CRANIA: Unlocking Data and Value Reuse in Iterative Neural Network Architectures

Maedeh Hemmat, Tejas Shah, Yuhua Chen and Joshua San Miguel
University of Wisconsin - Madison, Madison, WI, USA
{hemmat2, tyshah2, ychen664, jsanmiguel} @wisc.edu

***Abstract*—A common inefficiency in traditional Convolutional Neural Network (CNN) architectures is that they do not adapt to variations in inputs. Not all inputs require the same amount of computation to be correctly classified, and not all of the weights in the network contribute equally to generate the output. Recent work introduces the concept of iterative inference, enabling per-input approximation. Such an iterative CNN architecture clusters weights based on their importance and saves significant power by incrementally fetching weights from off-chip memory until the classification result is accurate enough. Unfortunately, this comes at a cost of increased execution time since some inputs need to go through multiple rounds of inference, negating the savings in energy. We propose *Cache Reuse Approximation for Neural Iterative Architectures (CRANIA)* to overcome this inefficiency. We recognize that the re-execution and clustering built into these iterative CNN architectures unlock significant temporal data reuse and spatial value reuse, respectively. CRANIA introduces a lightweight cache+compression architecture customized to the iterative clustering algorithm, enabling up to 9× energy savings and speeding up inference by 5.8× with only 0.3% area overhead.**

I. Introduction

Machine learning algorithms and deep learning have gained significant success in many applications including image and video classification to natural language processing. Convolutional Neural Networks (CNNs) are among the most widely used family of deep learning methods, providing unprecedented accuracy in many applications such as object localization and object detection [9]. The accuracy boost though comes with significant increase in required memory and computation resources, mandating efficient hardware implementation of these networks [3].

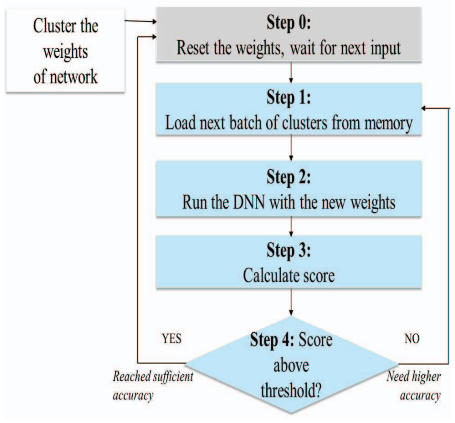
Challenge: Need for Input-Aware Inference. A key insight is that not all inputs require the same amount of computation to be correctly classified. While some inputs need to perform all computations in the network, the vast majority are easy to classify. Furthermore, not all of the weights in the network contribute equally to generate the output. Given these observations, several works [10], [8], [5] have been proposed that aim to dynamically control and decrease the power consumption at run-time while maintaining accuracy. In [8], a teacher-student scheme is exploited to propose Big/Little DNNs. Here, a Little network with fewer layers and a Big network with higher complexity (thus more accuracy) are trained. During inference, the Little network is executed first and the Big network is inferred only if the little one can not provide an acceptable result. In [10], the authors propose an incremental learning/inference framework in which the network is trained incrementally at the beginning by increasing the number of filters in convolutional layers at each iteration. During inference, a portion of the network can be turned off whenever the network can provide acceptable accuracy.

Promising Approach: Iterative CNN Inference. In [5], the authors introduce an iterative architecture for CNN inference that enables dynamic reconfiguration and approximation of the network on an input-by-input basis. To achieve per-input approximation, a clustering algorithm is used to group CNN weights based on their importance. The network is then iteratively inferred and at each iteration, only a fraction of the total weights are fetched from off-chip memory. The fetched weights are determined based on the input, with the rest of the weights kept at 0. This iterative inference leads to significant power savings given that the majority of inputs require only two or three iterations of inference, needing only about 40% of the total weights for correct classification. However, the key drawback to this iterative approach is that execution time is often increased, as each input may need to go through more than one round of inference to generate an acceptable result. This in turn can negate the benefits in energy efficiency.

