AxR-NN: Approximate Computation Reuse for Energy-Efficient Convolutional Neural Networks

Dongning Ma[‡], Xunzhao Yin[§], Michael Niemier[*], X. Sharon Hu[*], and Xun Jiao[‡]
[‡]Villanova University, [§]Zhejiang University, [*]University of Notre Dame

***Abstract*—The recent success of convolutional neural networks (CNN) has led its implementation in specialized accelerators such as graphics processing unit (GPUs). However, the intensive computing workloads of CNNs remain a challenge to existing accelerators. By leveraging the error tolerance of CNNs, we propose a novel method to design energy-efficient CNN accelerators using approximate computation reuse (ACR), referred to as `AxR-NN`. Computation reuse aims to reuse the previously computed results to avoid redundant executions. However, it cannot be applied directly to CNNs because CNNs do not have enough data locality. Thus, `AxR-NN` performs approximate computation reuse under relaxed precision requirements on input patterns and design a reconfigurable architecture to support the ACR. This reconfigurable pattern matching is central to achieve a "controllable approximation". We implement the `AxR-NN` using content addressable memory and integrate them with floating point units. Simulation results show that `AxR-NN` reduces the computation energy by 30-58% with only 1-2.5% accuracy degradation on MNIST, EMNIST, and CIFAR-10 dataset.**

I. Introduction

Convolutional neural networks (CNNs) have shown success in various application domains such as digit recognition [2], speech recognition [8], and image classification [11]. This versatility has led CNN implementations on various hardware platforms, e.g., GPUs [5]. However, the computationally intensive workloads of CNNs, which are mainly composed of addition and multiplication operations, incur significantly high energy consumption, posing a challenge for hardware implementation.

To combat this challenge, recent approaches focus on reducing computing workloads by exploiting the use of approximations in computed results, often referred to as "Approximate Computing." Approximate computing leverages error tolerance at the application level to allow inaccurate computations in a system while maintaining an acceptable output quality. Neural networks are naturally suited to such approximations because of their intrinsic error tolerance [3], [17], [22]. Approximate computing can be performed at both software and hardware levels. For example, pruning and compression-based techniques are used to shrink the original dense network, leading to a hardware-friendly sparse network structure [6]; Quantization are used to transform floating point operations into fixed point operations which requires less energy for computation [15]. On the hardware side, selective replacement of the less-critical neurons with approximate neurons is proposed in [22]. The substitution of exact multipliers with approximate multipliers is also proposed in [3], [17]. Although it is natural to apply approximation to neural networks due to their intrinsic properties, practically it still requires *retraining* to mitigate approximation-induced errors. Besides, the fixed physical hardware implementation cannot be flexibly adjusted during the runtime to control efficiency-accuracy tradeoff.

To overcome these limitations, we propose a reconfigurable and controllable approximation technique used in inference of CNNs by exploiting the reuse of computed results. Computation reuse has been adopted in various applications that demonstrate strong data locality. However, traditional computation reuse techniques cannot be directly applied to CNNs as CNNs exhibit a weak data locality (Section III). To overcome this challenge, we leverage the inherent error tolerance of CNNs, and perform approximate computation reuse (ACR) under relaxed precision requirements on input patterns. We design a reconfigurable architecture to support the ACR with reconfigurable precision using an efficiency content-addressable memory (CAM). The advantages of **`AxR-NN`** are three-fold: retraining-free, controllable accuracy loss, and complimentary to existing methods such as pruning and quantization. Besides, the proposed method can be applied to different hardware platforms such as GPU, FPGA, and ASIC.

Our contributions are as follows:

- We propose **`AxR-NN`**, a novel method that explores and enhances computation reuse in CNNs by performing layer-wise approximate pattern matching.
- We design and implement **`AxR-NN`** using an FeFET-based content addressable memory that can be reconfigured to systematically control the approximation effort to achieve balanced energy-accuracy trade-offs.
- We demonstrate the effectiveness of **`AxR-NN`** . The evaluation results indicate that **`AxR-NN`** achieves 30-58% computation energy consumption reduction, with only 1-2.5% accuracy degradation on MNIST, EMNIST, and CIFAR-10.

