Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators

Brandon Reagen Paul Whatmough Robert Adolf Saketh Rama
Hyunkwang Lee Sae Kyu Lee José Miguel Hernández-Lobato
Gu-Yeon Wei David Brooks
Harvard University

ABSTRACT

The continued success of Deep Neural Networks (DNNs) in classification tasks has sparked a trend of accelerating their execution with specialized hardware. While published designs easily give an order of magnitude improvement over general-purpose hardware, few look beyond an initial implementation. This paper presents Minerva, a highly automated co-design approach across the algorithm, architecture, and circuit levels to optimize DNN hardware accelerators. Compared to an established fixed-point accelerator baseline, we show that fine-grained, heterogeneous datatype optimization reduces power by 1.5×; aggressive, inline predication and pruning of small activity values further reduces power by 2.0×; and active hardware fault detection coupled with domain-aware error mitigation eliminates an additional 2.7× through lowering SRAM voltages. Across five datasets, these optimizations provide a collective average of 8.1× power reduction over an accelerator baseline without compromising DNN model accuracy. Minerva enables highly accurate, ultra-low power DNN accelerators (in the range of tens of milliwatts), making it feasible to deploy DNNs in power-constrained IoT and mobile devices.

1. INTRODUCTION

Deep Neural Networks (DNNs) are Machine Learning (ML) methods that learn complex function approximations from input/output examples [1]. These methods have gained popularity over the last ten years thanks to empirical achievements on a wide range of tasks including speech recognition [2, 3], computer vision [4], and natural language processing [5, 6]. Applications based on DNNs can be found across a broad range of computing systems, from datacenters down to battery-powered mobile and IoT devices. The recent success of DNNs in these problem domains can be attributed to three key factors: the availability of massive datasets, access to highly parallel computational resources such as GPUs, and improvements in the algorithms used to tune DNNs to data [7].

The parameters (weights) of a DNN are fitted to data in a process called *training*, which typically runs on a high-performance CPU/GPU platform. The training process depends on the characteristics of the available data and should ideally incorporate the latest ML advances, best suited to leverage the high flexibility of software. During the *prediction* phase, trained DNN models are used to infer unknown outputs from new input data. While training is largely a one-time cost, prediction computations run repeatedly once the DNN is deployed in a production environment. Therefore, speeding up prediction and minimizing its power consumption is highly desirable, especially for applications that run on battery-powered devices with limited computational capabilities. One obvious solution is to implement highly-customized hardware accelerators for DNN prediction. This paper presents a holistic methodology to automate the design of DNN accelerators that achieve minimum power consumption while maintaining high prediction accuracy.

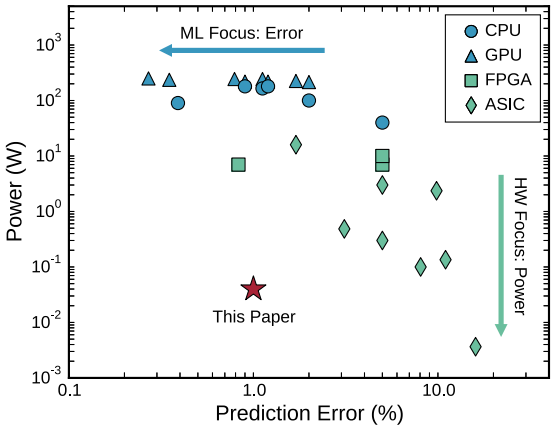


Figure 1: A literature survey of MNIST neural networks. Survey reveals the disconnect between ML research and designing DNN accelerators. References: CPUs [8, 9, 10, 11, 12], GPUs [13, 14, 15, 16, 8, 9, 11], FPGAs [17, 12], and ASICs [18, 19, 20, 21, 13, 22, 12, 23].

MNIST is a widely studied dataset used by the ML community to demonstrate state-of-the-art advances in DNN techniques. Figure 1 shows the results of a literature survey on reported values of MNIST prediction error and corresponding power consumption for different neural network implementations, as reported by the ML and hardware (HW) communities. The ML community (blue) focuses on minimizing prediction error and favors the computational power of GPUs over CPUs, with steady progress towards the top left of the plot. In contrast, solutions from the HW community (green) trend towards the bottom right of the figure, emphasizing practical efforts to constrain silicon area and reduce power consumption at the expense of non-negligible reductions in prediction accuracy.

1063-6897/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCA.2016.32

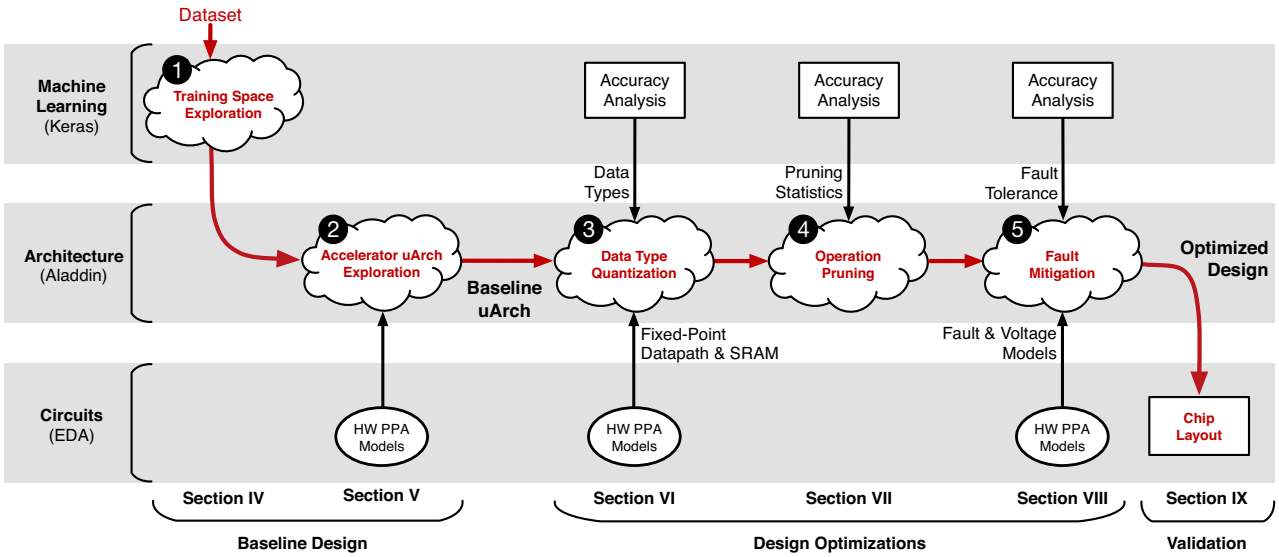


Figure 2: The five stages of Minerva. Analysis details for each stage and the tool-chain are presented in Section 3.

The divergent trends observed for published ML versus HW results reveal a notable gap for implementations that achieve competitive prediction accuracy with power budgets within the grasp of mobile and IoT platforms. Current-generation mobile devices already exploit DNN techniques across a range of applications. However, they typically off-load computation to backend servers. Solutions that can bring DNN computation to edge devices promise to improve latency, autonomy, power consumption, and security for the growing number of applications that look to use DNNs. For IoT devices in particular, it is often not possible to guarantee the availability of high-bandwidth communications, rendering off-loading impractical. To enable highly-accurate DNN predictions in mobile and IoT devices, novel power-reduction techniques are needed.

This paper presents *Minerva*: a highly automated co-design flow that combines insights and techniques across the algorithm, architecture, and circuit layers, enabling low-power accelerators for executing highly-accurate DNNs. The Minerva flow, outlined in Figure 2, first establishes a fair baseline design by extensively exploring the DNN training and accelerator microarchitectural design spaces, identifying an ideal DNN topology, set of weights, and accelerator implementation. The resulting design is competitive with the best DNN ASIC accelerators built today. Minerva then applies three cross-layer optimization steps to this baseline design including: (i) fine-grain, heterogeneous *data-type quantization*, (ii) dynamic *operation pruning*, and (iii) algorithm-aware *fault mitigation* for low-voltage SRAM operation. These techniques, both extensions of known optimizations as well as entirely new ones, result in an optimized design that demonstrates more than 8× power reduction compared to the baseline design while preserving its initial high prediction accuracy. Extensive simulations show Minerva can yield the ($\star$) in Figure 1 for MNIST. We also apply the Minerva flow to five common ML datasets and show that a continuum of designs is available: from a general-purpose DNN accelerator to a dataset specific hard-coded implementation with varying power/area trade-offs.