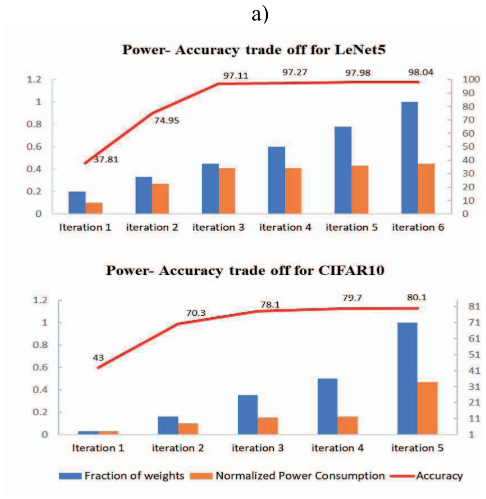
Our Solution: CRANIA. We propose *Cache Reuse Approximation for Neural Iterative Architectures (CRANIA)*. Our goal is to overcome the performance drawbacks of iterative CNN architectures and reap their benefits of per-input approximation and power savings. We uncover properties of data and value reuse that are unique to iterative CNN architectures and leverage them in our CRANIA design:

1) *Iterative execution of layers unlocks significant temporal data reuse.* CRANIA integrates a lightweight cache customized to the iterative clustering algorithm, saving up to 9× energy and speeding up inference by up to 5.8×.
2) *Clustering of weights per iteration unlocks significant spatial value reuse.* By clustering weights based on their importance, fetched values inherently exhibit value similarity more so than in non-iterative architectures. CRANIA exploits this property using a base-delta weight compression scheme customized to our clustering, reducing the cache overhead by 2.25×.

978-1-7281-4123-7/20/$31.00 © 2020 IEEE


Cluster the weights of network
Step 0: Reset the weights, wait for next input
Step 1: Load next batch of clusters from memory
Step 2: Run the DNN with the new weights
Step 3: Calculate score
Step 4: Score above threshold?
YES
Reached sufficient accuracy
NO
Need higher accuracy

a)



b)

Fig. 1: a) Overview of iterative CNN architecture; b) Power - Accuracy trade off for LeNet5 and CIFAR10 [5].

II. Iterative CNN Architectures

The iterative framework proposed in [5] provides a means to approximate the CNN dynamically per input, thus enabling early termination when applicable. This input-dependent approximation is performed via hardware reconfiguration by using only a fraction of the original (non-zero) weights, without altering the hardware structure of the core CNN.

Overview. Figure 1a shows the high-level flow of iterative inference for input-dependent approximation. First, as a pre-processing step, a clustering algorithm groups the weights of an already trained CNN into M clusters, where M is determined via an Elbow method [5]. The objective is to find clusters of weights such that the sum of squared distances from the centroid is minimized. Clustering is performed only once offline, and the clusters are stored as separate groups in memory.

After this one-time clustering, Step 0 begins upon receiving a new input for classification. In this step, all weights in the CNN are reset to 0. Next, an iterative procedure starts to approximate the CNN and conduct inference for that input. At Step 1, we load a new batch of weights from memory, which are fetched from clusters with the highest level of *importance*.

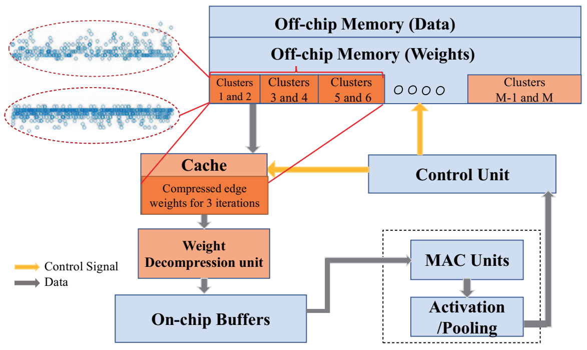


Fig. 2: Abstract hardware model of CRANIA. (Orange components are added to the base architecture by CRANIA.)

