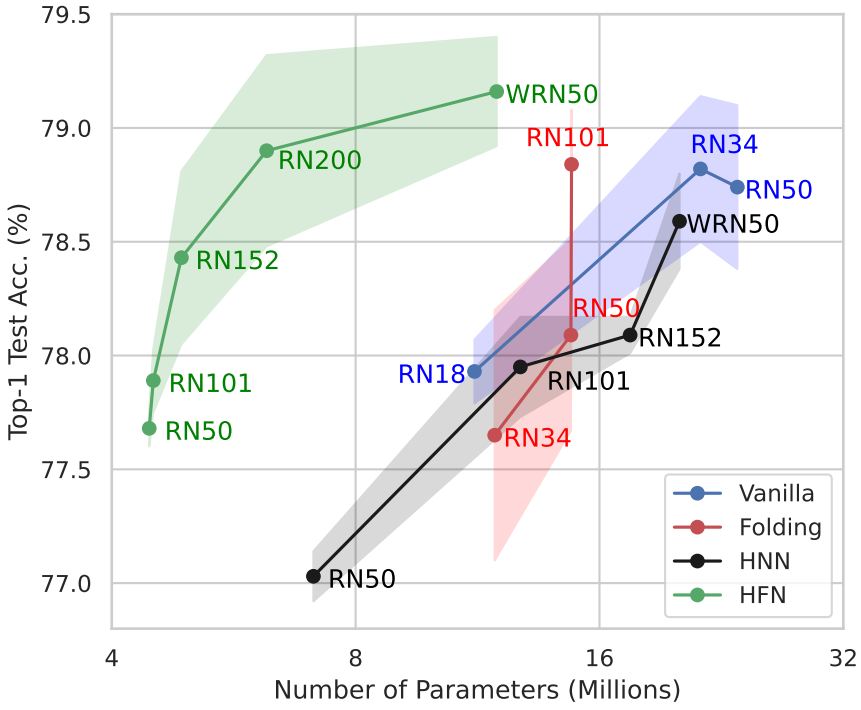
Fig. 4.15a compares the accuracy and parameter count of different ResNet sizes trained with the compared methods on CIFAR-100. HFN models are the most parameter-efficient: fewer parameters than equally performing models and more accurate than models with a similar parameter count. Furthermore, deeper HFNs and wider HFNs reach the highest accuracies overall.

Moreover, HFN's advantage is more evident when comparing in Fig. 4.15b model memory sizes under the compression scheme described in Section 4.3.5. HFNs are about half the size of their feedforward counterparts, with ResNet-50 fitting in under 2 MB. Wide-ResNet-50 overperforms dense models more than 30x larger, only matched by the wider dense models (Fig. 4.14b).

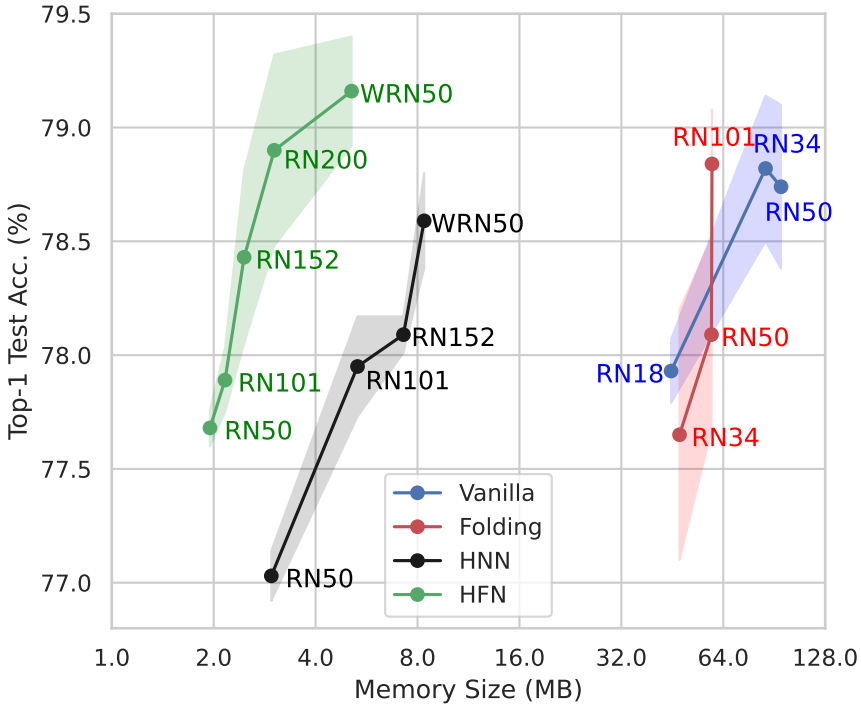
ImageNet

When the same models are compared on ImageNet, Fig. 4.16 shows that HFN's accuracy advantage scales poorly to a larger dataset. It is no longer the best-performing method, only matching the shallowest weight-learning architectures considered.

A suboptimal train schedule can explain this decay. As shown in Fig. 4.17, HFN converges to CIFAR-100 in 200 epochs, only needing additional epochs for larger models. However, HFN underfits on ImageNet and fails to reach convergence, even when doubling the number of epochs, using a different learning rate scheduler, or using RAdam [92]. This phenomenon is observed in all HFN sizes but in none of the other methods, making it evident that `edge-popup` is unable to exploit the full potential of HFN. Although doubling the number of epochs is enough to match the accuracy of feedforward strong tickets, it

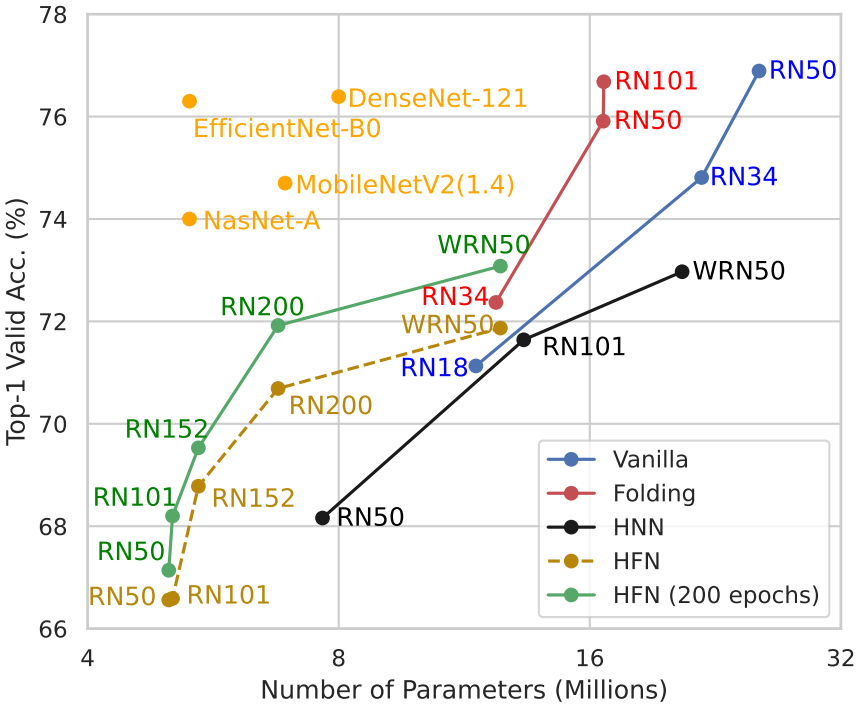


(a) Tradeoff between classification accuracy and number of parameters.

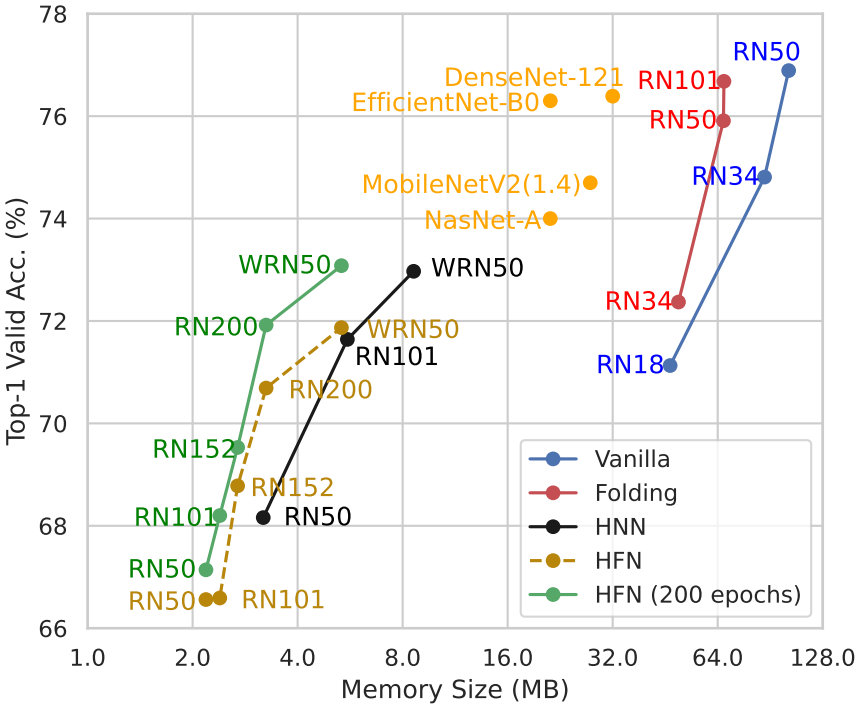


(b) Tradeoff between classification accuracy and model memory size.

Fig. 4.15: Comparison of the four methods using different model sizes on CIFAR-100. HFN is both the most accurate, the most parameter-efficient, and the tiniest.



(a) Tradeoff between classification accuracy and number of parameters.



(b) Tradeoff between classification accuracy and model memory size.

Fig. 4.16: Comparison of the four methods using different model sizes on ImageNet. HFN is both the most parameter-efficient and the tiniest. Also compared with popular efficient models: DenseNet [63], MobileNetV2 [133], NasNet [180], and EfficientNet [148].

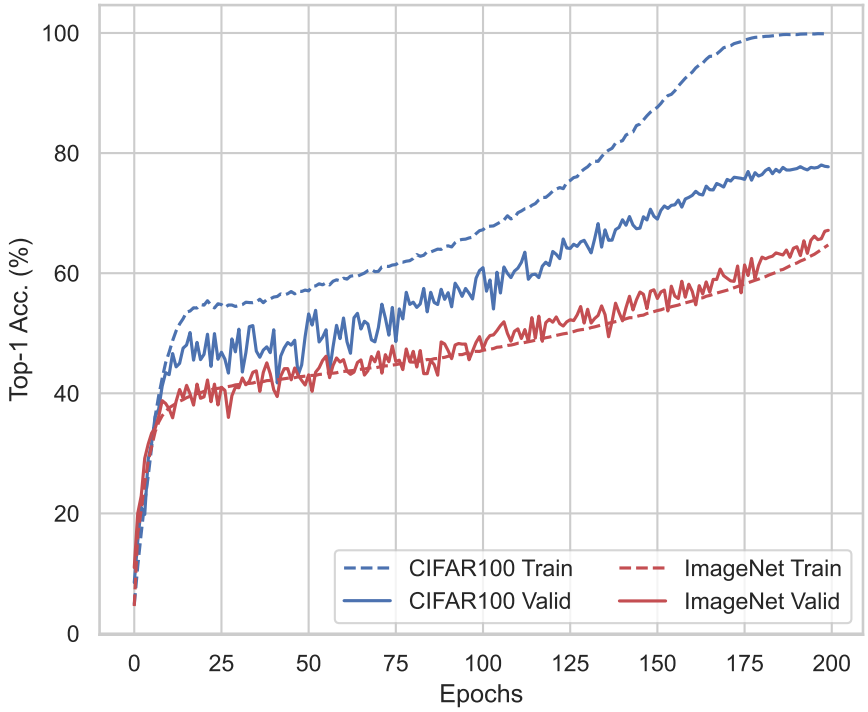


Fig. 4.17: Comparison of train graphs for the two datasets: HFN underfits on ImageNet and does not reach convergence.

contradicts the hypothesis that folded tickets are easier to find.

Nonetheless, HFN is still the most parameter-efficient and tiny: more accurate than models of similar parameter count and one order of magnitude smaller memory size than similarly performing dense models. Popular small models are even more parameter-efficient, but offer a much smaller memory size compression ratio†. Although not the focus of this chapter, it is also noticeable that folded ResNets with learned weights achieve higher accuracy than previously reported in both datasets, especially with deeper configurations.

4.4.5 Stretching HFN After Training

This Subsection explores the effect of changing the number of iterations after training. Since the goal is to investigate iterative generality, this section does not use UBN, so all

†Although Fig. 4.16 assumes 32 bit for all parameters of non-SLTH models, some 16 bit versions exist of the efficient models depicted in orange. It shall be noted that the X-axis is logarithmic, and that, even if assuming 16 bit parameters, HFN models are at least 2× smaller.

iterations apply strictly the same function.

As discussed above, folding transforms architecture depth into a new dimension: iterations. Where the parameters of a conventional block would receive a gradient per training update, a recurrent block propagates to its parameters a number of gradients equal to its number of iterations, as was covered in Section 4.3.2. It is then tempting to draw a parallelism between train iteration and recurrence iteration.

Fig. 4.18 investigates if additional gradients from increased iterations at train time have the same effect as additional gradients from increased train epochs. Training ResNet-50 with added iterations at individual folded stages has no positive impact, regardless of whether normalization is affine or not. The best performance is at the original number of iterations, but stretching in training has a very mild impact, suggesting that the learned functions have some degree of iterative generality.

Fig. 4.19 stretches ResNet-50 at inference time to further investigate this phenomenon, testing for more or fewer iterations than the models were trained for. Similar experiments

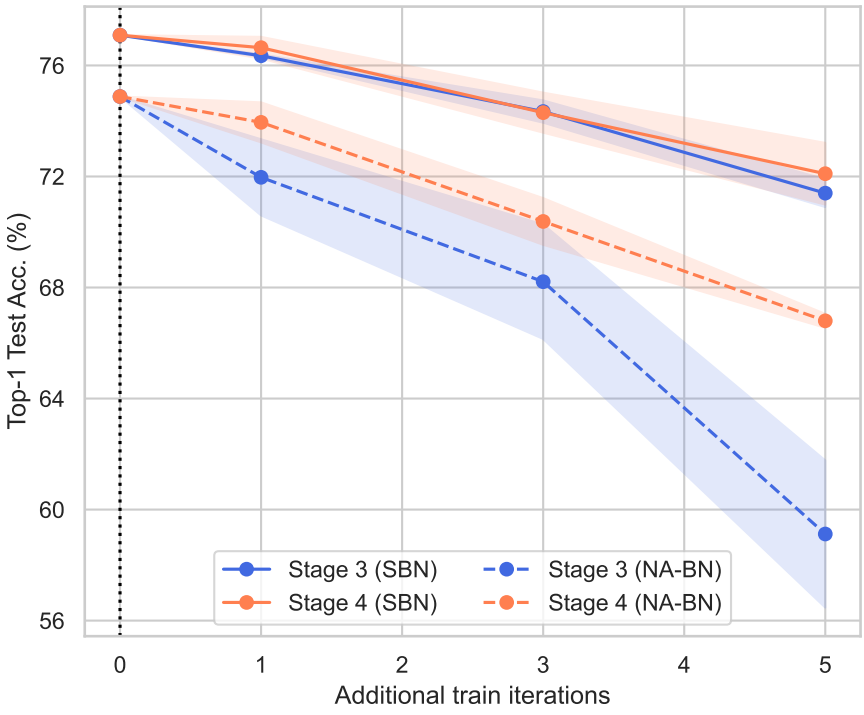
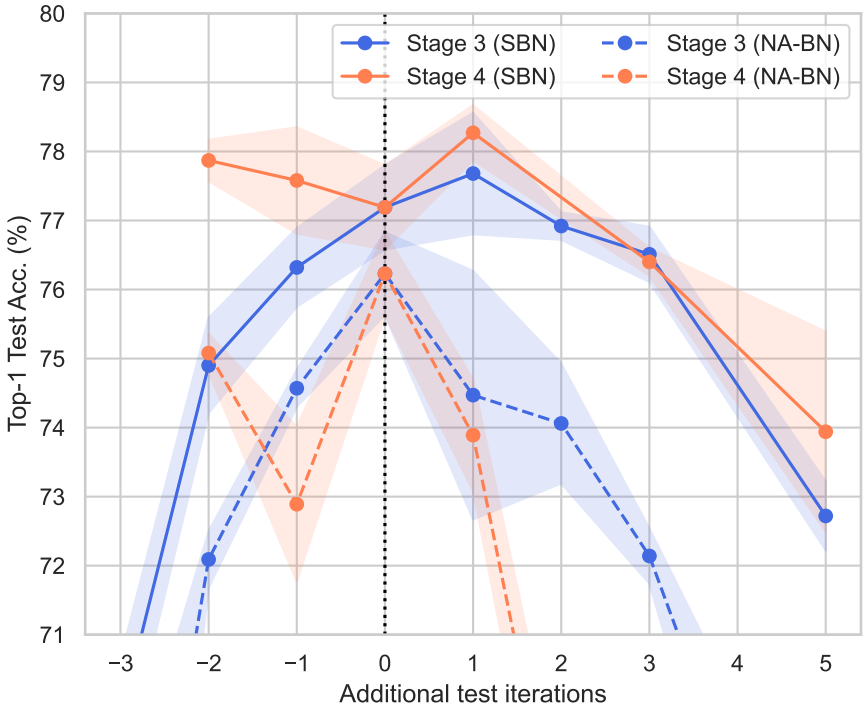
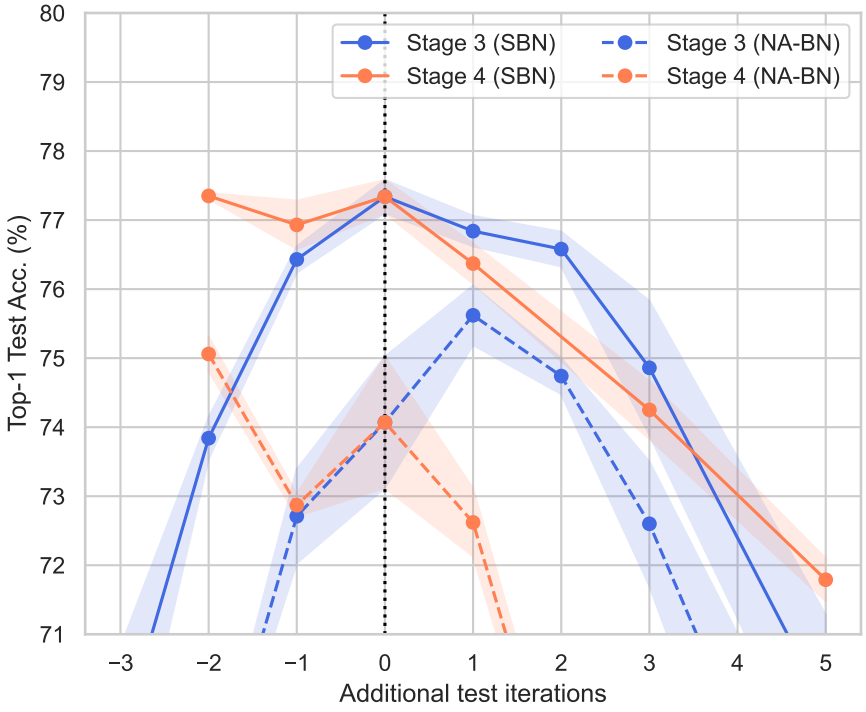


Fig. 4.18: Adding iterations to individual stages of ResNet-50 only at train time. SBN:shared affine BatchNorm; NA-BN: non-affine BatchNorm.



(a) Stretching with learned dense weights.



(b) Stretching with supermask training.

Fig. 4.19: Testing ResNet-50 for more iterations it was trained for. Additional iterations are added to an individual folded stage, on independent runs. SBN: shared affine BatchNorm; NA-BN: non-affine BatchNorm.

with weight learning were presented in [70, 89]. If ResNet is learning an ensemble, its accuracy would suffer little from removing blocks but unpredictably from adding additional blocks. However, if these blocks are learning generally iterative functions, a small change in the number of iterations, should have a small effect on performance.

The results show precisely that kind of behavior when the modification is small, sometimes even improving performance. Affine BatchNorm makes recurrent blocks more resilient to these alterations, and HFN shows similar behavior to its dense counterpart, other than a narrower sweet spot.

Strikingly, in both cases, higher accuracy is observed consistently when completely removing stage 4's folded block at test time, leaving only the downsampling block before the classifier. Although it confirms that these blocks do not perform a compositional transformation in the feature hierarchy, the reason for this phenomenon is not clear yet.

Since complex samples that benefit from additional iterations are a minority [70], increasing the number of iterations for all samples generally has a globally negative effect. A possible way of exploiting this iterative generality is to adjust the number of iterations dynamically on a per-sample basis as in [86] or [44], a mechanism that has also been observed in primate's visual cortex [73].

4.4.6 Result Analysis

Table 4.2 compares the four methods when applied to the same ResNet-50. In the case of CIFAR-100, the folded recurrent network hides a more accurate and smaller ticket than the feedforward model, even close in performance to the models with dense learned weights. Although HF-ResNet-50 falls slightly short compared to its feedforward version, HF-ResNet-101, with the same architecture but more iterations, matches it in performance.

The model compression potential of HFN is better appreciated when comparing models of similar accuracy in Table 4.3: even on ImageNet, an HFN with similar performance to a dense feedforward model occupies one order less memory size. As a technique for model compression, combining folding and supermask training results in a superior accuracy-size tradeoff that produces residual networks capable of fitting in just 2 MB, small enough for on-chip memory. From the standpoint of specialized hardware design, the sparsity, ternary nature, and, most importantly, the reduction in data transfers from external memory provides computation time and energy advantages that far outweigh the additional depth necessary to match in accuracy the baseline methods. Section 4.5.1 discusses in detail these benefits.

Although folding ResNet is a detriment to its performance when learning dense weights,

TABLE 4.2: Comparison of the four methods on the same model, ResNet-50.
F: folded; H: HNN; HF: HFN; RN:ResNet; WRN:ResNet of 2× width.

Dataset	Model	Top-1 Acc. (%)	Parameters (Millions)	Size (MB)	Size Reduction
CIFAR-100	RN50	78.74	23.68	94.82	-
	F-RN50	78.09	14.75	59.07	1.61×
	H-RN50	77.03	7.10	2.96	32.07×
	HF-RN50	77.68	4.45	1.95	48.55×
ImageNet	RN50	76.89	25.55	102.22	-
	F-RN50	75.91	16.60	60.47	1.54×
	H-RN50	68.16	7.65	3.19	32.07×
	HF-RN50	67.14	5.00	2.18	46.80×

TABLE 4.3: Size comparison of models of similar accuracy, using the proposed compression.

Dataset	Model	Top-1 Acc. (%)	Parameters (Millions)	Size (MB)	Size Reduction
CIFAR-100	RN34	78.82	21.32	85.31	-
	F-RN101	78.84	14.78	59.28	1.44×
	H-WRN50	78.59	20.10	8.37	10.19×
	HF-RN200	78.90	6.21	3.02	28.27×
ImageNet	RN18	71.13	11.68	46.75	-
	F-RN34	72.37	12.35	49.41	0.95×
	H-RN101	71.64	13.33	5.56	8.42×
	HF-RN200	71.92	6.77	3.25	14.39×

for supermask training it is beneficial. Independently of sparsity, depth, or width, folded residual networks contain smaller and stronger tickets than their feedforward counterparts. Furthermore, these models outperform densely learned models with the same number of parameters, and occupy one order of magnitude less memory than those of similar performance.

4.5 Discussion and Conclusion

This chapter proposes and tests the conjecture that recurrent residual networks contain stronger tickets than their feedforward counterparts. The results show that these strong tickets can be exploited to vastly reduce the memory footprint, and upscaled to match or surpass the accuracy of models with dense learned weights: ResNet-50 is compressed 48.55× to 2 MB without a drastic loss in performance, and a ResNet-200 with superior accuracy is compressed 28.27× to just 3 MB. However, more than the additional restrictions on the search space imposed by recurrence is needed: the current supermask training algorithm is unable to find the best possible tickets, especially in the case of a larger dataset. Nonetheless, this chapter finds that these recurrent subnetworks with random weights have similar iterative properties to their learned weight counterparts and, furthermore, can be fitted in on-chip memory with a straightforward compression scheme, encouraging efficient edge implementations of computer vision based on recurrent neural networks.

4.5.1 Hardware Acceleration Potential

HFNs are accurate models with few and highly reused random parameters. When considering specialized hardware inference accelerators with a random number generator and the capability of operating with binary supermasks [56], HFNs can be compressed enough to be stored on on-chip SRAM, making off-chip DRAM access for parameters unnecessary. This promises vast energy-efficiency advantages. The bulk of computations used for neural networks consists of energy-hungry multiplications. Still, this cost is minute compared to that of off-chip memory access. In the case of 45 nm CMOS, a 32 bit floating-point multiplication consumes 3.7 pJ, while a 32 bit DRAM read costs 640 pJ [45, 58]. Specialized hardware often operates with reduced numeric precision, making the contrast even starker: a 16 bit floating-point multiplication uses 291× less energy than DRAM access. Additionally, off-chip memory also suffers from a much longer latency. With random weights, highly compressed binary supermasks, and recurrent connections reusing parameters, memory reads can be reduced to a minimum.

Fig. 4.20 compares the estimated energy consumed by DRAM access for loading models of similar accuracy, considering an ideal accelerator implemented in 45 nm CMOS. On a specialized architecture such as [56], HFN's reduction of memory reads guarantees at least a proportional two-order of magnitude reduction of energy consumption, making it a promising candidate for implementing energy-efficient DNN acceleration hardware.

Alternatively, Chapter 7 presents an accelerator that, although it does not exploit supermasks' random nature with a random generator, exploits their sparsity for acceleration

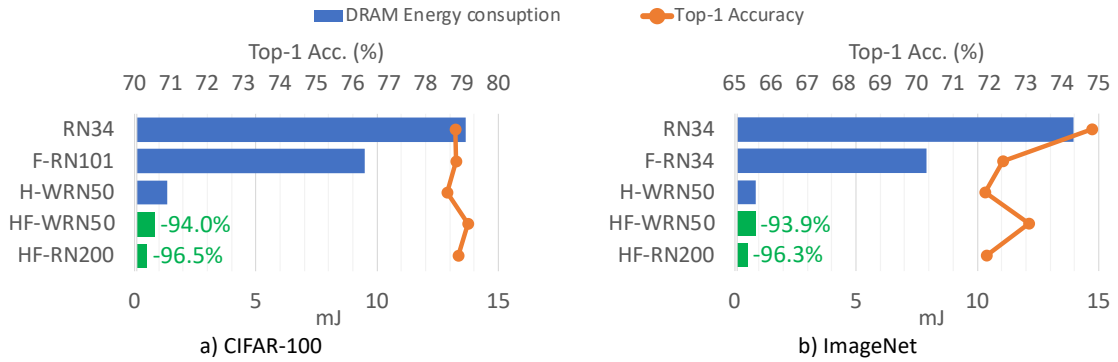


Fig. 4.20: Estimation of energy consumed for loading models of similar accuracy from DRAM, considering an ideal 45 nm CMOS inference accelerator.

and reduced power, while also exploiting HFN's model compression for the discussed off-chip memory access. Furthermore, Fig. 4.20 estimations do not consider data compression. Chapter 7 presents even larger reductions by exploiting HFN's entropy for lossless off-chip access compression with Huffman coding.

On the other hand, it shall be noted that HFN is not aimed at CPU/GPU implementation. Since the parameters must be uncompressed at runtime and folding does not change the number or size of convolutions, HFN-ResNet50's computational cost on standard processors is similar to ResNet-50's. HFN-ResNet-152 has similar accuracy to ResNet-50 while using 4.9× less parameters (Table 4.3, CIFAR-100), but its computational cost is almost 3× higher. This tradeoff between computation and size is avoided on specialized hardware, as the computational cost is also reduced by exploiting the model's sparsity and processing supermasks with binary operations. For software implementations, HFN could be coupled with techniques for reducing computational cost, such as reduced numerical precision, depth-wise separable convolutions [60], or AdderNets [15], or by generalizing folding to be compatible with the dense connections of DenseNet [63], which would reduce HFN's number of channels.

4.5.2 Future Work

Multiple improvements on the `edge-popup` algorithm have been proposed, which future work may apply to the method presented in this chapter: learning signs instead of using random binary weights [77] (explored in Chapter 6 and Chapter 7); learning weight scales instead of the fixed modulus of SC initialization [23]; annealing sparsity [30, 31] or enforcing it at a global level instead of per-layer [178](explored inChapter 7); an improved

straight-through estimator [27]; a better strategy for using biases [30]. Strategies for pruning trained models with masks could be adapted to supermask training, such as learning the pruning thresholds [91] instead of calculating them from top-k% at every update step.

HFN's effective weights can be equivalently regarded as ternary or binary, depending on whether the unselected weights are considered zero-weighted or simply not part of the model. Independently of this detail, arguably this trend should be reconnected with the literature on binary and ternary neural networks, where supermasks originated at first [101]. Essentially, supermasks are just a binarization—ternarization in [77]—of the scores, which act as latent weights. Recent advances in binarized neural network optimization have shown that learning can be done more efficiently without latent weights [55].

Pruning and quantization are two central techniques for implementing energy-efficient hardware inference accelerators. Supermask training concurrently learns and prunes a quantized network. Although many advances have been made by using ternary effective weights, Chapter 5 shows that supermask training can be expanded to learning weights quantized at a larger precision [118]. This points to a future trend of training methods that blend learning, pruning, and quantization into a concurrent process, explored in Chapter 6 and Chapter 7.

Previous work has shown that introducing recurrent connections in computer vision model is beneficial for weight learning. This chapter expands this argument to supermask training. ResNets with skip-connections and recurrent loops include most of the lateral connections observed in the visual cortex, but still miss one type: lateral feedback connections. Additionally, this chapter has only considered standard pyramidal ResNets. A study exploring isotropic ResNets with various complex connections was recently published in [29]. Their results could be leveraged for expanding HFN to modern isotropic architectures [152, 157]. This will be explored in Chapter 7.

Lastly, we have only evaluated HFNs on image classification tasks. Future work should evaluate them in time-series data tasks, considering standard RNN practices such as including LSTM units. Some previous work on RCNN for vision considered recurrent architectures that produced an output at every iteration [141, 170], whereas other methods, including the one studied in this paper, consider a single final output. Future work should also investigate the difference between these two readout strategies.

Impact

The work in this chapter was first presented as a conference paper [93] at the *32nd British Machine Vision Conference (BMVC)*, Online, November 2021 (also publicly available at [95]). Later, an expanded open access full paper [96] was published in *IEEE Access*, vol. 11, January 2023. It was also featured as a chapter [49] in the open access book *Photonic Neural Networks with Spatiotemporal Dynamics: Paradigms of Computing and Implementation* [146], October 2023. An early version of the implementation of this method and its experiments can be found publicly available at [94].

This research was supported by Japan Science and Technology Agency (JST) CREST Grant Number JPMJCR18K2, Japan.

Chapter 5

Multicoated Supermasks: Scalar Supermasks Enhance Strong Lottery Tickets

The previous chapter presented Hidden-Fold Networks (HFN), a novel method for transforming deep learning (DL) neural networks into reservoir computing (RC)-like models by leveraging the Strong Lottery Ticket Hypothesis (SLTH) and the `edge-popup` algorithm for training supermasks. Although HFNs are small, accurate, and scalable in both depth and width, they suffer in accuracy when applied to a larger dataset (ImageNet). Furthermore, the supermask learned by `edge-popup` is intrinsically limited to binary learning. This chapter presents Multicoated Supermasks, an enhancement of supermaks that improves classification accuracy in larger datasets without sacrificing any of the introduced RC-like characteristics by upgrading it to a method for concurrent training, pruning, and scalar quantization.

