PECAN: A Product-Quantized Content Addressable Memory Network

Jie Ran, Rui Lin, Jason Chun Lok Li, Jiajun Zhou, Ngai Wong
Department of Electrical and Electronic Engineering,
The University of Hong Kong, Hong Kong
Email Address: {jieran, linrui, u3524157, jjzhou, nwong}@eee.hku.hk

Abstract—**A novel deep neural network (DNN) architecture is proposed wherein the filtering and linear transform are realized solely with product quantization (PQ). This results in a natural implementation via content addressable memory (CAM), which transcends regular DNN layer operations and requires only simple table lookup. Two schemes are developed for the end-to-end PQ prototype training, namely, through angle- and distance-based similarities, which differ in their multiplicative and additive natures with different complexity-accuracy tradeoffs. Even more, the distance-based scheme constitutes a truly multiplier-free DNN solution. Experiments confirm the feasibility of such Product-QuantizEd Content Addressable Memory Network (PECAN), which has strong implication on hardware-efficient deployments especially for in-memory computing.**

Index Terms—**product quantization, DNN compression, in-memory computing.**

I. Introduction

Deep neural networks (DNNs) have achieved breakthroughs in various applications including classification [18], object detection [13] and semantic segmentation [24], etc. Nonetheless, the massive amount of parameters and computation make it difficult for both training and inference on edge devices with constrained hardware resources. Numerous efforts have been made to reduce the network complexity while preserving the output accuracy. Among various schemes, some are low-bitwidth neural networks using binary weights [4, 15, 22], replacing the expensive multiplications with cheaper sign flip operations during inference. Some approaches substitute multiplications with additions and bit-wise shifts. AdderNet [2] realizes convolution (in the sense of similarity matching) by l_1-distance between the activation and weights, and maintains competitive output accuracy. ShiftCNN [7] is based on a power-of-two weight representation for converting convolutional neural networks (CNNs) without retraining. Among works that aim to improve the memory efficiency and performance of shift neural networks, DeepShift [6] is a framework for training low-bitwidth neural networks from scratch to replace multiplication with bit-wise shift and sign flip. All these works, despite specific implementations, still adhere to the traditional DNN architecture. This work attempts to detach a neural network from its regular filtering operation and replace it with an associative memory, aka content addressable memory (CAM), whereby the content is derived from prototypes of product quantization [10]. Such framework, dubbed **P**roduct-QuantizEd **C**ontent **A**ddressable Memory **N**etwork (PECAN), combines the storage and compute into one place, and is particularly suitable for the fast-emerging in-memory computing. The codebook/table lookup during inference also makes PECAN hardware-friendly and positions it as a strong candidate for edge artificial intelligence (AI). This is also warranted by the readiness in commodity platforms like FPGAs with CAM support, as well as next-generation memristive microelectronics like resistive random-access memory (RRAM) wherein a CAM is inherent to an RRAM crossbar [11, 16].

Our proposed PECAN is inspired by the lately proposed MADDNESS [1] that utilizes product quantization and table lookup to truly omit multipliers in matrix-matrix products. However, the main contribution of MADDNESS, namely, the hash function for prototype matching, is heuristic and non-differentiable, thus making it incompatible with a learning framework. In fact, the authors also remark it will take several more papers to consolidate the framework for DNNs.

PECAN exactly fills this void by its end-to-end learnable PQ-based DNN architecture. The closest work to ours is differentiable product quantization (DPQ) [3], but *for the first time* we demonstrate its multi-layer feasibility and enrich DPQ prototype matching (viz. a similarity search) with an l_1-distance metric. The latter comes from the lately proposed AdderNet [2] wherein the l_1 metric is utilized in a different context of CNN filtering, whereas our work is the first to show its feasibility for training prototypes in the DPQ setting. To our best knowledge, PECAN is a brand new architecture that transcends regular DNN filtering and uses similarity search and table lookup for inference. This allows it to be compatible with simple hardware without the need of dedicated neural engines, especially edge devices where compute and storage resources are limited. Our major contributions are: 1) A first-of-its-kind, end-to-end learnable CAM-based DNN. PECAN is hardware-generic and friendly to almost all hardware platforms especially those with built-in CAM support, and represents a strong candidate for edge AI deployment; 2) Two similarity measures in PECAN, based on angle and distance, to investigate the trade-offs between computation complexity and accuracy; 3) Joint fine-tuning and co-optimization of weight matrices and PQ prototypes, which permits PECAN to train from scratch; 4) A *totally* multiplier-free DNN via the distance-based PECAN.