All other weights in the network remain 0. Each new batch of weights consists of two clusters, which contain the maximum positive and minimum negative average weights among all clusters. The CNN then executes with these weights in Step 2, which allows for calculating an accuracy score in Step 3. If the score is below a given threshold, a new iteration starts (for that same input) to fetch the next batch of important weights. The iterative algorithm terminates as soon as the score exceeds the specified threshold, indicating that an acceptable classification accuracy has been reached. Upon termination, the procedure returns to Step 0 and waits to receive the next input.

Advantages. Figure 1b shows the trade-off between accuracy, power and fraction of used weights for two networks, LeNet5 and CIFAR10. As the results show, most inputs only need two or three iterations to be correctly classified and achieve similar accuracy compared to the conventional non-iterative architecture (i.e., less than 2% degradation). More importantly, to achieve this accuracy, only 40% of weights are fetched from memory, leading to significant power/enrgy savings.

Drawbacks. Despite saving power, this iterative process significantly increases latency and negates energy efficiency. This is because each inference execution is divided into several iterations, and some inputs may need more than one iteration to reach acceptable accuracy.

III. CRANIA

In this section, we present our proposed solution, CRANIA, which aims to improve the execution time and energy efficiency of iterative CNN architectures. An abstract hardware model of our design is shown in Figure 2. Similar to conventional CNN architectures, weights and input data are stored off-chip. The weights are stored by their clusters. During inference, the network receives the weights and data from memory to perform computations and generates an output. A control unit is used to implement the iterative framework for inference. It receives the output of the inference and controls the memory and computational units of the neural network to either 1) load more clusters of weights and run the network for more iterations, or 2) generate the final output and terminate the execution of the network.

In CRANIA, we first leverage the increased *temporal data reuse* that is inherent to the iterative CNN architecture. We

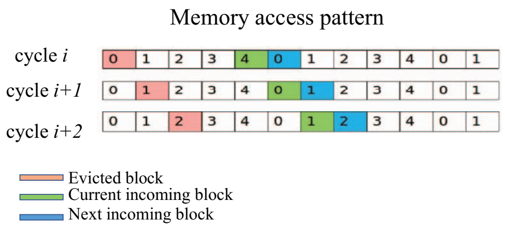


Fig. 3: LRU policy on cyclic sequential access pattern of iterative CNN architectures.

propose to integrate a custom cache into the memory hierarchy of the accelerator. While the weights are fetched to an on-chip buffer for inference, they are also pushed into a very small on-chip cache. Our motivations are twofold. First, as we observe in our experiments, the total number of weights required for the majority of inputs in the first three iterations are significantly reduced (to more than half). This opens up an opportunity to greatly reduce execution time and energy consumption by caching the weights. Second, in addition to intra-iteration weight reuse (which is the case for both non-iterative and iterative CNN architectures), CRANIA also exhibits inter-iteration weight reuse; the clusters of weights from earlier iterations are still needed in later ones. By this nature, the reuse distances of the weights are effectively reduced, making them more suitable for caching.

Next, we leverage the increased *spatial value reuse* that comes naturally from our clustering algorithm. We propose to take advantage of spatial compression techniques to further reduce the cache size and improve our design. Our insight is that by grouping weights together based on their importance to the classification accuracy, we expose significant value similarity among neighboring weights. As Figure 2 shows, the set of weights per cluster exhibits relatively low dynamic range and can be compressed efficiently. Then, to perform inference, we need to decompress the weights before fetching them into the on-chip buffers. A weight decompression unit is integrated into the architecture alongside the cache. In our experiments, we find this decompression overhead to be negligible.