5.1 Introduction

Hidden Networks [128] showed the possibility of finding accurate subnetworks within a randomly weighted neural network by training a connectivity mask, referred to as supermask. We show that the supermask stops improving even though gradients are not zero, thus underutilizing backpropagated information. To address this issue, we propose a method that extends Hidden Networks by training an overlay of multiple hierarchical supermasks—a Multicoated Supermask. This method shows that using multiple supermasks for a single task achieves higher accuracy without additional training cost. Experiments on CIFAR-10 and ImageNet show that Multicoated Supermasks enhance the tradeoff between accuracy

and model size. A ResNet-101 using a 7-coated supermask outperforms its Hidden Networks counterpart by 4%, matching the accuracy of a dense ResNet-50 while being an order of magnitude smaller.

Hidden Networks (HNN) [128] $*$ showed the possibility of finding accurate subnetworks within a randomly weighted neural network by training a connectivity mask, referred to as supermask. The most practical feature of Hidden Networks is that they can be reconstructed from a small amount of information: unlearned weights are obtained from a random number generator, and the only data that needs to be stored are the random seed and a sparse supermask. These features can be exploited for implementing efficient inference accelerators on specialized hardware [56]. There are compression, quantization, and pruning methods that provide similar advantages, such as pruning connections based on importance [47], Deep Compression [46], and coreset-based neural network compression [26], but Hidden Networks are unique in that supermasks handle training and pruning simultaneously. Furthermore, Hidden Networks make possible to hot swap supermasks for applying the same model to different tasks.

Nonetheless, Hidden Networks has room for improvement in accuracy, task scalability, and model compression. The original work on Hidden Networks provides no solution for compensating the inference accuracy loss other than using wider channels, which require models several times larger for only a slight accuracy improvement. Using iterative pruning with weight reinitialization [16], or ternary supermasks for learning both sign and connectivity [77], showed some promising results, but they both increase the learning cost and require some form of weight learning. Supermasks in Superposition [162] succeeded in handling multiple tasks with constant computation cost by using superposed multiple supermasks, and Hidden Fold Networks [93] reduced the model size of Hidden Networks by folding them into a recurrent structure.

This chapter addresses these issues by proposing *Multicoated Supermasks*, which use multiple supermasks for a single task to form a scalar mask that grants the random weights of strong tickets with virtually the same expressive power as trained weights. This method shows that using multiple supermasks for a single task achieves higher accuracy without additional training cost, while also offering a better tradeoff between model size and accuracy than wide channel models. Experiments on CIFAR-10 and ImageNet show that Multicoated Supermasks enhance the tradeoff between accuracy and model size. A ResNet-101 using a 7-coated supermask outperforms its Hidden Networks counterpart

$*$Although now commonly reffered to as *"Strong Lottery Tickets"*, much of the work in this dissertation, presented before this convention, used the term *"Hidden Networks (HNN)"*. This chapter uses this terminology to refer, specifically, to the seminal work presented in [128] using `edge-popup`.

by 4%, matching the accuracy of a dense ResNet-50 while being an order of magnitude smaller.

Additionally, this chapter present an analysis of the score evolution during the training of Hidden Networks that reveals that the supermask stops improving even though gradients are not zero, thus underutilizing backpropagated information a finding that forms the underpinning of the proposed method: Multicoated Supermasks allow `edge-popup` to reduce classification loss even after adge-swapping stops.

The remainder of the chapter is organized as follows. Section 5.2 recapitulates Hidden Networks and analyzes their shortcomings. This serves as the mathematical basis for Section 5.3, where the proposed Multicoated Supermasks are presented before being explored and evaluated on Section 5.4. Finally, Section 5.5 concludes this paper.

5.2 Reformulation of Strong Lottery Tickets

Hidden Networks is an intriguing approach that integrates training and pruning into a single form by learning connectivity instead of weights. Since the proposed method shares the mathematical basis of Hidden Networks, this section recaps its details.

After reviewing the structure and notation of Hidden Networks in Section 5.2.1, Section 5.2.2 summarizes the mathematical proof of Hidden Networks using a fully connected (FC) layer (illustrated in Fig. 5.1), similarly to [128]. This proof serves as basis in Section 5.3 for proving the proposed method in a similar manner. Additionally, in contrast to the original paper, which mainly focused on how swapping edges affects the loss, this section also provides an analysis of the score transition after swapping stops, leading to the underlying idea of the proposed method.

5.2.1 Structure of Hidden Networks

Fig. 5.1 describes the structure of a fully connected (FC) layer and the notation used in this chapter. The outputs $z^{(l)} = [\mathcal{Z}_1^{(l)}, \mathcal{Z}_2^{(l)}, \dots, \mathcal{Z}_{|\mathcal{V}^{(l)}|}^{(l)}]^\top$ of the l-th layer are calculated as

$$z^{(l)} = \mathrm{ReLU}(\boldsymbol{W}^{(l)} z^{(l-1)}), \tag{5.2.1}$$

where $\boldsymbol{W}^{(l)}$ is the weight matrix of layer l. In Hidden Networks, represented in Fig. 5.2, the weight matrix $\boldsymbol{W}^{(l)}$ is obtained from two matrices: a random weight matrix and a supermask matrix, which can be written as

$$\boldsymbol{W}^{(l)} = \boldsymbol{W}_{\text{rand}}^{(l)} \odot \mathcal{H}(\boldsymbol{S}^{(l)}), \tag{5.2.2}$$

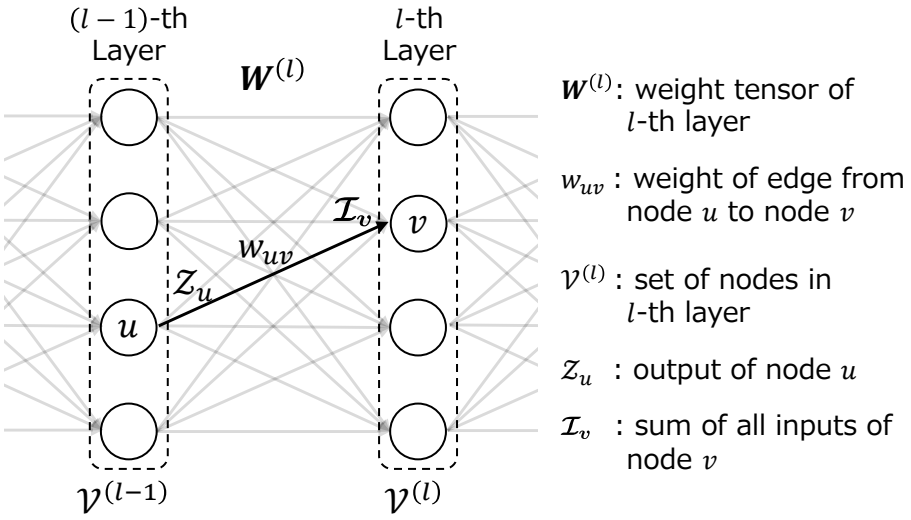


Fig. 5.1: Explanation of the notation used on a fully connected (FC) layer

where $\odot$ is the Hadamard product operator, and $\boldsymbol{W}_{\text{rand}}^{(l)}$ is a weight matrix randomly initialized by sampling uniformly from $\{-\sigma, \sigma\}$, in which σ is the standard deviation of Kaiming normal distribution [52]. $\mathcal{H}$ is the supermask generating function, defined as

$$\mathcal{H}(\boldsymbol{S}^{(l)}) = \begin{bmatrix} h(s_{1,1}^{(l)}) & h(s_{2,1}^{(l)}) & \cdots & h(s_{U,1}^{(l)}) \\ h(s_{1,2}^{(l)}) & h(s_{2,2}^{(l)}) & \cdots & h(s_{U,2}^{(l)}) \\ \vdots & \vdots & \ddots & \vdots \\ h(s_{1,V}^{(l)}) & h(s_{2,V}^{(l)}) & \cdots & h(s_{UV}^{(l)}) \end{bmatrix}, \tag{5.2.3}$$

in which $h(s_{uv})$ is a step function that determines mask values based on the score s_{uv} and a threshold score s_t:

$$h(s_{uv}) = \begin{cases} 1 & |s_{uv}| \geq s_t \\ 0 & \text{otherwise} \end{cases}. \tag{5.2.4}$$

Here, s_{uv} is the score assigned to the corresponding weight w_{uv}, and $\boldsymbol{S}^{(l)} \in \mathbb{R}^{U \times V}$, in (5.2.2) and (5.2.3), is the matrix formed by these scores.

The most distinct feature of Hidden Networks is that it uses randomly initialized frozen weights and selects a subset of them based on their scores. This makes the density of the supermask, $k\%$, a pivotal hyperparameter. Given $k\%$ and a tuple of sorted scores, written as

$$\begin{aligned} \text{sort}(\mathrm{S}^{(\mathrm{l})}) &= (s'_1, s'_2, s'_3, \cdots, s'_{U \times V}) \\ \text{s.t.} \quad |s'_1| &\geq |s'_2| \geq |s'_3| \geq \cdots \geq |s'_{U \times V}|, \end{aligned} \tag{5.2.5}$$

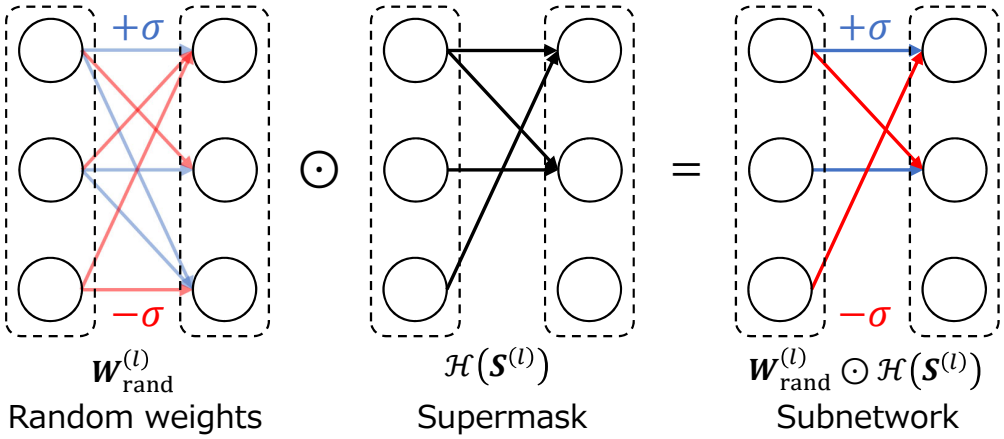


Fig. 5.2: Structure of Hidden Networks with binary weights randomly initialized to $\pm\sigma$, where σ is the standard deviation of Kaiming normal distribution. This initialization method is referred to as Signed Kaiming Constant (SKC) by [128].

the threshold score is calculated as $s_t = |s'_t|$, where

$$t = \lfloor k\% \times U \times V \rfloor \tag{5.2.6}$$

is used to select the top-k% scores $|s_{uv}| \geq s_t$, i.e., $s'_{\leq t}$.

5.2.2 Edge-Popup and Weight Scores

In the forward pass, `edge-popup` [128]—the training algorithm of Hidden Networks—only uses the weights selected by the supermask, while in the backward pass it applies backpropagation on the scores instead of the weights (i.e., weights are never updated). The authors of the original paper provided a simple and clear proof of the efficacy of `edge-popup`, which is briefly reviewed here to then expand it to the proposed Multicoated Supermasks in Chapter 5.3.

Using the notation in Fig. 5.1, the calculation of a FC layer on Hidden Networks can be written as

$$\mathcal{I}_v = \sum_{u \in \mathcal{V}^{(l-1)}} h(s_{uv}) w_{uv} \mathcal{Z}_u, \tag{5.2.7}$$

where $\mathcal{V}^{(l)} = \{v_1^{(l)}, \ldots, v_{n_l}^{(l)}\}$. In the forward pass, $h(s_{uv}) = 1$ if s_{uv} is one of the top-k% scores; otherwise $h(s_{uv}) = 0$. In the backwards pass, backpropagation uses a straight-through estimator (STE) [10] instead.

As mentioned above, Hidden Networks minimizes the loss by updating scores instead of weights. From (5.2.7), the differential of the loss w.r.t. the score is defined as

$$\frac{\partial \mathcal{L}}{\partial s_{uv}} = \frac{\partial \mathcal{L}}{\partial \mathcal{I}_v} \frac{\partial \mathcal{I}_v}{\partial s_{uv}} = \frac{\partial \mathcal{L}}{\partial \mathcal{I}_v} w_{uv} \mathcal{Z}_u. \tag{5.2.8}$$

Notice that the step function h can be omitted by assuming STE. Therefore, to decrease the loss, the score $\tilde{s}_{uv}$ can be updated as

$$\tilde{s}_{uv} = s_{uv} - \lambda \frac{\partial \mathcal{L}}{\partial \mathcal{I}_v} w_{uv} \mathcal{Z}_u, \tag{5.2.9}$$

where λ is the learning rate, and Hidden Networks takes the absolute value to ensure positive scores.

The remaining question is whether selecting the edges with the top-k% scores decreases the loss or not. If edge (i, p) replaces edge (j, p) at a gradient update, it means that $s_{ip} < s_{jp}$ before the swap and that $\tilde{s}_{ip} > \tilde{s}_{jp}$ after the update. Then, the following inequality holds:

$$\begin{aligned}
&(s_{ip} < s_{jp}) \wedge (\tilde{s}_{ip} > \tilde{s}_{jp}) \\
\Leftrightarrow\ & \tilde{s}_{ip} - s_{ip} > \tilde{s}_{jp} - s_{jp} \\
\Leftrightarrow\ & -\lambda \tfrac{\partial \mathcal{L}}{\partial \mathcal{I}_p} w_{ip} \mathcal{Z}_i > -\lambda \tfrac{\partial \mathcal{L}}{\partial \mathcal{I}_p} w_{jp} \mathcal{Z}_j \quad \because \text{ from (5.2.9)} \\
\Leftrightarrow\ & \tfrac{\partial \mathcal{L}}{\partial \mathcal{I}_p} \left(w_{ip} \mathcal{Z}_i - w_{jp} \mathcal{Z}_j \right) < 0.
\end{aligned} \tag{5.2.10}$$

As with all gradient-based optimization methods, the loss $\mathcal{L}(\tilde{\mathcal{I}}_p)$ can be approximated with its Taylor expansion, as shown in Fig. 5.3. When $\tilde{\mathcal{I}}_p$ is within the open neighborhood of $\mathcal{I}_p$, $\mathcal{L}(\tilde{\mathcal{I}}_p)$ can be approximated as

$$\begin{aligned}
\mathcal{L}(\tilde{\mathcal{I}}_p) &= \mathcal{L}(\mathcal{I}_p + (\tilde{\mathcal{I}}_p - \mathcal{I}_p)) \\
&\approx \mathcal{L}\left(\mathcal{I}_p\right) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_p} (\tilde{\mathcal{I}}_p - \mathcal{I}_p) \\
&= \mathcal{L}\left(\mathcal{I}_p\right) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_p} \left(w_{ip} \mathcal{Z}_i - w_{jp} \mathcal{Z}_j \right).
\end{aligned} \tag{5.2.11}$$

From (5.2.10), it is obvious that $\mathcal{L}(\tilde{\mathcal{I}}_p) < \mathcal{L}\left(\mathcal{I}_p\right)$, proving that `edge-popup` decreases the loss by swapping edges.

However, what would happen to scores if there was no swapping? If swapping stops, the score gradients $\frac{\partial \mathcal{L}}{\partial s_{uv}}$ are constant, and therefore scores will constantly increase or decrease until swapping occurs, as no swapping means no change in the subnetwork nor in the

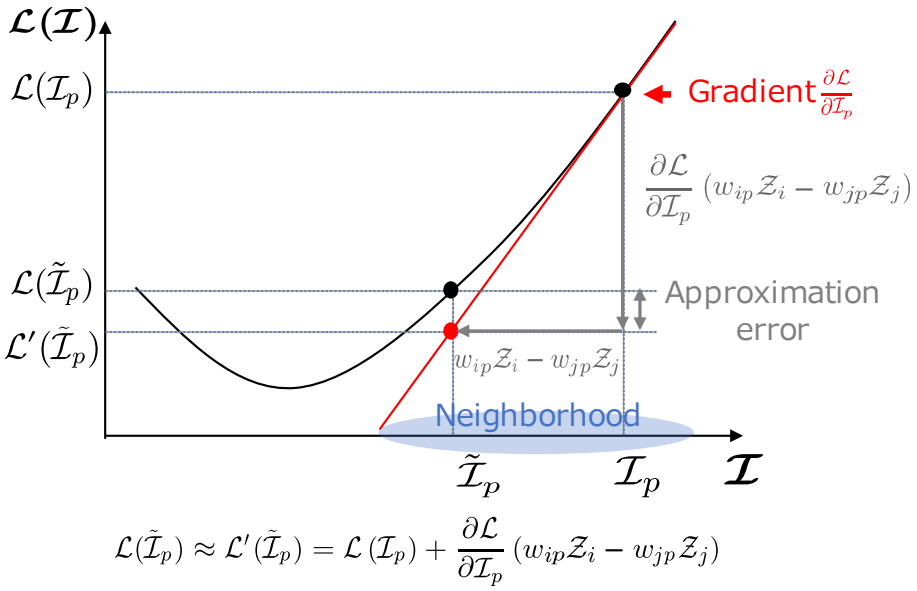


Fig. 5.3: First-order Taylor expansion of the loss on Hidden Networks.To keep the approximation error small, $\tilde{\mathcal{I}}_p$ needs to be within the neighborhood of $\mathcal{I}_p$.

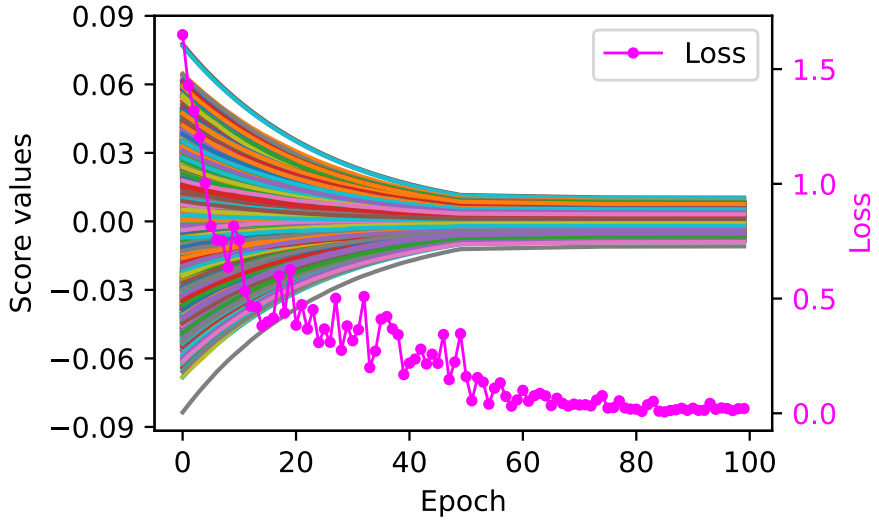


Fig. 5.4: Evolution of loss and `edge-popup` scores with weight decay during training. Solid lines correspond to 10 000 scores chosen randomly from the last convolutional layer of ResNet-18.

loss. In practice, however, this does not happen due to the weight decay applied to scores. Fig. 5.4 shows that most scores reach a plateau after 50 epochs, after which they do not constantly increase or decrease. But this does not mean that their gradients are zero: it is the effect of the balance between the score increase and the weight decay, i.e., their speeds of increment and decay are almost equal. Although the continuously decreasing loss reveals that the subnetwork still undergoes slight changes, these underutilized gradients can be exploited for further profit. The underlying idea of the proposed method is to take advantage of these non-zero score gradients for enhancing the performance of Hidden Networks.

5.3 Multicoated Supermasks

This section proposes a method that extends Hidden Networks to use multiple supermasks–bundled into a single Multicoated Supermask—instead of a single supermask. Although using multiple supermasks for multiple tasks (one mask for each task) has been proposed before [162], this work was the first to discuss multiple supermasks for a single task.

The proposed method, illustrated in Fig. 5.5, employs N supermasks—N coats—of descending density. That is, additional coats select a subset of the connections selected by the previous coat. Because of this information redundancy, and since all supermask coats use identical scores, this method has no additional training cost compared with Hidden Networks. Additionally, this redundancy is exploited to compress the Multicoated Supermasks.

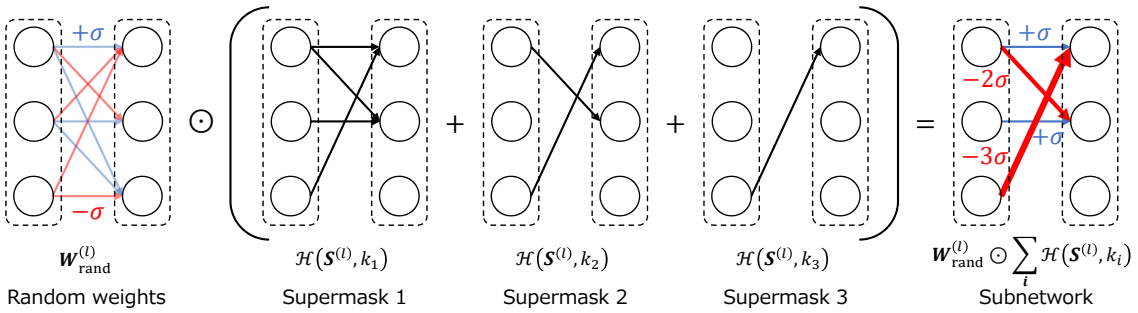


Fig. 5.5: An example of Multicoated Supermask with three coats and Signed Kaiming Constant weights. Each coat consists of a supermask with a different density top-k%: in this figure $k_1 > k_2 > k_3$. Since all coats use the same score matrix $\boldsymbol{S}^{(l)}$, all edges in the k_3 coat are included in the k_2 coat, which in turn is a subset of the k_1 coat.

5.3.1 Formulation and Proof

With the proposed Multicoated Supermasks, (5.2.2) can be rewritten as

$$\boldsymbol{W}^{(l)} = \sum_{k_n \in \mathcal{K}} \mathcal{H}(\boldsymbol{S}^{(l)}, k_n) \odot \boldsymbol{W}^{(l)}_{\text{rand}}, \tag{5.3.1}$$

where $\mathcal{K}$ is a set of N supermask densities. Denoting the element of matrix $\boldsymbol{A}$ at (x, y) as $(\boldsymbol{A})_{xy}$, the coordinates of non-zero elements in a supermask coat can be written as

$$\mathbb{H}^{(l)}_k = \left\{ (u, v) \middle| k > 0,\ \left(\mathcal{H}(\boldsymbol{S}^{(l)}, k) \right)_{uv} \neq 0 \right\}. \tag{5.3.2}$$

Then, the following statement holds between $\mathcal{H}_k$ with different density k_n:

$$k_i < k_j \Rightarrow \mathbb{H}^{(l)}_{k_i} \subset \mathbb{H}^{(l)}_{k_j}, \tag{5.3.3}$$

meaning that the non-zero elements of a supermask coat with smaller k_n are a subset of those in a coat with larger k_n, and so is the corresponding set of coordinates (i.e., this method "applies several coats" of the learned supermask†). These non-zero indices of each supermask coat are all the information necessary for constructing the desired subnetwork, and the redundancy of the subsets can be exploited for compressing them.

For calculating the output of a FC layer with a Multicoated Supermask, (5.2.7) is extended to

$$\begin{aligned} \mathcal{I}_v = \sum_{u \in \mathcal{V}^{(l-1)}} w_{uv} \mathcal{Z}_u \sum_{n=1}^{N} h_{k_n}(s_{uv}) \\ \text{s.t.} \quad k_1 > k_2 > \cdots > k_N > 0, \end{aligned} \tag{5.3.4}$$

where N is the number of total coats, and k_n is the density of the n-th coat. Notice that when $N = 1$, (5.3.4) is identical to (5.2.7), allowing to consider Hidden Networks as a special case of the proposed method.

The score update rule in (5.2.9) is rewritten just by introducing N as

$$\tilde{s}_{uv} = s_{uv} - \lambda \frac{\partial \mathcal{L}}{\partial \mathcal{I}_v} N w_{uv} \mathcal{Z}_u, \tag{5.3.5}$$

assuming STE over the summation of step functions in (5.3.4). Therefore, Multicoated Supermasks can be proved in the same manner as Hidden Networks based on (5.3.4) and

†Naming inspired in the multicoating of laquerware or paint, in which the same product is applied in multiple coats, progressively thinner.

(5.3.5). Here the proof is divided into two possible scenarios: the case of adding an additional coat, and the case of a swap between two existing coats.

In the case of adding a new coat, the loss $\mathcal{L}(\tilde{\mathcal{I}}_p)$ is approximated as

$$\begin{aligned}\mathcal{L}(\tilde{\mathcal{I}}_p) &\approx \mathcal{L}\left(\mathcal{I}_p\right) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_p}(\tilde{\mathcal{I}}_p - \mathcal{I}_p) \\ &= \mathcal{L}\left(\mathcal{I}_p\right) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_p}\left\{c w_{ip}\mathcal{Z}_i - (c-1)w_{ip}\mathcal{Z}_i\right\} \\ &= \mathcal{L}\left(\mathcal{I}_p\right) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_p} w_{ip}\mathcal{Z}_i, \end{aligned} \tag{5.3.6}$$

where $c \in \{0, 1, \ldots, N\}$ represents the number of coats accumulated in (5.3.4), and the term $\frac{\partial \mathcal{L}}{\partial \mathcal{I}_p} w_{ip}\mathcal{Z}_i$ is less than 0 because

$$\begin{aligned} &\tilde{s}_{ip} - s_{ip} > 0 \\ \Leftrightarrow\ &-\lambda \tfrac{\partial \mathcal{L}}{\partial \mathcal{I}_p} N w_{ip}\mathcal{Z}_i > 0 \quad \because \text{ from (5.3.5)} \\ &\Leftrightarrow \tfrac{\partial \mathcal{L}}{\partial \mathcal{I}_p} w_{ip}\mathcal{Z}_i < 0. \end{aligned} \tag{5.3.7}$$

Consequently, the updated loss $\mathcal{L}(\tilde{\mathcal{I}}_p)$ becomes smaller than the previous loss $\mathcal{L}\left(\mathcal{I}_p\right)$ by adding an additional coat.

In the case of an edge swap happening between two existing coats, the loss $\mathcal{L}(\tilde{\mathcal{I}}_p)$ can be written as

$$\begin{aligned}\mathcal{L}(\tilde{\mathcal{I}}_p) &\approx \mathcal{L}\left(\mathcal{I}_p\right) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_p}(\tilde{\mathcal{I}}_p - \mathcal{I}_p) \\ &= \mathcal{L}\left(\mathcal{I}_p\right) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_p}\left\{c w_{ip}\mathcal{Z}_i - (c-1)w_{ip}\mathcal{Z}_i\right\} \\ &\quad + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_p}\left\{(c-1) w_{jp}\mathcal{Z}_j - c w_{jp}\mathcal{Z}_j\right\} \\ &= \mathcal{L}\left(\mathcal{I}_p\right) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_p}\left\{w_{ip}\mathcal{Z}_i - w_{jp}\mathcal{Z}_j\right\}, \end{aligned} \tag{5.3.8}$$

which is identical to (5.2.11). As with (5.2.10), the updated loss $\mathcal{L}(\tilde{\mathcal{I}}_p)$ is also smaller than the loss before the update $\mathcal{L}\left(\mathcal{I}_p\right)$, as

$$\begin{aligned} &\tilde{s}_{ip} - s_{ip} > \tilde{s}_{jp} - s_{jp} \\ \Leftrightarrow\ &-\lambda N \tfrac{\partial \mathcal{L}}{\partial \mathcal{I}_p} w_{ip}\mathcal{Z}_i > -\lambda N \tfrac{\partial \mathcal{L}}{\partial \mathcal{I}_p} w_{jp}\mathcal{Z}_j \\ &\Leftrightarrow \tfrac{\partial \mathcal{L}}{\partial \mathcal{I}_p}\left(w_{ip}\mathcal{Z}_i - w_{jp}\mathcal{Z}_j\right) < 0. \end{aligned} \tag{5.3.9}$$

Therefore, in both the case of adding additional coats and the case of a swap between coats, the Multicoated Supermask approach works.

So far we have discussed Multicoated Supermasks for fully connected neural networks. Due to the similarity with Hidden Networks, Multicoated Supermasks also extends to convolutional neural networks in almost the same way as Hidden Networks [128](Section B.2), except for the additional N from the STE of the step functions used in Multicoated Supermasks.

5.3.2 Narrowing Down the Hyperparameter Space

Although Multicoated Supermasks can be trained with `edge-popup` as well, it is necessary to determine a density k_n for each of the N coats. To avoid the additional cost of exploring a bigger hyperparameter space, this section proposes two simple strategies for determining k_n from the total density of the model top-k% (i.e., the density of the first coat, k_1): Linear and Uniform.

These two methods, portrayed in Fig. 5.6, assume that the magnitude of learned scores follows a folded normal distribution (although, in reality, the concentration of scores around zero is slightly more prominent), and each uses a different way of partitioning this probability density function to obtain the necessary k_n. These two methods are compared experimentally in Section 5.4.2.

The Linear method (Fig. 5.6a) partitions the score magnitude segment $[s_{t_1}, \alpha]$ at even score magnitude intervals, where s_{t_1} is the threshold score with density k_1, and $\alpha = s_{t_1} + 3\sigma_s$

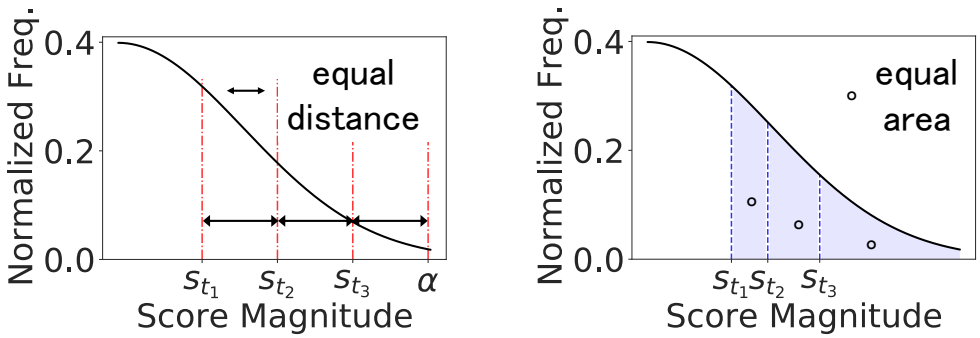


Fig. 5.6: Two methods for setting the density k_n of each coat n: (a) uses equidistant threshold scores s_{t_n}; (b) considers area under the curve of a folded normal distribution. $\alpha = s_{t_1} + 3\sigma_s$, where σ_s is the scores' standard deviation.

is defined as the upper bound for partitioning, in which σ_s is the standard deviation of the score distribution. The range of $3\sigma_s$ from s_{t_1} covers most of the scores range regardless of s_{t_1}. Then, the threshold score s_{t_n} for each density k_n is calculated by

$$s_{t_n} = s_{t_1} + (\alpha - s_{t_1})\frac{n-1}{N}, \tag{5.3.10}$$

and then, using (5.2.5) and (5.2.6),

$$k_n = t_n/UV. \tag{5.3.11}$$

The Uniform method (Fig. 5.6b), on the other hand, uses even portions of area under the curve to partition the score magnitude segment $[s_{t_1}, \infty)$. Thus, k_n is simply defined as

$$k_n = k_1\frac{N+1-n}{N}. \tag{5.3.12}$$

5.3.3 Multicoated Supermask Encoding

Since weights are random, it is only necessary to store the supermasks and the random seed used at training for generating weights. Supermask coats with higher density contain those with lower density (see (5.3.3)), so for an additional coat n it is only necessary to store the elements located at the coordinates of non-zero elements of the previous coat $n-1$, i.e., $\mathbb{H}^{(l)}_{k_n-1}$. Therefore, the total number of bits of a unary encoded Multicoated Supermask is

$$\begin{cases} UV, & \text{for } N = 1 \\ \left(1 + \sum_{n=2}^{N} k_{n-1}\right) UV, & \text{for } N > 1 \end{cases}. \tag{5.3.13}$$

The model sizes reported in Section 5.4 assume this optimal encoding.