2. OVERVIEW

Minerva consists of five stages, as shown in Figure 2. Stages 1–2 establish a fair baseline accelerator implementation. Stage 1 generates the baseline DNN: fixing a network topology and a set of trained weights. Stage 2 selects an optimal baseline accelerator implementation. Stages 3–5 employ novel co-design optimizations to minimize power consumption over the baseline in the following ways: Stage 3 analyzes the dynamic range of all DNN signals and reduces slack in data type precision. Stage 4 exploits observed network sparsity to minimize data accesses and MAC operations. Stage 5 introduces a novel fault mitigation technique, which allows for aggressive SRAM supply voltage reduction. For each of the three optimization stages, the ML level measures the impact on prediction accuracy, the architecture level evaluates hardware resource savings, and the circuit level characterizes the hardware models and validates simulation results.

The organization of the five stages is intended to minimize the possibility of compounding prediction error degradation. Appendix A provides a review of DNNs and terminology.

Stage 1: Training Space Exploration. Minerva first establishes a fair DNN baseline that achieves prediction accuracy comparable to state-of-the-art ML results. This stage leverages the Keras software library [24] to sweep the large DNN hyperparameter space. Of the thousands of uniquely trained DNNs, Minerva selects the network topology that minimizes error with reasonable resource requirements.

Stage 2: Microarchitecture Design Space. The optimal network from Stage 1 is then fed to a second stage that thoroughly explores the accelerator design space. This process exposes hardware resource trade-offs through microarchitectural parameters (e.g., clock frequency and memory bandwidth). Minerva then uses an optimal design point as the baseline implementation to which all subsequent optimizations are applied and compared against.

Stage 3: Data Type Quantization. Minerva optimizes DNN data types with linear quantization analysis, indepen-

dently tuning the range and precision of each DNN signal at each network layer. Quantization analysis minimizes bitwidths without exceeding a strict prediction error bound. Compared to a 16 bit fixed-point baseline, data type quantization reduces power consumption by 1.5×.

Stage 4: Selective Operation Pruning. The DNN kernel mostly comprises repeated weight reads and MAC operations. Analysis of neuron activity values reveals the vast majority of operands are close to zero. Minerva identifies these neuron activities and removes them from the prediction computation such that model accuracy is not affected. Selective pruning further reduces power consumption by 2.0× on top of bitwidth quantization.

Stage 5: SRAM Fault Mitigation. By combining inherent algorithmic redundancy with low overhead fault mitigation techniques, optimization Stage 5 saves an additional 2.7× power by aggressively scaling SRAM supply voltages. Minerva employs state-of-the-art circuits to identify potential SRAM read faults and proposes new mitigation techniques based on rounding faulty weights towards zero.

Minerva's optimizations reduce power consumption by more than 8× without degrading prediction accuracy.

3. EXPERIMENTAL METHODOLOGY

The Minerva design flow (Figure 2) integrates tools at the software, architecture, and circuit levels. Software level analysis lets Minerva quickly evaluate the accuracy impact of optimization trade-offs. Architectural simulation enables rapid design space exploration (DSE) and quantification of hardware optimization benefits. Circuit tools provide accurate power-performance-area (PPA) and reliability models, as well as validation of the higher-level simulation results.

3.1 Software Level: Keras

To understand how each Minerva stage affects prediction accuracy, Minerva uses a software model built on top of the Keras [24] ML library. This GPU-accelerated code enables us to explore the large hyperparameter space in Stage 1.

To evaluate the optimizations, we augment Keras with a software model for each technique. Evaluating fixed-point types in Stage 3 was done by building a fixed-point arithmetic emulation library and wrapping native types with quantization calls. To prune insignificant activities (Stage 4), a thresholding operation is added to the activation function of each DNN layer. This function checks each activity value and zeros all activities below the threshold, removing them from the prediction computation. To study faults in DNN weights (Stage 5), we built a fault injection framework around Keras. Before making predictions, the framework uses a fault distribution, derived from SPICE simulations for low-voltage SRAMs, to randomly mutate model weights. For statistical significance, both the model and the fault injection framework are sampled 500 times, and the resulting output distribution is shown.

3.2 Architecture Level: Aladdin

Stage 2 of Minerva automates a large design space exploration of microarchitectural parameters used for accelerator design (e.g., loop level parallelism, memory bandwidth, clock frequency, etc.) in order to settle on a Pareto-optimal starting point for further optimizations. The accelerator design space that we explore is vast, exceeding several thousand points. In order to exhaustively explore this space we rely on Aladdin, a cycle-accurate design analysis tool for accelerators [25].

To model the optimizations in Stages 3-5, Aladdin was extended in the following ways: first, fixed-point data types are modeled by adding support for variable types wherein each variable and array in the C-code is tagged to specify its precision. Aladdin then interprets these annotations by mapping them to characterized PPA libraries to apply appropriate costs. Overheads such as additional comparators associated with operation pruning and SRAM fault detection were modeled by inserting code representative of the overhead into the input C-code. The benefits of operation pruning are informed by the Keras software model which tracks each elided neuronal MAC operation. This information is relayed to Aladdin and used during an added activity trace post-processing stage where each skipped operation is removed to model dynamic power savings. To model power with respect to reduced SRAM voltages, Aladdin's nominal SRAM libraries are replaced with the corresponding low-voltage libraries.

3.3 Circuit Level: EDA

In order to achieve accurate results, Aladdin requires detailed PPA hardware characterization to be fed into its models as represented by the arrows from the circuit to architecture levels in Figure 2. Aladdin PPA characterization libraries are built using PrimePower for all datapath elements needed to simulate DNN accelerators with commercial standard cell libraries in 40nm CMOS. For SRAM modeling, we use SPICE simulations with foundry-supplied memory compilers. Included in the PPA characterization are the fixed-point types (for Stage 3) and reduced voltage SRAMs (for Stage 5). Fault distributions corresponding to each SRAM voltage reduction step are modeled using Monte Carlo SPICE simulation with 10,000 samples, similar to the methodology of [28].

We validate PPA estimates for Minerva's optimized design (Stage 5 output) by comparing against a fully place-and-routed implementation (using Cadence SoC Encounter) of hand-written RTL informed by the parameters determined by Aladdin. The power dissipation of the accelerator is dominated by datapath and memory elements which Aladdin models accurately using the detailed PPA libraries. Our validation results in Table 2 show that Aladdin estimates are within 12% of a fully place-and-routed design.

3.4 Datasets Evaluated

We validate the optimizations performed by our framework with five classification datasets commonly used by the ML community: 1) **MNIST**: images of hand-written digits from 0 to 9 [29]; 2) **Forest**: cartographic observations for classifying the forest cover type [26]; 3) **Reuters**-21578 (Distribution 1.0): news articles for text categorization [30, 27]; 4) **WebKB**: web pages from different universities [31]; 5) **20NG**: newsgroup posts for text classification and text clustering [32].