A. *Caching for Temporal Data Reuse*

Weight Cache. As discussed earlier, incorporating weight caching into the memory hierarchy of our iterative CNN architecture can be a favorable solution to speed up inference. Intuitively, cache capacity and associativity are critical factors in CRANIA's energy efficiency and performance. To determine the cache size, the trade-off between miss rate and latency/energy consumption per access should be carefully investigated. While larger caches can decrease the miss rate, they can adversely increase energy consumption and latency per access. They also incur more area overhead to the accelerators, which may not be acceptable for many use cases.

Determining Cache Size. To determine the appropriate cache size and associativity, we start our explorations from a set-associative cache given that the sequential access pattern of weights in neural networks can perfectly match the set indexing. We first generate the memory access trace for each CNN model in the first several iterations. Then, we simulate the cache under different capacities and associativities, measuring the miss rate. Generally, increasing the cache capacity decreases the miss rate but comes at the cost of more area and latency overhead. Based on our experiments, configuring the cache size for a miss rate of 1% yields significant energy savings and speedup while maintaining very low overhead.

Determining Cache Associativity. Next, we analyze the impact of cache associativity on miss rate. On general-purpose architectures, increasing associativity is expected to decrease the miss rate since more cache lines are available for replacement. However, our experiments show that in the case of iterative CNN architectures, increasing associativity offers little to no boost in cache performance and direct-mapped caching is sufficient. This is due to the cyclic sequential access pattern of weights that is inherent to the inference architecture, which is not suitably matched to LRU-based replacement policies that are commonly found in conventional caches. More specifically, LRU policies are counter-productive in architectures with cyclic sequential accesses since the lines accessed furthest in the past (which are the top candidates for eviction) are actually most likely to be accessed next. This is illustrated in Figure 3 for an example cache with 4 entries and LRU replacement. Here, addresses 0, 1, 2, 3, and 4 are accessed iteratively. The evicted lines are highlighted in red, and the new incoming block is highlighted in green. In order to cache address 4, an LRU-based policy would evict address 0 given that it has been accessed furthest in the past. However, due to the iterative cyclic access pattern of weights, in the next cycle, address 0 is expected to be accessed, as shown in blue. As a result, we integrate a direct-mapped weight cache, which is consequently more power-efficient and simpler in structure.

Data Cache. In addition to caching the weights, we investigate the merit of integrating a data cache for the inputs of each layer in the network. Configuring such a cache is different from the weight cache for two reasons. First, in contrast to the weight reuse over iterations, the feature maps calculated per layer are not reused. This is because once a new iteration starts, the feature maps are recalculated using newly fetched weights. Second, we do not need to write back feature maps to the memory upon eviction because the feature maps are only used as an input to the next layer. When layer *i*'s feature maps are calculated, they only need to be cached until layer *i+1* is processed completely, after which they are no longer used and can be discarded.

With this, we determine the input data size analytically. The data cache needs to be large enough to accommodate the largest feature maps across the layers (i.e., the largest input to all but the first layer) as well as the original input to the network (i.e., the input to the very first layer of the network). We note that, unlike per-layer feature maps, the network input is reused over iterations to start the inference. Hence, we can save energy from caching the input once when read from memory the first time.

B. Compression for Spatial Value Reuse

In CRANIA, we recognize that there is inherent spatial value reuse in our clustering; thus we propose to use spatial compression techniques to decrease the required cache storage for weights. In particular, we integrate mechanisms for base-delta compression, a scheme that has been shown to be simple and effective in general-purpose caches [2]. This technique exploits the value similarity and low dynamic range of the values within a cache line. Values are stored as a base value along with an array of deltas; i.e., the differences between each original value and the base. With high spatial value locality, these deltas are near zero and thus can be stored in much less bits than the original values.

Our motivation is that base-delta compression is naturally effective in compressing the weights of iterative CNN architectures due to clustering. Since the weights are grouped by their importance and are accessed and stored together, the fetched values inherently exhibit more value similarity and lower dynamic range. In contrast, base-delta compression is not very applicable to conventional non-iterative architectures where weights are stored and fetched from off-chip memory layer-by-layer, without any systematic value similarity.