5.3.4 Single coated and Multicoated Supermasks Comparison

As mentioned above, Hidden Networks can be considered a special case of Multicoated Supermasks that only use the first coat. Since additional coats are subsets of this first coat, they do not modify network connectivity. However, the scaling they provide expands the parameter search space, leading the learning algorithm to better quality models by exploiting the backpropagated information more effectively. As will be discussed in Section 5.4, Multicoated Supermasks achieve this by partially restoring the relationship between a connection's importance and its magnitude in the activation's linear combination—which Hidden Networks broke into score and random weight, respectively.

Since all coats are computed from the first one, multicoating has a trivial impact on training cost. Compared with the binary supermasks of Hidden Networks, encoding all coats into a single Multicoated Supermask makes them a scalar supermasks. In specialized hardware this may entail a negligible added cost in its application to the weights, but none in standard processors. The only significant inconvenience introduced by this method is the additional memory needed for storing more coats. However, as demonstrated in Section 5.4, increasing supermask size is more efficient than increasing model size, whether it is with extra layers or wider channels, as it results in models with similar memory size and less computational cost, but higher accuracy.

Previous work found success in reusing a single random network for multiple tasks by training a supermask for each task and hot swapping them accordingly [162]. Multicoated Supermasks differs in that it uses multiple supermasks for a single task. However, these two approaches are compatible: one Multicoated Supermask can be trained for each mask, raising the accuracy for each task while retaining the capacity of hot-swapping supermasks.

5.4 Experiments and Results

This section evaluates the performance of Multicoated Supermasks on image classification and compares it to Hidden Networks using the experimental settings described in section 5.4.1. First, a small-scale dataset is used to investigate the behaviour of the proposed method: Section 5.4.2 compares the different methods explained in Section 5.3.2 for setting the density of each supermask coat, while Section 5.4.3 explores the relationship between total model density and accuracy. Then, a large-scale dataset is used to evaluate the performance of the proposed method and its size to accuracy tradeoff by testing different supermask sizes in Section 5.4.4 and different model sizes in Section 5.4.5.

5.4.1 Experimental Settings

Multicoated Supermasks are hereafter evaluated for image classification using the CIFAR-10 [78] and ImageNet [132] datasets. In both cases residual networks [53] are trained for 100 epochs using stochastic gradient descent (SGD) with weight decay of 0.0001 and momentum of 0.9. As in Chapter 4, Wide ResNets [169] are used for wider channel models. In CIFAR-10 experiments the learning rate is decreased by 0.1 after 50 and 75 epochs, starting from 0.1 with a batch size of 128; in ImageNet experiments, the learning rate is reduced using cosine annealing starting from 0.1, with a batch size of 256. Following [128] and Chapter 4, these

experiments use Signed Kaiming Constant (SKC‡) weight initialization with a scaling factor of $\sqrt{1/k_1}$, depending on the sparsity k_1. However, instead of non-affine BatchNorm, our experiments use affine batch normalization (i.e., BatchNorm's learnable parameters are updated). All models and experiments are implemented using MMClassification [106], a toolbox based on PyTorch [123]. All reported model sizes for models using supermasks use the encoding described in Section 5.3.3. Reported accuracy is the average of three runs for CIFAR-10 experiments, and a single run for ImageNet experiments. An early version of the implementation of this method and its experiments can be found publicly available at [119].

5.4.2 Setting Each Coat's Density

Fig. 5.7 compares the two methods for setting k_n proposed in Section 5.3.2 (Linear and Uniform) by testing them on CIFAR-10 using ResNet-18 and a fixed $k_1 = 30\%$. This figure shows that the proposed method is effective: although only slightly, both methods surpass the accuracy of the baseline Hidden Networks, and accuracy grows with the number of coats. Notably, the biggest gap in accuracy is observed between the single-coated and the

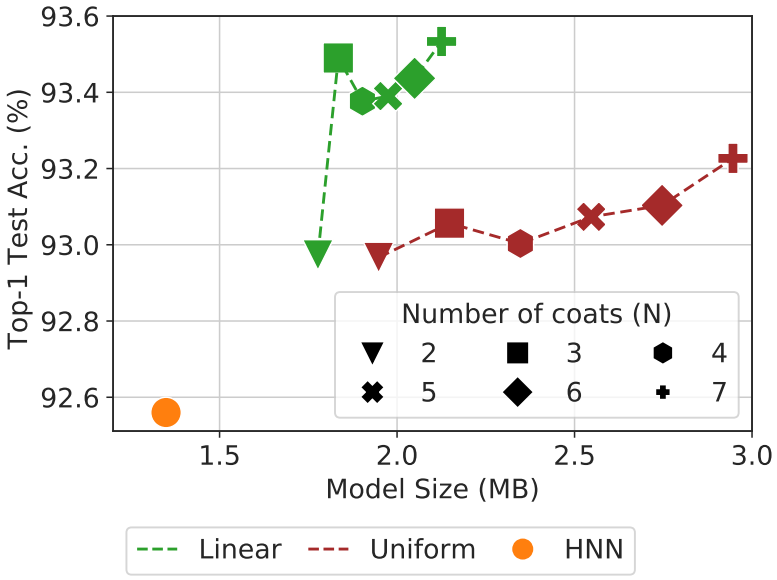


Fig. 5.7: Comparison between the proposed methods for setting k_n using ResNet-18 on CIFAR-10 with a fixed $k_1 = 30\%$. Linear produces smaller and more accurate models.

‡Alternatively abbreviated as SC in Chapter 4.

double-coated supermasks.

Even though the difference in accuracy between the two methods is narrow, Linear consistently outperforms Uniform. Furthermore, since Linear prefers larger additional threshold scores, it produces sparser coats, and thus significantly smaller models. Since Linear produces both the smallest and most accurate models, it is the method used hereafter.

This result suggests that the additional expressive power of Multicoated Supermasks comes from partially restoring the linear relationship between connectivity strength—scores—and weight magnitude of conventional weight learning. For all practical purposes, when applied to models initialized with Signed Kaiming Constant (see Chapter 4.3.3), a Multicoated Supermask extends the dynamic range of effective weights without weight learning, functioning as a concurrent blend of pruning and quantization.

5.4.3 Effect of Total Density on Accuracy

Fig. 5.8 shows the results of investigating the relationship between the total supermask density top-k% (i.e., the density of the first coat, k_1) and the accuracy of the resulting subnetwork. It shows that Multicoated Supermasks achieve higher accuracy than the single-coated Hidden Networks for practically any density value. The best results are obtained in the range $40 \leq k_1 \leq 20$, with a maximum at $k_1 = 40\%$ of 93.54%, close to the dense model's 94.14%.

It can also be appreciated that, although both single-coated and multicoated supermasks degenerate critically at high sparsity values, additional coats prevent accuracy degradation on dense supermasks ($k_1 > 70\%$). This result proves the hypothesis that Multicoated Supermasks help to mitigate the impact of using random weights.

5.4.4 Supermask Size to Accuracy Tradeoff

This section evaluates how supermask size affects accuracy and model size using ResNet-50 on ImageNet. The size of Multicoated Supermasks depends on two hyperparameters: the number of coats N and the total density k_1. This experiment probes the best ranges of N and k_1 found in the previous subsections.

Fig. 5.9 shows that not all ways of increasing supermask size are equal: while accuracy grows monotonically with number of coats, there is an optimal density value around $k_1 = 30\%$. Blending N and k_1 offers an almost linear tradeoff between accuracy and model size. The highest accuracy, obtained with $k_1 = 30$ and $N = 7$, overperforms Hidden Networks by 5.65%, while only requiring a 1.67× bigger model size.

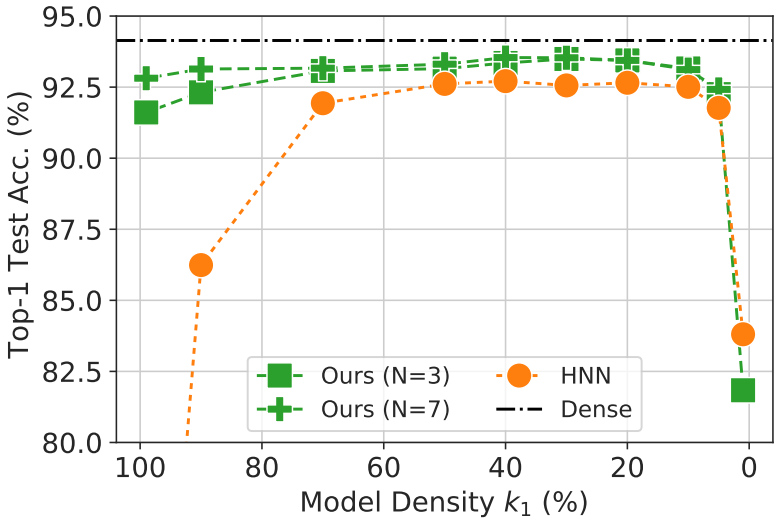


Fig. 5.8: Varying the value of density k_1% while setting the rest of k_n with the Linear method. ResNet-18 on CIFAR-10 with N-coated supermasks.

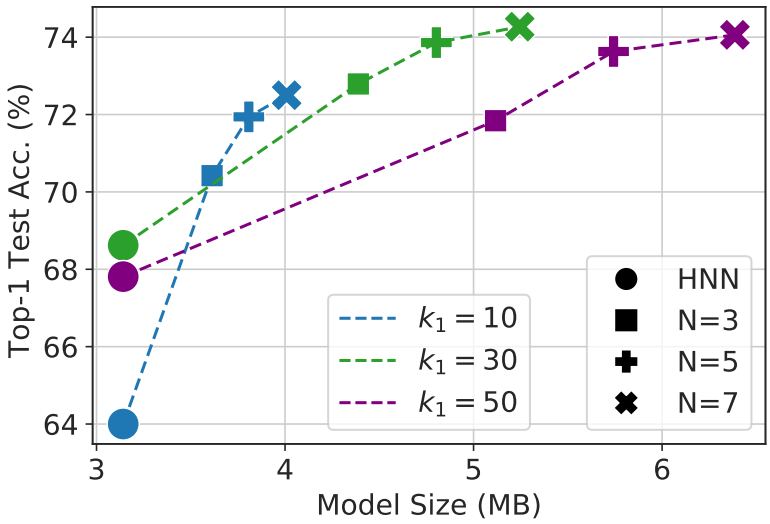


Fig. 5.9: Effect of supermask size (determined by first coat density k_1% and number of coats N) on accuracy and model size using ResNet-50 and ImageNet. HNN: Hidden Networks (i.e., $N = 1$).

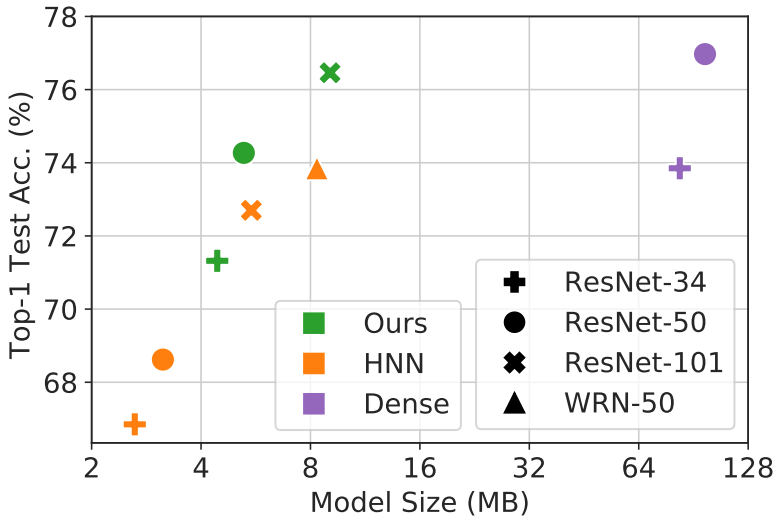


Fig. 5.10: Comparison of Multicoated Supermasks ($N = 7$, $k_1 = 30\%$, Linear), Hidden Networks ($k_1 = 30\%$) and dense ResNet models on ImageNet. WRN: Wide ResNet. The proposed method outperforms Hidden Networks, and matches the dense model, despite the 10× model size reduction.

5.4.5 Model Size to Accuracy Tradeoff

Fig. 5.10 evaluates the performance of Multicoated Supermasks ($N = 7$, $k_1 = 30\%$, Linear) when considering different model sizes, and compares it to the the baseline Hidden Networks ($k_1 = 30\%$) and dense models.

The enhanced expressive power granted by the additional supermask coats delivers an outstanding tradeoff between accuracy and model size. A multicoated ResNet-101 achieves higher accuracy (76.46%) than the best performing single-coated model, Wide ResNet-50 (73.85%), despite having similar model size (9.0 MB vs. 8.3 MB). Even more remarkably, this model's accuracy is only 0.5% lower than that of a dense ResNet-50, while having a model size 10.78× smaller. These results prove that Multicoated Supermasks achieve higher accuracy for the same model size and smaller size for the same accuracy even on a large-scale dataset, with no added training cost and negligible additional inference cost.

5.4.6 Effect of Weight Initialization

Fig. 5.11 shows the impact on accuracy of using standard Kaiming initialization (KN, [52]) instead of Signed Kaiming Constant initialization (SKC, see Chapter 4.3.3). Similarly

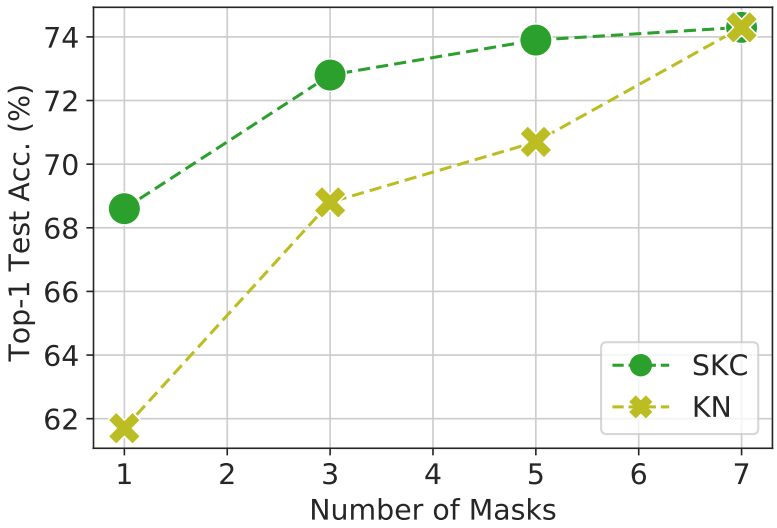


Fig. 5.11: Impact of weight initialization on ImageNet using ResNet-50 ($k_1 = 30\%$, linear). SKC is signed Kaiming constant initialization, and KN is Kaiming normal initialization.

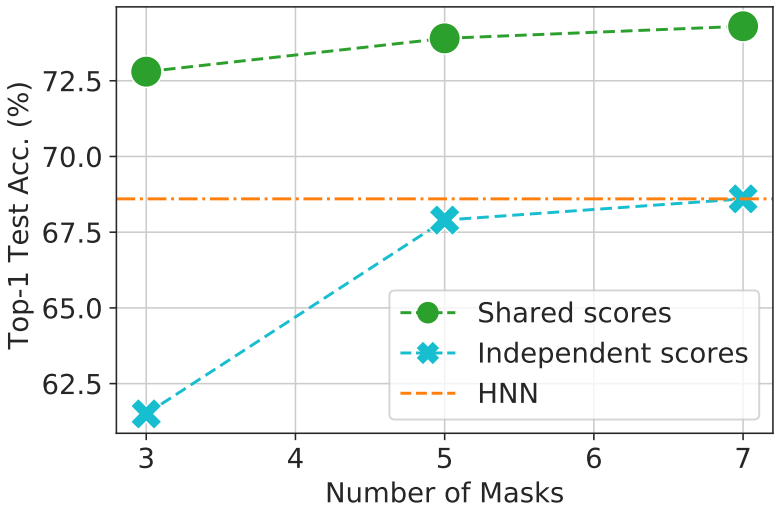


Fig. 5.12: Impact of learning masks with independent scores instead of a shared score. ResNet-50 ($k_1 = 30\%$, Linear, SKC) on ImageNet.

to [128], the results show that SKC achieves a higher accuracy than KN. However, it can be observed that, as the search space expands with the additional coats, this difference disappears.

5.4.7 Multicoated Supermasks vs. Multiple Supermasks

To demonstrate the effectiveness of using a Multicoated Supermask instead of multiple independent supermasks, an experiment is carried out using an independent score tensor for each mask instead of shared scores. Fig. 5.12 shows that, independently of the number of masks, shared scores result in higher accuracy than independent scores. Since the multiple coats have the combined effect of scaling each weight depending on its importance (i.e., its score), having multiple scores for each weight introduces redundant scores in the best case, and orthogonal scores in the worst case, which eliminate the scaling effect and produce denser subnets.

5.4.8 Comparison With Other Approaches

Table 5.1 compares the inference accuracy, inference FLOPS, and model size of the proposed method with pruning [38], and with two sparsity training methods: RigL [28] and MEST+EM&S [167]. Model sizes of the compared literature are calculated by considering nonzero elements as 32-bit [179]. Compared with approaches with the same sparsity and computation, Multicoated Supermasks result in models 5× smaller, although slightly less accurate. By increasing the number of layers and density, the proposed method outperforms

TABLE 5.1: Comparison on ImageNet of the proposed method with other sparsity training methods. Here GFLOPS consider ideal sparse processing. RN: ResNet.

Method	Model	Sparsity (%)	Inference GFLOPS	Model Size (MB)	Top-1 Acc. (%)
Dense	RN-50	-	8.2	98	77.1
Pruning [38]	RN-50	90	0.9	20	75.2
RigL [28]	RN-50	90	0.9	20	75.7
MEST+EM&S [167]	RN-50	90	0.9	20	76.1
MSup ($N = 7$)	RN-50	90	0.9	4	72.5
MSup ($N = 7$)	RN-50	70	2.5	5	74.3
MSup ($N = 7$)	RN-101	70	4.8	9	76.5

other works while still keeping the model size more than 50% smaller. These results show that Multicoated Supermasks are an efficient option for memory-constrained applications.

5.5 Discussion and Conclusion

This chapter analyzes Hidden Networks [128], showing that their limit in performance is due to supermask optimization stopping before score gradients converge to the minima. This finding that `edge-popup` is not uncovering the best possible connectivity pattern suggests that randomly weighted neural networks may be hiding even better performing subnetworks than previously thought, aligning with the conclusions of [31].

This problem is addressed by enhancing the random subnetwork's expressive power with Multicoated Supermasks, which use multiple supermasks to scale random weights. This method delivers a better accuracy to model size tradeoff, showing that increasing the number of supermasks can be more effective than adjusting sparsity or channel width for compensating the accuracy loss of models with random weights.

With a broader view, the high performance and small size of supermasked models point to a promising trend of lightweight neural networks combining pruning, quantization, and random weights.

Impact

The work in this chapter was presented as a conference paper [118] at the *39th International Conference on Machine Learning (ICML)*, Baltimore, Maryland, USA, July 2022. An early version of the implementation of this method and its experiments can be found publicly available at [119].

This research was supported by JST CREST Grant Number JPMJCR18K2 and JSPS KAKENHI Grant Numbers JP18H05288, JP22H03555, Japan.

Chapter 6

The *Ternary Strong Lottery Ticket Hypothesis*: A Novel Framework for Weight Randomness and Quantization

The previous chapters take the first steps towards reservoir computing (RC)-inspired deep learning (DL) by leveraging the capability of learning using random fixed weights of the Strong Lottery Ticket Hypothesis (SLTH). However, the SLTH is limited to random weights and *learned sparse connectivity*. Some reservoirs, like the ReCA presented in Chapter 2.1.1, employ a *dense network* of complex units. Others, like the ReFRET described in Chapter 2.1.2, obtain their complex behavior from a *random nonlinear connectivity*. Furthermore, the capacity to control the behavior of the reservoir (i.e., its degree of complexity or randomness) is an essential factor in RC design, as exemplified in the choice of rule in Chapter 3.2. The SLTH, so far, does not allow this control.

This chapter targets these two shortcomings by proposing a new way of looking at the SLTH: the *Ternary SLTH* (T-SLTH). This new look expands the types of randomness exploitable by the SLTH to three, including arbitrary connectivity. Furthermore, it allows straightforward control over the degree of randomness in RC-like DL models.

In addition to exploring randomness, the T-SLTH also bridges the SLTH with pure weight quantization, providing a novel framework for concurrent learning, pruning, and quantization. By building on the methods proposed in Chapter 4 and Chapter 5, this chapter presents a rich tradeoff between model size, accuracy, and randomness that spans from very tiny to highly accurate SLTH models, setting the SOTA in both ends. This framework is the foundation for the versatile neural engine accelerator proposed in Chapter 7, capable of processing quantized models with or without weight randomness.

6.1 Introduction

The Strong Lottery Ticket Hypothesis (SLTH) [128, 177] hypothesized that modern deep neural networks (DNN) are so overparameterized that they already contain highly efficient subnetworks in their randomly initialized state. Furthermore, it demonstrated that these subnetworks can be found without learning weights by simply training a binary pruning mask—a *supermask*.

Although the original SLTH models were less accurate than their weight-training dense counterparts, several improvements on supermasks have been proposed to raise their accuracy. Signed Supermasks (*SSup*) [77] improved the accuracy and sparsity range of the original method by adding learned signs in the supermask. Similarly, Multicoated Supermasks (*MSup*) [118] (proposed in Chapter 5) raised accuracy by adding learned scalar magnitudes to the supermask. Hidden-Fold Networks (HFN) [93, 96] (proposed in Chapter 4) exploited the nature of ResNet's [53] architecture to enhance its subnetworks by transforming it into a recurrent network.

Nonetheless, upon examination it becomes apparent that some of these methods are not exploiting any weight randomness. This chapter takes a look at the different supermask types and questions if the SLTH is actually finding lucky subnetworks, or is just a special case of weight quantization. For this examination, it proposes a reinterpretation of the SLTH–the *ternary SLTH* (T-SLTH). From this novel perspective, this chapter provides the following contributions:

- It proposes a new supermask type that surpasses the SOTA accuracy in image classification for supermask models—the *Signed-Multicoated Supermask* (SM).
- It analyzes the types and degrees of randomness of each supermask type.
- It connects the SLTH with quantization, and presents a novel framework for concurrent learning, quantization, and pruning.
- It extends supermask methods to dense connectivity, and then to random connectivity through random pruning at initialization.

The remainder of the chapter is organized as follows. After proposing the SM supermask, Section 6.2 analyzes the existing supermask methods and presents a reinterpretation of the STLH, which is then leveraged to extend supermasks to dense and random connectivity. Section 6.3 tests the proposed methods on image classification using ResNet. Finally, Section 6.4 concludes the chapter with a discussion.

6.2 A Reinterpretation of the SLTH

6.2.1 Existing Supermasks and the Signed-Multicoated Supermask

The Connectivity (*C*) Supermask∗ and the `Edge-Popup` Algorithm [128]:

The foundational work on the SLTH proposed finding the subnetworks that have "won the lottery at initialization" by training a binary pruning supermask. This supermasks is trained with backpropagation using the `edge-popup` algorithm. First, all weights are assigned an additional parameter: a *score*. Supermasks are applied during inference to uncover a subnetwork, but they are not applied during the backpropagation stage, using instead straight-through estimation. Since weights are fixed, their gradients are used to update their corresponding scores instead. The supermask is updated every iteration by zeroing as many positions as necessary to keep a predefined sparsity, removing the elements corresponding to the scores with smallest magnitudes. This process is applied repeatedly in epochs in similar fashion to standard weight learning. Equivalently, this can be viewed as to quantizing the magnitude of scores into binary values (i.e., $\{0, 1\}$).

The Signed Supermask (*SSup*) [77]:

SSup builds on the *C* supermask by adding the sign information of the learned scores to the supermask. This is done by simply copying the signs of the scores to their corresponding supermask elements. Effectively, this is equivalent to quantizing the scores into balanced ternary values (i.e., $\{-1, 0, +1\}$).

The Multicoated Supermask (*MSup*) [118]:

MSup obtains multiple *C* supermasks of different sparsity (coats) from the same learned scores and bundles them into a scalar supermask. This process can be interpreted as a quantization of score magnitudes into integers (i.e., $[0..N]$, where N is the number of coats).

The *Signed-Multicoated supermask (SMSup)*:

This chapter proposes to blend the *SSup* and *MSup* into a new type of supermask, simply by first computing the *MSup* and then copying to it score signs in the same manner as *SSup*. Since this supermask is obtained by quantizing scores into signed integers (i.e., $[-N..+N]$, where N is the number of coats), it raises an important question: *does it leave any randomness left in the model after training?* And more generally, *where does the SLTH end and standard quantization begin?*

∗Referred to as *Hidden Network (HNN)* in other chapters.

6.2.2 Analysis of SLTH Randomness

The original idea behind the SLTH was to uncover effective subnetworks of random fixed weights by learning the connectivity with a *C* supermask. That is, the network connectivity is learned, and weights are left random. *SSup* builds on this by additionally learning weight signs, leaving only weight magnitudes random. Alternatively, *MSup* learns weight scalars (magnitudes), and leaves weight signs random.

Although these methods primarily focus on enhancing the supermask they all capitalize largely on the accuracy gains brought by the Signed Kaiming Constant (SC) weight initialization method. As opposed to the Kaiming normal (KN) weight initialization, which initializes weights by sampling from the normal distribution $\mathcal{N}_l(0, \sigma^2)$ with average 0 and standard deviation σ, SC initializes by sampling uniformly from $\{-\sigma, +\sigma\}$. Since σ can be viewed as a scaling factor, and be absorbed by the normalization layer scaling factor [56], SC can be interpreted to be initializing weights to the unit magnitude with a random sign (i.e., $\{-1, +1\}$).

Although seemingly just a technical detail, this initialization choice, has a fundamental consequence: it reduces the randomness of the initial weights to only their signs. Therefore, in the case of SC weight initialization, the only randomness in the subnetworks uncovered by a *C* supermask or *MSup* is the weight signs. The models resulting from training an *SSup* or an *SM* supermask, however, have *no randomness*: they are essentially equivalent to pure weight quantization (ternary and integer quantization, respectively). In these cases it is easy to see that `edge-popup`'s *scores* are acting essentially equivalently to standard quantization's *latent weights*.

TABLE 6.1: Supermaks with learned connectivity.
*: stronger randomness due to normal noise.

Supermask Type	Weight Initialization	Randomness Type			Degree of Randomness
		CX	SIGN	MAG	
C	KN	✗	✓	✓	2
	SC			✗	1
SSup ($C \cap S$)	KN		✗	✓	1
	SC			✗	0
MSup ($C \cap M$)	KN		✓	✗	1*
	SC				1
SMSup ($C \cap S \cap M$)	KN		✗		0*
	SC				0

Table 6.1 summarizes the types of randomness of each supermask type for each weight quantization method. Although most supermasks offer 1 or 2 degrees of randomness, signed supermasks with SC initialization offer none. Although even for KN initialization *MSup* supermasks make the biggest contribution to weight magnitudes, compared to their SC counterparts they also have an added normal noise, and therefore can be considered to have stronger degree of randomness, marked with a * for differentiation. This analysis has three consequences:

1. It offers a way of adjusting a model's degree of randomness from 0 to 2.
2. The SLTH models with no randomness cannot be considered RC-like.
3. It connects the SLTH with pure quantization through supermask learning.

6.2.3 The *Ternary* SLTH

SSup expands the *C* supermask's learned *connectivity* with learned *signs*, while *MSup* expands it with learned *magnitudes*. This chapter proposes detaching these three elements into *three fundamental supermasks*:

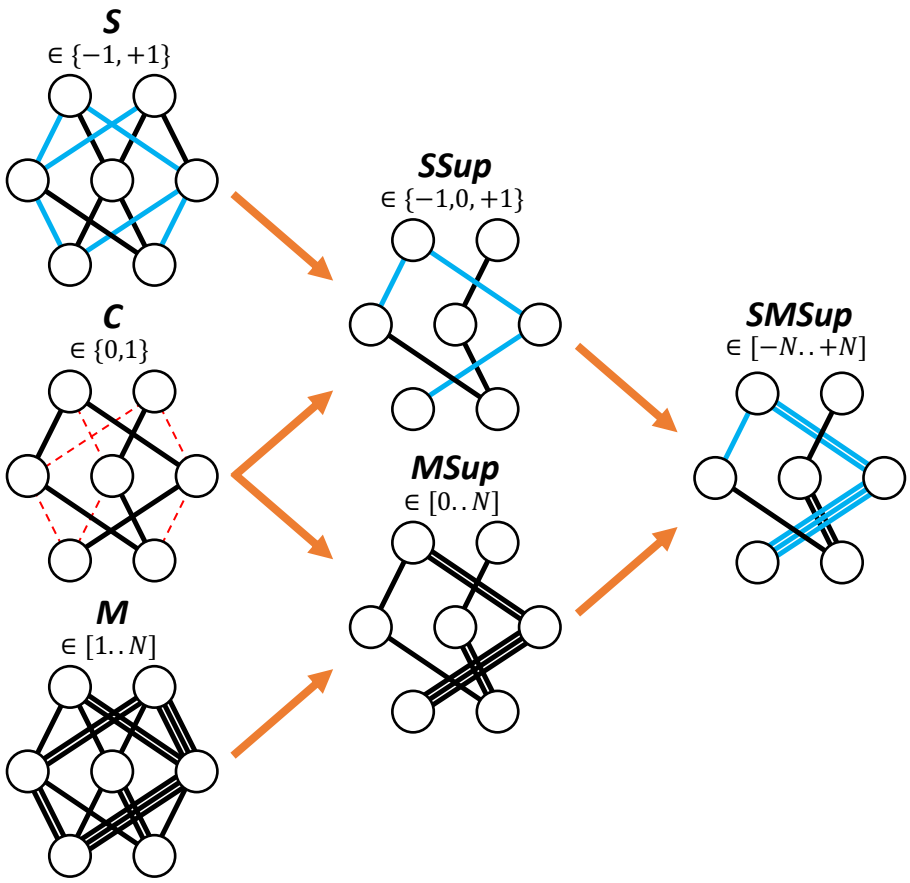


Fig. 6.1: Combination of the *C*, *S*, and *M* fundamental supermasks into the *SSup*, *MSup*, and *SM* supermasks.

- The C supermask learns connectivity by quantizing score magnitudes to 0, 1 according to a threshold or target sparsity, leaving weight magnitudes and signs untouched.
- The S supermask, that learns signs by quantizing scores to their sign $-1, +1$, leaving magnitudes and connectivity untouched.
- The M supermask, that learns magnitudes by quantizing scores to $[1..N]$, where N is an integer, leaving signs and connectivity untouched.

Then, *SSup* can be seen as a combination of C and S supermasks, and *MSup* as a combination of C and M supermasks. This reinterpretation of the supermasks is illustrated in Fig. 6.1.

Under this view S and M supermasks require no sparsity, and therefore it should be possible to train them with *dense connectivity*, and also to expand the SLTH to *random connectivity*. Now with three types of fundamental supermask, the SLTH can be expanded into **the *Ternary Strong Lottery Ticket Hypothesis (T-SLTH)*:**

> ***A randomly initialized DNN of arbitrary sparsity, if sufficiently overparameterized, contains enough useful information to perform accurately on an arbitrary task after only training its connectivity, signs, magnitudes, or any combination of them.***

Fig. 6.2 illustrates the T-SLTH. The three elemental supermasks can be combined in multiple ways. In the extreme case of combining the three of them, or the cases in which SC initialization reduces randomness to zero, the learning process is equivalent to concurrent learning, pruning, and quantization.