II. Related Work

Approximate computing trades intrinsic error tolerance at the application level for improved performance and energy efficiency. Venkataramani *et al.* proposed an approach to evaluate the criticality of different neurons and selectively replaces less-critical neurons with approximate neurons with dynamic accuracy [22]. A similar work [25] replaces the less-critical neurons with approximate ones and skip some neuron operations. These two works focus on finding the opportunities for approximate computing without significantly degrade the accuracy. Another two works focus on designing

inexact hardware to trade for energy efficiency. Du *et al.* proposed an inexact multiplier design using an inexact logic minimization method, and emphasizes the need to approximate multipliers rather than adders [3]. A hardware optimization approach was proposed in [17] to design multipliers in a uniform way that suits physical VLSI implementation. However, such approaches can result in an accuracy drop and ask for *retraining* to mitigate the accuracy loss. Furthermore, such inexact logic design is not flexible in the sense that it is not reconfigurable once it has been physically implemented, making it less general to different types and architectures of neural networks.

Due to value locality and similarity presented in CNNs, computation reuse has been exploited to improve efficiency [10], [9], [20], [7]. For example, Riera et al. utilizes the input similarity in a streaming applications such as speech and video and design a complicated layer-level reuse mechanism. Hegde et al.utilizes weight similarity to enable a dot product factorization and activation group reuse [7]. A more similar set of works to ours are the ones utilizing operation-level similarity. For example, Bloom filters [10] and STT-RAM based TCAM [9] are used for computation reuse but they are limited in evaluating small datasets, i.e., MNIST. In addition, they both suffer from the lack of a systematic control of accuracy loss: [10] introduces false positives and [9] introduces random bit mismatch based on hamming distance which may mismatch most significant bits.

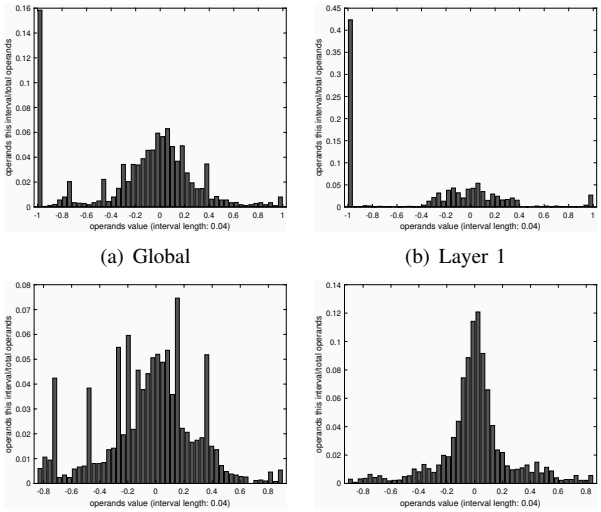
Compared with existing works, **AxR-NN** exploits operand silimilarity in CNN computations and does not require retraining. Further, **AxR-NN** can be systematically reconfigured to control the accuracy loss. By enaling a layer-wise limited bitwidth pattern matching, **AxR-NN** can enhance the computation reuse even under an original low data locality, thus is more universal over existing approaches.

III. Case Study on Data Locality

We perform a case study on CNN data locality using a LeNet-5 [13], a widely used CNN consisting of six layers, among which three are convolutional layers. We use the MNIST database of handwritten digits as the dataset.

To highlight the distribution of computation operands, we draw histograms of multiplication operands from the three convolutional layers as shown in Fig. 1. We profile the distribution using two settings: *global-based* and *layer-based*. For *global-based*, we profile the input operands in all layers. For *layer-based*, we profile input operands for each convolutional layer separately. For *global-based*, as shown in Fig. 1(a), around 15% of input operands fall into the interval near -1 while the rest exhibit a Gaussian-like distribution with a center around 0. For *layer-based*, per Fig. 1(b)-(d), operands from each layer have more unique characteristics: layer 1 contains most of the operands near -1; layer 3 has a more concentrated distribution around 0; layer 2 has a distribution with more variation, with several spikes around specific numbers.