Base and delta values need to be computed for each line as they are inserted into the cache. In CRANIA, each line stores up to K weights, to be determined by the configurable cache line size and precision of each individual weight. The base value can be chosen in two different ways. It can either be the first value or the average of all values in the cache line. The former case requires storing *K-1* deltas while the latter one requires K deltas. In this work, we opt for the first weight in the cache line as the base, calculated as follows:

$$Compression\ Ratio = \frac{K \times W_{bits}}{W_{bits} + (K-1) \times \Delta_{bits}}$$

where W_{bits} is the precision of each weight and Δ_{bits} is the required number of bits for storing deltas.

Determining *K*, and equivalently the cache line size, is an important step in CRANIA. On one hand, having more weights (i,e., increasing *K*) in each cache line can be beneficial to improve the compression ratio, because more weights can share one base value. Also, a larger cache line can significantly decrease the cache tag overhead. On the other hand, increasing the number of weights to be compressed in each cache line can reduce the similarity between the weights, hence increasing the number of bits per delta and degrading the compression ratio. Our experiments explore this trade-off in determining *K*.

IV. Evaluation and Discussion

A. Simulation Framework

In our experiments, we used two well-known CNNs namely, LeNet5 and CIFAR10. The architecture of these network and their input and weight matrix sizes are summarized in Table I. The networks are first implemented and trained with Neupy [4]. Then, we quantized the trained floating point weights to 16-bit fixed point ones and imported each network in Matlab to verify post-quantization accuracy. To build iterative CNN architectures, we have used K-means clustering algorithm and divided the edge weights to twelve and ten clusters for LeNet5 and CIFAR10, respectively, as proposed in [5]. We note that 12 and 10 clusters lead to at most six and five iterations of inference per input, because at each iteration two clusters of edge weights are fetched and used for inference. This is while majority of inputs require three iterations of inference, as discussed in Section II. Hence, an accuracy almost similar to the non-iterative network can be achieved after at most 3 iterations and using less 50% of the weights.

TABLE I: Information on experimented neural networks

	LeNet5		CIFAR10	
	Weight	Input	Weight	Input
CS1	5×5×1×20	28×28×1	5×5×3×32	32×32×3
CS2	5×5×20×50	12×12×20	5×5×32×32	14×14×32
CS3	-	-	5×5×32×64	9×9×64
FC1	800×500	4×4×50	1024×10	4×4×64
FC2	500×10	500×1	-	-

TABLE II: Energy cost and latency of various arithmetic operations and memory accesses in 45nm technology.

Component	Energy (pJ)	Latency (clock cycle)
16-bit adder	0.4	1
16-bit multiplier	1.0	1
Max Pool	1.2	2
ReLU	0.9	1
18 KB cache	13.5	3
1 KB cache	1.8	1
0.5 KB cache	1.3	1
DRAM	1950	120

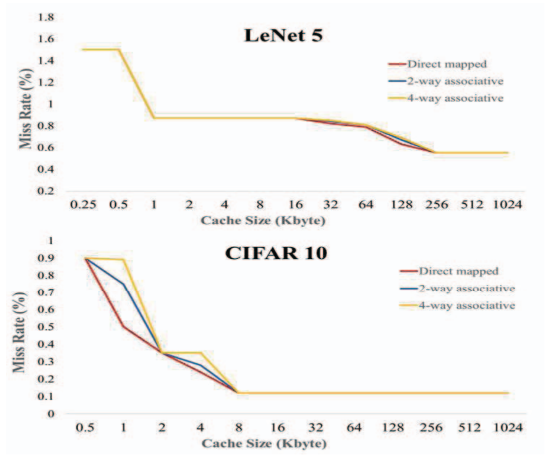
To measure the latency and energy consumption of the network for inference, we first adapted the energy consumption and latency of each individual computation unit from [9], [7]. We have also used CACTI 6.0 [6] to measure energy consumption and latency of DRAM memory and various caches of different size. The results are summarized in Table II. Then, we constructed an analytical model based on the CNN network architecture that computes the number of required operations for inference, including the number of memory accesses and the number of MAC operations. It finally calculates total network energy and latency. Note that latency is measured as the number of clock cycles required to finish one or several iterations of inference.