This chapter expands the SLTH to random connectivity, and implements it by with

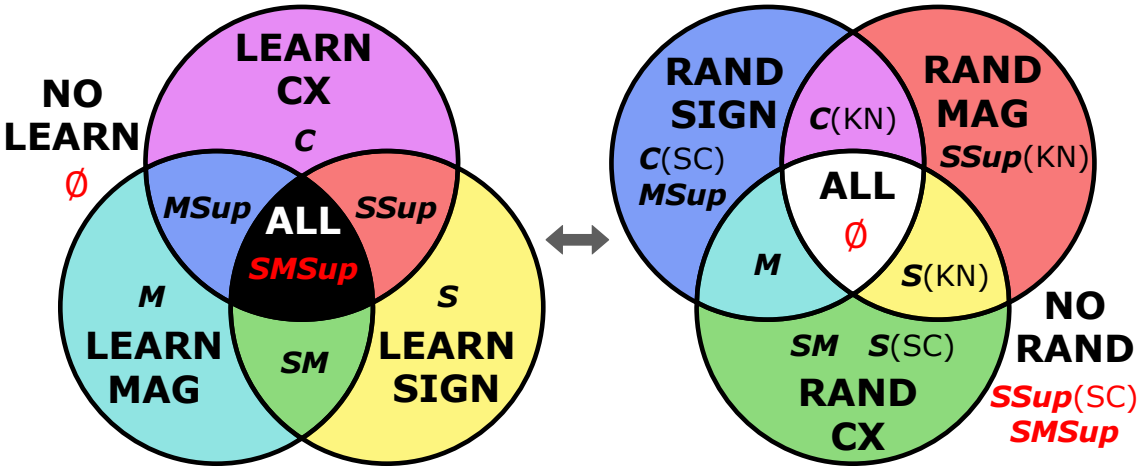


Fig. 6.2: Venn diagrams of the learned (left) and random (right) elements of each supermask type. CX: connectivity; MAG: magnitude.

random pruning at initialization (RPaI). The RPaI counterparts of *SSup*, *MSup*, and *SMSup* are the *S*, *M*, and *SM* supermasks, respectively. The respective dense counterparts (whose connectivity arguably cannot be considered learned nor random) are at the boundary between them.

Table 6.2 collects a randomness analysis of supermasks with random connectivity analogous to the one presented in Table 6.1 for supermasks with learned connectivity. This expansion of the SLTH allows to use at least one degree of randomness with any supermask and weight initialization method, opening a wide range of possibilities for RC-inspired DL models.

TABLE 6.2: Supermaks with random connectivity.
*: stronger randomness due to normal noise.

Supermask Type	Weight Initialization	Randomness Type			Degree of Randomness
		CX	SIGN	MAG	
C	KN	N.A.			
	SC				
S	KN	✓	✗	✓	2
	SC			✗	1
M	KN		✓	✗	2*
	SC				2
SM ($S \cap M$)	KN		✗		1*
	SC				1

6.3 Experiments and Results

This section tests the hypothesis proposed in this chapter on the CIFAR-100 and ImageNet image classification datasets using ResNet-50. The methodology is in general identical to that used in Chapter 4. The sparsity of each *MSup* coat is calculated following Chapter 5, and for dense or randomly pruned scenarios an additional dense coat is added. Similarly to the encoding method discussed in Chapter 5.3.3 for *MSup*, model size for signed supermasks only considers the signs of non-zero elements.

6.3.1 Fundamental Supermasks With Learned and Random Pruning

This section examines all the discussed supermask types by testing them for a double range of sparsity: learned connectivity and random connectivity are set on the same axis, separated by the point of dense connectivity.

Fig. 6.4 compares on said axis the *C*, *S*, and *M* fundamental supermasks for the case of SC weightinitialization, while Fig. 6.3 makes the same comparison for the case of KN weight initialization. As reported by previous work, SC initialization results in a higher classification accuracy for all models, although the difference is not critical. Therefore, following experiments only employ SC.

The results on the learned connectivity side show that adding *S* and *M* supermasks to the *C* supermask (i.e., *SSup* and *MSup*) have a similar effect of boosting accuracy and extending the effective sparsity range. From the view of the T-SLTH, this is natural, since *S* and *M* supermasks do not require learned connectivity. Indeed, at the point of dense connectivity they only suffer a small drop in accuracy, demonstrating that learned connectivity only makes a minor contribution to their efficacy.

The results on the random pruning side confirm the proposed T-SLTH: high accuracy can be achieved by only learning part of the elements of the network and leaving the rest randomly initialized; this is not limited to the learned connectivity and random weights of the SLTH. Although the accuracy with random connectivity is lower than with learned connectivity, it does not necessarily mean that there is some intrinsic advantage in finding subnetworks: where the randomness of weights uses the more sophisticated method of Kaiming initialization [52], the RPaI implemented here is purely random.

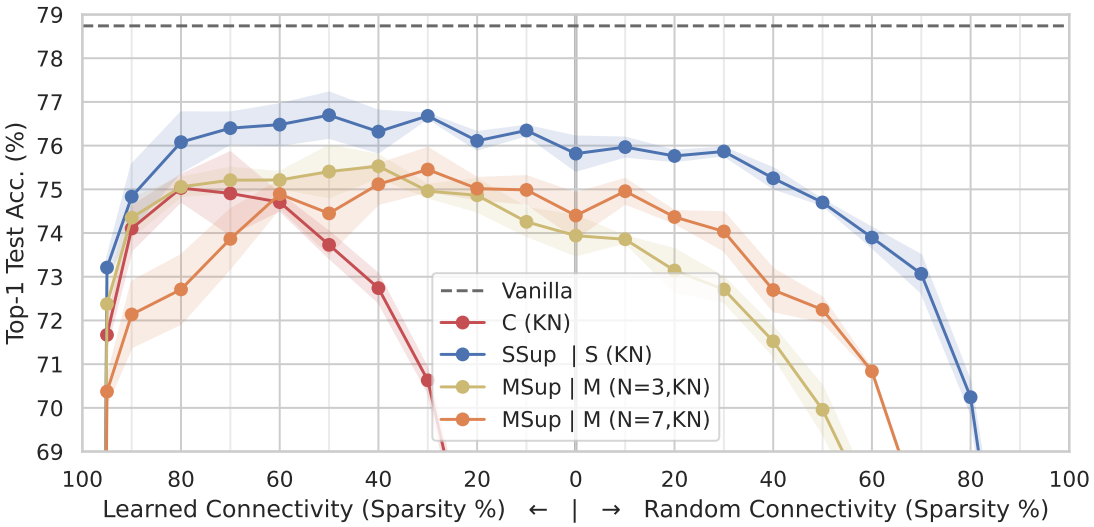


Fig. 6.3: Fundamental supermasks with learned and random pruning, KN initialization.

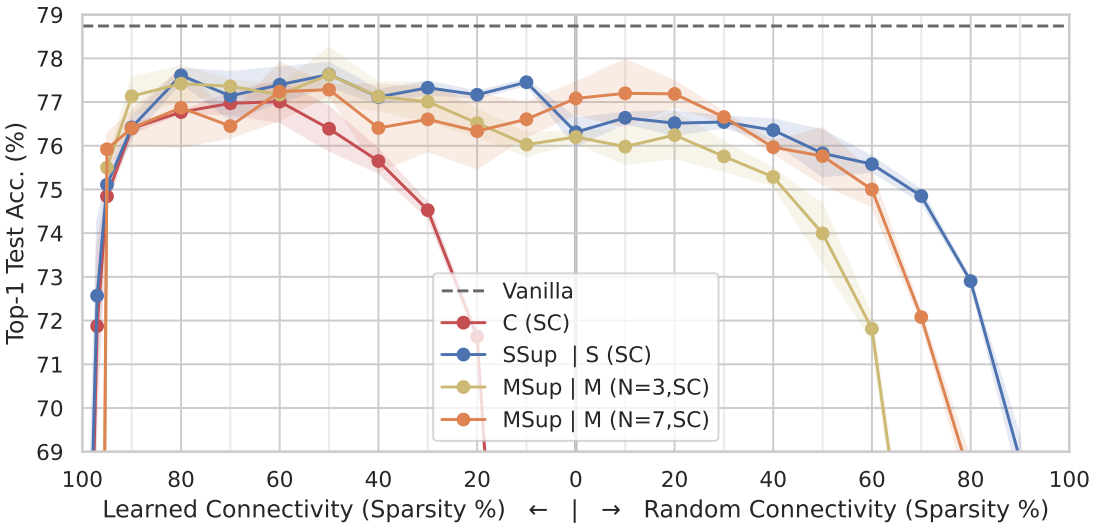


Fig. 6.4: Fundamental supermasks with learned and random pruning, SC initialization.

6.3.2 *SM* Supermasks With Learned and Random Pruning

Fig. 6.6 tests the *SM* and *SMSup* supermasks, and also includes the *S* and *SSup* results, since they can be seen as special cases of their multicoated versions when $N = 1$. These supermasks, with no randomness on the left side of the dense point (i.e., equivalent to pure quantization), predictably achieve the highest accuracies. Most notably, even on the random connectivity side they raise accuracy to almost match learned connectivity results.

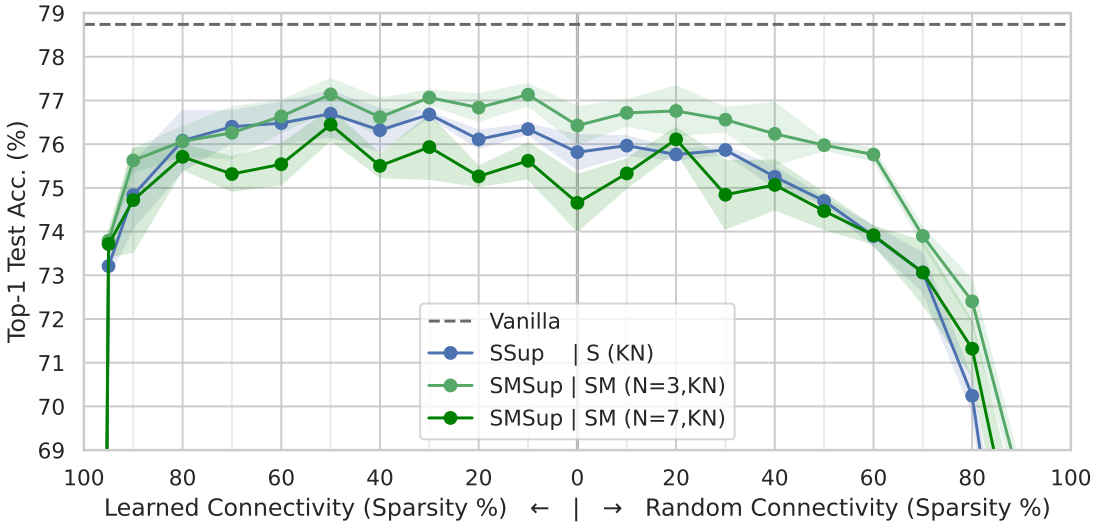


Fig. 6.5: *SM* supermasks with learned and random pruning, KN initialization.

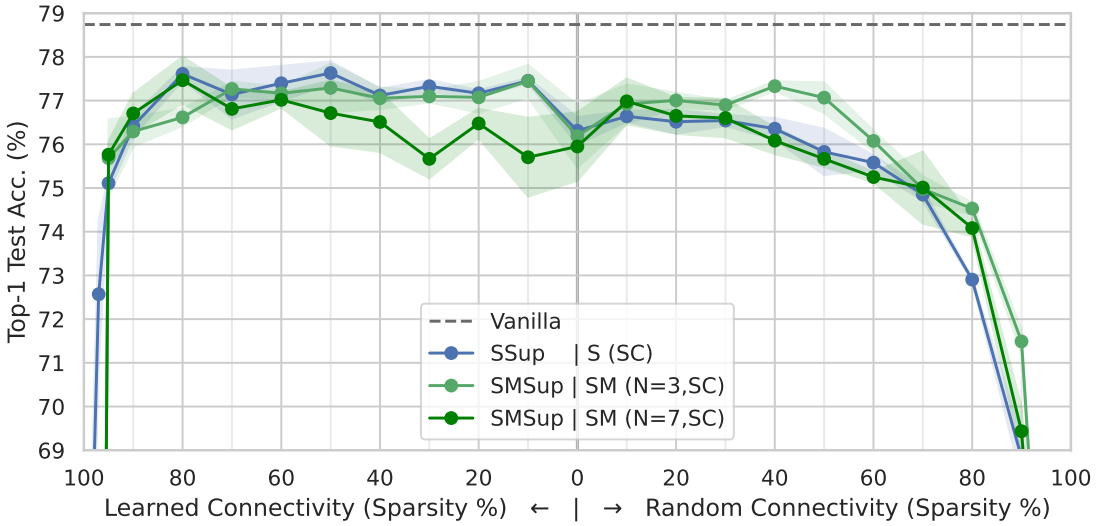


Fig. 6.6: *SM* supermasks with learned and random pruning, SC initialization.

6.3.3 Experiments with Folded Supermasks

Fig. 6.7 and Fig. 6.8 expand the previous experiments to the folded supermasks proposed in Chapter 4. Despite the reduction in parameters, accuracy generally improves in all cases across the entire connectivity range. Remarkably, the model trained with a folded *SSup* of 10–70% sparsity, although equivalent to a ternarized neural network, reaches almost the same accuracy of the original ResNet-50 with trained FP32 weights. Additionally, it outperforms its feedforward ternary counterpart (*C*(SC) in Fig. 6.4).

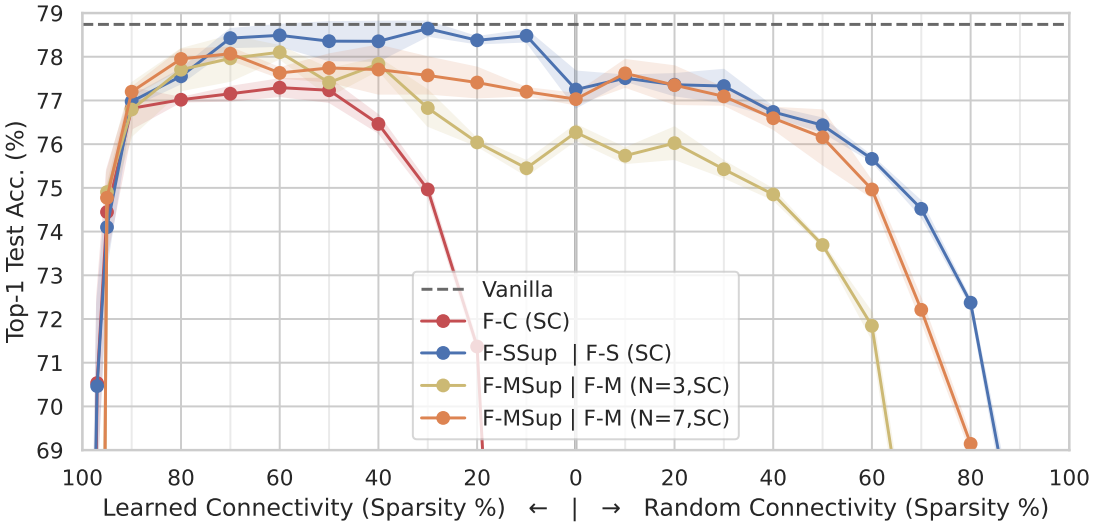


Fig. 6.7: Folded fundamental supermasks with learned and random pruning.

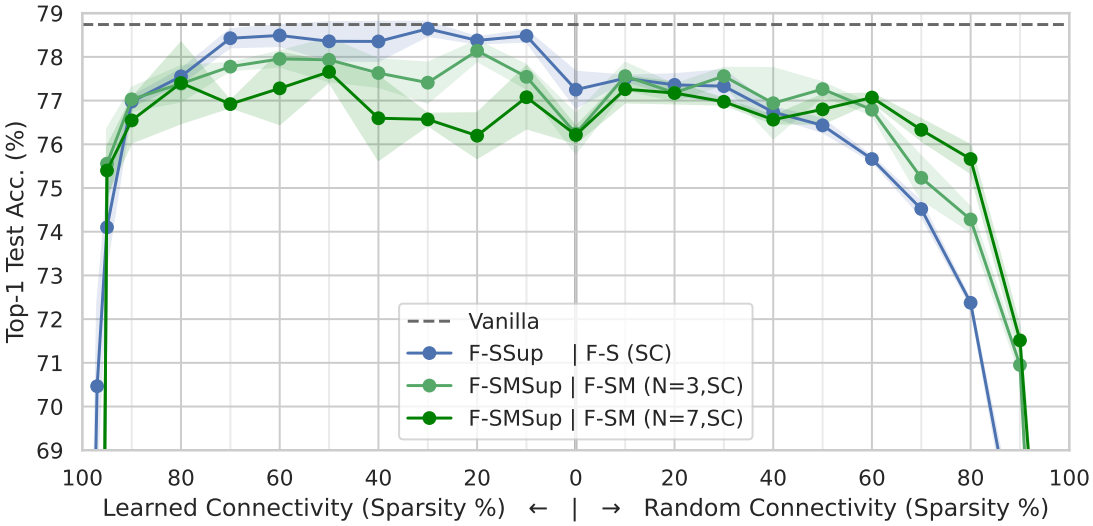


Fig. 6.8: Folded *SM* supermasks with learned and random pruning.

6.3.4 Comparison on ImageNet

Fig. 6.9 shows the results of applying the discussed supermasks with 30% learned sparsity to ResNet-50, comparing the memory size of the resulting models and their respective classification accuracies on ImageNet. Results corresponding to supermasks where the only difference is the number of coats are shown connected with a dotted line, arranging them in four groups that emphasize the impact of signing or folding the supermasks.

Although in the smaller dataset the accuracy gains provided by the *SSup* and *MSup* supermasks do not compound when combined into the *SMSup* supermask, in the larger dataset the accuracy grows monotonically with the number of coats and supermasks learned. Signing a 7-coated *MSup* boosts its 74.43% accuracy to 75.32%, setting the SOTA for a SLTH-based ResNet-50 on this dataset. Even for the folded version, which is known to suffer from lower accuracy on ImageNet (see Chapter 4.4), the compound benefits of signing and multicoating raise its accuracy from 67.14% to 75.28%, almost matching its feedforward counterpart despite being 1.49× smaller. Compared to the 76.89% accuracy of the standard ResNet-50, the 7-coated *SMSup* and folded-*SMSup* reduce model size by

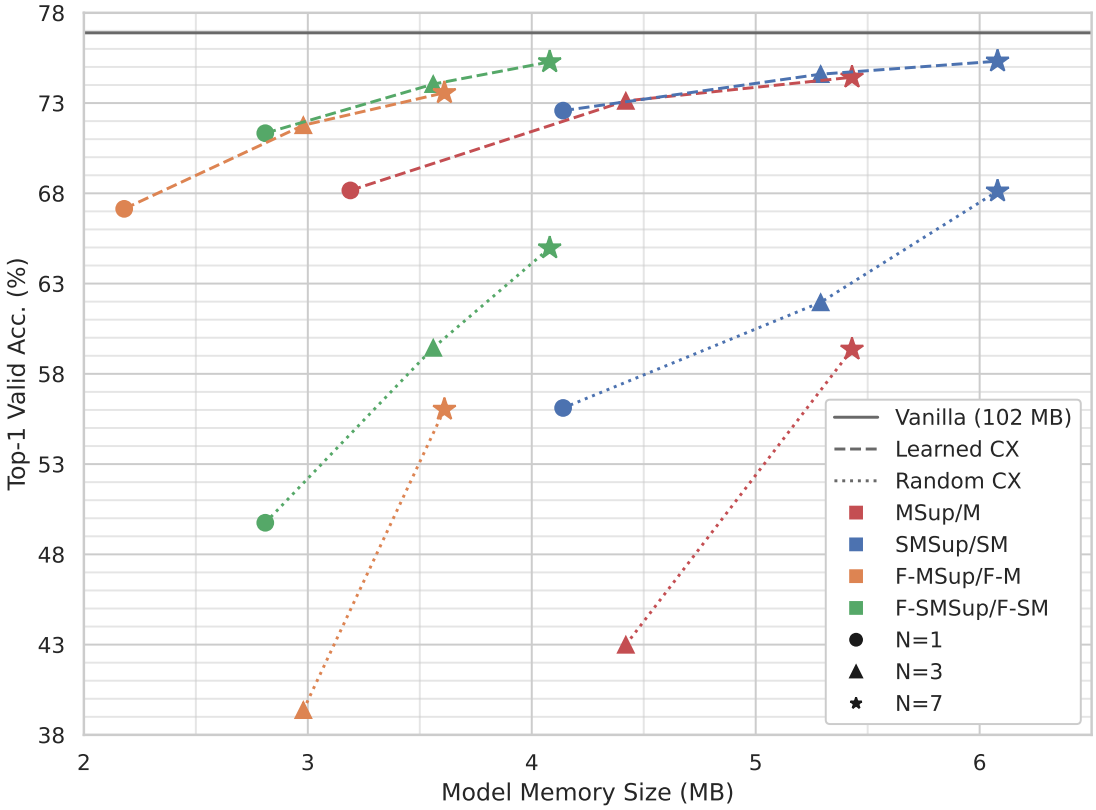


Fig. 6.9: Comparison on ImageNet of the accuracy-model size tradeoff offered by the different supermasks. When N=1, *MSup* and *SMSup* are equivalent to *C* and *SSup*, respectively. F: folded.

16.82× and 25.04×, respectively, while only suffering an accuracy drop of 1.5 percentage points.

6.3.5 Control Over Degree of Randomness

The T-SLTH offers a straightforward control over the degree of randomness of a supermask-based model. Fig. 6.10 illustrates an example, comparing the accuracy on CIFAR-100 of four settings of the *S* supermask applied to ResNet-50 with different degrees of randomness, following the analyses of Table 6.1 and Table 6.2:

- Randomness 2: *S* with KN initialization, with random connectivity and magnitudes.
- Randomness 1: *S* with SC initialization, with random connectivity.
- Randomness 1: *S* with *C* (i.e., *SSup*) and KN initialization, with random magnitudes.
- Randomness 0: *S* with *C* (i.e., *SSup*) and SC initialization, with no randomness.

If considering the naive randomness of RPaI stronger than the sophisticated weight randomness of KN, Fig. 6.10 can be interpreted as suggesting a negative correlation between accuracy and degree of randomness, in agreement with the hypothesis that it can be used in RC-like models to mimic the control of a reservoir.

These results additionally suggests that although models with a stronger RC resemblance suffer from a bigger accuracy drop, this deterioration is not critical, as they shows that

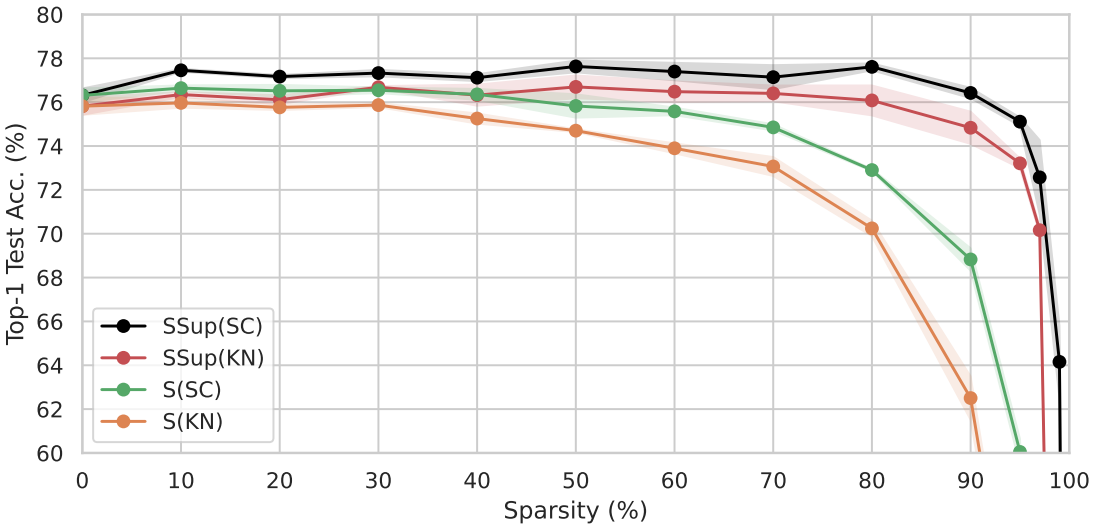


Fig. 6.10: Accuracy-randomness tradeoff comparison for a *S* supermask with a degree of randomness from 0 to 2 adjusted by choice of weight initialization and connectivity type.

even the model with highest degree of randomness is capable of achieving a competitive accuracy of over 70% at 80% sparsity.

6.4 Discussion and Conclusion

This chapter expands the SLTH on two fronts: on one end, it expands it to a framework for handling effectively three different types of randomness; on the other, it connects it with pure quantization.

For the field of deep learning, it contributes the most accurate supermask-based models proposed so far. Additionally, it points to a connection between the relatively new field of SLTH research and the established area of neural network quantization, in hopes that the rich know-how of the latter boosts the progress of the former.

In the context of this dissertation, the analysis of randomness presented in this chapter points out that not all SLTH models can be considered RC-like. Most importantly, it makes the research presented in previous chapters compatible with both the dense and random connectivity observed in some reservoirs, and opens a rich new range of possibilities for RC-like DL models based on supermasks with a novel flexible control over three randomness types.

Table 6.3 summarizes the RC-like characteristics of each of the 7 supermask models discussed compared to weight learning, in addition to their folded versions. Models are considered RC-like if they have all of the RC-like elements considered.

TABLE 6.3: Summary of the discussed models.
▲: depends on weight initialization.

Model	RC-like elements						RC-like
	Randomness			Recurrent	Shallow	Learned w/	
	CX	SIGN	MAG	CX	Arch.	Simple Output	
Weight learning	✗	✗	✗	✗	✗	✗	✗
HNN (***C***)	✗	✓	▲	✗	✗	✓	✗
SSup (***C*** ∩ ***S***)	✗	✗	▲	✗	✗	✓	✗
MSup (***C*** ∩ ***M***)	✗	✓	✗	✗	✗	✓	✗
SMSup (***C*** ∩ ***S*** ∩ ***M***)	✗	✗	✗	✗	✗	✓	✗
S	✓	✗	▲	✗	✗	✓	✗
M	✓	✓	✗	✗	✗	✓	✗
SM (***C*** ∩ ***M***)	✓	✗	✗	✗	✗	✓	✗
Folding (***F***)	✗	✗	✗	✓	✓	✗	✗
HFN (***F-C***)	✗	✓	▲	✓	✓	✓	✓
F-SSup (***F-(C*** ∩ ***S)***)	✗	✗	▲	✓	✓	✓	▲
F-MSup (***F-(C*** ∩ ***M)***)	✗	✓	✗	✓	✓	✓	✓
F-SMSup (***F-(C*** ∩ ***S*** ∩ ***M)***)	✗	✗	✗	✓	✓	✓	✗
F-S	✓	✗	▲	✓	✓	✓	✓
F-M	✓	✓	✗	✓	✓	✓	✓
F-SM (***F-(S*** ∩ ***M)***)	✓	✗	✗	✓	✓	✓	✓

Impact

This research was supported by Japan Science and Technology Agency (JST) CREST Grant Number JPMJCR18K3, Japan.

Chapter 7

WhiteDwarf: 40-nm Strong Lottery Ticket Accelerator With Triple Sparsity Exploitation

The previous chapters introduce reservoir computing (RC) inspired algorithmic elements into deep learning (DL) models that potentially raise their efficiency. However, these benefits cannot be harvested using standard processors (i.e., CPU or GPU). Neural inference accelerator design is a thriving area of research. However, rather than a generic neural engine, a specialized accelerator is necessary to capitalize on the particular RC-like characteristics introduced in this dissertation.

This chapter presents that accelerator: *WhiteDwarf*. Nonetheless, this chapter does not present a mere specialized accelerator for the algorithms introduced in the previous chapters. *WhiteDwarf* is a holistic software-hardware co-design approach applicable not only to the RC-like DL models proposed but also to general DL models. Furthermore, *WhiteDwarf* brings additional RC-like elements: multiplier-less computation and bit-level pruning.

7.1 Introduction

The Strong Lottery Ticket Hypothesis (SLTH) has upgraded the role of sparsity, commonly exploited by algorithms and sparse architectures as an advantageous by-product of training, to the main driver of neural training—i.e., sparsity-driven learning—opening new co-design opportunities for efficient neural execution. *WhiteDwarf*$*$ is a holistic ultra-compact approach that first uses a triple model compression algorithm to reduce convolutional

$*$Inspired by the low-mass, ultra-dense, low-energy white dwarf stars.

neural network (CNN) and multi-layer perceptron (MLP) models to under 10% of their original off-chip size and then uses a triple unstructured sparsity exploitation architecture for under 1% on-chip size. For example, a ResNet-50 is compressed to 5.3% of the original size while maintaining 74.7% accuracy. The fabricated 40-nm 1K-PE chip, with FP8 activations, INT4 weights, and FP16 normalization, achieves 12.24 TFLOPS/W at 99% weight sparsity.

Although SLTH with binary supermasks presented an opportunity for on-chip weight generation through random number generation [56], they suffer from lower accuracy and limited scalability. The ResNet-50 presented in [128] cannot reach 70% top-1 classification accuracy on ImageNet, and barely surpasses it even with the improved distillation process proposed by [56]. Furthermore, `edge-popup` increases the cost of training, since it requires, in addition to holding weights and computing their gradients, to hold scores and sort them for updating the supermask. Several improvements have been proposed to improve the accuracy and sparsity of strong tickets.

Hidden-Fold Networks (HFN) [93, 96] introduced in Chapter 4, exploit ResNet's overparameterization through folding [89], a neuro-inspired technique that transforms multi-block stages into recurrent blocks via weight sharing with minimal impact on accuracy. When combined with the SLTH, folding increases the number of strong tickets hidden within a randomly initialized ResNet [96]. However, it does not reduce computational cost, and has been shown to inflict an accuracy drop on larger datasets.

Multicoated Supermasks (MSup) [118] introduced in Chapter 5, applies multiple thresholds to the scores learned by `edge-popup` to produce several binary masks—multiple *coats*—as in Fig. 7.1b (left), improving accuracy without additional training cost.

Signed Supermasks (SSup) [77] introduced in Chapter 6, take further advantage the learned `edge-popup` scores by adding their signs to the supermask with an additional coat (Fig. 7.1b, right). However, additional supermask coats increase the size of the model, and still suffer from the additional training cost of supermasks.

While SLTH models are a promising approach to drastically reduce the memory and energy footprints, to achieve high execution efficiency they should be condensed as much as possible without hampering the ongoing growth of neural model diversity. In addition to efficiently handling reduced precision data formats, it is imperative to support extremely fine-grained but efficient sparse processing in the architecture. Most of the prior architectural approaches for sparsity exploitation (i.e., compression and sparse dataflow design) fall short because they stop at the value-level. It is known that there is even higher sparsity at the bit-level, but approaches that focus on its exploitation target general sparse neural models and fail to capitalize on the unique opportunities presented by sparsity-driven

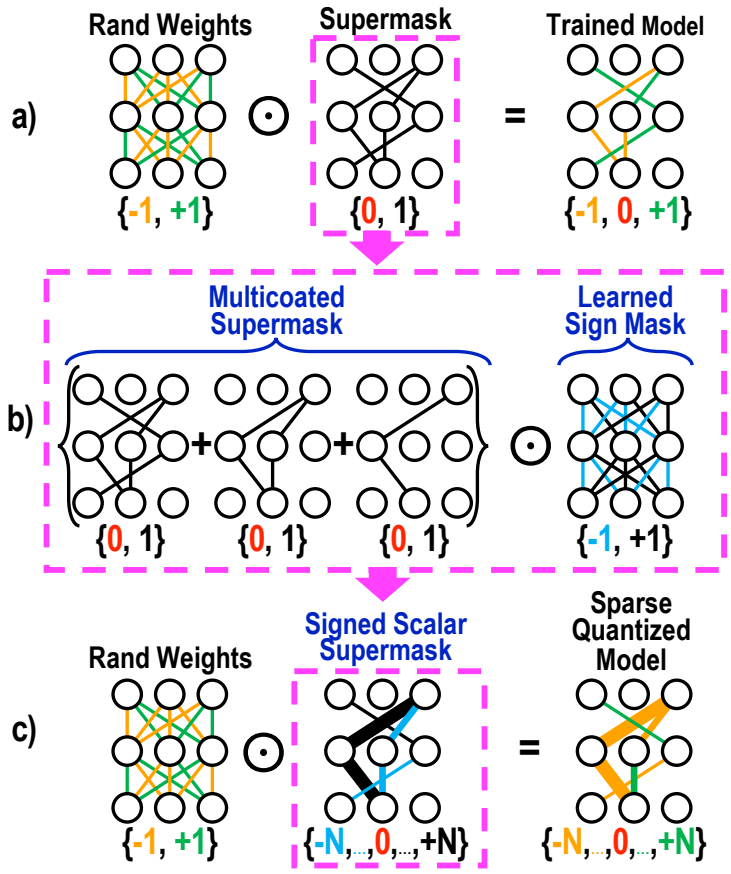


Fig. 7.1: The signed scalar supermask used by *WhiteDwarf*. (a) The original binary supermask proposed by [128] (a.k.a. Hidden Networks). (b) *left*: The multicoated (scalar) supermask proposed in Chapter 5 [118]; *right*: the signed supermask proposed in [77]. (c) The signed multi-coated supermask used by *WhiteDwarf*. The Hadamard product is represented as $\odot$.

models. Conversely, algorithm architects, unaware of the acceleration potential of SLTH models, have generally proposed only GPU-centric model optimizations.