II. Related Work

For efficient edge deployment, binary neural networks (BNNs) [9, 15] exclusively make use of the logical XNOR operation that obviates regular multipliers, but in principle they

978-3-9819263-7-8/DATE23/© 2023 EDAA

are still doing 1-bit multiplication. Moreover, though BNNs have gone through major improvements in recent years, their top-1 accuracies measured on large-scale datasets are still noticeably lower than their full-precision counterparts. Indeed, most BNN implementations are only partial in the sense that the first and final layers are still using full-precision weights and activations [21, 22].

Other works replace multiplication with addition [2] or bit-shift operations [6, 7], or both [23]. Specifically, AdderNet makes novel use of l_1-norm difference and adders to do template matching required in a CNN. Yet it still employs multipliers for the necessary batch normalization to bring back signed pre-activations. Progressive kernel based knowledge distillation (PKKD) AdderNet [20] improves the performance of the vanilla AdderNet. AdderNet with Adaptive Weight Normalization (AWN) [5] further alleviates the curse of instability of running mean and variance in batch normalization layers. Applying bitwise shift on an element is mathematically equivalent to multiplying it by a power of two, and sign flipping is introduced to represent negative numbers. Although these works focus on largely multiplier-free DNNs, they still build on the traditional architectures.

The proposed PECAN is motivated by MADDNESS which realizes multiplier-free matrix-matrix product using hashing and table lookup rather than multiply-add operations. Although it achieves orders of speedups compared to existing approximate matrix multiplication (AMM) methods, the proposed hashing functions are not differentiable and not amenable to DNN training. DPQ [3] is proposed for end-to-end embedding, but it is only single-layer and targets word embedding, and still requires full-precision multiplication to obtain distances between the input and matching keys.

III. PECAN

The convolution operation in a CNN is conceptually illustrated as a window sliding across the c_{in}-channel input feature (cf. Fig. 1(a)). Actual implementations often unfold the convolution into a matrix-matrix product (cf. Fig. 1(b)). Specifically, the `im2col` command stretches the input entries covered in each filter stride into a column and concatenates the columns into a matrix X, whereas the kernel tensors are reshaped into a filter matrix F, such that PQ can be used to approximate FX. For an intermediate CNN layer, consider the flattened feature matrix $X \in \mathbb{R}^{c_{in}k^2 \times H_{out}W_{out}}$, where c_{in} and k are the number of input channels and the kernel size, H_{out} and W_{out} are height and width of the output feature, codebooks $C \in \mathbb{R}^{c_{in}k^2 \times p}$ are assigned with parameters to construct an embedding table for the features, where p is the number of choices for each *codebook* $C^{(j)}$, $j = 1, 2, \ldots, D$. $C_m^{(j)} \in \mathbb{R}^d$ are called *prototypes*, $m = 1, 2, \ldots, p$ (cf. Fig. 1(c)). It is natural to set each prototype in PQ to be a $k^2 \times 1$ subvector (viz. same size as a vectorized kernel), with p prototypes in each of the c_{in} input channels according to the patterns of flattened matrices. With this setting, there are two main components in a trained PECAN that require memory storage in each layer, namely, i) pc_{in} prototypes for "quantizing" the input subvectors; ii) $c_{out}c_{in}p$ inner product values between the (sub)rows in F and each prototype.

In short, PECAN is mapping (quantizing) the original input features onto prototype patterns in compact codebooks, then multiplication between weights (F) and features (X) can be approximated by lookup table operation during inference. Below we elaborate two content addressing techniques (i.e. similarity matching) approaches based respectively on angle (dot product) and distance (l_1-norm) which are both end-to-end learnable. Accordingly, these two schemes are dubbed PECAN-A and PECAN-D, which cover both ends of complexity-accuracy spectrum: The angle-based scheme uses multiplicative operations and generally leads to higher output accuracy, whereas the distance-based one uses additive operations and is much more lightweight at the expense of slight accuracy loss.