B. Determining Cache Size

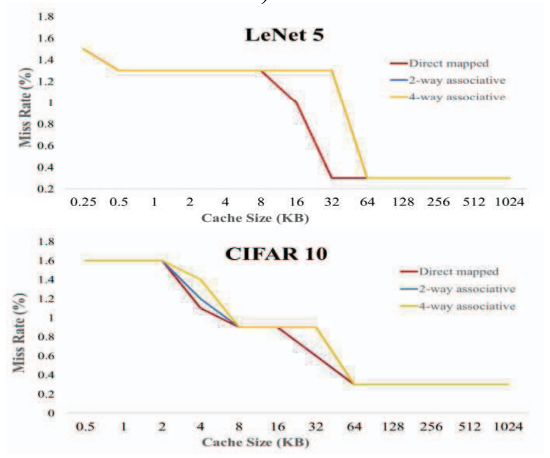
Weight Cache. Here, we determine the weight cache size for LeNet5 and CIFAR10 using the procedure explained in Section III-A. Figure 4a shows the miss rate versus cache size for these two iterative networks in the first three iterations. As the figure shows, the required cache size for a reasonably small miss rate (i.e., 1% in this work) is 1KB and 0.5KB for LeNet5 and CIFAR10, respectively.

Figure 4a also shows the miss rate for different configurations of the cache. As the results show, direct - mapped caches show better performance compared to set-associative ones, as we discussed in Section III-A.

Data Cache. To determine the size of data cache, we have used the analysis discussed in Section III-A. The cache needs to accommodate the input to very first layer *and* the largest



a)



b)

Fig. 4: Miss rate under different configurations of cache associativity and size for a)iterative and b)non-iterative networks.

input across all remaining layers. As Table I shows, for both networks, the second layer, CS2, has the largest input size, thus should be cached along with the original input to the network (i.e., CS1 input). With 16-bit precision for input and feature maps, the data cache size is 8KB and 18KB for LeNet5 and CIFAR10, respectively.

C. Compression Results

To further reduce the weight cache size for iterative CNN architectures, we have applied Base-delta compression technique to store the edge weights.

Evaluating Compression Ratio. To compress the edge weights, we first need to determine the cache block size (i.e., equivalent to K) and then evaluate the number of bits required to represent the deltas and base in each cache line. Recall that required number of bits for representing deltas is changing as K varies. To investigate the trade off between compression ratio and cache block size, we changed K from 4 to 64 (i.e., equivalent to changing the number of weights in a cache line from 4 to 64). Then, using the first weight in each block as the base, we have calculated the delta array. Our experiments show that for 16-bit weight precision, the required number of bits per delta is increased from 6 bits to 7 bits as K is increased. Given that larger K will reduce the tag overhead of the cache and can be also beneficial for improving the compression ratio, we have chosen K= 64.With 7 bits per delta and using the first weight as the base, the compression ratio for the network will be 2.25 using the equation in III-B. With this compression ratio, the cache size is decreased to 0.5KB and 0.25KB for LeNet5 and CIFAR10, respectively.

Decompression Overhead. Once the weights are read from the cache, they should be decompressed before being fetched to the computational units to do the inference. In the case of Base-Delta compression, weights can be decompressed by adding deltas to the base value in each cache line. Hence, the required decompression logic is 16-bit fixed-point adders which is relatively cheap compared to the other components of the network. In addition, the energy saving achieved from reducing the cache size outperforms the energy consumption incurred by decompressing. As a result, the decompression overhead will not degrade the energy efficiency of iterative CNN architectures.