MNIST is the dataset most commonly used and reported

Table 1: Application Datasets, Hyperparameters, and Prediction Error

Dataset				Hyperparameters				Error (%)		
Name	**Domain**	**Inputs**	**Outputs**	**Topology**	**Params**	**L1**	**L2**	**Literature**	**Minerva**	**σ**
MNIST	Handwritten Digits	784	10	256×256×256	334 K	10^{-5}	10^{-5}	0.21 [8]	1.4	0.14
Forest	Cartography Data	54	8	128×512×128	139 K	0	10^{-2}	29.42 [26]	28.87	2.7
Reuters	News Articles	2837	52	128×64×512	430 K	10^{-5}	10^{-3}	13.00 [27]	5.30	1.0
WebKB	Web Crawl	3418	4	128×32×128	446 K	10^{-6}	10^{-2}	14.18 [27]	9.89	0.71
20NG	Newsgroup Posts	21979	20	64×64×256	1.43 M	10^{-4}	1	17.16 [27]	17.8	1.4

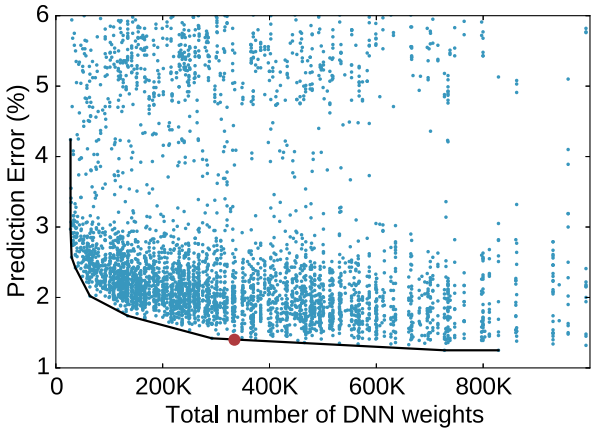


Figure 3: Each point is a uniquely trained MNIST DNN. The black line indicates the Pareto frontier, minimizing DNN weights and prediction error. The red dot indicates the chosen network. Parameters swept: 3–5 hidden layers, 32–512 nodes per layer, L1/L2 weight parameters from 0–10^{-6}.

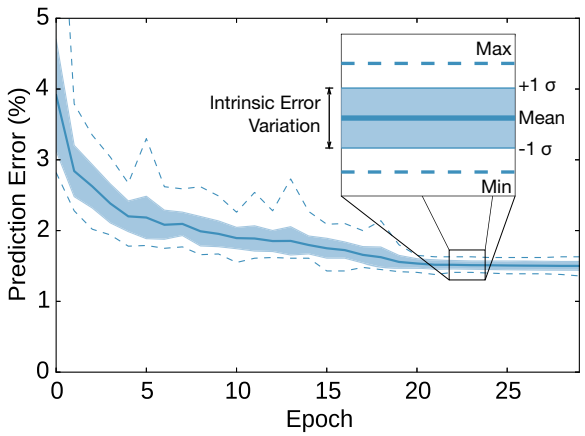


Figure 4: By training the same network using many random initial conditions, we can measure the intrinsic error variation in its converged state. All our optimizations are designed to have an accuracy degradation below this threshold, so their effects are indistinguishable from noise.

by the ML community when evaluating the performance of DNN methods. Hence, we use MNIST as the main reference to evaluate each of the optimization operations performed by Minerva. To demonstrate optimization generality, Section 9 presents results for the remaining four datasets.

4. TRAINING SPACE EXPLORATION

Stage 1 of Minerva explores the DNN training space, identifying hyperparameters that provide optimal predictive capabilities. There is currently no standard process for setting/tuning DNN hyperparameters, which remains somewhere between a black art and cutting-edge research [33]. To explore this configuration space, Minerva considers the number of hidden layers, number of nodes per layer, and L1/L2 weight regularization penalties. Minerva then trains a DNN for each point and selects the one with the lowest prediction error. The weights for the trained network are then fixed and used for all subsequent experiments.

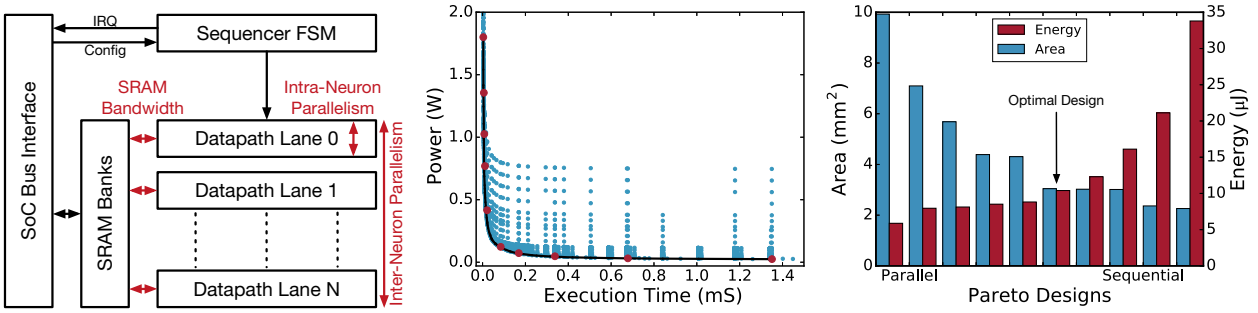
4.1 Hyperparameter Space Exploration

Figure 3 plots the resulting prediction errors as a function of the number of DNN weights for MNIST. Larger networks often have smaller predictive error. However, beyond a certain point, the resources required to store the weights dominates the marginal increase in prediction error. For example, a DNN with three hidden layers and 256 nodes per layer requires roughly 1.3MB of storage, while 512 nodes per layer requires 3.6MB (assuming 4B weights). This 2.8× storage increase only improves absolute model accuracy 0.05%. The red dot in Figure 3 corresponds to an optimal network that balances memory requirements versus incremental accuracy improvements. The resulting network has a prediction error of 1.4%. Selected DNN hyperparameter settings for MNIST and other datasets can be found in Table 1.

4.2 Bounding Prediction Error

Minerva modifies the calculations performed by the original DNN in order to optimize power and chip area for the resulting hardware accelerator. This comes with the possibility of a small increase in prediction error. To maintain DNN accuracy, we constrain the cumulative error increase from all Minerva optimizations to be smaller than the intrinsic variation of the training process. This interval is not deterministic, but sensitive to randomness from both the initialization of the pre-training weights and the stochastic gradient descent (SGD) algorithm.

Figure 4 shows the average prediction error and a corresponding confidence interval, denoted by ±1 standard deviations, obtained across 50 unique training runs. We use these confidence intervals to determine the acceptable upper bound on prediction error increase due to Minerva optimizations. For MNIST, the interval is ±0.14%. Table 1 enumerates the intervals for the other datasets.



(a) Accelerator architecture (b) DSE of accelerator implementations (c) Energy and Area of Pareto designs

Figure 5: Shown above is a high-level description of our accelerator architecture, results from a DSE over the accelerator implementation space, and energy and area analysis of resulting Pareto frontier designs.

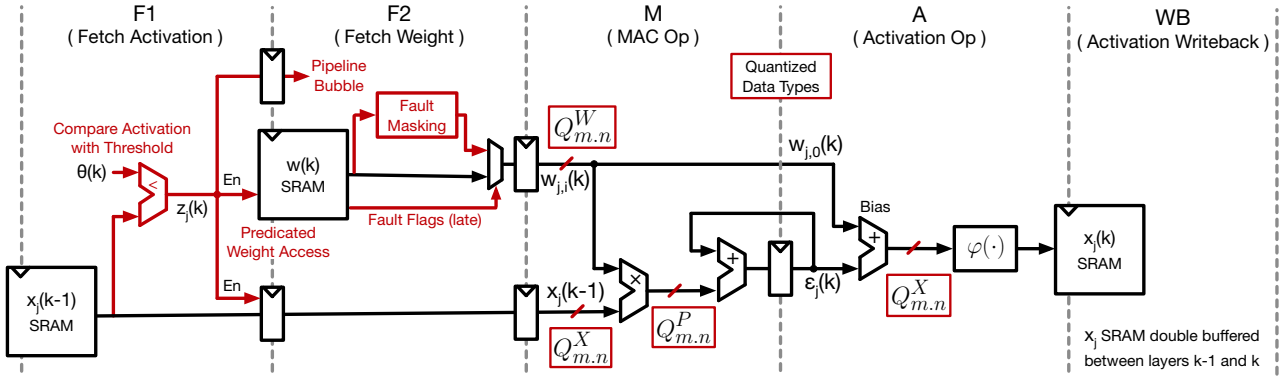


Figure 6: The microarchitecture of a single datapath lane. Modifications needed for optimizations are shown in red.