Therefore, this chapter revisits the entire DNN-HW co-design optimization stack and presents *WhiteDwarf*, a holistic co-design approach to ultra-compact neural inference with SLTH models, illustrated in Fig. 7.2, to empower DNNs with an implosion towards ultra-compact execution of sparsity-driven models.

First, three key algorithmic techniques are introduced to enable triple model compression:

(I) A recurrent network architecture with better affinity to SLTH, using the Folding

technique [89, 96] to reduce model size significantly;

(II) Learned signs [77] and magnitudes [118] that elevate SLTH to sparse-quantization-driven training;

(III) Mixed precision with FP8 activations and FP16 normalization for a better bit precision to inference accuracy tradeoff [104, 174], and INT4 weights for enhanced bit-sparse computation. Collectively, these models set the SOTA for SLTH-based models, delivering sparser and more accurate supermasks that achieve comparable or superior accuracy to the original dense models.

Second, it proposes the *WhiteDwarf* architecture, which:

(IV) Minimizes off-chip access with optimal Huffman-encoded weight transportation, exploiting the low entropy of sparse parameters with lossless compression;

(V) Features a 1K-Processing Element (PE) Tensor customized to harness triple unstructured sparsity at the values of activation and weight via a non-zero gathering datapath, and at the bit representation of weights through a bit-serial decomposition of multiplication;

(VI) Employs a hybrid controller with a custom RISC-V processor to support a broad range of neural network models.

Finally, the performance of a *WhiteDwarf* chip fabricated with a commercial 40 nm technology is evaluated. On ImageNet, a ResNet-50 [53] is compressed to 5.7% of the original memory size while maintaining 74.7% accuracy, and on CIFAR-100, a custom MLP-Mixer S24 [152] is compressed to 0.6% of the original without accuracy loss.

The remainder of the chapter is organized as follows. Section 7.2 outlines the *WhiteDwarf* co-design holistic approach, whose detailed explanation is broken down into that algorithmic part, accelerator design, and fabricated chip and evaluation, covered in Section 7.3, Section 7.4, and Section 7.5, respectively. Finally, Section 7.6 discusses *WhiteDwarf* in comparison with other works, and concludes the chapter.

7.2 The *WhiteDwarf* Holistic Approach

WhiteDwarf is a holistic algorithm-accelerator co-design approach that first produces tiny, ultra-sparse, sparsity-driven neural models using a novel triple-compression algorithm (see

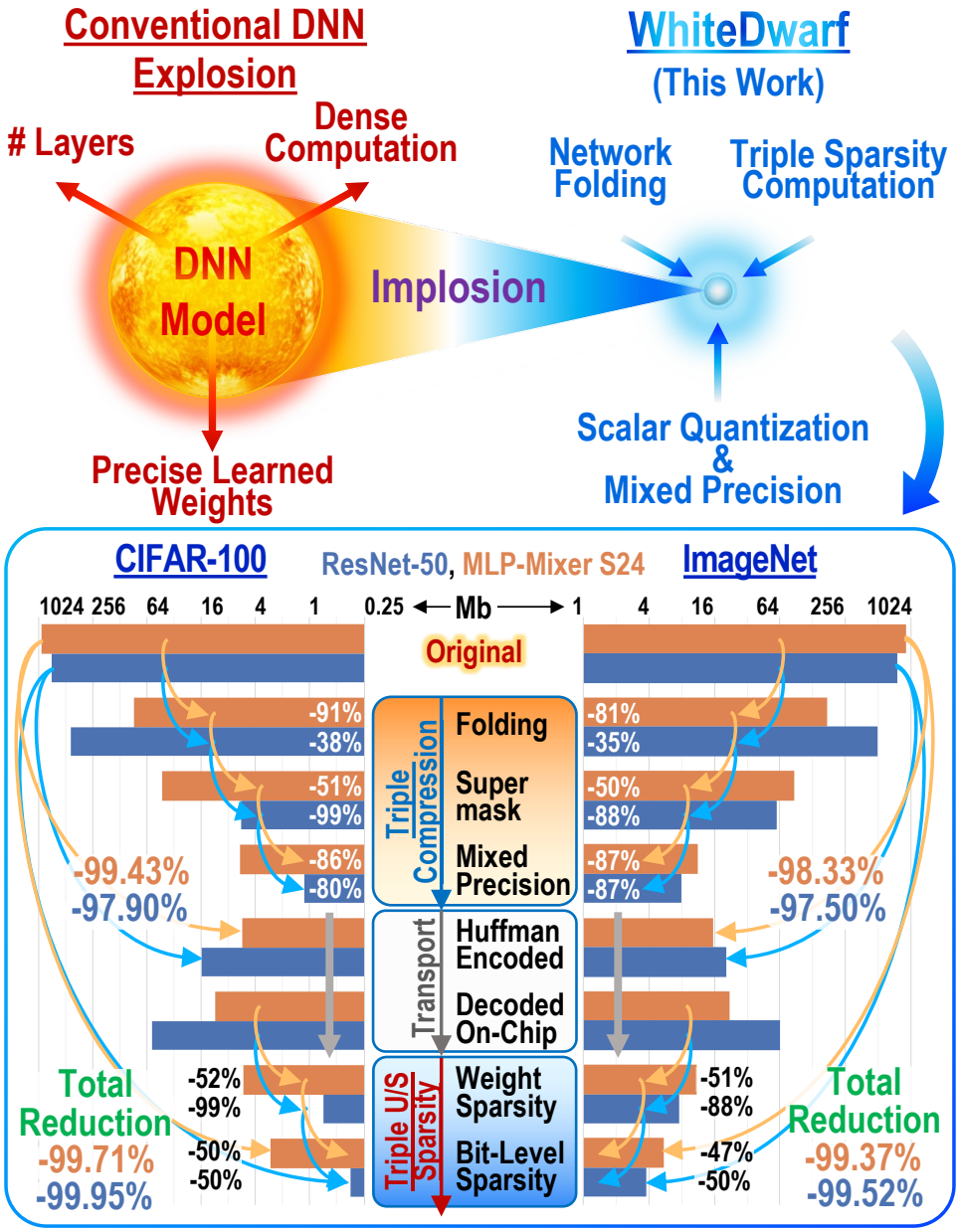


Fig. 7.2: The *WhiteDwarf* holistic concept. Contrary to the conventional DNN explosion (represented by a red giant star), *WhiteDwarf* seeks small, dense, and low-energy inference. This co-design approach achieves competitive accuracy while only processing under 1% of the original weights. U/S: unstructured.

Section 7.3) and then executes them in an ultra-compact manner in a novel triple sparsity exploitation architecture (see Section 7.4) manufactured in 40 nm CMOS (see Section 7.5). Fig. 7.2 illustrates how neural models are compressed along the *WhiteDwarf* approach, a process that begins even before neural training and culminates at bit-level processing, reaching model reductions of up to 99.95%.

The process begins by folding the neural structure, a method that has high synergy with SLTH models [96], raising accuracy, reducing model size, and cutting computational costs (see Chapter 4). Instead of using INT8 precision for both weights and activations to accommodate general models like other sparse architectures [6, 22, 90, 138], or exploiting the original binary masks for random weight generation [56], *WhiteDwarf* recognizes an opportunity on the still very low bit-width, but high accuracy, of the additional Multicoated Supermask (MSup) [118] and signed supermask (SSup) [77] coats (see Chapter 5 and Chapter 6, respectively), and combines them into INT4 weights. This novel supermask, together with an optimized training schedule, allows to train models with up to 99% weight sparsity while retaining high accuracy thanks to the use of FP8 activations, which confer higher inference accuracy with a marginal additional hardware cost [104, 174]. Additionally, this sparsity can be exploited for lossless weight compression.

This first leg of the process places on off-chip memory very sparse models that are ~90% smaller than the original. Although the weights are then expanded on-chip, the reduced size at transport is crucial, as off-chip memory access is the primary bottleneck in time and energy consumption for CMOS processors [58]. Fig. 7.3 shows a breakdown of three types of sparsity observed in the trained models. On top of the expected ~50% activation sparsity and the introduced weight sparsity, our training method allows enough control over the weights to reach over 96% bit-level sparsity, exposing an opportunity to carry on compressing the model down the process.

Once the decompressed model enters the core, its size is further reduced by zero-value skipping and processing MACs as shift-add operations, similarly to [6, 22, 90, 138]. This contrasts with architectures that process models in their compressed representations (e.g., CSC [45, 61], CSR [7, 122, 172], Booth encoding [138]). The latter approach, focused on high utilization, requires complex control hardware and computation overhead for both distributing loads at the input and reordering data at the output. In comparison, shift-add architectures, with a focus on low energy, are much simpler. *WhiteDwarf*, which (1) compensates the weight expansion with the model compression of Folding, (2) takes higher advantage of shift-add operations with bit-sparse INT4 weights, and (3) exploits idle PEs for power reduction with distributed clock gating, places a strong bet on the shift-add architecture.

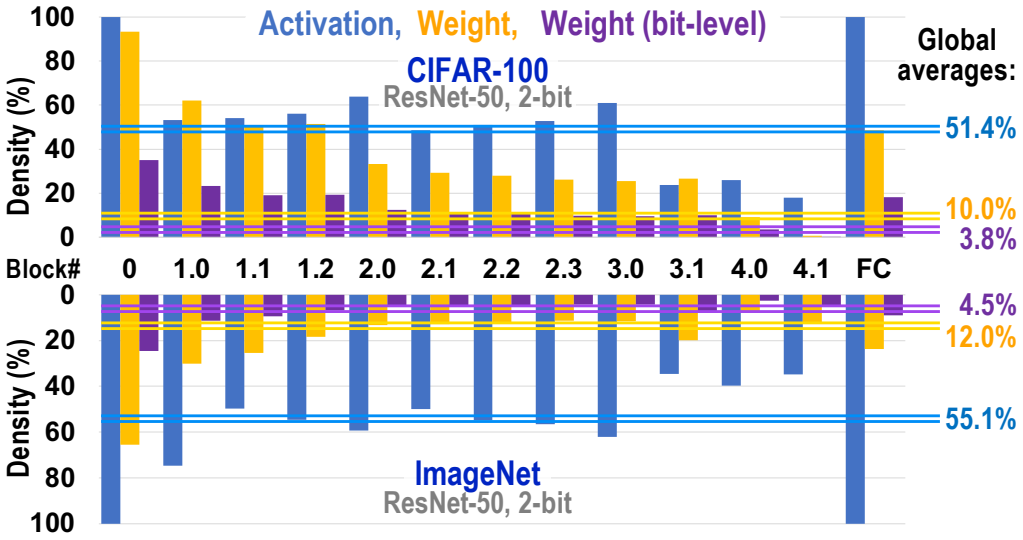


Fig. 7.3: Breakdown of the three sparsity types exploited by *WhiteDwarf* at each block of a ResNet-50 trained with fixed threshold.

After the datapath predicts and skips the weights and activations that produce zero-valued products, *WhiteDwarf* skips zero-bits in the weights too, using only the non-zero bits for computation. In the end, *WhiteDwarf* performs computation less than 1% of the original dense FP32 ResNet-50 and MLP-Mixer S24, in both CIFAR-100 and ImageNet.

7.3 *WhiteDwarf* Algorithm: Triple Model Compression

WhiteDwarf's triple model compression algorithm is composed of: (1) folding, (2) signed multicoated supermasks, and (3) mixed-precision with Huffman encoding. This compression is enhanced by a novel training strategy that improves the accuracy-sparsity tradeoff and compensates for the mixed numerical precision used on the chip. Additionally, this section extends supermask training techniques, generally applied exclusively to ResNet, to a deep MLP model.

7.3.1 Combining Folded, Multicoated, and Signed Supermasks

WhiteDwarf combines HFN's [93, 96] (see Chapter 4) Folding with the multicoating of MSup [118] (see Chapter 5) and the learned sign coat of SSup [77] (see Chapter 6), forming a recurrent model with a signed scalar supermask, as illustrated in Fig. 7.1c. The

resulting trained network has optimized weight connectivity, magnitude, and sign, thus deviating from the Strong Lottery Ticket Hypothesis and conforming to a novel general neural framework that performs concurrent training, pruning, and quantization.

The network structure is first folded in the same way as in Chapter 4, converting a series of repeated blocks of equal size into a single recurrent block by sharing weights in the forward pass, as illustrated in Fig. 7.4(left), and propagating one gradient per iteration in the backward pass.

Following the findings in Chapter 4, folded blocks use batch normalization with independent affine parameters for each iteration, as it has been shown to increase accuracy with marginal additional memory size. For optimal size-accuracy tradeoff, only the last two stages of ResNet-50 are folded [93]. Then, a collection of N thresholds is used to form an N-coated scalar supermask, as proposed in Chapter 5. Supermask scalars are then signed with the sign of the respective score.

Folding reduces the number of weights in ResNet-50 by 1.6×, compensating both the additional memory size and additional computational costs of signing and multicoating the supermask.

7.3.2 Extension to MLP-Mixer

The original work on HFN only applied Folding and supermask training to ResNets [53,169], based on evidence that residual connections approximate unrolled recurrent networks [70, 89]. However, although [96] found a size-accuracy threshold on the number of folded

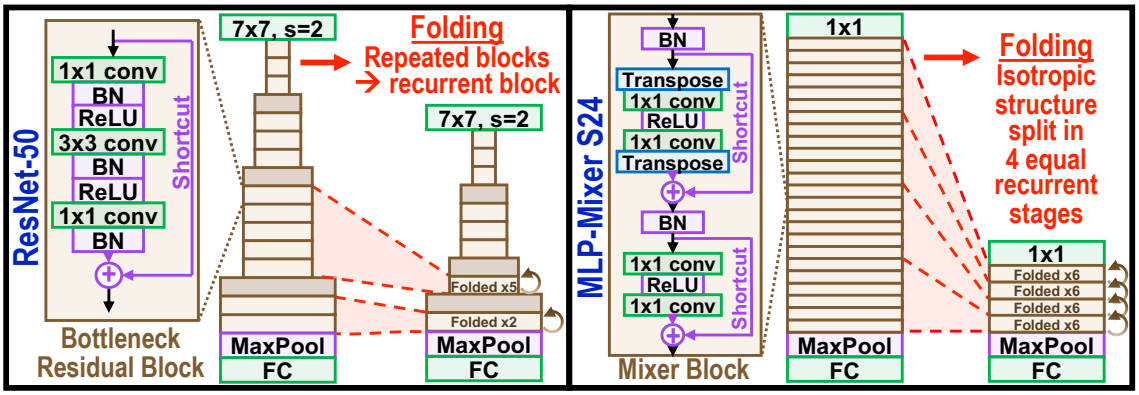


Fig. 7.4: Structure of ResNet-50 (*left*) and custom MLP-Mixer S24 (*right*), their basic building blocks, and their folded versions. Layers are colored corresponding to the part of the architecture used to process them in Fig. 7.9.

stages, ResNet's pyramidal structure (i.e., 4 distinct stages) limits the granularity in which blocks can be folded.

Recently deep MLP-based architectures such as deep MLPs [152, 153] and transformers [24,157] have gained popularity for their simplicity and accuracy in various applications. Although they are generally larger models than CNNs, unlike the multi-stage ResNets (pyramidal structure), they usually employ blocks of the same size (isotropic structure), providing an opportunity for deeper folding with fine-grained granularity.

In addition to ResNet-50, this chapter also uses for demonstration MLP-Mixer S24 [152], a simple variant of deep MLP model shown in Fig. 7.4 (right). Here the S24 size from [153] is adapted to MLP-Mixer, since it achieves higher accuracy than the original sizes and has a computational cost and number of parameters similar to ResNet-50. Folding it in 4 down to 1 stages achieves weight reductions of 5.7× up to 18.8×, respectively.

Additionally, the original LayerNorm normalization layer and GeLU activation function are substituted to the same BatchNorm and ReLU as ResNet. These changes, which do not inflict a big decrease in accuracy, allow the *WhiteDwarf* chip to process this architecture without additional dense post-processing hardware, only requiring the inclusion of a simple input transposer (see Section 7.4).

7.3.3 Mixed Precision and Huffman Coding

To support up to 7-coated supermasks—scalars 0 to 7—in addition to a learned sign coat, it suffices to use INT4 precision for weights. Since, as covered in Chapter 5.4, there are further gains for more than 7 coats, this reduced numerical precision choice comes at no cost. This precision, half of the INT8 used by other bit-sparsity accelerators [6,22,90,138], is targeted at enhancing the efficiency of *WhiteDwarf*'s shift-add multiplication (see Section 7.4.3).

To further improve bit-level sparsity exploitation while retaining the accuracy benefits of scalar supermasks, a logarithmic 3-coat MSup is also implemented, where additional coats double the supermask scalars of their connections. The INT4 representation of the resulting possible scalars ($\pm(0, 1, 2, 4)$) always has a single raised bit—i.e., optimal bit-sparsity. Unlike other approaches for optimizing weight bit-level sparsity [164], logarithmic MSup requires no additional training cost.

The high sparsity of the supermasks can be exploited for lossless compression with entropy coding, as implemented by [56] for binary supermasks using zero run-length encoding. Since MSup assigns exponentially higher thresholds to higher scalars, thus maintaining low entropy, this strategy can be expanded to scalar supermasks with Huffman coding, like [46].

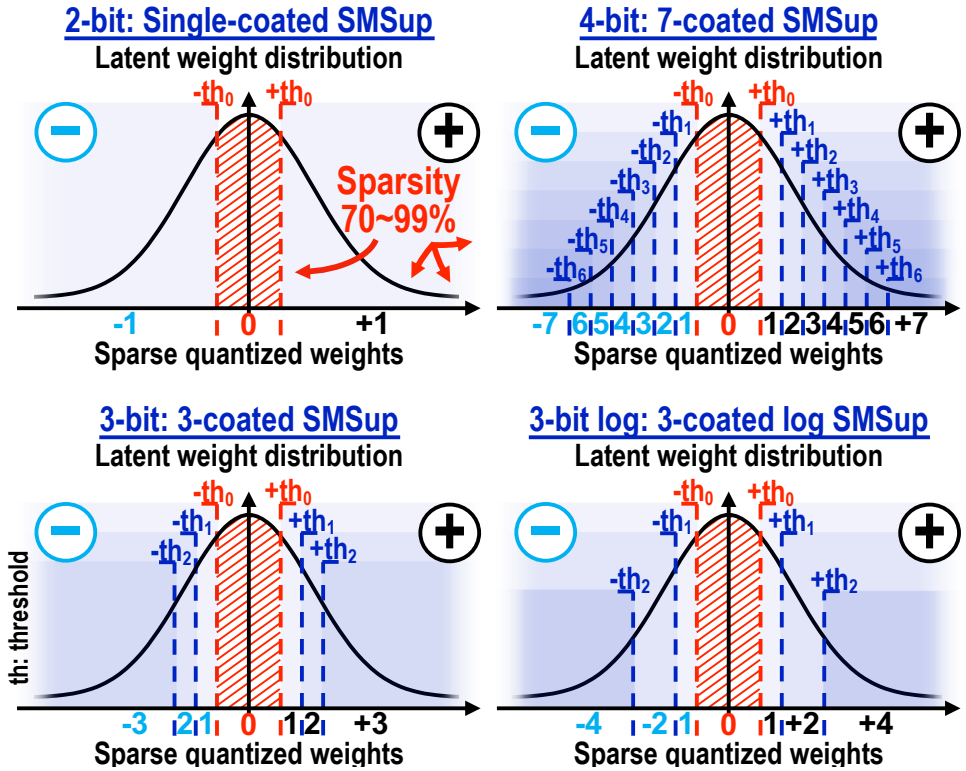


Fig. 7.5: Relationship between MSup, quantization, and Huffman coding for the four settings supported by *WhiteDwarf*.

After minimimal off-chip memory access, Huffman-compressed models are expanded on-chip using an array of decoders. Since Huffman codes are hardwired in the chip for low hardware overhead, the number of codes and their length is a relevant design decision. To retain versatility while maximizing the compression ratio, four Huffman codes are employed: 2-bit (ternary), 3-bit, logarithmic 3-bit (log), and 4-bit, which correspond to four MSup configurations: single-coated, 3-coated, logarithmic 3-coated, and up to 7-coated signed supermasks, respectively. These four settings are illustrated in Fig. 7.5, which highlights the similarity between MSup and quantization.

7.3.4 Training Algorithm and Schedule

Like previous work, *WhiteDwarf* uses the `edge-popup` algorithm [128] to train the supermasks, which counts and sorts scores to find the threshold value that will maintain a target top-k% density throughout all training. Although most commonly applied locally

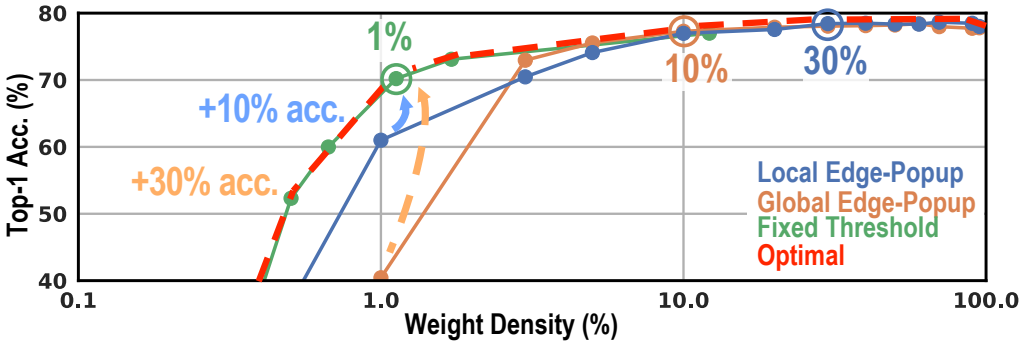


Fig. 7.6: Comparison on CIFAR-100 of three SLTH training methods using a 2-bit folded ResNet-50. *WhiteDwarf* uses the optimal for each target sparsity.

to keep all layers at the same target sparsity, some literature has proposed to apply it globally, allowing layers to have different sparsity values [178]. Alternatively, using a fixed threshold value, without controlling target sparsity, reportedly achieves higher sparsity [77].Furthermore, using a fixed threshold removes the need for sorting scores, reducing training cost.

Comparing the accuracy-sparsity tradeoff achieved by the local `edge-popup`, global `edge-popup`, and fixed threshold training methods, shown in Fig. 7.6, reveals that the best training strategy depends on the sparsity range. This means that the tradeoff can be optimized by switching the training method depending on the target sparsity. Employing the optimal method for each target sparsity allows us to, for example, raise the top-1 accuracy on CIFAR-100 of a 99%-sparse (1% density) ResNet-50 by 10% to reach 71.5%, the same performance [56] achieves at 70% sparsity.

Fig. 7.7 shows the training schedule employed. Models are trained from scratch for 200 epochs on the CIFAR-100 [78] and ImageNet [132] image classification tasks using FP32 weights and activations. Learning rate is scheduled with cosine annealing with linear decay. When using fixed threshold training (models sparser than 90% on CIFAR-100 and 80% on ImageNet), weight sparsity is annealed intrinsically by the learning algorithm, resulting in optimally low weight density. An additional 50 fine-tuning epochs are then run using the same reduced numerical precision used on the chip—FP8 activations, INT4 weights, and FP16 normalization. This choice of mixed precision is enough to fully compensate for the resulting accuracy drop after fine-tuning.

The rest of hyperparameters follow Chapter 4.4, with the addition of regularization-augmentation on CIFAR-100 (RandAugment [20] with magnitude 15, and CutMix [168] with $\alpha = 0.5$ and 80% probability). MLP-Mixers are trained on CIFAR-100 by upscaling

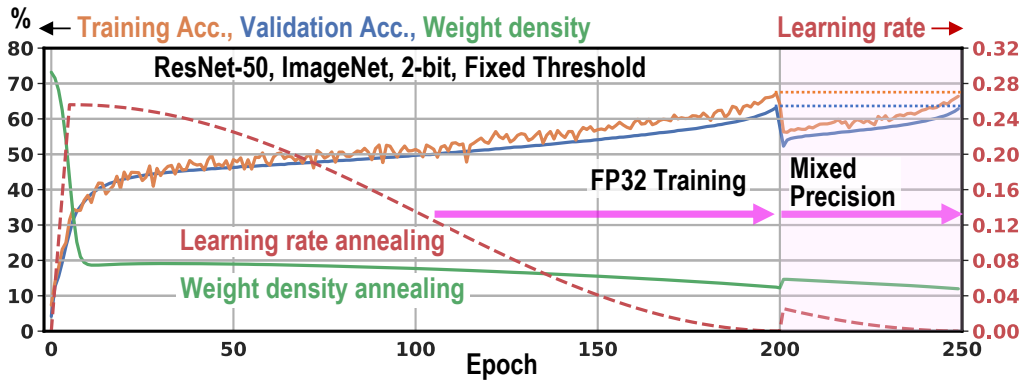


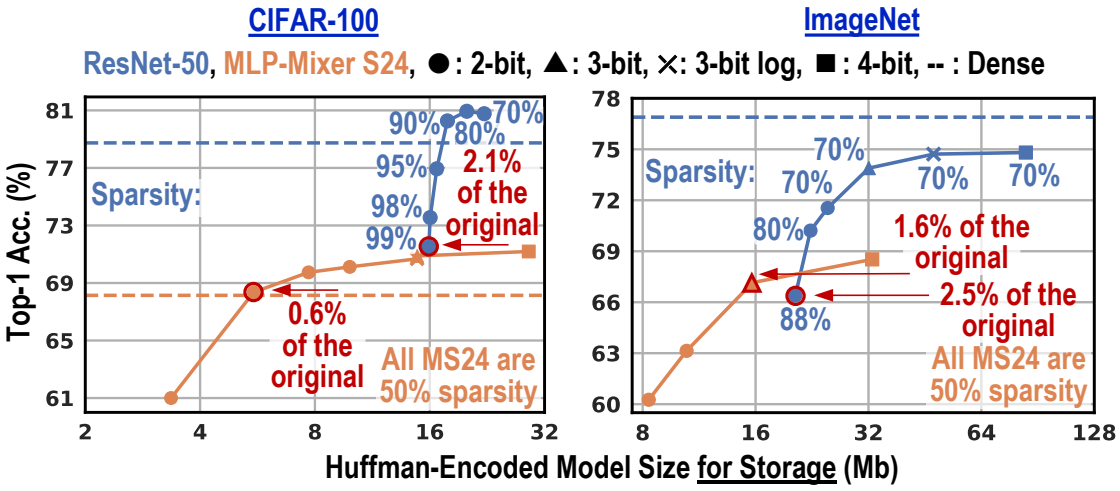
Fig. 7.7: Training and fine-tuning schedule. Weight density is optimally annealed by the training the supermask with a fixed threshold.

inputs to the same size as ImageNet. Mixed precision training is implemented using [173].

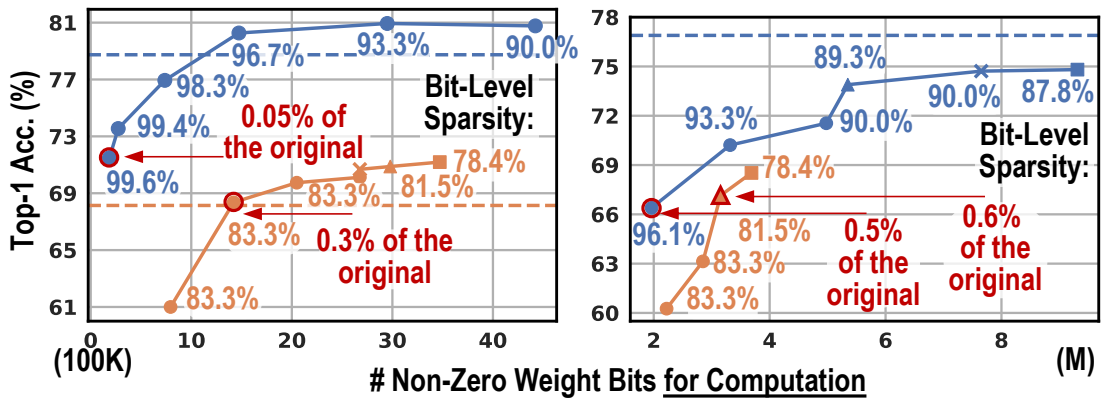
7.3.5 Results: Classification Accuracy vs. Model Size

Fig. 7.8a shows image classification results of some of the best configurations of the proposed triple-compression algorithm and their memory size after Huffman encoding. On CIFAR-100 both ResNet and MLP surpass their respective dense baseline accuracies. The proposed method is able to get ResNet-50 as sparse as 99%—down to 15.9 Mb, only 2.1% of the original model size—while maintaining a competitive accuracy of 71.5%. Even on ImageNet, models can be compressed to under 3% of the original size, training a 80% sparse ResNet-50 to 70.2% accuracy while occupying only 22.4 Mb.

Splitting MLP-Mixer's isotropic structure in 1 to 4 stages uncovers a tradeoff between model size and accuracy. A 2-stage, 2-bit MLP Mixer S24 is compressed to only 0.6% of its original size without accuracy loss. Accuracy grows with the number of folding stages and supermask coats, reaching 72.65% top-1 accuracy with 4-stages and 4-bit weights before the mixed precision fine-tuning, which is, currently, the highest for an MLP model trained from scratch on CIFAR-100 [59, 98]. Although its accuracy peaks at only 50% supermask sparsity, this additional folding granularity compensates for it, achieving even higher compression ratios than ResNet-50, as illustrated on Fig. 7.2.



(a) Top-1 accuracy vs. off-chip model memory size.



(b) Top-1 accuracy vs. number of bits used on-chip for computation.

Fig. 7.8: Tradeoff of top-1 classification accuracy to model size, using the same mixed precision settings as the *WhiteDwarf* chip. Results highlighted in red correspond to those shown in Fig. 7.2.

7.4 *WhiteDwarf* Architecture: Triple Unstructured Sparsity Exploitation

The *WhiteDwarf* architecture, shown in Fig. 7.9, is designed to execute the explained ultra-sparse DNN models in a ultra-compact manner to achieve high efficiency. It is imperative for the architecture to handle unstructured sparsity because (1) SLTH models feature high unstructured sparsity, and (2) sparsity in activations is also intrinsically unstructured. The *WhiteDwarf* architecture goes one step further to exploit even the bit-level sparsity of weights.