Such observations indicate that the data locality is more notable at each layer and the input operands tend to cluster



(a) Global (b) Layer 1

(c) Layer 2 (d) Layer 3

Fig. 1. Histograms of Multiply Operands from Convolutional Layers

within a small range. This suggests two hypotheses summarized below, which are subsequently verified in Section IV:

- *Hypothesis 1*: Since each layer tends to have unique distribution, computation reuse can be enhanced with layer-wise matching.
- *Hypothesis 2*: Since the distribution of operands is clustered, computation reuse can be enhanced with approximate matching.

IV. Proposed Method

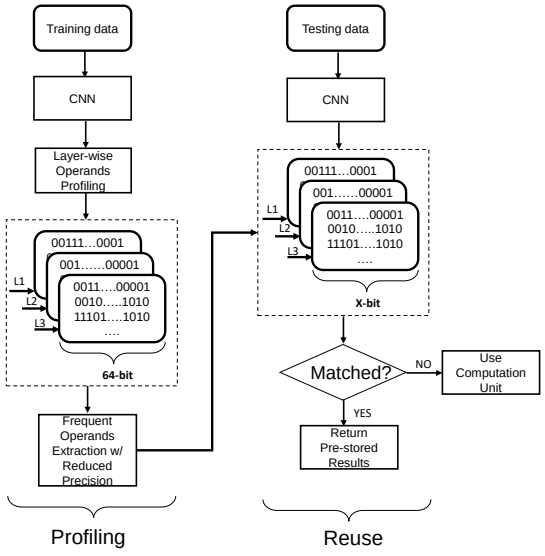


Fig. 2. **AxR-NN** Overview.

A. Layer-Wise Pattern Matching

Since each arithmetic operation has two inputs, we define the combined vector of two input vectors as one input pattern. To substantiate *Hypothesis 1*, we profile and select a set of

most frequently used global and layer-wise operand patterns, respectively. These most frequently used patterns as well as their operation results will be stored in the memory. Once a new input pattern from testing data are found to be identical (match) with the pre-stored patterns in the memory, we call it a "hit". Subsequently we use "hit rate" to quantitatively measure the chances of "hit", as defined in Eq. 1.

$$r = \frac{m}{n} \tag{1}$$

In Eq. 1, r is the hit rate, while m is the number of matched patterns and n is the total number of patterns.

As shown in Fig. 3, the hit rate increases as we store more patterns, and the layer-based matching methods always have slightly higher hit rates over global-based hit rates. This validates *Hypothesis 1* that a layer-based matching approach could more effectively exploit data locality to enhance computation reuse.

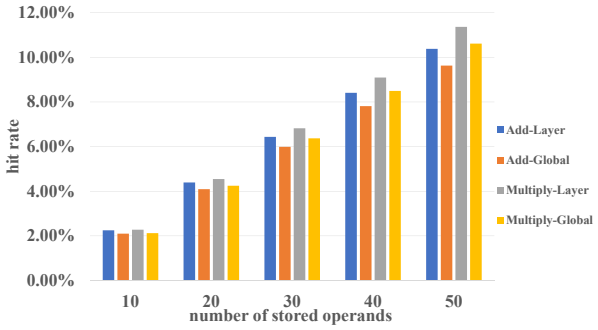


Fig. 3. Hit Rate of Global and Layer-based Matching

However, the hit rate is still very low: even 50 stored patterns yield only 10 - 12% hit rate. With such low hit rates, the energy saving achieved by computation reuse may not compensate the extra hardware cost. The reason for such low hit rate is as follows: According to the IEEE 754 format, a single precision (32-bit) floating point number consists of three fields: 1-bit sign, an 8-bit exponent field and a 23-bit mantissa field. All 32 bits will be compared and only if all bits match will it be considered as a successful match on one operand (only). Note that we need to match the combined vector of two operands. This matching is referred to as exact matching.