5. ACCELERATOR DESIGN SPACE

Stage 2 of Minerva takes the DNN topology from Stage 1 and searches the microarchitectural design space for a superior DNN accelerator. This exploration entails generating and evaluating thousands of unique implementations using Aladdin and ultimately yields a power-performance Pareto frontier. We select a baseline design from this frontier that balances area and energy. We then apply all remaining optimizations to this optimal baseline design.

The high-level DNN accelerator architecture is shown in Figure 5a, representing the machine specified by the description fed to Aladdin. The accelerator consists of memories for input vectors, weights, and activities as well as multiple datapath lanes that perform neuron computations. Figure 6 shows the layout of a single datapath lane, consisting of two operand fetch stages (**F1** and **F2**), a MAC stage (**M**), a linear rectifier activation function unit (**A**), and an activation writeback (**WB**) stage. The parts in black are optimized by Aladdin to consider different combinations of intra-neuron parallelism (i.e., parallelism and pipelining of per-neuron calculations), internal SRAM bandwidth for weights and activities (**F1** and **F2**), and the number of parallel MAC operations (**M**). These features, in addition to inter-neuron parallelism (i.e., how many neurons are computed in parallel), collectively describe a single design point in the design space shown in Figure 5b. The red wires and boxes in Figure 6 denote additional logic needed to accommodate the optimizations described in this paper.

Red dots in Figure 5b correspond to design points along the power-performance Pareto frontier. The area and energy consumed by each of these Pareto design points is further shown in Figure 5c. The DNN kernel is embarrassingly parallel within a single layer, and the bottleneck to performance- and energy-scaling quickly becomes memory bandwidth. In order to supply bandwidth, the SRAMs must be heavily partitioned into smaller memories, but once the minimum SRAM design granularity is reached, additional partitioning becomes wasteful as there is more total capacity than data to store. This scaling effect levies a heavy area penalty against excessively parallel designs and provides little significant energy improvement, as seen in the most parallel designs in Figure 5c (left side). The baseline, indicated as the Optimal Design in Figure 5c, is a balance between the steep area increase from excessive SRAM partitioning versus the energy reduction of parallel hardware. Under these constraints, the chosen design maximizes performance and minimizes power.

6. DATA TYPE QUANTIZATION

Stage 3 of Minerva aggressively optimizes DNN bitwidths. The use of optimized data types is a key advantage that allows accelerators to achieve better computational efficiency than general-purpose programmable machines. In a DNN accelerator, weight reads and MAC operations account for the majority of power consumption. Fine-grained per-type, per-layer optimizations significantly reduce power and resource demands.

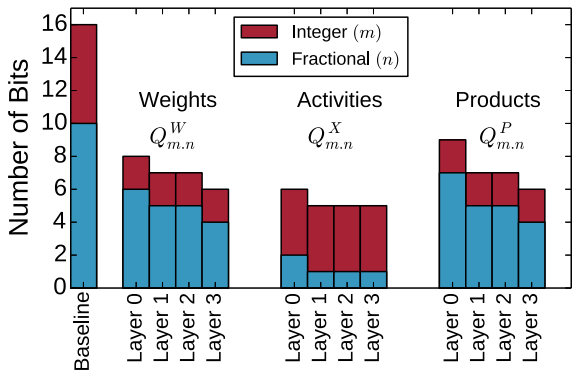


Figure 7: Minimum precision requirements for each datapath signal while preserving model accuracy within our established error bound.

6.1 Fixed-Point Datapath Design

Figure 6 identifies three distinct signals that we independently quantize: $x_j(k-1)$, neuron activity (SRAM); $w_{j,i}(k)$, the network's weights (SRAM); and the product $w_{j,i}(k) \cdot x_j(k-1)$, an intermediate value that determines the multiplier width. The notation $Q_{m.n}$ describes a fixed-point type of m integer bits (including the sign bit) and n fractional bits (i.e., bits to the right of the binary point). For each independent signal, we consider a separate fixed-point representation, named accordingly: $Q^X_{m.n}$, $Q^W_{m.n}$, and $Q^P_{m.n}$.

6.2 Optimized Fixed-Point Bitwidths

The conventional approach to using fixed-point types in DNNs is to select a single, convenient bitwidth (typically 16 bits [34, 19, 21]) and verify the prediction error is within some margin of the original floating-point implementation. To improve upon this approach, Minerva considers all possible combinations and granularities of m and n for each signal within each network layer, independently.

Figure 7 presents the resulting number of bits $(m+n)$ from fine-grained bitwidth tuning for each signal on a per-layer basis for MNIST. The required number of bits is computed by iteratively reducing the width, starting from a baseline type of $Q_{6.10}$ for all signals (shown in Figure 7 as Baseline), and running the Keras SW models to measure prediction error. The minimum number of bits required is set to be the point at which reducing the precision by 1 bit (in either the integer or fraction) exceeds MNIST's error confidence interval of $\pm 0.14\%$.

The datapath lanes process the DNN graph sequentially, in a time multiplexed fashion. Hence, even though per-layer optimization provides the potential for further width reductions, all datapath types are set to the largest per-type requirement (e.g., all weights are 8 bits). While this may seem wasteful, it is in fact the optimal design decision. With respect to weight SRAM word sizes, the savings from removing one to two additional bits saves 11% power and 15% area but requires a larger number of unique SRAMs. Instantiating two different word sized SRAMs tailored to each layer, as opposed to having a single 8 bit word SRAM, results in a 19% increase in area. In a similar vein, it is difficult to modify the MAC stage logic to allow such fine-grained reconfiguration without incurring significant overheads.

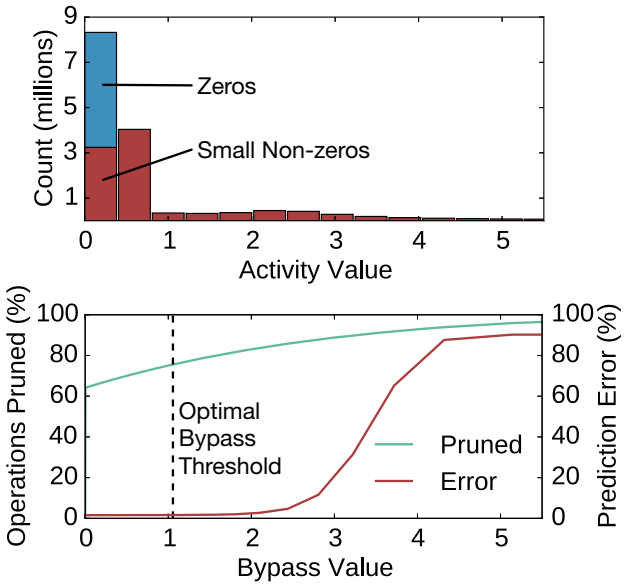


Figure 8: Analysis of neuron activities and sensitivity of prediction error to pruning. The vertical line corresponds to the point our error bound is exceeded.

Using per-type values of $Q^W_{2.6}$, $Q^X_{2.4}$, and $Q^P_{2.7}$ provides a power saving of 1.6× for MNIST and an average of 1.5× across all datasets compared to a traditional, global fixed-point type of $Q_{6.10}$ as used in [21, 34].

7. SELECTIVE OPERATION PRUNING

Stage 4 of Minerva reduces the number of edges that must be processed in the dataflow graph. Using empirical analysis of neuron activity, we show that by eliminating operations involving small activity values, the number of weight fetch and MAC operations can be drastically reduced without impacting prediction accuracy.