The whole dataflow is directed by the Core Controller (CTRL), formed by a Hardware Controller (HWC) configured by a custom RISC-V processor that facilitates support for various models. Within the HWC, a Clock Gating Controller (CGC) exploits sparsity for

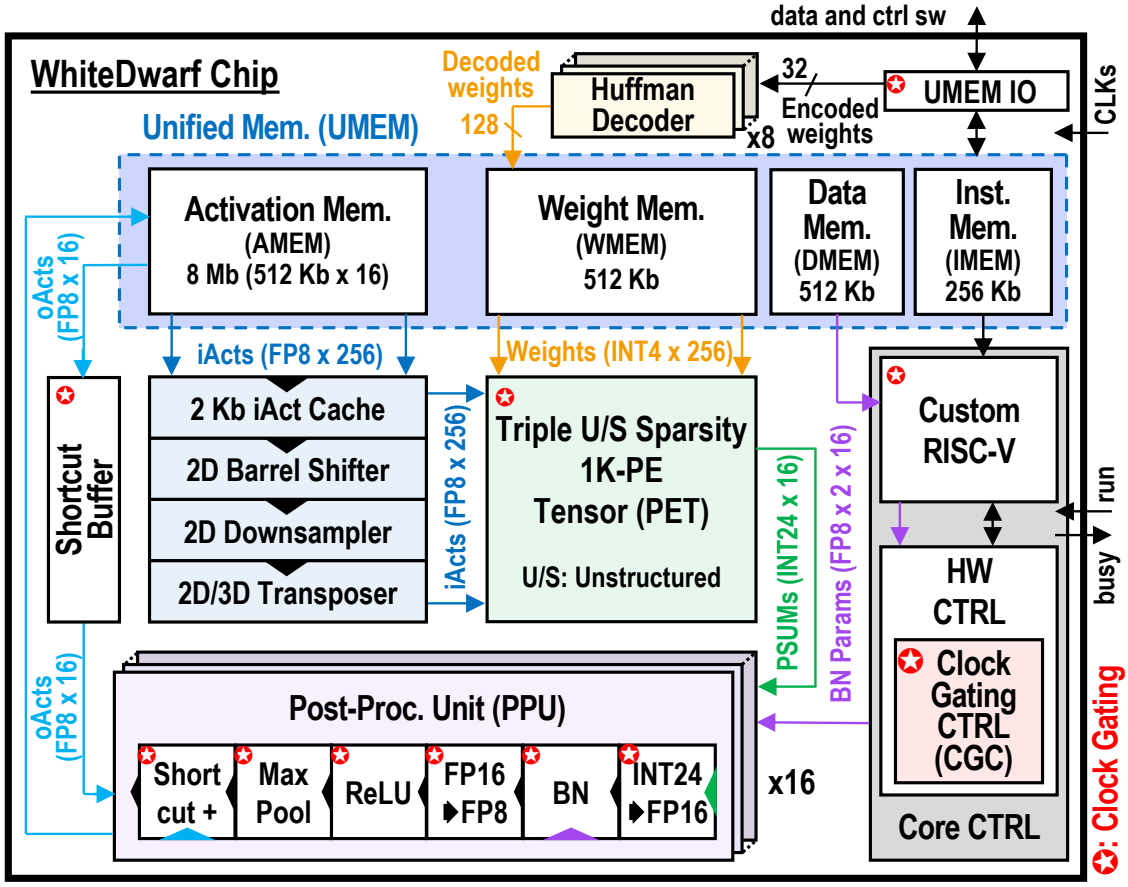


Fig. 7.9: Top level of the *WhiteDwarf* architecture.

power reduction by coordinating fine-grained clock gating at the PE level of the PET, the PPUs, the RISC-V, and the Memory IO interface.

After the Huffman-encoded weights are expanded to INT4 by an array of eight Huffman decoders, they are written to the 512 Kb Weight Memory (WMEM). Before being supplied along the weights to the Processing Element (PE) Tensor (PET), the FP8 input activations (iActs), stored in the 8 Mb Activation Memory (AMEM), first pass through a stack of a 2D Barrel Shifter, a 2D Downsampler, and a 2D/3D Transposer, which allow transformations necessary for a variety of neural models, including the target MLP-Mixer, in addition to serving as a 2 Kb iAct cache.

Once the PET finishes processing a 4×4×16 INT24 partial sum (PSUM), it is written back to the AMEM through an array of sixteen Post-Processing Units (PPUs), each processing an output channel (oCh). PPUs optionally perform FP16 Batch Normalization (BN), quantization scaling, ReLU, Max Pooling, and the addition of a residual shortcut connection before writing the FP8 output activation (oAct) back to the AMEM. The PPUs receive the learned FP16 BN bias and scale parameters (the latter of which doubles as the weight scaling factor for binary/ternary models, as in [56]) from the CTRL, stored in the RISC-V's 512 Kb Data Memory (DMEM).

7.4.1 Triple Unstructured Sparsity 1K-PE Tensor Core (PET)

Fig. 7.10 details *WhiteDwarf* 's output stationary dataflow. The PET is formed by 16 PE Matrices (PEMs), which correspond to a 4×4 oAct surface (1×16 in the case of 1×1 convolution.) Each PEM is formed by 16 PE Vectors (PEVs), corresponding to 16 oCh. The PET receives a 4×4×16 iAct chunk at a time, broadcasted within each of the 4×4 PEMs to all PEVs, and 1×1×16×16 weights, broadcasted across all the PEMs to the 16 PEVs, to produce a 4×4×16 PSUM chunk.

Each PEV distributes the 16 iCh workload in parallel across 4 lanes of Non-Zero Condensing PEs (NZC-PEs), each receiving 4 unique pairs of iActs, as detailed in Fig. 7.11. *WhiteDwarf* performs MAC as bit-series shift-add multiplications. While shift operations are performed at the PEs within the NZC-PEs, both the add and accumulation of the partial multiplications (PMULs) are performed at the PEV level.

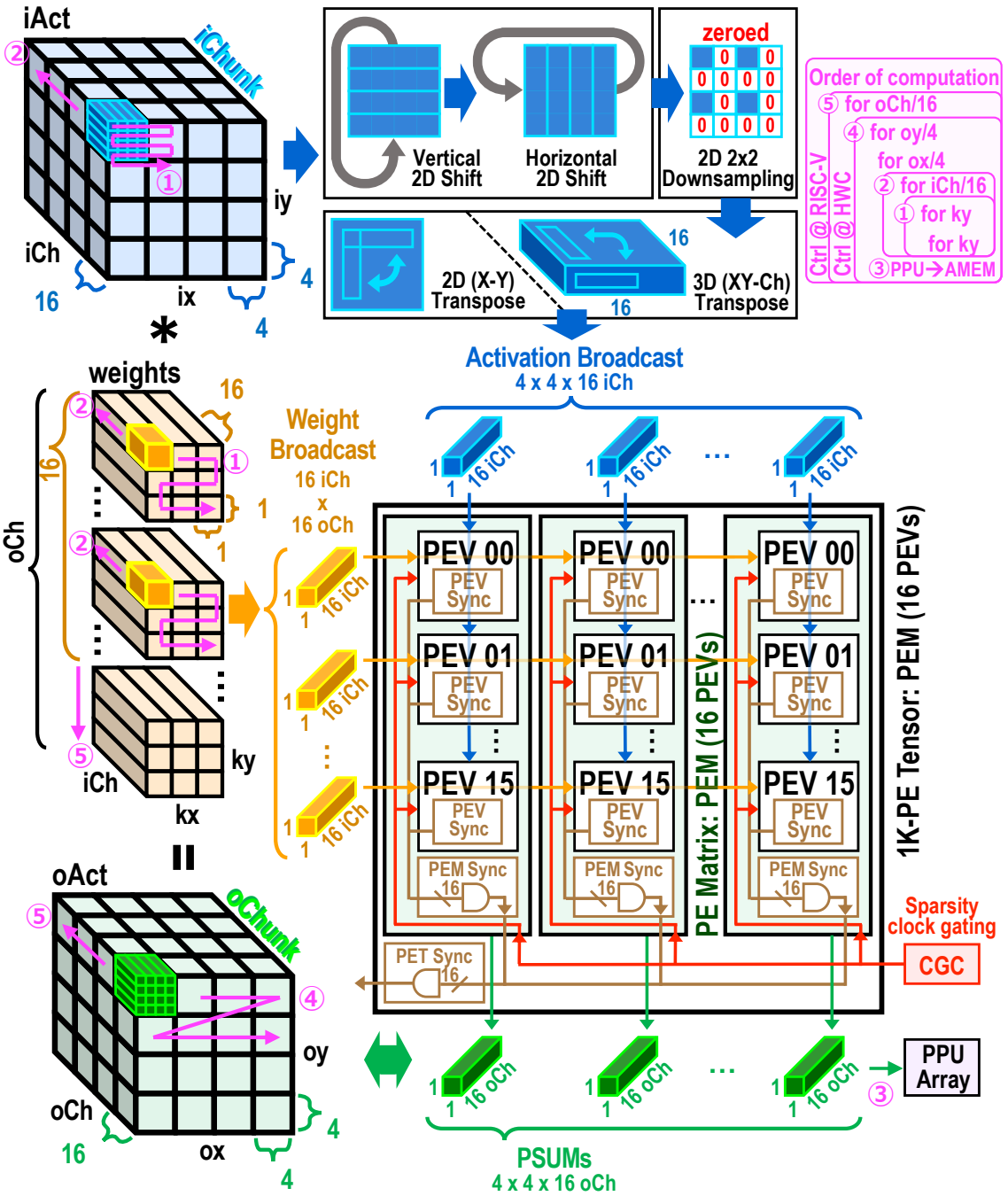


Fig. 7.10: Output stationary dataflow and hierarchical structure of the 1K-PE Tensor (PET), PE Matrix (PEM), and PE Vector (PEV), along with the hierarchical synchronization and clock gating signal networks.

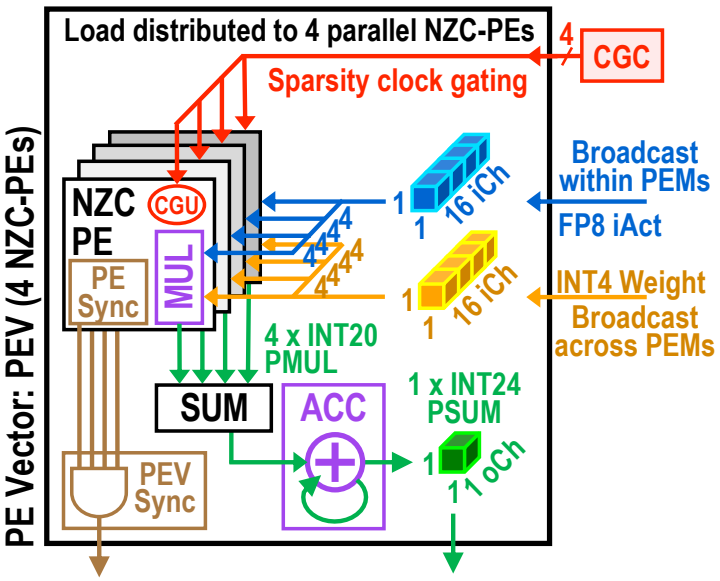


Fig. 7.11: Each PEV splits its load parallel across 4 NZC-PEs (Fig. 7.12).

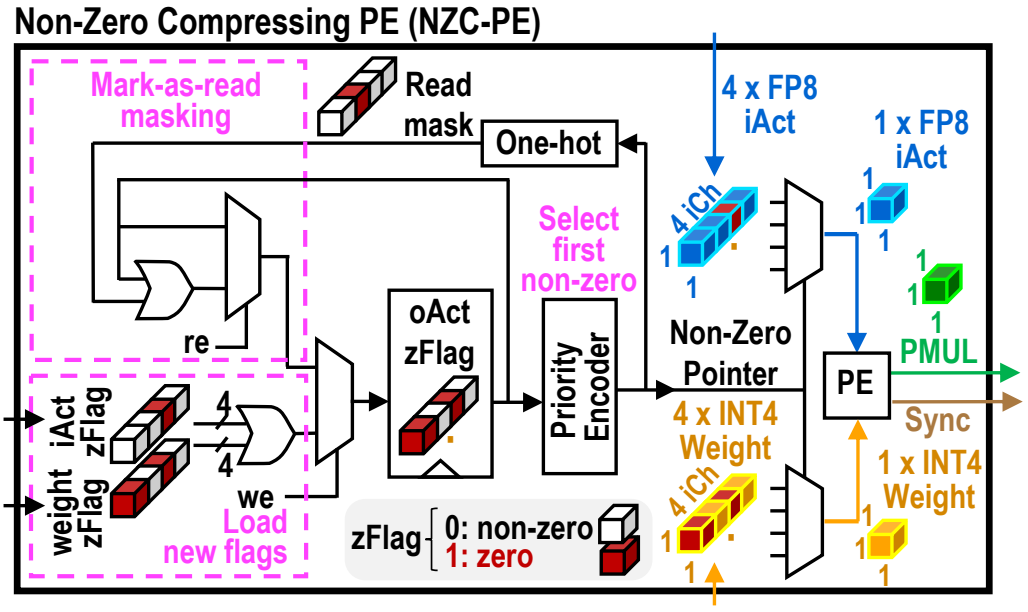


Fig. 7.12: The NZC-PE implements a zero-value-skipping datapath that only feeds to the PE (Fig. 7.13) the non-zero activation-weight pairs.

7.4.2 Value-Level Sparsity Exploitation of Activations and Weights

Fig. 7.12 explains the non-zero gathering datapath performed by the NZC-PE, a wrapper of the PE that exploits the first two elements of *WhiteDwarf*'s triple unstructured sparsity exploitation—value-level sparsity of activations and weights.

In addition to 4 pairs of iActs and weights, the NZC-PE receives from the PET a broadcast of their respective Zero Flags (zFlags), which are produced with a NOR of all their bits. zFlags are ORed in pairs to predict which iAct-weight products will produce a zero—these multiplications are skipped. A priority encoder computes the distance to the first non-zero pair—a non-zero pointer—to select and feed it to the PE. Once the PE is done, the non-zero pointer is re-used for masking the already-processed pair in the next iteration. Once all non-zero iAct-weight pairs (0 to 4, depending on sparsity) are read, the NZC-PE computation is done.

This simple zero-skipping dataflow requires no control overheads. Instead of load balancing between adjacent PEs, as proposed in [90], *WhiteDwarf* combines simple datapath control with clock gating of idle PEs for power reduction.

Optionally, iAct and weight zFlags can be independently deactivated in the RISC-V configuration for performing dense computation or single-value sparsity exploitation, modes which are used in Section 7.5 as baselines for evaluation.

7.4.3 Bit-Level Sparsity Exploitation of Weights

Upon receiving a non-zero iAct-weight pair, the PE, portrayed in Fig. 7.13, exploits bit-level sparsity to reduce the multiplication cost with a multi-cycle process. The absolute value of the INT4 weights (i.e., 3 bits) is serialized to process multiplication as a series of shift-adds that takes 1 to 3 cycles, depending on bit-level sparsity. First, the UINT3 magnitude is converted into a list of up to 3 shift-widths of 0 to 2 bits, which is stored in a register. The Hamming weight of the magnitude (i.e., its number of raised bits) is also registered in unary representation. A LUT performs both of these conversions.

Every cycle, one of the shift operations is performed at the same time as an INT conversion of the iAct, for PMUL accumulation, by left-shifting the iAct mantissa by the sum of the iAct exponent and the position of the first shift-width in the registered list. Then, both registers are left-shifted to the next list element for the next cycle. This process takes a number of iterations equal to the Hamming weight. By storing it in unary and left-shifting it, the register works as a count of the remaining iterations. In the case of single-coat or logarithmic supermasks, when all weights have an optimal Hamming weight of 1, all multiplications get a single cycle.

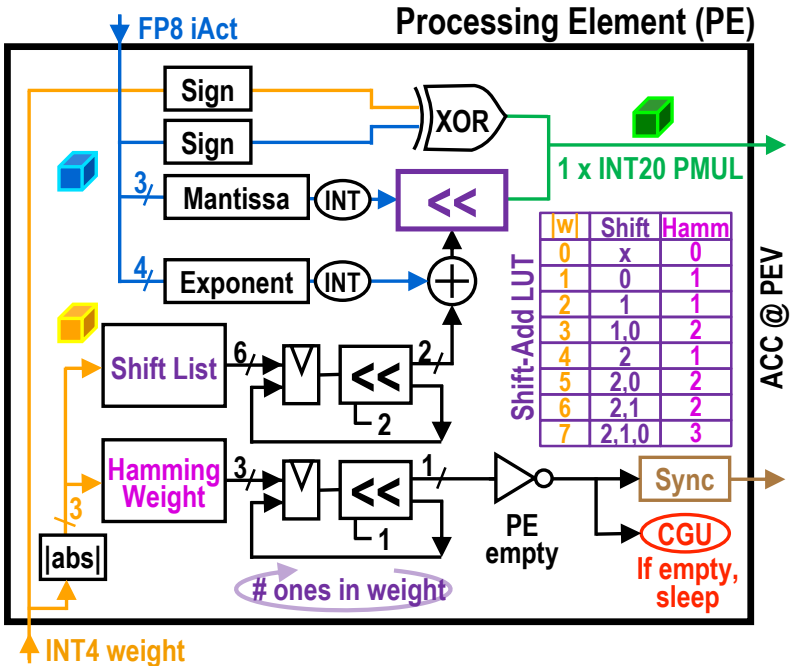


Fig. 7.13: *WhiteDwarf*'s PE decomposes multiplication into a series of up to 3 shift operations. It performs one shift per cycle.

Compared to other shift-add architectures that serialize INT8 iActs [6, 22], *WhiteDwarf* can afford using more accurate FP8 iActs (4-bit exponent and 3-bit mantissa [104]) by serializing weights. Compared to other weight-serializing designs [90, 138], *WhiteDwarf* uses INT4 instead of INT8 weights, reducing operations by up to 2.3× in the worst case (dense) and by up to 7× in the case of the proposed optimal bit-sparsity weights.

Bit-level sparsity exploitation can also be deactivated at the RISC-V configuration for bit-dense computation, as used in Section 7.5 for evaluation.

7.4.4 Clock Gating and Hierarchical Synchronization

The target unstructured sparsity causes PEs to finish their computation at irregular times and spatial patterns. *WhiteDwarf* handles this irregularity with a hierarchical synchronization network, and at the same time exploits it for harvesting power reduction with clock gating. Fig. 7.14 depicts an example time chart of synchronization and clock gating across the PET, detailing the computation reduction gained by the sparse computation and the power reduction harvested by the clock gating network. Once a PE finishes the computation of all raised bits, it sets itself for clock gating in a distributed manner. PEs then sleep until the

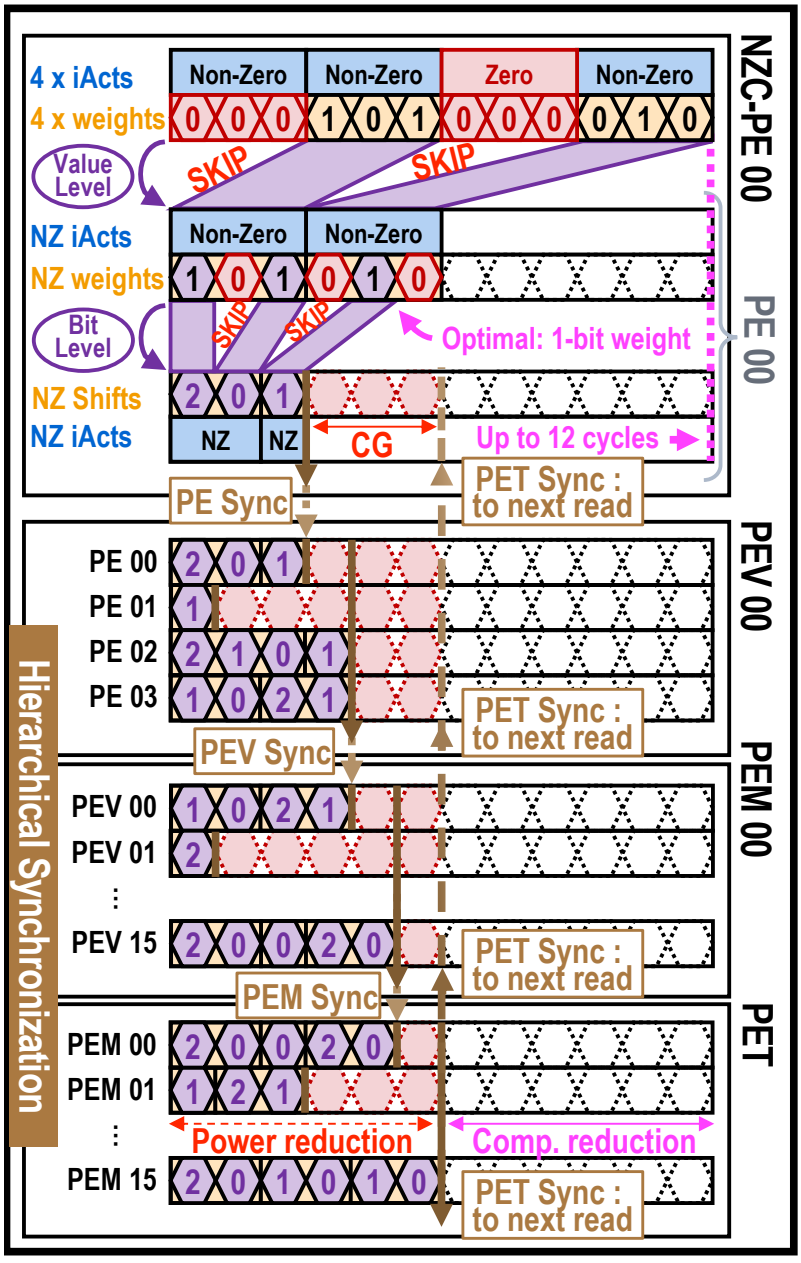


Fig. 7.14: Hierarchical synchronization time chart. After the NZC-PE skips multiplications with zero-values, PEs skip zero-bits in the serialized shift-add multiplication. Idle PEs are clock gated while waiting, for power reduction.

last PE has finished processing the last raised bit of its last non-zero iAct-weight pair, a hierarchical synchronization process that takes 1 to 12 cycles (4 pairs×3 bits).

Synchronization signals are collected hierarchically at the PEV, PEM, and PET levels through the AND gates depicted in Fig. 7.11 and Fig. 7.10. Once the HWC receives the PET synchronization signal, a new pair of input iAct and weight chunks enters the PET, and the CGC resets the PET clock gating signals globally. Apart from exploiting sparsity, the CGC also sets to sleep the UMEM IO, the RISC-V, the shortcut buffer, the entire PET, and individual stages of the PPU array when they are not being used (see Fig. 7.9).

The asymmetry of distributed zero-skipping is not only exploited for power reduction but also for speedup. In the case of 30% density, on average only 1.2 weights are processed at each PE. In the optimal case of single-coated or logarithmic supermasks, whose weights always have a single raised bit, the computation takes an average of 1.2 cycles, a theoretical 10× speedup with respect to the 12 cycle dense computation.

Both the power and speedup benefits are measured on the fabricated chip and reported on Section 7.5.

7.4.5 Additional Design Choices for Optimal AMEM Usage

To optimize the usage of the limited AMEM available in the fabricated chip, *WhiteDwarf* uses a "snake" scan pattern, which always shifts the iAct by 1, instead of the conventional raster scan (see Fig. 7.15) to optimize iAct cache reuse and thus minimize AMEM access.

Like other neural inference accelerator chips [39,56], *WhiteDwarf* employs a slice-based layer-fusion to accommodate models whose intermediate activations would not fit in its

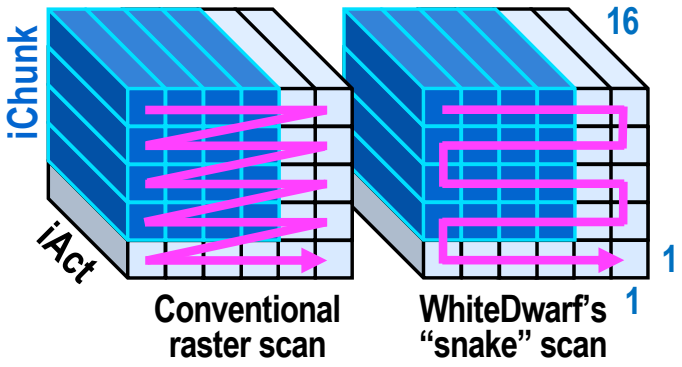


Fig. 7.15: The conventional raster scan pattern used for processing iActs (*left*) and *WhiteDwarf*'s "snake" pattern (*right*), which optimizes AMEM usage.

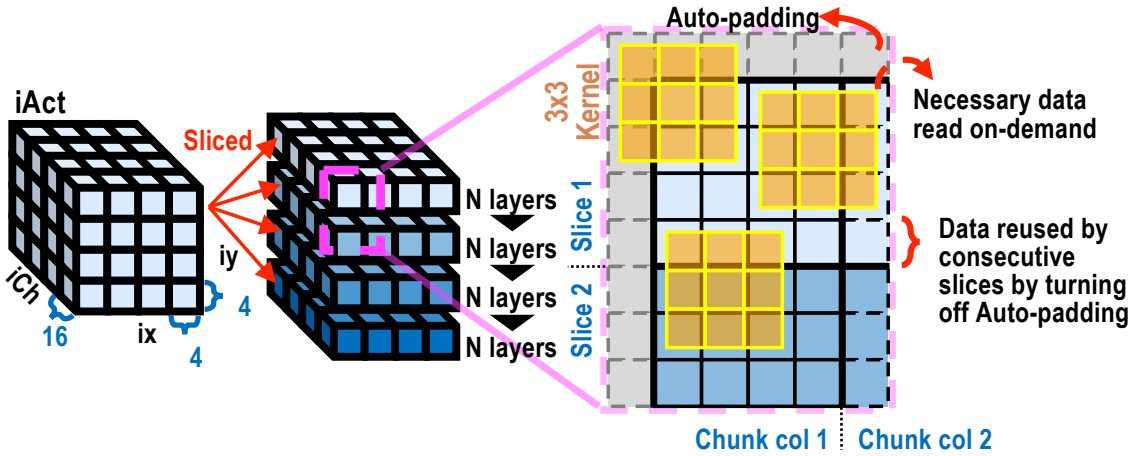


Fig. 7.16: *WhiteDwarf*'s slice-based layer-fusion. An Auto-padding signal lets the core know if it is processing an input border (to add padding) or an inter-slice border (to read adjacent data).

AMEM. However, unlike other accelerators, which compute a spurious line in the inter-slice border due to insufficient loaded input data [56], *WhiteDwarf*'s upgraded layer-fusion, illustrated in Fig. 7.16, always loads necessary data at the inter-chunk borders and employs an Auto-padding signal to either add padding at the input edges or load additional data on-demand at the inter-slice borders, preventing wasteful computation.

7.4.6 Results: Model Computation Reduction On-Chip

As shown on Fig. 7.8a, the proposed algorithm reaches > 10× off-chip memory compression ratio in all cases after Huffman encoding. Although model size grows on-chip after the Huffman decoders expand all the weights to INT4, as shown on the model compression breakdown in Fig. 7.2 (bottom), the model is then further compressed. *WhiteDwarf* triple sparsity exploitation does not affect the model memory size (i.e., WMEM and DMEM compression), but it does shrink the model as it travels down the PET hierarchy.

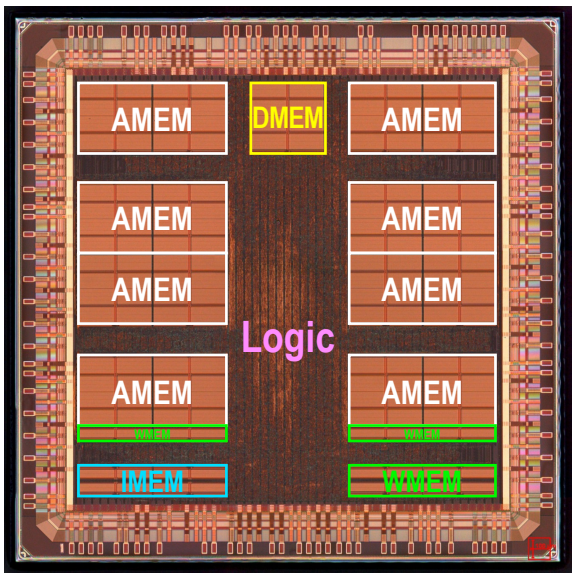
After passing through the non-zero gathering datapath of the NZC-PEs, value-level sparsity compression discards at least as many weights as zeroes in the supermask—i.e., ≥ 99% of the original weights in the case of a model trained to 99% sparsity. These results do not take into account the additional weights skipped due to iAct value-level sparsity, as they are input dependent. Once non-zero weights are input into the PE, bit-level sparsity discards up to 50% of the INT4 weight bits (best case scenario of only 1 raised bit, plus sign bit).

In the optimal case of 2-bit or 3-bit log weights, this totals to a reduction of > 99%, meaning PEs process < 1% of the original FP32 dense model. As collected in Fig. 7.8b, in the smallest case tested, *WhiteDwarf* only processes 0.05% and 0.3% of the original ResNet-50 and MLP-Mixer S24 bits for CIFAR-100, and even for ImageNet PEs only use 0.5% and 0.6% of the original models, respectively. Even for the best performing ResNet-50 on ImageNet, only 7.3 Mb out of the original 25.5 M FP32 weights (779.1 Mb) are processed on-chip. It shall be noted that these results do not take into account the BatchNorm parameters, which are processed in a dense fashion, and in the case of the discussed folded ResNet-50 account for an additional 0.42 Mb.

7.5 Evaluation of the Fabricated *WhiteDwarf* Chip

Fig. 7.17 collects a micrograph and specification table of the fabricated *WhiteDwarf* chip (40-nm CMOS, 9 mm^2), whose core area breakdown is specified in Fig. 7.18, with a detailed logic area breakdown. Out of the 5.34 mm^2 core area, 77% is occupied by the 9.25 Mb SRAM and 17% by the PET, which combined account for around 80% of the power (see Fig. 7.21).

The measurement environment depicted in Fig. 7.19 is used to verify and evaluate several *WhiteDwarf* chips, whose results are reported in this section. It comprises an FPGA board,



Technology	TSMC 40 nm CMOS (LP)
Package	QFN80 (48 Signal Pins)
Chip Size	3 mm × 3 mm
Core Area	SRAM: 3.95 mm^2 Logic: 1.39 mm^2
Core VDD	0.76—1.2 V
I/O VDD	3.3 V
Gate Count	986K Gates
SRAM	AMEM: 1 MB WMEM: 64 KB IMEM: 32 KB DMEM: 64 KB Total: 1.16 MB

Fig. 7.17: Fabricated *WhiteDwarf* chip micrograph and specification table.

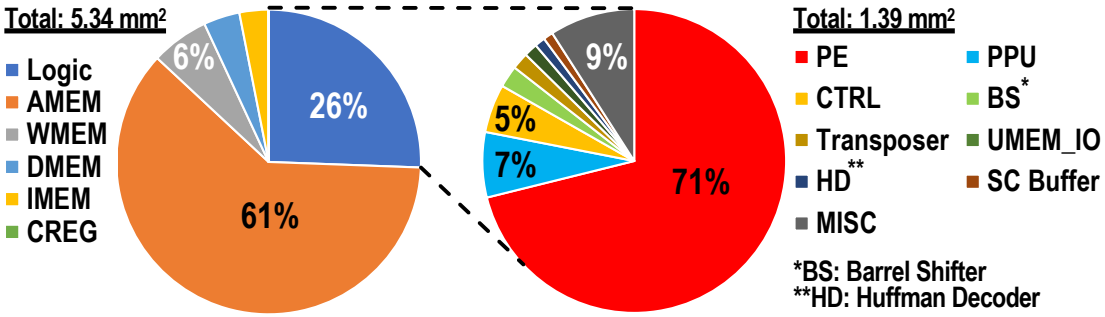


Fig. 7.18: Chip core area breakdown, and detailed logic area breakdown.