D. CRANIA for Non-Iterative CNN Architectures

CRANIA exploits the inherent features of iterative CNN architectures to improve its performance by unlocking significant temporal data reuse and spatial value reuse. To show this, we have evaluated the efficiency of CRANIA on conventional non-iterative architectures.

Determining Cache Size. To show how iterative CNN architectures can benefit more from integrating the cache, we have measured the required cache size for non-iterative architectures. Similar to the iterative case, we first generated the memory trace for non-iterative architectures and measured the miss rate of the network under different cache sizes. We note that the memory trace for non-iterative CNN architectures has two main differences compared to iterative ones: First, it has significantly larger number of weights given that all weights in the network are used for inference. Second, for non-iterative architectures, we only have intra-iteration weight reuse which leads to larger distance reuse, particularly in fully connected layers.

Figure 4b shows the cache miss rate for non-iterative networks, LeNet5 and CIFAR10, under different cache sizes and configuration. Similar to iterative networks, increasing the cache size has reduced the miss rate. However, in order to achieve the same miss rate with iterative CNN architectures (i.e., 1% miss rate), the cache size should be 32KB and 8KB for LeNet5 and CIFAR10, respectively, which is increased $32\times$ and $16\times$ in comparison to its iterative counterpart.

Evaluating Compression Efficiency. Unlike iterative CNN architectures in which clustering has significantly increased the value similarity, in conventional architectures, the weights are stored and fetched layer by layer from/to memory. Hence, we do not expect any significant value similarity between neighboring weights. To show this, we have applied base-delta compression to non-iterative CNN architectures and measured the deltas for each cache line. The results show higher dynamic range of deltas for non-iterative network architecture in comparison to the iterative ones. Hence, base-delta compression can not be as effective in reducing the cache size for non-iterative networks.

E. Area Overhead

We have measured the area overhead incurred by integrating a cache into CNN accelerator. For this, we have used the area numbers from DaDianNao [1], obtained in 28nm technology. For a fair comparison, we then have scaled the area numbers to 45nm technology.

The area for each processing unit (PU) consisting of multipliers, adders, and ReLU is $0.78mm^2$ in 28nm technology [1], which is scaled up to 1.29 mm^2 for 45nm technology. Sixteen PEs are used in each accelerator. Hence, the total area of processing units is 20.76 mm^2. On the other hand, the area consumption of the largest cache required in our design (which is 18KB data cache for CIFAR10) is 0.065 mm^2, measured by CACTI 6.0 [6]. Hence, the area overhead incurred by the cache is around than 0.3%, thus negligible.

F. Network Evaluation

Here, we will evaluate the efficiency CRANIA on latency and energy consumption of iterative CNN architectures over the first three iterations. To achieve this, we have measured the energy saving and speedup achieved by CRANIA for each iteration in comparison to the base iterative network (i.e., the network without caching and compression). Note that energy savings and speedup are normalized to iterative CNNs instead of non-iterative ones; the former has been shown to be more energy-efficient [5]. Figure 5 shows the results for LeNet5 and CIFAR10, respectively. As can be seen, CRANIA is able to improve energy efficiency and performance of iterative network architectures.

More specifically, for LeNet5 (CIFAR10), integrating the cache into CNN design can speed up the network by at least $3.5\times$ $(2\times)$ and save energy up to $8\times(1.5\times)$. Compression is able to further improve speedup and energy saving up to $5.8\times(2.5\times)$ and $9\times$ $(2\times)$ compared to the base case. Note that further energy saving/speedup is possible after compression, because compression has reduced the cache size which in turn decreases energy and latency per cache access.