7.1 Analysis of DNN Activity

Figure 8 shows a histogram of neuron activities over all of the test vectors for the MNIST dataset. Immediately visible is the overwhelming number of zero- and near-zero values. Mathematically, zero-valued activities are insignificant: guaranteed to have no impact on prediction error. Skipping operations involving these values can save power both from avoided multiplications and SRAM accesses. Intuitively, it also stands to reason that many near-zero values have no measurable impact on prediction error. Thus, if we loosen the definition of "insignificant" to encompass this large population small values, we stand to gain additional power savings with no accuracy penalty.

Stage 4 quantifies the relationship between activity value and overall error. Our software model elides operations involving any neuron activity below a given threshold and evaluates the network as if those activities were zero. The resulting curves in Figure 8 show that even if we remove all activities with magnitude less than 1.05, the overall prediction error is unaffected. The green pruned-operation curve tracks the cumulative number of activities smaller than a given value. A threshold value of 1.05 is larger than ap-

proximately 75% of activities, meaning we can safely prune three out of every four MAC and SRAM read operations.

While this may sound surprising, there is some intuition behind this phenomenon. Rectifier activation functions eliminate negative products, so we should expect approximately half of the internal activity values to be zero from this effect alone. This is largely responsible for the high number of operations pruned even with a threshold close to zero (the y-intercept of the pruned-operations curve in Figure 8). Additionally, DNNs based on rectifier activation functions are known to grow increasingly sparse with deeper topologies as a result of successive decimation [35]. The combination of these effects results in a remarkable number of insignificant operations. In addition to quantization, selective pruning further reduces power 1.9× for MNIST and an average of 2.0× over all datasets.

7.2 Predicating on Insignificant Operations

The sparsity patterns in the activities is a dynamic function of the input data vector. As such, it is not possible to determine *a priori* which operations can be pruned. Therefore, it is necessary to inspect the neuron activities as they are read from SRAM and dynamically predicate MAC operations for small values. To achieve this, the datapath lane (Figure 6) splits the fetch operations over two stages. **F1** reads the current neuron activation ($x_j(k-1)$) from SRAM and compares it with the per-layer threshold ($\theta(k)$) to generate a flag bit, $z_j(k)$, which indicates if the operation can be skipped. Subsequently, **F2** uses the $z_j(k)$ flag to predicate the SRAM weight read ($W_{ij}(k)$) and stall the following MAC (**M**), using clock-gating to reduce the dynamic power of the datapath lane. The hardware overhead for splitting the fetch operations, an additional pipeline stage and a comparator, are negligible.

8. SRAM FAULT MITIGATION

The final stage of Minerva optimizes SRAM power by reducing the supply voltage. While the power savings are significant, lowering SRAM voltages causes an exponential increase in the bitcell fault rate. We only consider reducing SRAM voltages as they account for the vast majority of the remaining accelerator power. To enable robust SRAM scaling, we present novel co-designed fault mitigation techniques.

8.1 SRAM Supply Voltage Scaling

Scaling SRAM voltages is challenging due to the low noise margin circuits used in SRAMs, including ratioed logic in the bitcell, domino logic in the bitline operation, and various self-timed circuits. Figure 9 shows SPICE simulation results for SRAM power consumption and corresponding bitcell fault rates when scaling the supply voltage of a 16KB SRAM array in 40nm CMOS. The fault rate curve indicates the probability of a single bit error in the SRAM array. This data was generated using Monte Carlo simulation with 10,000 samples at each voltage step to model process variation effects, similar to the analysis in [28], but with a modern technology.

SRAM power decreases quadratically as voltage scales down while the probability of any bitcell failing increases

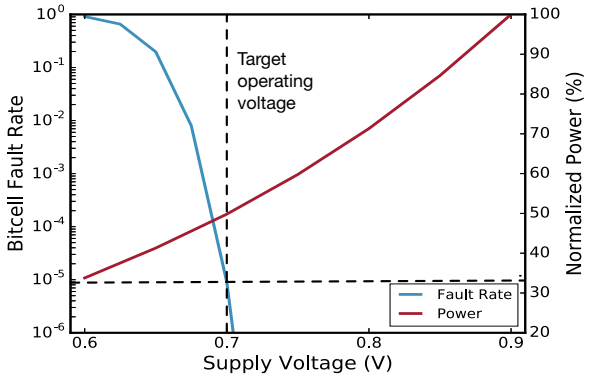


Figure 9: SRAM supply voltage scaling trends for fault rate and power dissipation.

exponentially. If we target an operating voltage of 0.7V, we could approximately halve the power consumption of the SRAM and, according to Figure 9, operate with seemingly negligible fault probability. However, many practical effects make this fault rate far higher than the ideal case, including process variation, voltage noise, temperature, and aging effects. As a result, it is important to be able to tolerate fault rates beyond this point.

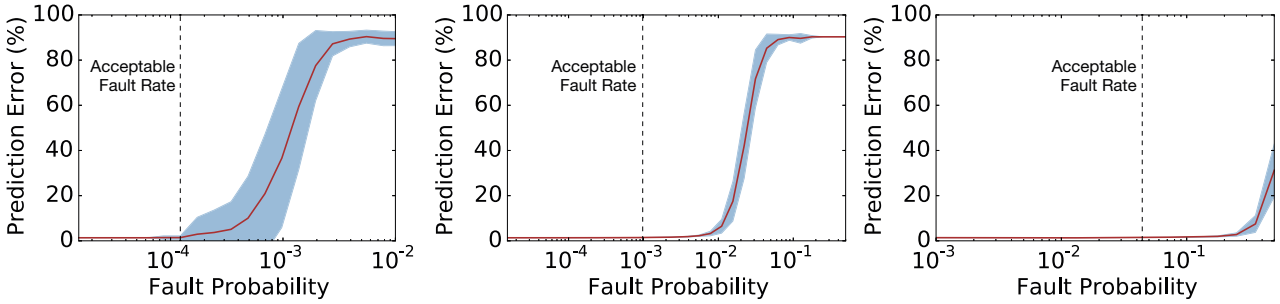
8.2 SRAM Fault Detection

A number of techniques have been described to detect faults arising from voltage scaling, including parity bits [36], Razor double-sampling [37, 38], Razor transition-detection [39], and various canary circuits [40]. A single parity bit enables fault detection simply through inspecting the read SRAM data. In contrast, Razor and canary circuits provide fault detection by monitoring delays in the circuits. None of these solutions correct suspected-faulty SRAM data but instead indicate when a fault may have occurred. The overheads of these fault detection methods vary relative to the protection provided. Parity can only detect an odd number of errors and provides no information as to which bit(s) in the word the fault(s) may have affected. In the case of DNN accelerators, the word sizes are relatively small (Section 6); the overhead of a single parity bit is approximately 11% area and 9% power. Thus, anything more than a single bit is prohibitive.

In this work, we instead employ the Razor double-sampling method for fault detection. Unlike parity, Razor monitors each column of the array individually, which provides two key advantages: (1) there is no limit on the number of faults that can be detected, and (2) information is available on which bit(s) are affected. The relative overheads for Razor SRAM fault detection are modeled in our single-port weight arrays as 12.8% and 0.3% for power and area respectively [38]. This overhead is modeled in the SRAM characterization used by Aladdin.

8.3 Mitigating Faults in DNNs

Razor SRAMs provide fault detection, not correction. Correction mechanisms require additional consideration, and are built on top of detection techniques. In fault-tolerant CPU applications a typical correction mechanism is to recompute using checkpoint and replay [39, 41, 42], a capability specific to the sophisticated control-plane logic of



(a) No Protection (b) Word Masking (c) Bit Masking

Figure 10: The sensitivity of weight matrices to proposed fault mitigation techniques. The figures show the impact faults have on accuracy when mitigating faults with no protection (a), word masking (b), and bit masking (c). The vertical dashed lines correspond to the maximum tolerable fault rate that satisfies our 0.14% absolute increase to prediction error bound.

a CPU employing pipeline speculation. In an accelerator, implementing replay mechanisms incur significant, and in this case unnecessary, overheads. To avoid this, we use a lightweight approach which does not reproduce the original data but instead attempts to mitigate the impact of intermittent bit-flips to the DNN's model accuracy. This is reminiscent of an approach studied for fault-tolerant DSP accelerators [43, 44].