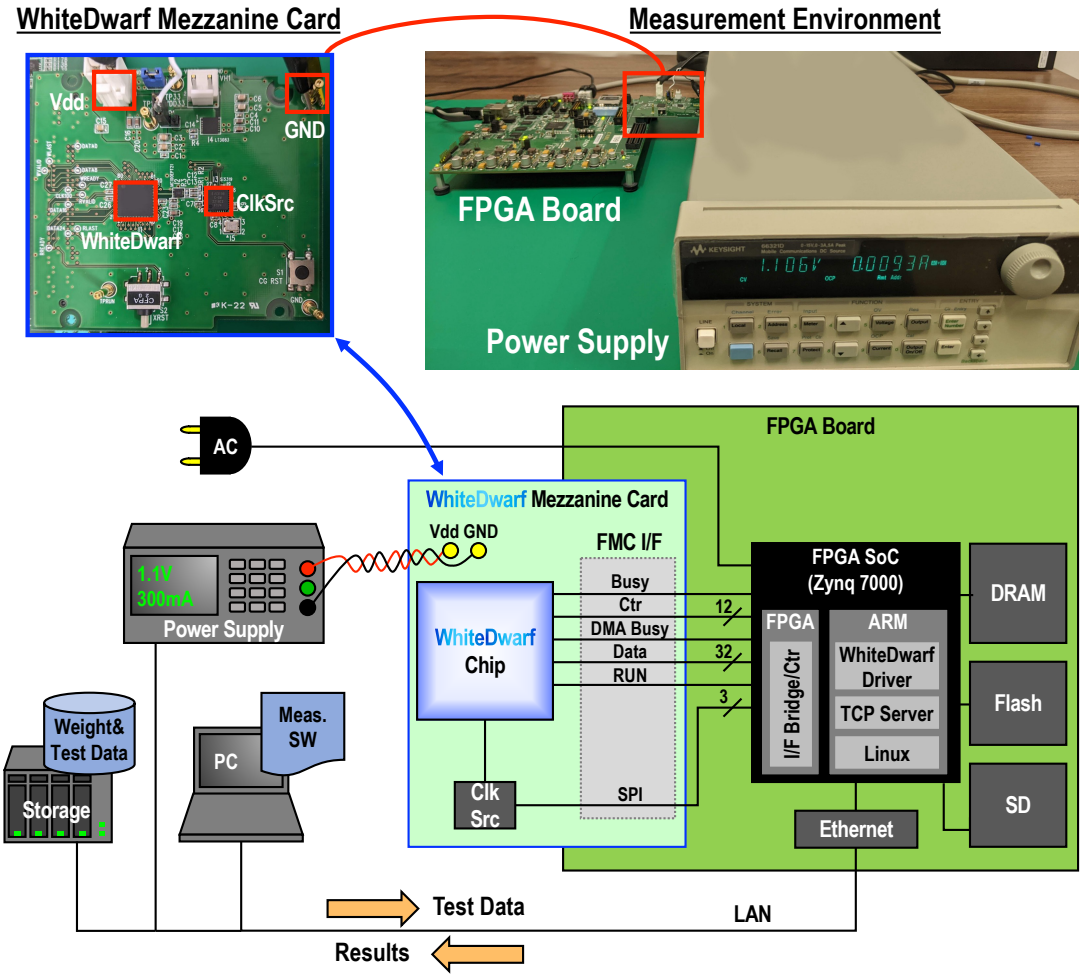


Fig. 7.19: *WhiteDwarf* chips are connected through a custom Mezzanine card to an FPGA and power supply for measurement and evaluation.

a *WhiteDwarf* Mezzanine card, a power supply, and a PC. The PC sends the test data to the FPGA board, which transmits it to *WhiteDwarf* via FMC I/F. The power supply provides 0.77–1.20 V voltage to the chip and measures operational current during the process. These results are sent to the PC through the FPGA board, where analysis is conducted.

Fig. 7.20 reports results measured on the fabricated chip of 3×3 and 1×1 convolutions—the primary components of CNNs and MLPs—for different algorithmic configurations and power settings. Performance per watt is measured in the environment described earlier, while normalized execution time and internal power ratios are obtained from RTL simulation.

7.5.1 Impact of Sparsity on Efficiency

Fig. 7.20a shows the impact of weight value-level sparsity, controlled by the supermask, on efficiency, measured in TFLOPS/W. It achieves 3.12 and 2.64 TFLOPS/W at 0.77 V on 3×3 and 1×1 convolutions, respectively, in the case of dense computation (0% sparsity), progressively improving with higher sparsity. At 99% sparsity it achieves up to 12.24 and 8.17 TFLOPS/W at 0.77 V on 3×3 and 1×1, respectively.

An ablation experiment on the triple unstructured sparsity exploitation strategy, reported in Fig. 7.20b, compares the efficiency and execution time of four sparsity exploitation strategies: dense, double value-level, weight bit-level, and *WhiteDwarf*'s triple sparsity exploitation. It reveals that both the value and bit levels of exploitation provide exponential efficiency benefits, with the weight bit-level providing the most significant advantage until weight sparsity surpasses 70%. At 99% sparsity, the triple exploitation sparsity boosts TFLOPS/W by 10.3×.

With optimal bit-sparsity, bit-level exploitation provides a constant 3× speedup, only surpassed by value-level exploitation once weight sparsity surpasses 90%. At 0.77 V and 110 MHz, using 3-coat logarithmic weights, the triple sparsity exploitation provides execution time speedups of 7.7× and 6.3× on 3×3 and 1×1 convolutions, respectively.

Fig. 7.20c shows a comparison of the four available weight configurations, revealing that using the proposed optimal bit-sparsity improves performance per watt by up to 7.9% and 7.2% while reducing computation time by up to 22.4% and 20.1% on 3×3 and 1×1 convolutions, respectively. While these efficiency gains come at a negligible accuracy cost, as discussed in Section 7.3 (see Fig. 7.8), they improve *WhiteDwarf*'s flexibility, as the optimal weight configuration depends on the model (see Fig. 7.8). By raising weight sparsity from 90% to 99%, performance improves by 38.7–62.0% and computation time is reduced by 22.2–37.3% for all weight configurations, highlighting the exponential effect of

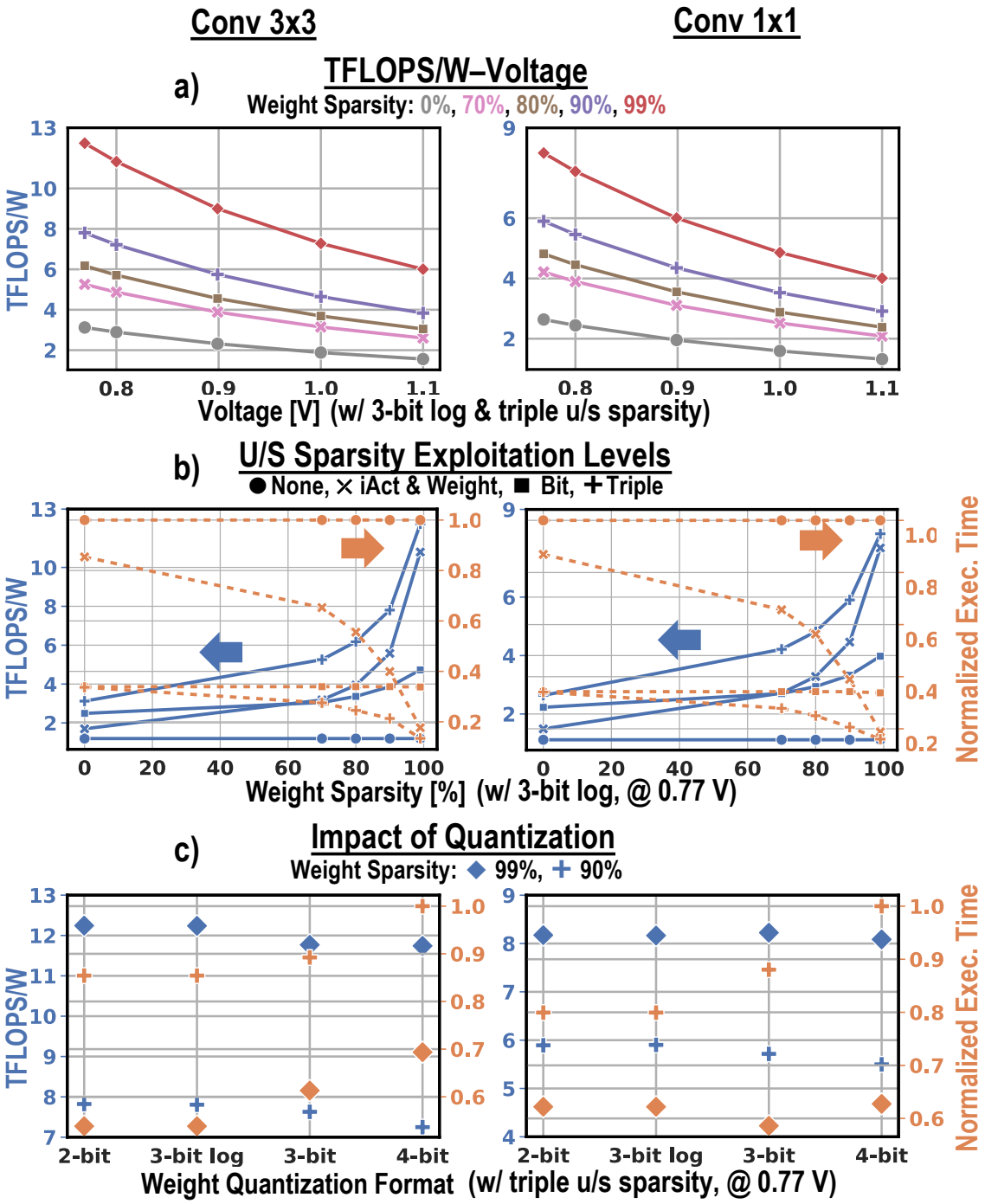


Fig. 7.20: Results measured on the fabricated *WhiteDwarf* chip using conv 3×3 (16×16×256 input, 16×16×16 output) and conv 1×1 (16×16×1024 input, 16×16×64 output) with 50% activation sparsity. (a) Impact of weight sparsity level on efficiency. (b) Impact of sparsity exploitation strategy on efficiency and execution time. (c) Impact of weight quantization setting on efficiency-performance. Measurements at 0.77, 0.80, 0.90, 1.00, and 1.10 V are taken at 110, 150, 215, 290, and 370 MHz, respectively. U/S: unstructured.

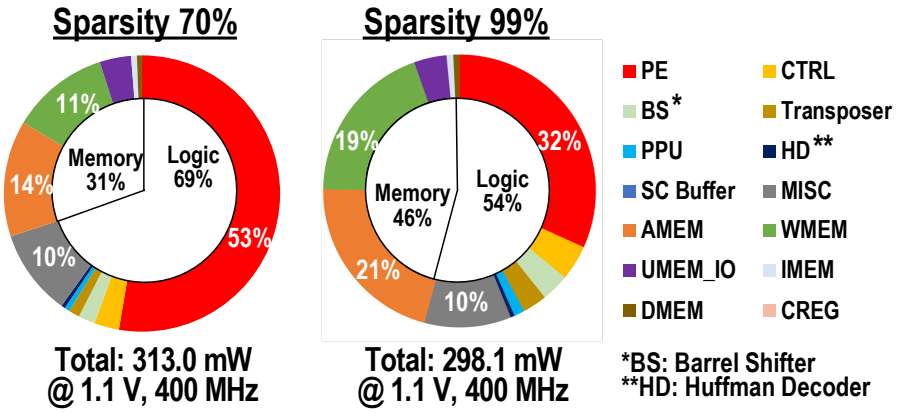


Fig. 7.21: Impact of additional sparsity on the internal power ratio when processing a conv 3×3 (iCh 256, oCh 16), using 4-bit weights, and triple sparsity exploitation.

sparsity on efficiency. Although these efficiency gains may be critical for some applications, and our 99%-sparse ResNet-50 still scores a competitive 71.66% top-1 image classification accuracy on CIFAR-100 (surpassing SOTA on SLTH accelerators [56]), they come at a loss of 8.77 percentage points, which may be too much for other applications.

7.5.2 Impact of Sparsity on Internal Power Ratio

Fig. 7.21 shows *WhiteDwarf*'s internal power ratio when processing a conv 3×3 using triple sparsity exploitation and weight sparsities of 70% (left) and 99% (right). In both cases, the PET and SRAM combinedly account for around 80% of the total consumption. Fig. 7.20b shows that increasing weight sparsity from 70% to 99% reduces computation time by 52% thanks to the triple sparse exploitation. The comparison of power ratios in Fig. 7.21 reveals that this increment in sparsity also reduces core power by 27%, thanks to the distributed clock gating of inactive PEs. Although the reduced computation time leads to a higher AMEM access rate, which causes 46% memory power growth, the net power is cut by 5%, totaling an estimated 54.4% reduction in energy consumption.

7.5.3 Full-Model Evaluation

Fig. 7.22 shows performance and frequency graphs for multiple *WhiteDwarf* chips at various operating voltages when processing inference on ImageNet with a full ResNet-50 of 70% weight sparsity and 4-bit weights, corresponding to our highest scoring ResNet-50 on ImageNet (74.8% top-1 accuracy, see Fig. 7.8). It is processed at a global 3.48 TFLOPS/W

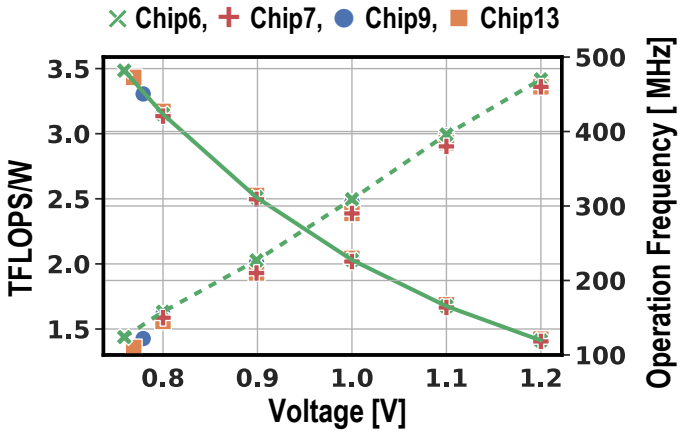


Fig. 7.22: Results for a full ResNet-50 measured on multiple *WhiteDwarf* chips, using sparsity 70%, 4-bit weights, and triple sparsity exploitation.

and 114 MHz when operated at 0.76 V, and at 1.41 TFLOPS/W and 470 MHz when operated at 1.20 V.

7.5.4 Comparison With Prior Art

Table 7.1 compares *WhiteDwarf* with the SOTA sparse and SLTH accelerators. In terms of energy efficiency and chip area, *WhiteDwarf* demonstrates superior performance over Eyeriss v2 [61]. This advantage becomes particularly evident when comparing a dense model to its counterpart with maximized sparsity, highlighting the significant impact of fine-grained sparsity exploitation and clock gating on energy efficiency. As for accuracy, Eyeriss v2 achieves 79.7% top-5 ImageNet accuracy with a sparse MobileNet [60], similar

TABLE 7.1: Performance and ImageNet accuracy compared with prior work. RN: ResNet; NR: not reported.

Chip				Full model evaluation (ImageNet)					Conv 3x3 evaluation		
Name	Process (nm)	Gates (K)	Fabricated	Model	Sparsity	Top-1 (%)	Top-5 (%)	GOPS/W GFLOPS/W†	Size	Sparsity	GOPS/W GFLOPS/W†
Eyeriss v2 [61]	65	2695	No	AlexNet	"Sparse"	NR	79.56	962.9	i:12×12×384; o:12×12×384	Dense "Sparse"	392.0 1423.2
Hiddenite [56]	40	746	Yes	RN-50	70%	70.09	NR	16 000.0	NR		
WhiteDwarf	40	986	Yes	RN-50	70%	74.81	92.36	†3484.8	i:16×16×256; o:16×16×16	Dense 99%	†3122.9 †12 238.5

to their sparse AlexNet [79] in the table, indicating a significant advantage of *WhiteDwarf*'s ultra-sparse models in accuracy compared to traditional methods. When contrasted with Hiddenite [56], *WhiteDwarf* offers a broader accuracy-size tradeoff and a wider range of supported models. *WhiteDwarf* stands out as the chip with best GFLOPS/W among high-accuracy neural engines fabricated to silicon.

7.5.5 Structured vs. Unstructured Sparsity

Although *WhiteDwarf* is focused on the more demanding challenge of unstructured sparsity, and all evaluation results presented so far use the optimal unstructured sparsity resulting of the algorithm presented in Section 7.3, the *WhiteDwarf* architecture is also capable of processing structured sparsity. Furthermore, it is even more efficient in this case. Fig. 7.23 shows a comparison of speedup when processing unstructured and structured models, for different sparsity exploitation configurations.

The unstructured weight sparsity offers 34.7–60.1% execution time reduction compared to the dense baseline at 70, 80, and 90% weight sparsity points when using the double

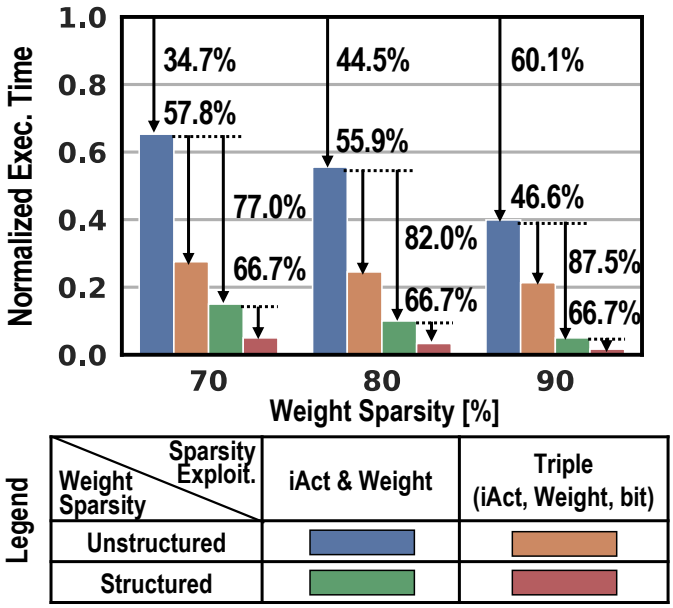


Fig. 7.23: Comparison of speedup of structured and unstructured sparsity for different sparsity exploitation strategies, using conv 3×3 (16×16×256 input, 16×16×16 output), 70% weight sparsity, 50% activation sparsity, and 3-bit log weights. Normalized to dense execution time.

value-level sparsity exploitation. In the case where weight sparsity is structured perfectly over the 1K-PEs, i.e., zero cycles are spent on hierarchical synchronization, the reduction is further magnified by 77.0 – 87.5%. In both unstructured and structured sparsity cases, also enabling the bit-level sparsity option enhances the reduction by a further 57.8 – 46.6% and 66.7%, underlining the importance of *WhiteDwarf* 's triple sparsity exploitation capability.

7.6 Discussion and Conclusion

This is the first work proposing a holistic co-design for an ultra-compact neural network acceleration based on the insights from SLTH.

Compared to other value-level sparse accelerators (see Section 2.4.1), *WhiteDwarf* features a simpler dataflow, eliminating the need for complex control for intersection and accumulation. Despite lower PE utilization (roughly half of Eyeriss v2's 91–100% PE utilization [61]), *WhiteDwarf*'s ultra-compact execution and fine-grained clock gating collectively contribute to better energy efficiency.

Compared to other value-level sparse accelerators (see Section 2.4.2), since *WhiteDwarf*'s algorithm only requires at most INT4 precision for weights (half of the INT8 allocated by other shift-add architectures), *WhiteDwarf* serializes over weights instead of activations, requiring at most 3 shifts per weight instead of 7. At the same time, not serializing activations allows *WhiteDwarf* to upgrade them to FP8 precision for higher inference accuracy [104]. *WhiteDwarf* is the first bit-sparse accelerator using FP8, and the first manufactured chip in that camp [39, 107, 110, 145].

Compared to other SLTH accelerators (see Section 2.4.3), *WhiteDwarf* solves both their low accuracy and scalability shortcomings with an algorithmic improvement that sets the SOTA on SLTH models. Although the additional supermask coats increase the model size and require off-chip weight loads, this is compensated with triple model compression: a 70%-sparse ResNet-50 on [56] occupies 25 Mb and achieves 70.09% top-1 accuracy on ImageNet, while *WhiteDwarf*'s achieves 71.53% with the same model memory size, and finds both sparser and more accurate SLTH models. *WhiteDwarf* is able to accommodate for scalar supermasks, not only without sacrificing the multiplier-less computation, but also bringing both value and bit-level unstructured sparsity exploitation to SLTH accelerators.

The 40-nm *WhiteDwarf* chip, although fabricated in a relatively old process, places a strong focus on research by featuring flexible options for different sparsity exploitation levels and weight quantization configurations, and support for multiple neural models, achieving 12.24 TFLOPS/W at 99% weight sparsity.

Impact

The work in this chapter was submitted as a conference paper to the *51st Annual International Symposium on Computer Architecture (ISCA)*, Buenos Aires, Argentina, June 2024, and is currently under review.

This research was supported by Japan Science and Technology Agency (JST) CREST Grant Number JPMJCR18K3, Japan.

Chapter 8

Conclusion

This dissertation undertakes the ambitious task of bringing immediate solutions to the urgent problem of deep learning's (DL)'s efficiency by borrowing elements from reservoir computing (RC), a research field still in the early stage of fundamental research. In the process, it proposes RC-like DL, a new approach that attempts to open a new line of research while also contributing to DL and RC from the conceptual, algorithmic, and hardware fronts.

Chapter 3 demonstrates an application of RC based on cellular automata for image classification. By substituting the linear classifier with a random forest classifier, operating with single-bit features, and pruning iterations with little contribution to accuracy, it is possible to achieve a multiplier-less image classifier and also reduce 96% of addition operations and 82% of necessary features with respect to the baseline system, in addition to reducing 60% of logic area with the proposed FPGA reservoir accelerator. These improvements come at a negligible 0.6% accuracy cost, proving RC's viability for computer vision (CV). However, this system faces numerous challenges for being upscaled to a deep reservoir computer, as discussed in this chapter from both the algorithmic and physical reservoir fronts. Here, this dissertation proposes *RC-like DL*, a new approach developed in the following chapters to take the key efficiency elements from RC and apply them to DL models.

Chapter 4 proposes *Hidden-Fold Networks* (HFN), the first RC-like DL model. By mixing the fixed random weights and supermask-based learning of the Strong Lottery Ticket Hypothesis (SLTH) with the shallow dynamical architecture resulting from Folding, this chapter presents a version of the residual neural network (ResNet) that bears strong similarities with RC. Furthermore, HFN compresses ResNet model memory size by 48.55× down to 2 MB, smaller than similar small-sized approaches, which promises enormous energy savings from reduced data movement. On the other end of the accuracy-size tradeoff, HFN is also high-performing, achieving top-1 classification accuracies of 78.90%

and 71.92% in CIFAR-100 and ImageNet, respectively. This chapter also proposes *the ensemble of unrollings view* for a better understanding of ResNet.

Despite many advantages, HFN has one flaw: it suffers from a lower accuracy on a larger dataset (ImageNet) due to an inefficient training algorithm. Chapter 5 proposes an upgrade to said training algorithm—`edge-popup`—by enhancing its supermask. *Multicoated Supermasks* (MSup), a scalar supermask that blends training, pruning, and quantization, helps `edge-popup` reduce loss even after edge swapping stops. Although MSup achieves a lower compression ratio of 19.6×, it boosts accuracy the accuracy on ImageNet of ResNet-50 from 68.16% with a single-coated supermask to 74.3% with a 7-coated MSup, and with a deeper architecture it achieves 76.5%, almost matching the accuracy of a dense ResNet.

Chapter 6 mixes the high-compression of HFN and the high-accuracy of MSup: a 7-coated HFN achieves 73.57% accuracy on ImageNet while occupying only 3.6 MB, 28.4×. Furthermore, combining this research with the existing signed supermask (SSup) raises the accuracy of the 7-coated HFN to 75.28%—close to the original's 76.89%—while keeping a compression ratio 25.0× (corresponding to a 4 MB model memory size), completing a broad size-accuracy tradeoff spectrum ranging from the tiny 2 MB HFN to the high-accuracy—yet still small—4 MB version. In CIFAR-100, a signed HFN surpasses the accuracy of the baseline ResNet-50, achieving 79.53% accuracy with 38.8× compression (2.5 MB). It shall be noted that, until the next chapter, reported results do not consider data augmentation strategies nor model compression through encoding.

Furthermore, this chapter proposes the *Ternary Strong Lottery Hypothesis* (T-SLTH), an extension of the SLTH that allows it to operate with three types of sparsity, including the more reservoir-like random connectivity. The T-SLTH opens the door to six types of supermask compatible with RC-like DL, and an additional supermask type that realizes a novel framework for concurrent training, pruning, and quantization for generic DL.

Chapter 7 proposes WhiteDwarf, a holistic algorithm-hardware approach to efficient neural inference. Its *triple compression algorithm*, on top of building on the models proposed in previous chapters, employs a mixed-precision format of FP8 activations, INT4 weights, and FP16 normalization, in addition to Huffman coding for lossless compression in the off-chip memory access. Additionally, it extends the proposed approach, so far only applied to ResNet, to a deep MLP model. With a carefully tuned learning schedule, this model design achieves accuracies as high as 80.77% and 74.81%, and off-chip compression ratios of 34.1–290.5× and 9.7–190.0× on CIFAR-100 and ImageNet, respectively.

Compression continues once the model enters the *triple unstructured sparsity* WhiteDwarf *architecture*. The datapath removes zero elements at the value and the fine-grained bit

levels, and the low bit-width of the proposed models is exploited with a multiplier-less architecture, similar to the RC image classifier proposed in Chapter 3. Furthermore, sparsity is further exploited for power reduction with a fine-grained network of clock-gating signals. In the end, only 0.05–1.2% and 0.5–2.3% of the model weights are used for computation in CIFAR-100 and ImageNet, respectively, despite achieving high accuracy, demonstrating that most of the computation in DNN is unnecessary.

The fabricated 40-nm CMOS *WhiteDwarf* chip achieves 12.24 TFLOPS/W and demonstrates through a series of ablation studies how the proposed algorithm's weight quantization and the architecture's sparsity exploitation improve efficiency and reduce power. Although *WhiteDwarf* is evaluated with the more challenging fully-learned triple-supermask models with unstructured sparsity, it is also capable of accelerating all of the RC-like DL models with arbitrary connectivity proposed in Chapter 6, making it the first RC-like DL inference accelerator.

Table 8.1 compares the most accurate ResNet-50 version of each chapter. Note that these are not necessarily the most accurate nor the most efficient proposed models, and this table does not capture the rich tradeoffs in size, accuracy, sparsity, and randomness presented in this dissertation. However, it does highlight that even if prioritizing accuracy, by applying RC elements at the algorithmic and hardware level, it is possible to compress significantly the memory size of current computer vision backbones without sacrificing accuracy.

TABLE 8.1: Comparison of the most accurate ResNet-50 of each chapter. Compression rates consider total model memory size with the coding scheme used in each chapter.

Most accurate ResNet-50	CIFAR-100		ImageNet	
	Top-1 Acc.	Compression	Top-1 Acc.	Compression
Vanilla	78.74%	1.0×	76.89%	1.0×
HNN	77.03%	32.1×	68.16%	32.1×
Chapter 4	77.68%	46.8×	67.14%	46.8×
Chapter 5	77.24%	17.3×	74.30%	19.6×
Chapter 6	79.53%	38.8×	75.28%	25.0×
Chapter 7	**80.77%**	off-chip: 34.1× on-chip: **81.6×**	**74.81%**	off-chip: 9.7× on-chip: **42.9×**

Acknowledgements

First and foremost, I would like to express my gratitude to my doctoral supervisor, Prof. Masato Motomura, and all the committee members, Prof. Atsushi Takahashi, Prof. Tsuyoshi Isshiki, Assoc. Prof. Hiroshi Sasaki, Assoc. Prof. Yuko Hara-Azumi, and Assoc. Prof. Daichi Fujiki, Tokyo Institute of Technology, for making possible this dissertation.

This dissertation has come to fruition thanks to the input and support of all the members of the AI Computing Research Unit, Tokyo Institute of Technology. Special thanks to the indispensable collaboration of my brilliant teammate Mr. Yasuyuki Okoshi. I would also like to extend my gratitude to my master's thesis supervisor, Prof. Masanori Hashimoto, Kyoto University, and all the members of the former VLSI Laboratory, Osaka University. Much of the work presented was started with their collaboration.

Further, I am grateful to the CREST project directed by Prof. Hideyuki Suzuki, Osaka University, which has inspired and nurtured most of this research. The support of the CREST project directed by Prof. Motomura was also fundamental. I would also like to acknowledge the contributions of the co-authors, reviewers, and editors of the publications resulting from this work and to thank the input from the researchers at the NEC, KIOXIA, and NTT corporations with whom I had the opportunity to discuss my research.

My interest in research was first sparked during the friendly welcome I received at the Intelligent Networking Systems Laboratory, Osaka University, and stimulated into a career path by the engaging opportunity I received at the High Performance Computing and Networking research group, Autonomous University of Madrid. Warm thanks to Prof. Takashi Watanabe, Osaka University, Prof. Kazuhiko Kinoshita, Tokushima University, Prof. Gustavo Daniel Sutter Capristo and Prof. Jorge E. López de Vergara, Autonomous University of Madrid, and Dr. Mario Ruiz Noguera, Dr. Tobías Alonso Pugliese, Dr. David Muelas Recuenco, Dr. Rafael Leira Osuna, and Dr. José Fernando Zazo Rollón.

The six years I have spent in graduate school have been sponsored by the Japanese Government (MEXT) Scholarship. I am honored and thankful to the Japanese public and the Ministry of Education, Culture, Sports, Science and Technology of Japan for the financial support, and to the Embassy of Japan in Spain and Prof. Hashimoto for recommending me. I would have never been able to access this opportunity without my

excellent Japanese-language teachers, Ms. Laura Rivas Campos and Mr. Takuzo Hirono. Additionally, this research was partially supported by Japan Science and Technology Agency (JST) CREST Grant Numbers JPMJCR18K2 and JPMJCR18K3, and Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Numbers JP18H05288 and JP22H03555.

Finally, I am deeply indebted to the care and support of family, friends, and bandmates. Special thanks to my friend Dr. Hector Bjoljahn Hougaard, Yukawa Institute for Theoretical Physics, whose advice and encouragement were instrumental to surviving this odyssey. I want to express my deepest appreciation to my greatest ally, my wife Manami, who has been by my side every second of this journey. I also wish to thank her family for their kind support. Lastly, I cannot begin to express my gratitude to my parents and my sisters.

Bibliography

[1] Data centres metered electricity consumption 2022. Statistical publication, Central Statistics Office, Republic of Ireland, June 2022.

[2] Google environmental report 2023. Report, Google, 2023.

[3] Nuscale small modular reactor design certification. Rule C1-2023-00729, US Nuclear Regulatory Commission, February 2023.

[4] M. Abdi and S. Nahavandi. Multi-residual networks: Improving the speed and accuracy of residual networks. *arXiv preprint arXiv:1609.05672*, 2016.

[5] M. M. Aladago and L. Torresani. Slot machines: Discovering winning combinations of random weights in neural networks. In *Proc. Int. Conf. Mach. Learn.*, pages 163–174, 2021.

[6] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos. Bit-Pragmatic deep neural network computing. In *Proc. IEEE/ACM Int. Symp. Microarch.*, pages 382–394, 2017.

[7] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Proc. Int. Symp. Comp. Archit.*, pages 1–13, Seoul, Republic of Korea, 2016. IEEE Press. doi:10.1109/ISCA.2016.11.

[8] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[9] S. Baek, M. Song, J. Jang, G. Kim, and S.-B. Paik. Face detection in untrained deep neural networks. *Nat. Commun.*, 12(1):1–15, 2021.

[10] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[11] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[12] Boost. Boost C++ Libraries. `http://www.boost.org/`, 2015. Last accessed 2015-06-30.

[13] I. Caswell, C. Shen, and L. Wang. Loopy neural nets: Imitating feedback loops in the human brain. Technical Report 110 cs231n, Stanford University, Stanford, CA, 2016.

[14] V. S. Charlie Giattino, Edouard Mathieu and M. Roser. Computation used to train notable artificial intelligence systems. https://ourworldindata.org/grapher/artificial-intelligence-training-computation, 2023.

[15] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, and C. Xu. AdderNet: Do we really need multiplications in deep learning? In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 1468–1477, 2020.

[16] D. Chijiwa, S. Yamaguchi, Y. Ida, K. Umakoshi, and T. Inoue. Pruning randomly initialized neural networks with iterative randomization. In *Proc. Adv. Neural Inform. Process. Syst.*, 2021.