B. Approximate Pattern Matching

To improve the hit rate, we follow *Hypothesis 2* by exploiting the clustering characteristics of input patterns. That is, we profile the most frequent operands with reduced precision and accordingly adopt approximate pattern matching. Specifically, during profiling, we only compare "x" highest order bits and store the most common highest order bits in memory; during ACR, we only compare "x" highest order bits of new input patterns with the stored patterns — this relaxed matching standard will only require a limited number of bits to be the same for a match rather than requiring all 32 bits to be identical. (We call it "x-bit matching").

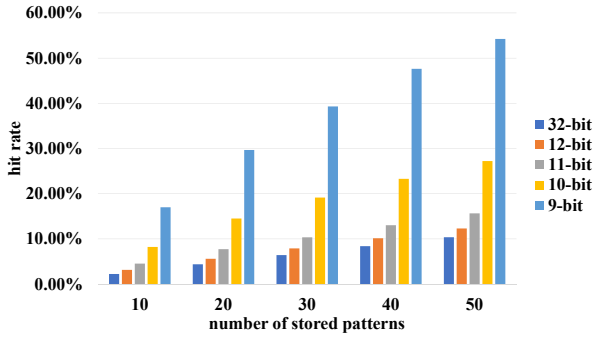
For example, we perform profiling with 10-bit matching and find 0011111111 is one of the most common 10 highest order bits across all the profiled operands. We then pre-store this 10-bit vector in CAM. Then if the new testing operand is 1.5375, with IEEE-754 format as "00111111110001001100110011001101", we consider it a match because their first conflict only appears at the 11th bit. Therefore, any operand with their first 10 bits as 0011111111 will have a match. Note that again this is just for one operand match and we need match on both operands to claim a pattern match.

Combining layer-based and approximate pattern matching, the hit rate could be significantly improved as illustrated in Fig. 4. We examine five different matching modes: 9-bit, 10-bit, 11-bit, 12-bit, and 32-bit (exact matching). We can observe that the hit rate is in negative correlation with the bit length for pattern matching. For add operations, the hit rate of 10-bit length matching mode doubles that of the exact matching mode, and the 9-bit even further increases it to around 50% with 50 stored patterns. For multiply operations, statistics are more optimistic as the 50 pattern, 9-bit matching method could increase hit rate to more than 80%, compared with the less-than-10% hit rate of exact matching.

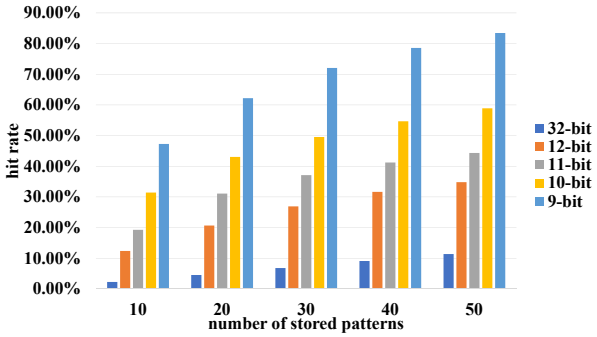
C. Reconfigurable Approximation

Increased hit rates do come with accuracy loss since **AxR-NN** returns approximate results rather than exact results. Fortunately, the reconfigurable property of **AxR-NN** allows us to systematically control the approximation effort by tuning the number of bits required for a match and the number of stored patterns.

Table I presents the accuracy loss under different number of stored pattern (*#match_patterns*) and number of bits required for matching (*#match_bits*), where several observations can be made. First, 8-bit matching leads to significant accuracy loss because it introduces aggressive approximation that ignores the last bit in the exponent field, which could cause a notable deviation from the exact results. Second, we can regain accuracy by tuning the matching bit length. For example, 9-bit matching is able to achieve acceptable accuracy loss because it covers at least the sign bit and the exponent field. Moreover, across all different numbers of patterns, 10-bit matching always exhibits insignificant accuracy loss because the returned approximated results are closer to the exact results. This intrinsic error tolerance of CNNs even without retraining allows us to systematically control the accuracy loss by tuning matching settings. Third, as we store more patterns, CNN suffers more accuracy loss because of more returned approximate results. For example, (9-bit matching, 8 stored patterns) does not cause noticeable accuracy loss but (9-bit matching, 64 stored patterns) reduce the accuracy to 91.3%. More parameter space explorations are presented in Section V to explore the best energy-accuracy trade-off.