We extend the Keras DNN model to study the impact of SRAM faults incurred from low-voltage operation on prediction accuracy. Faults are modeled as random bit-flips in the weight matrix (**w**). Figure 10 shows how these faults result in degraded prediction accuracy as an exponential function of fault probability. Figure 10a shows weight fault tolerance without any protection mechanism; here, even at relatively small fault probabilities ($< 10^{-4}$), the prediction error exceeds our established error confidence interval. Once the fault rate goes above 10^{-3}, the model is completely random with 90% prediction error. This result intuitively follows from the observations regarding the sparsity of the network activity in the previous section—if the majority of neuron outputs are zero, a fault which flips a high-order bit can have a catastrophic affect on the classification.

To prevent prediction accuracy degradation, we combine Razor fault detection with mechanisms to *mask* data towards zero when faults are detected at the circuit level. Masking can be performed at two different granularities: **word masking**: when a fault is detected, all the bits of the word are set to zero; and **bit masking**: any bits that experience faults are replaced with the sign bit. This achieves a similar effect to rounding the bit position towards zero. Figure 11 gives a simple illustration of word masking and bit masking fault mitigation.

Figure 10b illustrates the benefits of word masking. Masking the whole word to zero on a fault is equivalent to removing an edge from the DNN graph (Figure 14), preventing fault propagation. However, there is an associated second-order effect of neuron activity being inhibited by word masking when a fault coincides with a large activity in the preceding layer. Compared to no protection, word masking is able to tolerate an order of magnitude more bitcell faults.

By masking only the affected bit(s), rather than the whole word, bit masking (Figure 10c) is the strongest solution.

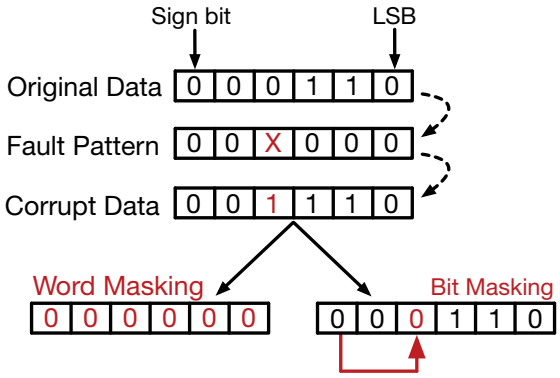


Figure 11: Illustration of word masking (faulty weights set to zero) and bit masking (faulty bits set to sign bit) fault mitigation techniques.

Bit masking limits error propagation through the fully-connected network of a DNN, and its bias towards zero complements the natural sparsity of ReLU-activated networks. The combination of Razor fault detection and bit masking fault mitigation allows the weight SRAMs to tolerate 44× more faults than word masking. With bit masking, 4.4% of SRAM bitcells can fault without loss of prediction error. This level of fault tolerance allows us to confidently drop our SRAM supply voltage by more than 200mV, reducing overall power consumption of the accelerator by an additional 2.7× on average and 2.5× for MNIST.

8.4 HW Modifications for Mitigation

Bit masking requires modifications to the weight fetch stage (**F2**) of the datapath lane, as illustrated in Figure 6. The Razor SRAM provides flags to indicate a potential fault for each column of the array, which are used to mask bits in the data, replacing them with the sign bit using a row of two-input multiplexers. The flag bits are prone to metastability, because (by definition) timing will not be met in the case of a fault. However, metastability is not a big concern as the error flags only fan-out into datapath logic, as opposed to control logic, which can be problematic. The flags are also somewhat late to arrive, which is accommodated by placing the multiplexers at the end of the **F2** stage (Figure 6).

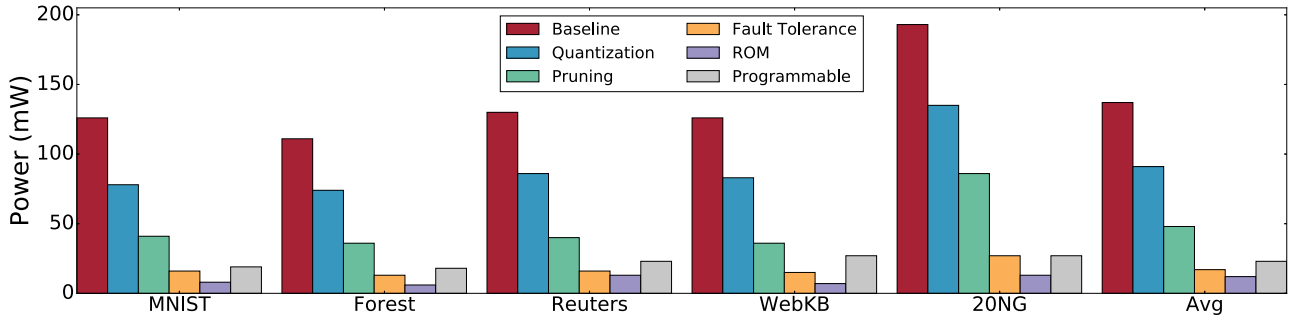


Figure 12: Results from applying the Minerva design flow to five application datasets to investigate generality. Each successive optimization insures compounding error does not exceed the established threshold. The bars labeled ROM show the benefit of full customization, wherein weights are stored in ROMs rather than SRAMs. Those labeled programmable show the overhead each dataset incurs when an accelerator is designed to handle all 5 datasets.

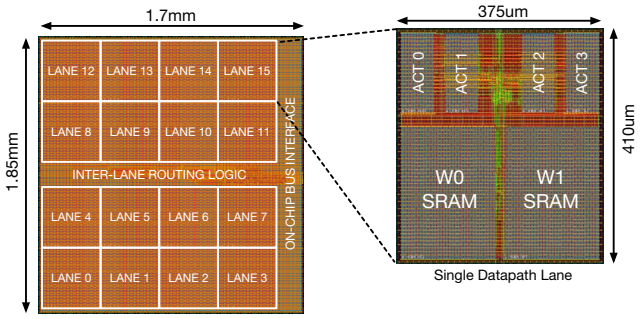


Figure 13: Layout of the optimized Minerva accelerator.

Table 2: Validation of Minerva compared to a chip layout.

Metrics	Minerva	Layout
Clock Freq (MHz)	250	250
Performance (Pred/s)	11,820	11,820
Energy (μJ/Pred)	1.3	1.5
Power (mW)	16.3	18.5
Weights (mm^2)	1.3	1.3
Activities (mm^2)	0.53	0.54
Datapath (mm^2)	0.02	0.03

9. DISCUSSION

9.1 Sensitivity Analysis of Optimizations

To this point, we focused only on MNIST for clarity in the presentation of the Minerva design flow and optimizations. In this section we demonstrate the generality of Minerva by considering four additional datasets; each is run through the Minerva design flow described in Sections 4–8. Figure 12 presents the resulting accelerator designs, indicating the power reduction of each with respect to each optimization stage. On average, Minerva generated DNN accelerators dissipate 8.1× less power, operating at the 10s rather than 100s of mWs.

While the overall power reduction for each dataset is similar, the relative benefits from each optimization differ. For example, MNIST benefits more from quantization (1.6×) than WebKB (1.5×) while WebKB is far more amenable to operation pruning—2.3× versus MNIST's 1.9×. This is caused by differences in the application domain and input vectors.

9.2 Balancing Specialization and Flexibility

With Minerva, we are able to consider the trade-offs of building a fully-optimized accelerator capable of performing only one function against building a programmable accelerator able to run any of the datasets using our assumed network topologies. In Figure 12, the bars labeled ROM indicate the power saved when the weights are stored using ROM instead of SRAM. This additional optimization further reduces power on average by 1.9×.