V. Conclusion

In this work, we have proposed CRANIA, a lightweight cache and compression architecture customized for iterative execution, enabling fast and energy-efficient inference. CRANIA exploits significant temporal data reuse and spatial value reuse inherent to iterative CNN architectures, which originate from clustering weights based on importance and dynamically approximating and re-executing the network during inference. Our results on two popular networks show up to $9\times$ energy savings and $5.8\times$ speedup in iterative inference with CRANIA.

Acknowledgments

We thank reviewers for their valuable feedback. This work is supported by the Wisconsin Alumni Research Foundation and the University of Wisconsin-Madison.

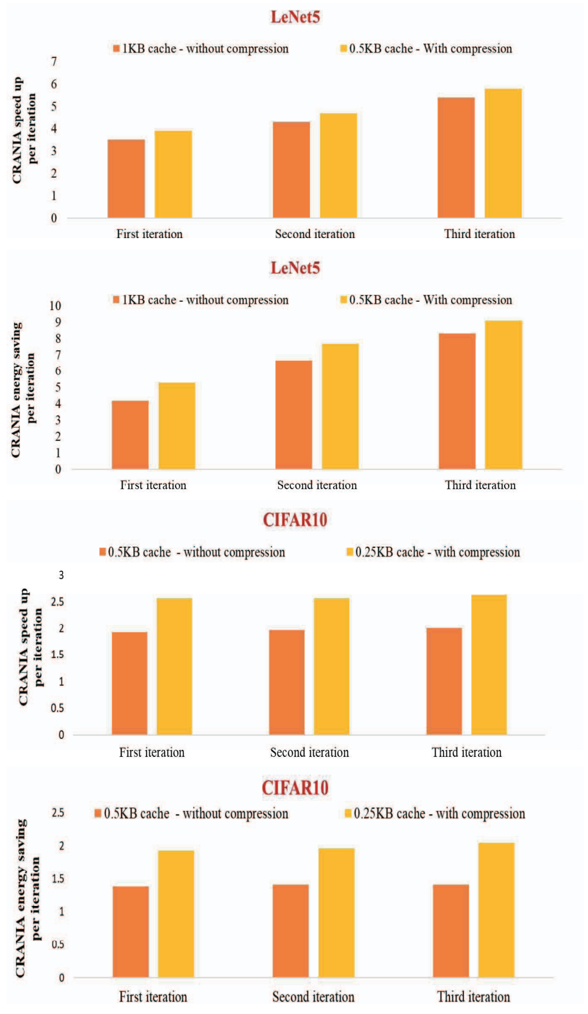


Fig. 5: CRANIA energy savings and speedup.

References

[1] 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 *MICRO*, pages 609 –622, 2014.

[2] G.Pekhimenko, V. Seshadri, O. Multu, M. Kozuch, P. Gibbons, and T. Mowry. Base-delta-immediate compression: practical data compression for on-chip caches. In *PACT*, pages 377 – 388, 2012.

[3] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In *DATE*, pages 1474–1479, 2017.

[4] http://neupy.com/pages/home.html.

[5] M.Hemmat and A. Davoodi. Dynamic reconfiguration of CNNs for input-dependent approximation. In *ISQED*, pages 176 – 182, 2019.

[6] N. Muralimanohar, R. Balasubramonian, and N.Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *MICRO*, pages 3 – 14, 2007.

[7] M. Nazemi, G. Pasandi, and M. Pedram. Energy-eicient, low-latency realization of neural networks through boolean logic minimization. In *ASPDAC*, pages 274 – 279, 2019.

[8] E. Park, D. Kim, S. Kim, Y. Kim, G. Kim, S. Yoon, and S. Yoo. Big/little deep neural network for ultra low power inference. In *CODES +ISSS*, pages 1624–132, 2015.

[9] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. Strachan, M. Hu, R. Williams, and V. Srikumar. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *ISCA*, pages 14–26, 2016.

[10] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda. Runtime configurable deep neural networks for energy-accuracy trade-off. In *CODES + ISSS*, pages 34:1–34:10, 2016.