(a) Add



(b) Multiply

Fig. 4. Hit Rate of Approximate Layer-wise Matching

TABLE I
NN ACCURACY UNDER DIFFERENT MATCHING SETTINGS.

#match_patterns		#8	#16	#32	#64
#match_bits	8-bit	53.2%	41.1%	34.0%	29.4%
	9-bit	99.1%	98.7%	94.6%	91.3%
	10-bit	99.2%	99.1%	98.9%	98.1%

D. FeFET-based CAM Implementation

CAM can be implemented with different technology, such as CMOS [18], MTJs [16], and ReRAMs [14]. Recently, FeFET-based Tenary CAMs (TCAMs) have been proposed, and could deliver better energy efficiency, performance, and area with comparisons to other technology implementations [23]. Therefore, we evaluate our approximate memoization approach in hardware assuming a 4T-2FeFET TCAM.

The schematic of 4T-2FeFET TCAM cell shown in Fig. 5 is identical to the cell design in [24]. However, unlike previous ideal, single-domain Landau Khalatnikov (LK) FeFET model [1] used in [24], in this paper we employ the latest Preisach FeFET multi-domain model which has been calibrated and validated with experiments measurements [4]. The Preisach FeFET model assumes a higher write voltage

TABLE II
OPERATIONS OF 4T-2FEFET TCAM CELL

V_{write}=4V V_{search}=1V		WL	$BL/\overline{BL}$	$SL/\overline{SL}$	S	$\overline{S}$
Write '1'	Step 1	V_{write}	V_{write}/0	0	'1'	hold
	Step 2	0	0/-V_{write}	0	hold	'0'
Write '0'	Step 1	0	-V_{write}/0	0	'0'	hold
	Step 2	V_{write}	0/V_{write}	0	hold	'1'
Write $don't\ care$		0	-V_{write}	0	'0'	'0'
Search '1'		V_{search}	V_{search}	V_{search}/0	–	–
Search '0'		V_{search}	V_{search}	0/V_{search}	–	–

V_{write} (4V) and a read voltage of 1V, thus resulting in quite different write and search operations, which are summarized in Table II. The write phase consists of two steps. To write a logic '1', we first apply V_{write} to WL, and the access transistors T_3 and T_4 are activated. Since logic state S is stored in M_1, we apply V_{write} to BL, and keep $\overline{BL}$ at 0. The gate voltage of M_1 is raised and sets the polarization of M_1 to logic '1'. Then we set the WL to 0, and apply 0/-V_{write} to M_1/M_2, respectively. The gate-source voltage of M_2 is negative and the polarization of M_2 is set to logic '0'. To write a logic '0', we write logic '1' to M_2 and logic '0' to M_1 respectively. For the $don't\ care$ state where the state of the TCAM cell represents a wild card, we write both logic '0' to both FeFETs by applying $-V_{write}$ to both BL and $\overline{BL}$. During the write phase, $SL/\overline{SL}$ are set to 0 to eliminate static current. In the search phase, we apply V_{search} to WL and both bitlines to turn the FeFETs on. The FeFET with state '1' will conduct current, which turns on the pull down path, while the FeFET with state '0' turns off the pull down path. When searching for a logic '1', we apply V_{search}/0 to $SL/\overline{SL}$. In contrast, we apply 0/V_{search} to $SL/\overline{SL}$ to search for a logic '0'.

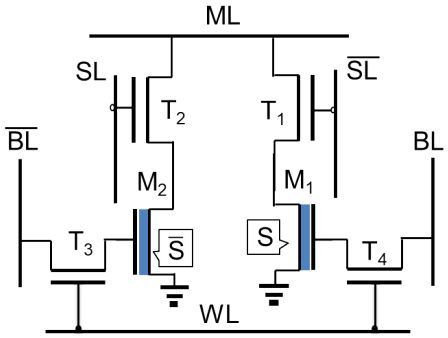


Fig. 5. Schematic of 4T-2FeFET TCAM cell.