We also consider building a configurable accelerator. To accommodate all datasets, the configurable accelerator's parameters are set to the maximum of each individually optimized design (i.e., it supports 20NG's 21979 input size and up to 256 × 512 × 512 nodes per layer). This design consumes an average of 24mW across all datasets and uses 1.4× and 2.6× more power than dataset specific SRAM and ROM implementations respectively. The largest overhead introduced by the configurable design relative to building individual accelerators for each dataset is due to memory leakage. With selective pruning, the dynamic power remains minimal. The inefficiencies of generality, compared to an accelerator tuned to each individual dataset, are reflected in the bars labeled Programmable in Figure 12.

9.3 Post-Optimization Implementation

The Aladdin analysis used throughout Minerva provides simulated estimates for fully-implemented accelerators. Therefore, to validate the PPA results and gain practical insight, we implemented the Minerva-optimized DNN accelerator in RTL and place-and-routed it using a 40nm CMOS technology. The resulting layout is shown in Figure 13.

Table 2 summarizes the implementation results. Compared to Aladdin's estimates, we find that our simulation results are within 12% power. The performance difference is negligible. Considering all pieces modeled by Aladdin, the simulation results are well within the previously published 7% PPA error. As for the entire design, the true area is larger, due to a number of things that are not modeled by Aladdin, namely the bus interface logic. Despite the area mismatch,

average power is not proportionally increased due to relatively low bus activity (i.e., all of the weights are stored locally rather than continuously streamed over the bus) and dissipates negligible leakage power. This design is the ($\star$) in Figure 1. We intend to fabricate a DNN accelerator using the Minerva design flow as part of a practical demonstration of the potential for low-power DNN accelerators.

10. RELATED WORK

Early investigations into data type precision focused purely on the relationship between reduced precision and prediction error [45, 46]. More recently, with the revival of DNNs, fixed-point bitwidth reduction has been revisited as a method to reduce power and area for resource-constrained computing [47, 48, 13, 49, 21, 17]. Others have proposed reducing computation and data movement costs with in-situ processing using memristors [50].

Other works have begun to exploit network sparsity to save power. Weight sparsity was considered in [51], where the authors co-design training and prediction to favor weight sparsity then statically prune the network to remove unimportant edges. In [52], the authors focus on compressing DNN weights, but also propose skipping activities of zero value, eliding up to 70% of the neuron computations. The reconfigurable CNN accelerator in [53] clock gates processing elements when activities are zero. In addition to pruning neurons with zero activities, Minerva also prunes small activities. This enables us to eliminate up to 95% of all neuron computations in the best case with no accuracy loss.

In [54], authors present and evaluate techniques for removing network weights without compromising model accuracy. DNN fault tolerance has also been discussed in [34, 55]; these papers exploit the fact that it is possible to re-train a DNN to work around a permanent hardware fault. In [34], the authors found that small sigmoid neural networks were able to tolerate 12-20 hardware defects without significantly increasing prediction error after re-training with the static fault present. This approach was demonstrated to work well (at least on small networks), but requires the exact fault to be known, and requires each ASIC to be individually re-trained, which is not scalable. Our approach mitigates arbitrary fault patterns, does not require re-training, and is able to tolerate several orders of magnitude more faults.

While DNNs are arguably the most generally applicable deep learning model, there are others that outperform them in specific domains. Convolutional Neural Networks (CNNs) [56], for example, demonstrate state-of-the-art prediction error in many image recognition tasks. CNNs have some similarities to DNNs, but reuse a smaller number of weights across the image. Although this tends to stress computation rather than weight storage, we believe the Minerva design flow and optimizations should readily extend to CNNs. Many of the features of DNNs we exploit, such as neuron output sparsity, hold true for CNNs, and so we anticipate similar gains to those described here for DNNs. Spiking Neural Networks (SNNs) [57, 58, 59, 60] are popular in ASIC designs, largely due to their compact mapping onto fully-parallel hardware. However, the prediction error of SNNs is not competitive for the datasets considered here.

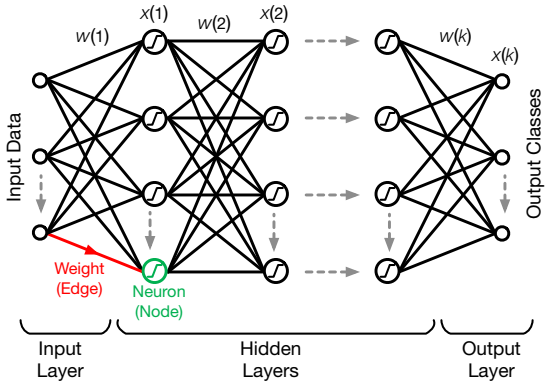


Figure 14: The operation of a DNN during the prediction phase, represented as a directed acyclic graph.

11. CONCLUSION

In this paper we have considered the problem of designing and building optimized hardware accelerators for deep neural networks that achieve minimal power consumption while maintaining high prediction accuracy. Minerva is a holistic, highly automated co-design flow that combines insights and techniques across the algorithm, architecture, and circuit levels, enabling low-power accelerators for highly accurate DNN prediction. We present three accelerator optimizations that substantially improve the power efficiency of DNN hardware accelerators. By aggressively optimizing data types, selectively pruning operations, and reducing SRAM voltages safely with novel fault mitigation techniques, Minerva is able to reduce the overall power consumption across five diverse ML datasets by an average of 8.1$\times$ without impacting prediction error. Minerva makes it possible to deploy DNNs as a solution in power-constrained mobile environments.

APPENDIX

A. DEEP NEURAL NETWORKS

Neural networks are a biologically inspired machine learning method for learning complex functions which apply multiple nonlinear transformations to an input vector to obtain a corresponding output vector. Deep neural networks (DNNs) are defined as networks with one or more hidden layers [61, 62]. The nonlinear transformations are performed by consecutive layers of fully connected artificial neurons. The first layer is called the input layer, and it has a neuron for each component in the input vector (e.g., each pixel in an image). The last layer is called the output layer and contains one neuron for each component in the output vector (e.g., the class probabilities for the object contained in the image). Between input and output layers, there are additional layers of hidden neurons. The strength of the connection between neurons is determined by a collection of weights whose values are tuned to minimize the prediction error of the network on some training data. Figure 14 depicts a DNN, represented as a weighted directed acyclic graph. Following the biological metaphor, edges and nodes represent synapses and neurons respectively.

Each neuron's output is obtained by computing a linear combination of the outputs of the neurons in the previous layer and then applying a nonlinear function, where the first layer is fed by the input vector. Mathematically, the output $x_j(k)$ of the j^{th} neuron in the k^{th} layer is

$$\varepsilon_j(k) = \sum_i w_{j,i}(k) \cdot x_i(k-1), \quad (1)$$

$$x_j(k) = \varphi(\varepsilon_j(k)). \quad (2)$$

Each weight $w_{j,i}(k) \in \mathbb{R}$ represents the connection strength between the i^{th} neuron in layer $k-1$ and the j^{th} neuron in layer k. The nonlinear function φ allows DNNs to become universal approximators. While many different nonlinear functions have been proposed, recent research favors the rectifier because of its simplicity and superior empirical performance [63].

A neural network is trained by iteratively adjusting weights to minimize a loss function over labeled data. Training is often performed using stochastic gradient descent (SGD), and the loss function is usually a combination of the prediction error and regularization terms [64]. This process requires hyperparameter values related to the network topology (e.g., number of layers and neurons per layer) and the configuration of the SGD procedure (e.g., regularization parameters). These hyperparameter values are often tuned by selecting, amongst a grid of candidates, values that minimize the prediction error of the corresponding trained DNN. This can then be used to make predictions on new inputs for which the corresponding outputs are not available.

Acknowledgments

This work was partially supported by C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and DARPA under Contract #: HR0011-13-C-0022. J.M.H.L. acknowledges support from the Rafael del Pino Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[3] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," *arXiv:1412.5567 [cs.CL]*, 2014.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.