A. PECAN-A: Angle-Based Similarity Measure

A scaled dot-product attention [19], widely used in Transformers, computes the dot products of queries and keys, followed by a row-wise softmax to obtain the weighted values:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V, \tag{1}$$

where d_k is the dimension of keys, which serves as a scaling factor. Generally, Q, K and V are obtained from three distinct learned projection matrices. However, different from self-attention, we learn the keys K (viz. prototypes in PQ) directly without the intermediate linear transforms, and make V equal to K. For PECAN-A, we compute the approximated matrix $\tilde{X}$ by splitting its rows into $D = c_{in}$ groups, each with subvectors of dimension $d = k^2$, and get the attention scores $K_i^{(j)}$ to formulate the combination of prototypes $C_m^{(j)}$:

$$K_i^{(j)} = \text{softmax}((C^{(j)})^T X_i^{(j)}),\ \tilde{X}_i^{(j)} = C^{(j)} K_i^{(j)}, \tag{2}$$

where $i = 1, 2, \ldots, H_{out}W_{out}$. Since the dot product distance function with softmax is differentiable, mapping features to prototypes can be learned end-to-end. It is worth noting that all intermediate features are replaced with the combination of learned prototypes after training.

B. PECAN-D: Distance-Based Similarity Measure

Now we attempt to get rid of all multipliers. To achieve this, we make use of only l_1-norm difference for the so-called template matching, namely, finding the closest match through absolute difference which involves only subtraction. Specifically, in this distance-based framework, l_1-norm is applied in order to discard multiplication:

$$k_i^{(j)} = \arg\max_m -\|X_i^{(j)} - C_m^{(j)}\|_1, \tilde{X}_i^{(j)} = C^{(j)} \text{one\_hot}(k_i^{(j)}), \tag{3}$$

where $K_i^{(j)}$ = one_hot $(k_i^{(j)})$ denotes a p-dimensional vector with the $k_i^{(j)}$-th entry as 1 and others 0. To enable optimization for prototypes with the non-differentiable function argmax, we approximate it with a differentiable softmax function:

$$\tilde{K}_i^{(j)} = \frac{\exp(-\|X_i^{(j)} - C_m^{(j)}\|_1/\tau)}{\sum_{m'} \exp(-\|X_i^{(j)} - C_{m'}^{(j)}\|_1/\tau)}, \tag{4}$$

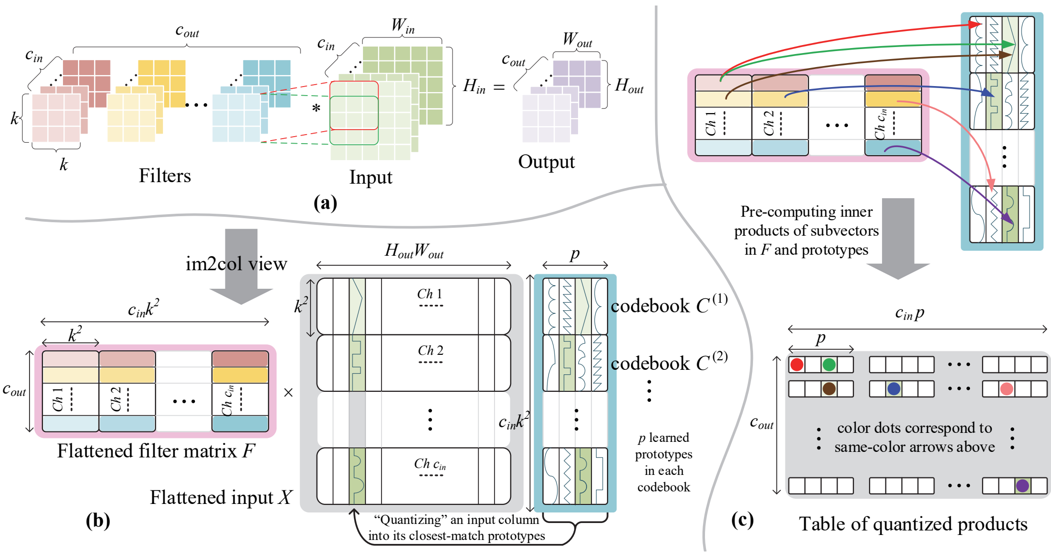

Fig. 1. CNN convolution in its (a) conceptual form; (b) equivalent matrix-matrix-multiply by flattening the filters and input features via `im2col`, with mapping of input data sub-columns onto the closest prototypes in different codebooks; (c) Precomputed inner products of F-subvectors and prototypes in a lookup table.