We implement the TCAM array architecture as shown in Fig. 6. The architecture consists of the TCAM core, the write/search buffers, the wordline drivers, the output sense amplifier (SA) and the encoder, which are similar components in [24]. A simple inverter-based SA is used for the TCAM array. In the write phase, we follow the write steps in Table II and apply corresponding voltages to the wordlines (WLs) and bitlines (BLs/$\overline{BL}$s) associated with a selected row. To inhibit write disturbance, we drive the wordlines WLs associated with the unselected rows to disable the write operation, which has

been validated by simulations. For example, when the selected WL is enabled by V_{write} (step 1 in write '1' and step 2 in write '0' in Table II), 0 is applied to other WLs. When the selected WL is enabled by 0 (step 2 in write '1' and step 1 in write '0'), we apply $-V_{write}$ to other WLs. In the search phase, all the wordlines and bitlines are set to the search voltage V_{search} to turn the FeFETs on, and the search buffer drives the search lines according to the input data. When a match occurs, the matchline associated with the matched word remain at a high voltage, and the encoder receives the match signal from the SA, and sends a "hit" signal along with the matched entry.

Reconfigurability: We can reconfigure the approximate efforts by tuning two parameters: number of matching bits (*#match_bits*) and number of stored patterns (*#match_patterns*). To reconfigure *#match_patterns*, we can just drive the wordlines to deactivate unselected rows. To reconfigure *#match_bits*, we can just write $don't\ care$ into the TCAM cells. The commands can be initiated via system kernel or software.

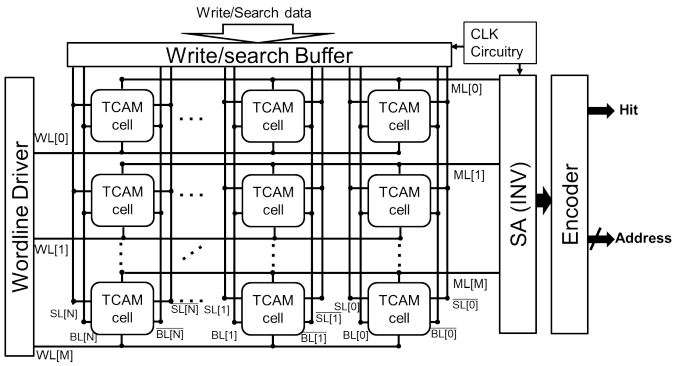


Fig. 6. Architecture of an $M \times N$ TCAM array

V. Experimental Results

A. Experimental Setup

We use three datasets: MNIST, EMNIST, and CIFAR-10 as our benchmark. MNIST has a training set of 60,000 examples, and a test set of 10,000 examples of handwritten digits. EMNIST dataset [2] is an extension of MNIST to handwritten *letters* and contains 145,600 characters within 26 balanced classes. CIFAR-10 image classification dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class [12]. We implement FeFET-based TCAM in 45nm and evaluate the energy using HSPICE. The TCAM array can be tightly integrated to the floating point units (FPUs) in a given architecture, which has been implemented in GPU [24] and FPGA [21]. Energy consumptions of FPUs are extracted from [24]. The TCAM arrays use the minimum transistor sizes for buffer, cell and SA to save energy. Wiring parasitics are extracted for 45nm technology node from DESTINY [19].

B. Energy Saving

In this section, we use different configurations of **AxR-NN** to explore energy savings and accuracy loss. We defined a tuple of four controllable parameters $< b_a, b_m, p_a, p_m >$, where b_a and b_m represent the *#match_bits* for add and multiply, and p_a and p_m represent the *#match_patterns* of add and multiply. Note that, the *#match_bits* determines the number of cells at each TCAM row and the *#match_patterns* determines the number of TCAM rows. For example, storing 32 patterns under 10-bit matching will require 32 rows with each row containing 20 cells (two 10-bit vectors).