[5] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*, pp. 160–167, ACM, 2008.

[6] I. Sutskever, O. Vinyals, and Q. V. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27*, pp. 3104–3112, 2014.

[7] M. Ranzato, G. E. Hinton, and Y. LeCun, "Guest editorial: Deep learning," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 1–2, 2015.

[8] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine learning*, 2013.

[9] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in *Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 317–324, Feb 2010.

[10] C. Poultney, S. Chopra, and Y. Lecun, "Efficient learning of sparse representations with an energy-based model," in *Advances in Neural Information Processing Systems (NIPS 2006)*, MIT Press, 2006.

[11] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pp. 27–40, ACM, 2015.

[12] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems.," in *ISCAS*, pp. 257–260, IEEE, 2010.

[13] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 609–622, Dec 2014.

[14] D. Ciresan, U. Meier, L. Gambardella, and J. Schmidhuber, "Convolutional neural network committees for handwritten character classification," in *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pp. 1135–1139, Sept 2011.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machachine Learning Research*, vol. 15, pp. 1929–1958, Jan. 2014.

[16] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, pp. 3207–3220, Dec. 2010.

[17] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of The 32nd International Conference on Machine Learning*, pp. 1737–1746, 2015.

[18] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 640m pixel/s 3.65mw sparse event-driven neuromorphic object recognition processor with on-chip learning," in *Symposium on VLSI Circuits*, pp. C50–C51, June 2015.

[19] J. Kung, D. Kim, and S. Mukhopadhyay, "A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses," in *ISPLED*, 2015.

[20] J. Arthur, P. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. Esser, N. Imam, W. Risk, D. Rubin, R. Manohar, and D. Modha, "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," in *The International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, June 2012.

[21] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ASPLOS*, pp. 269–284, ACM, 2014.

[22] S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla, S. Chandra, N. B. S. Carpin, T. Zimmerman, F. Zee, R. Alvarez-Icaza, J. A. Kusnitz, T. M. Wong, W. P. Risk, E. McQuinn, T. K. Nayak, R. Singh, and D. S. Modha, "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *The International Joint Conference on Neural Networks (IJCNN)*, 2013.

[23] E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S.-C. Liu, and S. Furber, "Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker," in *International Joint Conference on Neural Networks*, pp. 1–8, IEEE, 2015.

[24] "Keras: Theano-based deep learning library." http://keras.io, 2015.

[25] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proceeding of the 41st*

Annual International Symposium on Computer Architecuture, ISCA '14, (Piscataway, NJ, USA), pp. 97–108, IEEE Press, 2014.

[26] J. A. Blackard, *Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types*. PhD thesis, Colorado State University, Fort Collins, CO, USA, 1998. AAI9921979.

[27] A. Cardoso-Cachopo, *Improving Methods for Single-label Text Categorization*. PhD thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

[28] A. Agarwal, B. Paul, S. Mukhopadhyay, and K. Roy, "Process variation in embedded memories: failure analysis and variation aware architecture," *IEEE Journal of Solid-State Circuits*, vol. 40, pp. 1804–1814, Sept 2005.

[29] Y. Lecun and C. Cortes, "The MNIST database of handwritten digits." http://yann.lecun.com/exdb/mnist/.

[30] D. D. Lewis, "Reuters-21578 text categorization collection data set." https://archive.ics.uci.edu/ml/datasets/Reuters-21578+Text+Categorization+Collection.

[31] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery, "Learning to extract symbolic knowledge from the world wide web," in *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pp. 509–516, 1998.

[32] T. Joachims, "A probabilistic analysis of the rocchio algorithm with tfidf for text categorization," in *ICML*, pp. 143–151, 1997.

[33] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Neural Information Processing Systems*, 2012.

[34] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *ISCA*, pp. 356–367, June 2012.

[35] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[36] S. Jahinuzzaman, J. Shah, D. Rennie, and M. Sachdev, "Design and analysis of a 5.3-pj 64-kb gated ground sram with multiword ecc," *IEEE Journal of Solid-State Circuits*, vol. 44, pp. 2543–2553, Sept 2009.

[37] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A self-tuning dvs processor using delay-error detection and correction," *IEEE Journal of Solid-State Circuits*, vol. 41, pp. 792–804, April 2006.

[38] J. Kulkarni, C. Tokunaga, P. Aseron, T. Nguyen, C. Augustine, J. Tschanz, and V. De, "4.7 a 409gops/w adaptive and resilient domino register file in 22nm tri-gate cmos featuring in-situ timing margin and error detection for tolerance to within-die variation, voltage droop, temperature and aging," in *ISSCC*, pp. 1–3, Feb 2015.

[39] D. Bull, S. Das, K. Shivashankar, G. S. Dasika, K. Flautner, and D. Blaauw, "A power-efficient 32 bit arm processor using timing-error detection and correction for transient-error tolerance and adaptation to pvt variation," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 18–31, 2011.

[40] A. Raychowdhury, B. Geuskens, K. Bowman, J. Tschanz, S. Lu, T. Karnik, M. Khellah, and V. De, "Tunable replica bits for dynamic variation tolerance in 8t sram arrays," *IEEE Journal of Solid-State Circuits*, vol. 46, pp. 797–805, April 2011.

[41] K. Bowman, J. Tschanz, S. Lu, P. Aseron, M. Khellah, A. Raychowdhury, B. Geuskens, C. Tokunaga, C. Wilkerson, T. Karnik, and V. De, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE Journal of Solid-State Circuits*, vol. 46, pp. 194–208, Jan 2011.

[42] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, "Razorii: In situ error detection and correction for pvt and ser tolerance," *IEEE Journal of Solid-State Circuits*, vol. 44, pp. 32–48, Jan 2009.

[43] P. Whatmough, S. Das, D. Bull, and I. Darwazeh, "Circuit-level timing error tolerance for low-power dsp filters and transforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, pp. 989–999, June 2013.

[44] P. Whatmough, S. Das, and D. Bull, "A low-power 1-ghz razor fir accelerator with time-borrow tracking pipeline and approximate error correction in 65-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 49, pp. 84–94, Jan 2014.

[45] J. Holt and T. Baker, "Back propagation simulations using limited precision calculations," in *The International Joint Conference on Neural Networks (IJCNN)*, vol. ii, pp. 121–126 vol.2, Jul 1991.

[46] S. Sakaue, T. Kohda, H. Yamamoto, S. Maruno, and Y. Shimeki, "Reduction of required precision bits for back-propagation applied to pattern recognition," *IEEE Transactions on Neural Networks*, vol. 4, pp. 270–275, Mar 1993.

[47] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS*, 2011.

[48] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," in *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, (New York, NY, USA), pp. 164–174, ACM, 2011.

[49] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "Approxann: An approximate computing framework for artificial neural network," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, DATE, (San Jose, CA, USA), pp. 701–706, 2015.

[50] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016.

[51] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

[52] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *ISCA*, 2016.

[53] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *IEEE International Solid-State Circuits Conference*, pp. 262–263, 2016.

[54] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, pp. 598–605, Morgan Kaufmann, 1990.

[55] J. Deng, Y. Fang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, and C. Wu, "Retraining-based timing error mitigation for hardware neural networks," in *IEEE Design, Automation and Test in Europe (DATE)*, March 2015.

[56] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, 1995.

[57] J. M. Brader, W. Senn, and S. Fusi, "Learning real-world stimuli in a neural network with spike-driven synaptic dynamics," *Neural computation*, vol. 19, no. 11, pp. 2881–2912, 2007.

[58] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, pp. 295–308, 2009.

[59] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[60] M. Tsodyks, K. Pawelzik, and H. Markram, "Neural networks with dynamic synapses," *Neural computation*, vol. 10, no. 4, pp. 821–835, 1998.

[61] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[62] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends in Machine Learning*, vol. 2, pp. 1–127, Jan. 2009.

[63] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML*, vol. 30, 2013.

[64] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.