[17] B. Chu, D. Yang, and R. Tadinada. Visualizing residual networks. *arXiv preprint arXiv:1701.02362*, 2017.

[18] J. Cluzeau, X. Henriquel, G. Rebender, G. Soudain, L. van Dijk, A. Gronskiy, D. Haber, C. Perret-Gentil, and R. Polak. Concepts of design assurance for neural networks (CoDANN). Public report extract, European Union Aviation Safety Agency (EASA) and Daedalean AG, March 2020.

[19] M. Cook. Universality in elementary cellular automata. *Complex Syst.*, 15(1):1–40, 2004.

[20] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. RandAugment: Practical automated data augmentation with a reduced search space. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 702–703, 2020.

[21] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, 1998.

[22] A. Delmás Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos. Bit-Tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks. In *Proc. Conf. Arch. Sup. Prog. Lang. Op. Sys.*, pages 749–763, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3297858.3304041.

[23] J. Diffenderfer and B. Kailkhura. Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network. In *Proc. Int. Conf. Learn. Repr.*, 2021.

[24] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proc. Int. Conf. Learn. Repr.*, 2021.

[25] A. B. Dror, N. Zehngut, A. Raviv, E. Artyomov, R. Vitek, and R. Jevnisek. Layer folding: Neural network depth reduction using activation linearization. *arXiv preprint arXiv:2106.09309*, 2021.

[26] A. Dubey, M. Chatterjee, and N. Ahuja. Coreset-based neural network compression. In *Proc. European Conf. Comput. Vis.*, pages 454–470, 2018.

[27] R. Dupont, M. A. Alaoui, H. Sahbi, and A. Lebois. Extracting effective subnetworks with Gumbel-softmax. *arXiv preprint arXiv:2202.12986*, 2022.

[28] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. In *Proc. Int. Conf. Mach. Learn.*, pages 2943–2952, 2020.

[29] W. Feng, X. Zhang, Q. Song, and G. Sun. The incoherence of deep isotropic neural networks increases their performance in image classification. *Electronics*, 11(21):3603, 2022.

[30] J. Fischer and R. Burkholz. Lottery tickets with nonzero biases. *arXiv preprint arXiv:2110.11150*, 2021.

[31] J. Fischer and R. Burkholz. Plant'n'seek: Can you find the winning ticket? *arXiv preprint arXiv:2111.11153*, 2021.

[32] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proc. Int. Conf. Learn. Repr.*, 2019.

[33] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proc. Int. Conf. Mach. Learn.*, pages 3259–3269, 2020.

[34] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *Proc. Int. Conf. Learn. Repr.*, 2021.

[35] J. Frankle, D. J. Schwab, and A. S. Morcos. Training batchnorm and only BatchNorm: On the expressive power of random features in CNNs. In *Proc. Int. Conf. Learn. Repr.*, 2021.

[36] C. Freitag, M. Berners-Lee, K. Widdicks, B. Knowles, G. S. Blair, and A. Friday. The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. *Patterns*, 2(9):100340, 2021. doi:10.1016/j.patter.2021.100340.

[37] D. Fujiki, X. Wang, A. Subramaniyan, and R. Das. *In-/near-memory Computing*. Springer Cham, 2021. doi:10.1007/978-3-031-01772-8.

[38] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[39] K. Goetschalckx and M. Verhelst. DepFiN: A 12nm, 3.8TOPs depth-first CNN processor for high res. image processing. In *IEEE Symp. VLSI Circuits*, pages 1–2, 2021. doi:10.23919/VLSICircuits52068.2021.9492338.

[40] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar. SparTen: A sparse tensor accelerator for convolutional neural networks. In *Proc. IEEE/ACM Int. Symp. Microarch.*, MICRO '52, pages 151–165, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3352460.3358291.

[41] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[42] K. Greff, R. K. Srivastava, and J. Schmidhuber. Highway and residual networks learn unrolled iterative estimation. In *Proc. Int. Conf. Learn. Repr.*, 2017.

[43] K. Grosse and M. Backes. How many winning tickets are there in one DNN? *arXiv preprint arXiv:2006.07014*, 2020.

[44] Q. Guo, Z. Yu, Y. Wu, D. Liang, H. Qin, and J. Yan. Dynamic recursive neural network. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 5147–5156, 2019.

[45] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. EIE: Efficient inference engine on compressed deep neural network. *Proc. Int. Symp. Comp. Archit.*, 2016.

[46] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *Proc. Int. Conf. Learn. Repr.*, 2016.

[47] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Proc. Adv. Neural Inform. Process. Syst.*, pages 1135–1143, 2015.

[48] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi:10.1038/s41586-020-2649-2.

[49] M. Hashimoto, Á. López García-Arias, and J. Yu. *Bridging the Gap Between Reservoirs and Neural Networks*, pages 245–258. Springer Nature Singapore, 2023. doi:10.1007/978-981-99-5072-0_12.

[50] M. Hashimoto, T. Matsumoto, M. Tanaka, R. Shirai, N. Tate, M. Nakagawa, T. Tokuda, K. Sasagawa, J. Ohta, and J. Yu. *Exploring Integrated Device Implementation for FRET-Based Optical Reservoir Computing*, pages 89–108. Springer Nature Singapore, Singapore, 2024. doi:10.1007/978-981-99-5072-0_5.

[51] S. Hayou, J.-F. Ton, A. Doucet, and Y. W. Teh. Robust pruning at initialization. In *Proc. Int. Conf. Learn. Repr.*, 2021.

[52] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. IEEE Int. Conf. Comput. Vis.*, pages 1026–1034, 2015.

[53] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 770–778, 2016.

[54] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *Proc. European Conf. Comput. Vis.*, pages 630–645. Springer, 2016.

[55] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. volume 32, 2019.

[56] K. Hirose, J. Yu, K. Ando, Y. Okoshi, Á. López García-Arias, J. Suzuki, T. V. Chu, K. Kawamura, and M. Motomura. Hiddenite: 4K-PE hidden network inference 4D-tensor engine exploiting on-chip model construction achieving 34.8-to-16.0TOPS/W for CIFAR-100 and ImageNet. In *Proc. IEEE Int. Solid-State Circuits Conf.*, volume 65, pages 1–3, 2022.

[57] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[58] M. Horowitz. 1.1 computing's energy problem (and what we can do about it). In *Proc. IEEE Int. Solid-State Circuits Conf.*, pages 10–14, 2014.

[59] Q. Hou, Z. Jiang, L. Yuan, M.-M. Cheng, S. Yan, and J. Feng. Vision Permutator: A permutable mlp-like architecture for visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(1):1328–1334, 2022.

[60] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[61] Y. hsin Chen, T.-J. Yang, J. S. Emer, and V. Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Sel. Top. Circuits Syst.*, 9(2):292–308, 2019.

[62] K. Hu. ChatGPT sets record for fastest-growing user base - analyst note. *Reuters*, Feb. 2023.

[63] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 4700–4708, 2017.

[64] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *Proc. European Conf. Comput. Vis.*, pages 646–661. Springer, 2016.

[65] G.-B. Huang, L. Chen, C. K. Siew, et al. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Netw.*, 17(4):879–892, 2006.

[66] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[67] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. Int. Conf. Mach. Learn.*, pages 448–456, 2015.

[68] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. Technical Report 34, 2001.

[69] A. Jalalvand, K. Demuynck, W. De Neve, and J.-P. Martens. On the application of reservoir computing networks for noisy image recognition. *Neurocomputing*, 277:237–248, 2018.

[70] S. Jastrzębski, D. Arpit, N. Ballas, V. Verma, T. Che, and Y. Bengio. Residual connections encourage iterative inference. In *Proc. Int. Conf. Learn. Repr.*, 2018.

[71] F. Juefei-Xu, V. Naresh Boddeti, and M. Savvides. Perturbative neural networks. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 3310–3318, 2018.

[72] W. Kang and D. Kim. Deeply shared filter bases for parameter-efficient convolutional neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Proc. Adv. Neural Inform. Process. Syst.*, 2021.

[73] K. Kar, J. Kubilius, K. Schmidt, E. B. Issa, and J. J. DiCarlo. Evidence that recurrent circuits are critical to the ventral stream's execution of core object recognition behavior. *Nat. Neurosci.*, 22(6):974–983, 2019.

[74] N. Karvonen, J. Nilsson, D. Kleyko, and L. L. Jiménez. Low-power classification using FPGA—an approach based on cellular automata, neural networks, and hyperdimensional computing. In *Proc. Int. j. mach. learn. appl.*, pages 370–375. IEEE, 2019.

[75] D. Kleyko, S. Khan, E. Osipov, and S.-P. Yong. Modality classification of medical images with distributed representations based on cellular automata reservoir computing. In *Proc. IEEE Int. Symp. Biomed. Imag.*, pages 1053–1056. IEEE, 2017.

[76] O. Köpüklü, M. Babaee, S. Hörmann, and G. Rigoll. Convolutional neural networks with layer reuse. In *Proc. IEEE Int. Conf. Image Process.*, pages 345–349. IEEE, 2019.

[77] N. Koster, O. Grothe, and A. Rettinger. Signing the supermask: Keep, hide, invert. In *Proc. Int. Conf. Learn. Repr.*, 2022.

[78] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, Toronto, 2009.

[79] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

[80] J. Kubilius, M. Schrimpf, A. Nayebi, D. Bear, D. L. Yamins, and J. J. DiCarlo. CORnet: Modeling the neural mechanisms of core object recognition. *BioRxiv preprint BioRxiv:408385*, 2018.

[81] C. G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Phys. D: Nonlinear Phenom.*, 42(1):12–37, 1990. doi:10.1016/0167-2789(90)90064-V.

[82] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[83] Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010. URL: http://yann.lecun.com/exdb/mnist/.

[84] N. Lee, T. Ajanthan, S. Gould, and P. H. Torr. A signal propagation perspective for pruning neural networks at initialization. In *Proc. Int. Conf. Learn. Repr.*, 2020.

[85] N. Lee, T. Ajanthan, and P. Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *Proc. Int. Conf. Learn. Repr.*, 2019.

[86] S. Leroux, P. Molchanov, P. Simoens, B. Dhoedt, T. Breuel, and J. Kautz. IamNN: Iterative and adaptive mobile neural network for efficient image classification. *arXiv preprint arXiv:1804.10123*, 2018.

[87] G. Li, W. Xu, Z. Song, N. Jing, J. Cheng, and X. Liang. Ristretto: An atomized processing architecture for sparsity-condensed stream flow in CNN. In *Proc. IEEE/ACM Int. Symp. Microarch.*, pages 1434–1450, 2022. doi:10.1109/MICRO56248.2022.00097.

[88] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 3367–3375, 2015.

[89] Q. Liao and T. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

[90] X. Lin, G. Li, Z. Liu, Y. Liu, F. Zhang, Z. Song, N. Jing, and X. Liang. AdaS: A fast and energy-efficient CNN accelerator exploiting bit-sparsity. In *Proc. ACM/IEEE Des. Autom. Conf.*, pages 1–6. IEEE, 2023.

[91] J. Liu, Z. Xu, R. Shi, R. C. Cheung, and H. K. So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. *arXiv preprint arXiv:2005.06870*, 2020.

[92] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. On the variance of the adaptive learning rate and beyond. In *Proc. Int. Conf. Learn. Repr.*, April 2020.

[93] Á. López García-Arias, M. Hashimoto, M. Motomura, and J. Yu. Hidden-Fold Networks: Random recurrent residuals using sparse supermasks. In *Proc. British Mach. Vis. Conf.*, 2021.

[94] Á. López García-Arias, M. Hashimoto, M. Motomura, and J. Yu. Hidden-Fold Networks: Random recurrent residuals using sparse supermasks. https://github.com/Lopez-Angel/hidden-fold-networks, 2021. GitHub repository.

[95] Á. López García-Arias, M. Hashimoto, M. Motomura, and J. Yu. Hidden-Fold Networks: Random recurrent residuals using sparse supermasks. *arXiv preprint arXiv:2111.12330*, 2021.

[96] Á. López García-Arias, Y. Okoshi, M. Hashimoto, M. Motomura, and J. Yu. Recurrent residual networks contain stronger lottery tickets. *IEEE Access*, 11:16588–16604, 2023. doi:10.1109/ACCESS.2023.3245808.

[97] Á. López García-Arias, J. Yu, and M. Hashimoto. Low-cost reservoir computing using cellular automata and random forests. In *Proc. IEEE Int. Symp. Circuits and Syst.*, pages 1–5. IEEE, 2020.

[98] T. Lv, C. Bai, and C. Wang. MDMLP: Image classification from scratch on small datasets with mlp. *arXiv preprint arXiv:2205.14477*, 2022.

[99] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comput.*, 14(11):2531–2560, 2002. doi:10.1162/089976602760407955.

[100] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *Proc. Int. Conf. Mach. Learn.*, pages 6682–6691, 2020.

[101] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proc. European Conf. Comput. Vis.*, pages 67–82, 2018.

[102] O. Martin, A. M. Odlyzko, and S. Wolfram. Algebraic properties of cellular automata. *Commun. Math. Phys.*, 93(2):219–258, 1984.

[103] N. McDonald. Reservoir computing & extreme learning machines using pairs of cellular automata rules. In *Proc. IEEE Proc. Int. Jt. Conf. Neural Netw.*, pages 2429–2436. IEEE, 2017.

[104] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, et al. FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

[105] M. Mitchell, P. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *arXiv preprint adap-org/9303003*, 1993.

[106] MMClassification Contributors. OpenMMLab's image classification toolbox and benchmark. https://github.com/open-mmlab/mmclassification, 2020.

[107] S. Moon, H.-G. Mun, H. Son, and J.-Y. Sim. A 127.8TOPS/W arbitrarily quantized 1-to-8b scalable-precision accelerator for general-purpose deep learning with reduction of storage, logic and latency waste. In *Proc. IEEE Int. Solid-State Circuits Conf.*, pages 21–23, 2023. doi:10.1109/ISSCC42615.2023.10067615.

[108] A. Morán, C. F. Frasser, M. Roca, and J. L. Rosselló. Energy-efficient pattern recognition hardware with elementary cellular automata. *IEEE Trans. Comput.*, 69(3):392–401, 2019.

[109] A. Morán, C. F. Frasser, and J. L. Rosselló. Reservoir computing hardware with cellular automata. *arXiv preprint arXiv:1806.04932*, 2018. arXiv:1806.04932.

[110] H. Mun, H. Son, S. Moon, J. Park, B. Kim, and J.-Y. Sim. A 28 nm 66.8 TOPS/W sparsity-aware dynamic-precision deep-learning processor. In *IEEE Symp. VLSI Circuits*, pages 1–2, 2023. doi:10.23919/VLSITechnologyandCir57934.2023.10185264.

[111] M. Nakagawa. *FRET Networks: Modeling and Analysis for Computing*, pages 109–138. Springer Nature Singapore, Singapore, 2024. doi:10.1007/978-981-99-5072-0_6.

[112] M. Nakagawa, Y. Miyata, N. Tate, T. Nishimura, S. Shimomura, S. Shirasaka, J. Tanida, and H. Suzuki. Spatiotemporal model for fret networks with multiple donors and acceptors: multicomponent exponential decay derived from the master equation. *J. Opt. Soc. Am. B*, 38(2):294–299, Feb 2021. URL: https://opg.optica.org/josab/abstract.cfm?URI=josab-38-2-294, doi:10.1364/JOSAB.410658.

[113] S. Nichele and M. S. Gundersen. Reservoir computing using nonuniform binary cellular automata. *Complex Syst.*, 26, 2017.

[114] S. Nichele and A. Molund. Deep learning with cellular automaton-based reservoir computing download pdf. *Complex Syst.*, 26(4), 2017.

[115] S. Nichele and G. Tufte. Evolution of incremental complex behavior on cellular machines. In *Proc. Artif. Life Conf.*, pages 63–70. MIT Press, 2013.

[116] T. Nishimura. *Fluorescence Energy Transfer Computing*, pages 51–70. Springer Nature Singapore, Singapore, 2024. doi:10.1007/978-981-99-5072-0_3.

[117] NVIDIA. Deeplearningexamples/pytorch/. https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/ConvNets/resnet50v1.5, 2021.

[118] Y. Okoshi, Á. López García-Arias, K. Hirose, K. Ando, K. Kawamura, T. Van Chu, M. Motomura, and J. Yu. Multicoated supermasks enhance hidden networks. In *Proc. Int. Conf. Mach. Learn.*, pages 17045–17055, 2022.

[119] Y. Okoshi, Á. López García-Arias, K. Hirose, K. Ando, K. Kawamura, T. Van Chu, M. Motomura, and J. Yu. Multicoated supermasks enhance hidden networks. `https://github.com/yasu0001/multicoated-supermasks`, 2022.

[120] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[121] L. Orseau, M. Hutter, and O. Rivasplata. Logarithmic pruning is all you need. In *Proc. Adv. Neural Inform. Process. Syst.*, pages 2925–2934, 2020.

[122] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proc. Int. Symp. Comp. Archit.*, pages 27–40, 2017. `doi:10.1145/3079856.3080254`.

[123] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Proc. Adv. Neural Inform. Process. Syst.*, pages 8024–8035, 2019.

[124] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

[125] A. Pensia, S. Rajput, A. Nagle, H. Vishwakarma, and D. S. Papailiopoulos. Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient. In *Proc. Adv. Neural Inform. Process. Syst.*, 2020.

[126] P. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *Proc. Int. Conf. Mach. Learn.*, pages 82–90, 2014.

[127] S. Pontes-Filho, A. Yazidi, J. Zhang, H. Hammer, G. Mello, I. Sandvig, G. Tufte, and S. Nichele. A general representation of dynamical systems for reservoir computing. *arXiv preprint arXiv:1907.01856*, 2019.

[128] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. What's hidden in a randomly weighted neural network? In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 11893–11902, 2020.

[129] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. What's hidden in a randomly weighted neural network? `https://github.com/allenai/hidden-networks`, 2020.

[130] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. In *Proc. Int. Conf. Mach. Learn.*, pages 8821–8831, 2021.

[131] H. Ritchie. Climate change and flying: what share of global co2 emissions come from aviation? *Our World in Data*, 2020. https://ourworldindata.org/co2-emissions-from-aviation.

[132] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015.

[133] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobilenetV2: Inverted residuals and linear bottlenecks. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 4510–4520, 2018.

[134] P. Savarese and M. Maire. Learning implicitly recurrent cnns through parameter sharing. *arXiv preprint arXiv:1902.09701*, 2019.

[135] M. Schrimpf, J. Kubilius, H. Hong, N. J. Majaj, R. Rajalingham, E. B. Issa, K. Kar, P. Bashivan, J. Prescott-Roy, K. Schmidt, D. L. K. Yamins, and J. J. DiCarlo. Brain-score: Which artificial neural network for object recognition is most brain-like? *BioRxiv preprint BioRxiv:407007*, 2020.

[136] M. Schüle and R. Stoop. A full computation-relevant topological dynamics classification of elementary cellular automata. *Chaos*, 22(4):043143, 2012.

[137] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos. Compute trends across three eras of machine learning. In *Proc. IEEE Proc. Int. Jt. Conf. Neural Netw.* IEEE, 2022. `doi:10.1109/ijcnn55064.2022.9891914`.

[138] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos. Laconic deep learning inference acceleration. In *Proc. Int. Symp. Comp. Archit.*, pages 304–317, 2019.

[139] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. volume 28, 2015.

[140] E. Silverman. Convolutional neural networks for cellular automata classification. In *Proc. Artif. Life Conf.*, pages 280–281. MIT Press, 2019.

[141] C. J. Spoerer, P. McClure, and N. Kriegeskorte. Recurrent convolutional neural networks: a better model of biological object recognition. *Front. Psychol.*, 8:1551, 2017.

[142] K. Sreenivasan, S. Rajput, J.-Y. Sohn, and D. Papailiopoulos. Finding nearly everything within random binary networks. In *Proc. Int. Conf. on Artif. Intell. and Stat.*, pages 3531–3541, 2022.

[143] K. Sreenivasan, J.-y. Sohn, L. Yang, M. Grinde, A. Nagle, H. Wang, K. Lee, and D. Papailiopoulos. Rare gems: Finding lottery tickets at initialization. 2022.

[144] E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*, 2019.

[145] W. Sun, X. Feng, C. Tang, S. Fan, Y. Yang, J. Yue, H. Yang, and Y. Liu. A 28nm 2D/3D unified sparse convolution accelerator with block-wise neighbor searcher for large-scaled voxel-based point cloud network. In *Proc. IEEE Int. Solid-State Circuits Conf.*, pages 328–330, 2023. doi:10.1109/ISSCC42615.2023.10067644.

[146] H. Suzuki, J. Tanida, and M. Hashimoto. *Photonic Neural Networks with Spatiotemporal Dynamics: Paradigms of Computing and Implementation*. Springer Nature Singapore, 2023. doi:10.1007/978-981-99-5072-0.

[147] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. *Efficient Processing of Deep Neural Networks*. Synthesis Lectures on Computer Architecture. Springer International Publishing, Cham, 2020. URL: https://link.springer.com/10.1007/978-3-031-01766-7, doi:10.1007/978-3-031-01766-7.

[148] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proc. Int. Conf. Mach. Learn.*, pages 6105–6114, 2019.

[149] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose. Recent advances in physical reservoir computing: A

review. *Neural Netw.*, 115:100–123, Jul 2019. doi:10.1016/j.neunet.2019.03.005.

[150] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Proc. Adv. Neural Inform. Process. Syst.*, volume 33, pages 6377–6389, 2020.

[151] N. Tate. *Quantum-Dot-Based Photonic Reservoir Computing*, pages 71–87. Springer Nature Singapore, Singapore, 2024. doi:10.1007/978-981-99-5072-0_4.

[152] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al. MLP-Mixer: An all-MLP architecture for vision. volume 34, pages 24261–24272, 2021.

[153] H. Touvron, P. Bojanowski, M. Caron, M. Cord, A. El-Nouby, E. Grave, G. Izacard, A. Joulin, G. Synnaeve, J. Verbeek, et al. ResMLP: Feedforward networks for image classification with data-efficient training. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(4):5314–5321, 2022.

[154] F. Triefenbach, A. Jalalvand, B. Schrauwen, and J.-P. Martens. Phoneme recognition with large hierarchical reservoirs. In *Proc. Adv. Neural Inform. Process. Syst.*, pages 2307–2315, 2010.

[155] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[156] R. S. van Bergen and N. Kriegeskorte. Going in circles is the way forward: the role of recurrence in visual inference. *Curr. Opin. Neurobiol.*, 65:176–193, 2020.

[157] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. volume 30, 2017.

[158] A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. volume 29, 2016.

[159] C. Wang, G. Zhang, and R. Grosse. Picking winning tickets before training by preserving gradient flow. In *Proc. Int. Conf. Learn. Repr.*, 2020.

[160] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu. Pruning from scratch. In *Proc. AAAI Conf. on Artif. Intell.*, volume 34, pages 12273–12280, 2020.

[161] S. Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

[162] M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi. Supermasks in superposition. volume 33, 2020.

[163] Y. Wu and K. He. Group normalization. In *Proc. European Conf. Comput. Vis.*, pages 3–19, 2018.

[164] H. Yang, L. Duan, Y. Chen, and H. Li. BSQ: Exploring bit-level sparsity for mixed-precision neural network quantization. In *Proc. Int. Conf. Learn. Repr.*, 2021.

[165] O. Yilmaz. Symbolic computation using cellular automata-based hyperdimensional computing. *Neural Comput.*, 27(12):2661–2692, 2015.

[166] H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, R. G. Baraniuk, Z. Wang, and Y. Lin. Drawing early-bird tickets: Toward more efficient training of deep networks. In *Proc. Int. Conf. Learn. Repr.*, 2020.

[167] G. Yuan, X. Ma, W. Niu, Z. Li, Z. Kong, N. Liu, Y. Gong, Z. Zhan, C. He, Q. Jin, S. Wang, M. Qin, B. Ren, Y. Wang, S. Liu, and X. Lin. MEST: Accurate and fast memory-economic sparse training framework on the edge. In *Proc. Adv. Neural Inform. Process. Syst.*, 2021.

[168] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. CutMix: Regularization strategy to train strong classifiers with localizable features. In *Proc. IEEE Int. Conf. Comput. Vis.*, pages 6023–6032, 2019.

[169] S. Zagoruyko and N. Komodakis. Wide residual networks. In *Proc. British Mach. Vis. Conf.*, 2016.

[170] A. R. Zamir, T.-L. Wu, L. Sun, W. B. Shen, B. E. Shi, J. Malik, and S. Savarese. Feedback networks. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 1308–1317, 2017.

[171] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang. SNAP: An efficient sparse neural acceleration processor for unstructured sparse deep neural network inference. volume 56, pages 636–647, 2021. doi:10.1109/JSSC.2020.3043870.

[172] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen. Cambricon-X: An accelerator for sparse neural networks. In *Proc. IEEE/ACM Int. Symp. Microarch.*, pages 1–12, 2016. doi:10.1109/MICRO.2016.7783723.

[173] T. Zhang, Z. Lin, G. Yang, and C. D. Sa. QPyTorch: A low-precision arithmetic simulation framework. *arXiv preprint arXiv:1910.04540*, 2019.

[174] Y. Zhang, L. Zhao, S. Cao, W. Wang, T. Cao, F. Yang, M. Yang, S. Zhang, and N. Xu. Integer or floating point? new outlooks for low-bit quantization on large language models. *arXiv preprint arXiv:2305.12356*, 2023.

[175] Z. Zhang and C. Jung. Recurrent convolution for compact and cost-adjustable neural networks: An empirical study. *arXiv preprint arXiv:1902.09809*, 2019.

[176] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, and H. Li. Learning N:M fine-grained structured sparse neural networks from scratch. *arXiv preprint arXiv:2102.04010*, 2021.

[177] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Proc. Adv. Neural Inform. Process. Syst.*, pages 3597–3607, 2019.

[178] X. Zhou, W. Zhang, H. Xu, and T. Zhang. Effective sparsification of neural networks with global sparsity constraint. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, 2021.

[179] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[180] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. and Pattern Recognit.*, pages 8697–8710, 2018.

List of Publications

Journal Papers

1. **Á. López García-Arias**, Y. Okoshi, M. Hashimoto, M. Motomura and J. Yu, "Recurrent Residual Networks Contain Stronger Lottery Tickets," *IEEE Access*, vol. 11, pp. 16588–16604, Jan. 2023, doi:10.1109/ACCESS.2023.3245808.

International Conference Proceedings

1. **Á. López García-Arias**, Y. Okoshi, J. Yu, J. Suzuki, H. Otsuka, T. Van Chu, K. Kawamura, D. Fujiki, and M. Motomura, "WhiteDwarf: A Holistic Co-Design Approach to Ultra-Compact Neural Inference Acceleration," *The 51st Annual International Symposium on Computer Architecture (ISCA)*, Buenos Aires, Argentina, Jun. 2024. *(Submitted Nov. 22, 2023. Under review.)*

2. Y. Okoshi*, **Á. López García-Arias***, K. Hirose, K. Ando, K. Kawamura, T. Van Chu, M. Motomura, and J. Yu*, "Multicoated Supermasks Enhance Hidden Networks," *The 39th International Conference on Machine Learning (ICML)*, Baltimore, Maryland, USA, Jul. 2022. (Also collected in *Proceedings of Machine Learning Research, PMLR*, vol. 162, pp. 17045–17055, Jul. 2022.) *: Equal contribution.

3. **Á. López García-Arias**, M. Hashimoto, M. Motomura, and J. Yu, "Hidden-Fold Networks: Random Recurrent Residuals Using Sparse Supermasks," *The 32nd British Machine Vision Conference (BMVC)*, Online, Nov. 2021.

4. **Á. López García-Arias**, J. Yu and M. Hashimoto, "Low-Cost Reservoir Computing using Cellular Automata and Random Forests," *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Online, pp. 1–5, Oct. 2020, doi:10.1109/ISCAS45731.2020.9180742.

Book Chapters (Coauthored)

1. M. Hashimoto, **Á. López García-Arias**, and J. Yu, "Bridging the Gap Between Reservoirs and Neural Networks," in *Photonic Neural Networks with Spatiotemporal Dynamics: Paradigms of Computing and Implementation*, Springer Nature Singapore, pp. 245–258, Oct. 2023, doi:10.1007/978-981-99-5072-0_12.

Journal Papers (Coauthored)

1. H. Otsuka, Y. Okoshi, **Á. López García-Arias**, K. Kawamura, T. Van Chu, D. Fujiki, M. Motomura, "Restricted Random Pruning at Initialization for High Compression Range," *Transactions on Machine Learning Research (TMLR)*, 2024. *(Submitted Jan. 9, 2024. Under review.)*

2. J. Suzuki, J. Yu, M. Yasunaga, **Á. López García-Arias**, Y. Okoshi, S. Kumazawa, K. Ando, K. Kawamura, T. Van Chu, and M. Motomura, "Pianissimo: A Sub-mW Class DNN Accelerator With Progressively Adjustable Bit-Precision," *IEEE Access*, vol. 12, pp. 2057–2073, Jan. 2024, doi:10.1109/ACCESS.2023.3347578.

International Conference Proceedings (Coauthored)

1. H. Otsuka, D. Chijiwa, **Á. López García-Arias**, Y. Okoshi, K. Kawamura, T. Van Chu, D. Fujiki, S. Takeuchi, M. Motomura, "Partial Search in a Frozen Network is Enough to Find a Strong Lottery Ticket," *The 41st International Conference on Machine Learning (ICML)*, Vienna, Austria, Jul. 2024. (Submitted Feb. 2, 2024. Under review.)

2. H. Otsuka, Y. Okoshi, **Á. López García-Arias**, K. Kawamura, T. Van Chu, M. Motomura, "Ramanujan Edge-Popup: Finding Strong Lottery Tickets with Ramanujan Graph Properties for Efficient DNN Inference Execution," *The 25th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI 2024)*, Taipei, Taiwan, Mar. 2024.

3. J. Yan, H. Ito, **Á. López García-Arias**, Y. Okoshi, H. Otsuka, K. Kawamura, T. Van Chu, and M. Motomura, "Multicoated and Folded Graph Neural Networks with Strong Lottery Tickets," *The Second Learning on Graphs Conference (LoG 2023)*, Online, Nov. 2023.

4. J. Suzuki, J. Yu, M. Yasunaga, **Á. López García-Arias**, Y. Okoshi, S. Kumazawa, K. Ando, K. Kawamura, T. V. Chu, and M. Motomura, "Pianissimo: A Sub-mW Class DNN Accelerator with Progressive Bit-by-Bit Datapath Architecture for Adaptive Inference at Edge," *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, Kyoto, Japan, pp. 1–2, Jun. 2023, doi:10.23919/VLSITechnologyandCir57934.2023.10185293.

5. K. Kawamura, J. Yu, D. Okonogi, S. Jimbo, G. Inoue, A. Hyodo, **Á. López García-Arias**, K. Ando, B. H. Fukushima-Kimura, R. Yasudo, T. Van Chu, and M. Motomura, "Amorphica: 4-Replica 512 Fully Connected Spin 336MHz Metamorphic Annealer with Programmable Optimization Strategy and Compressed-Spin-Transfer Multi-Chip Extension," *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA, USA, pp. 42–44, Feb. 2023, doi:10.1109/ISSCC42615.2023.10067504.

6. K. Hirose, J. Yu, K. Ando, Y. Okoshi, **Á. López García-Arias**, J. Suzuki, T. Van Chu, K. Kawamura, and M. Motomura, "Hiddenite: 4K-PE Hidden Network Inference 4D-Tensor Engine Exploiting On-Chip Model Construction Achieving 34.8-to-16.0TOPS/W for CIFAR-100 and ImageNet," *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA, USA, pp. 1–3, Feb. 2022, doi:10.1109/ISSCC42614.2022.9731668.

7. T. Alonso, M. Ruiz, **Á. López García-Arias**, G. Sutter and J. E. López de Vergara, "Submicrosecond Latency Video Compression in a Low-End FPGA-based System-on-Chip," *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, Dublin, Ireland, pp. 355–3554, Aug. 2018, doi:10.1109/FPL.2018.00067.