TABLE III
MNIST Energy Saving using **AxR-NN**

b_a	p_a	b_m	p_m	accuracy loss	energy saving
9	8	9	8	0.1%	33.01%
10	16	10	16	0.2%	29.30%
9	16	9	16	0.6%	44.81%
10	32	10	32	0.4%	37.36%
9	32	9	32	4.7%	58.00%
10	32	9	32	2.0%	52.00%
9	32	10	32	1.9%	43.36%
10	**64**	**9**	**64**	**1.3%**	**58.71%**
9	64	9	64	8.0%	65.99%

TABLE IV
EMNIST Energy Saving using **AxR-NN**

b_a	p_a	b_m	p_m	accuracy loss	energy saving
9	8	9	8	30.8%	34.02%
9	8	10	8	4.1%	23.79%
10	8	10	8	0.9%	21.98%
9	16	9	16	32.7%	45.52%
9	16	10	16	3.6%	33.76%
10	**16**	**10**	**16**	**2.4%**	**30.66%**
10	32	10	32	4.6%	41.02%
9	32	10	32	18.0%	45.55%
10	64	10	64	13.1%	43.93%

TABLE V
CIFAR-10 Energy Saving using **AxR-NN**

b_a	p_a	b_m	p_m	accuracy loss	energy saving
9	8	9	8	10.1%	15.48%
9	16	9	16	18.4%	16.42%
10	16	9	16	4.6%	14.93%
10	**8**	**9**	**32**	**1.8%**	**31.22%**
10	16	9	32	5.5%	30.88%
10	8	9	64	15.6%	37.61%
10	8	10	64	1.0%	7.36%

We compare the energy consumption using **AxR-NN** with the baseline implementation without **AxR-NN**, i.e. all computations are performed via FPUs. The energy saving comes from two parts: **AxR-NN** avoids redundant floating point operations; Low-wordwidth (reduced *#match_bits*) design reduces TCAM energy itself. These two mechanisms work in synergy. When the wordwidth is reduced, the possibility of a match is simultaneously increased: for 32-bit matching, there can be 2^{32} different situations in the matching space while for a reduced-bit mode like 10-bit matching, the matching space

size is only 2^{10} which is reasonable for a TCAM size like 64 that can significantly increase the hit rate and maximum the use of the proposed TCAM architecutre. Table III, Table IV and Table V show the results of CNN tradeoffs between accuracy loss and energy-saving under various configurations of approximate **AxR-NN** for MNIST and EMNIST, from which we can observe several important trends.

First, for a fixed bit length matching, the energy savings are improved if the number of stored patterns increases because the hit rate is increased. For example, for MNIST, the energy saving is 33.01% at $< 9, 8, 9, 8 >$ and is increased to 65.99% at $< 9, 64, 9, 64 >$.

Second, increasing the number of stored patterns would introduce more approximation into neural networks, resulting in an accuracy degradation (For example $< 9, 64, 9, 64 >$ leads to 8.0% accuracy drop for MNIST). Therefore, we should reduce the number of stored patterns to ensure acceptable accuracy (e.g., by using $< 9, 16, 9, 16 >$, accuracy loss can be reduced to 0.6% for MNIST). Alternatively, we can also change the bit length for matching — by switching to 10-bit matching, the accuracy loss could always be maintained at an acceptable level (below 2%). The best configuration for MNIST is found under 10-bit matching at $< 10, 64, 9, 64 >$, which yields 1.3% accuracy loss while enabling 58.7% energy saving.

Third, the best configuration of approximation varies with different datasets. However, accuracy loss is still controllable as we are able to tune the approximate configurations. For example, $< 10, 16, 10, 16 >$ exhibits 30.66% energy saving while only triggering 2.4% accuracy loss for EMNIST and $< 10, 8, 9, 32 >$ exhibits 31.22% energy saving with 1.8% accuracy loss for the more sophisticated CIFAR-10.

We also compare our approximate matching scheme with traditional exact matching approaches [24] on MNIST dataset. Results show that exact matching only provides very limited energy saving, e.g., 1.04% (1.25%) saving with *#match_patterns* at 8 (16) patterns. This is largely due to the low hit rate of exact matching as shown in Fig. 4.

VI. Conclusion

This paper presents **AxR-NN**, an approximate computation reuse approach in CNNs to reducing computation workloads. **AxR-NN** uses a layer-wise approximate data pattern matching with relaxed data precision requirements, leading to an enhanced opportunity for computation reuse and can be reconfigured to systematically control the accuracy loss. We implement **AxR-NN** in hardwares using 4T-2FeFET TCAMs. By tuning the parameters such as bit length required for matching and number of stored patterns, **AxR-NN** is able to achieves 30-58% computation energy consumption reduction with only 1-2.5% accuracy degradation on MNIST, EMNIST, and CIFAR-10 dataset. Our future work focuses on an optimized reuse configuration in various types of CNNs like quantized CNNS as well as better CAM designs that can be readily used to achieve better energy-accuracy trade-off.

References

[1] Ahmedullah Aziz, Swapnadip Ghosh, Suman Datta, and Sumeet Kumar Gupta. Physics-based circuit-compatible spice model for ferroelectric transistors. *IEEE Electron Device Letters*, 37(6):805–808, 2016.

[2] Gregory Cohen et al. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[3] Zidong Du et al. Leveraging the error resilience of neural networks for designing highly energy efficient accelerators. *IEEE TCAD*.

[4] S Dünkel et al. A fefet based super-low-power ultra-fast embedded nvm technology for 22nm fdsoi and beyond. In *(IEDM)*, 2017.

[5] Song Han et al. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

[6] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[7] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher W Fletcher. Ucnn: Exploiting computational reuse in deep neural networks via weight repetition. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 674–687. IEEE Press, 2018.

[8] Geoffrey Hinton et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 2012.

[9] Mohsen Imani et al. Efficient neural network acceleration on gpgpu using content addressable memory. In *DATE*. IEEE/ACM, 2017.

[10] Xun Jiao, Vahideh Akhlaghi, Yu Jiang, and Rajesh K Gupta. Energy-efficient neural networks using approximate computation reuse. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1223–1228. IEEE, 2018.

[11] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[12] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[13] Yann LeCun et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[14] Jing Li et al. 1mb 0.41 μm 2 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing. In *VLSIC*. IEEE, 2013.

[15] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[16] Shoun Matsunaga et al. Design of a nine-transistor/two-magnetic-tunnel-junction-cell-based low-energy nonvolatile ternary content-addressable memory. *Japanese Journal of Applied Physics*, 2012.

[17] Vojtech Mrazek et al. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *ICCAD*. IEEE/ACM, 2016.

[18] Kostas Pagiamtzis et al. Content-addressable memory (cam) circuits and architectures: A tutorial and survey. *IEEE JSSC*, 2006.

[19] Matt Poremba et al. Destiny: A tool for modeling emerging 3d nvm and edram caches. In *DATE*. IEEE/ACM, 2015.

[20] Marc Riera, Jose-Maria Arnau, and Antonio González. Computation reuse in dnns by exploiting input similarity. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 57–68. IEEE Press, 2018.

[21] Farkhanda Syed, Zahid Ullah, and Manish K Jaiswal. Fast content updating algorithm for an sram-based tcam on fpga. *IEEE Embedded Systems Letters*, 10(3):73–76, 2017.

[22] Swagath Venkataramani et al. Axnn: energy-efficient neuromorphic systems using approximate computing. In *ISLPED*. ACM, 2014.

[23] Xunzhao Yin, Xiaoming Chen, Michael Niemier, and Xiaobo Sharon Hu. Ferroelectric fets-based nonvolatile logic-in-memory circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(1):159–172, 2018.

[24] Xunzhao Yin, X Sharon Hu, et al. Design and benchmarking of ferroelectric fet based tcam. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 1448–1453. European Design and Automation Association, 2017.

[25] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. Approxann: An approximate computing framework for artificial neural network. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 701–706. EDA Consortium, 2015.