where τ is the temperature to relax the softmax function. Note that Eq. (4) can be considered as the proportion of Laplacian kernels when $\tau \neq 0$. It relies on the observation that the positive definite function $k(X_i^{(j)}, C_m^{(j)}) = \exp(-\|X_i^{(j)} - C_m^{(j)}\|_1/\tau)$ here defines an inner product and a lifting function ϕ such that the inner product $\langle \phi(X_i^{(j)}), \phi(C_m^{(j)}) \rangle$ can be computed quickly using the kernel trick [14].

Now the approximated index $\tilde{K}_i^{(j)}$ is fully differentiable when $\tau \neq 0$. However, this yields the combination of prototypes for $\tilde{X}_i^{(j)}$ again, while we need $\tau \to 0$ to get discrete indices during the forward inference. To this end, we follow [3] and define a new index to solve both non-differentiable and discrete problems in one go. Specifically, in the forward and backward passes during training, we adopt

$$\tilde{K}_i^{(j)}(\tau \neq 0) - sg\left(\tilde{K}_i^{(j)}(\tau \neq 0) - \tilde{K}_i^{(j)}(\tau = 0)\right), \quad (5)$$

where sg is *stop gradient*, which takes the identity function in the forward pass and drops the gradient inside it in the backward pass. Based on this, we can now use the argmax function in the forward pass and softmax function during backpropagation. However, the partial derivative of the distance $d_{im}^{(j)} = -\|X_i^{(j)} - C_m^{(j)}\|_1$ with respect to codebook subvector $C_m^{(j)}$ is a sign function:

$$\frac{\partial d_{im}^{(j)}}{\partial C_m^{(j)}} = \operatorname{sgn}(X_i^{(j)} - C_m^{(j)}), \quad (6)$$

where $\operatorname{sgn}(\cdot)$ is the sign function and takes the values of $\{+1, 0, -1\}$. Such zero gradient almost everywhere makes it impossible to train a neural network. In this regard, we adopt Eq. (7) to replace the gradient, where e is the current epoch and E the total number of training epochs.

$$\frac{\partial d_{im}^{(j)}}{\partial C_m^{(j)}} = \tanh\left(a(X_i^{(j)} - C_m^{(j)})\right) \text{ where } a = \exp(\frac{4e}{E}), \quad (7)$$

TABLE I
INFERENCE COMPLEXITIES OF PECAN-A AND PECAN-D.

Method	Layer	#Add.	#Mul.
Baseline	CONV	$c_{in}H_{out}W_{out}k^2c_{out}$	$c_{in}H_{out}W_{out}k^2c_{out}$
	FC	$c_{in}c_{out}$	$c_{in}c_{out}$
PECAN-A	CONV	$pDH_{out}W_{out}(d + c_{out})$	$pDH_{out}W_{out}(d + c_{out})$
	FC	$pD(d + c_{out})$	$pD(d + c_{out})$
PECAN-D	CONV	$DH_{out}W_{out}(2pd + c_{out})$	0
	FC	$D(2pd + c_{out})$	0

This epoch-aware approximation to the sign function *w.r.t.* values of $\frac{e}{E}$ as epoch increases during training. In the early stage, the function is smoother for stable training. As the training progresses, the approximation gradually turns into the sign-like function.

C. Inference Details and Complexity

For the original `im2col` convolution, the computation complexity is $O(c_{in}H_{out}W_{out}k^2c_{out})$. During inference, our method includes two stages, the first is to get the indices by computing the distance between the flattened features and prototypes, while the second is to retrieve the product between weights and prototypes computed in advance, i.e., a simple table lookup. The inference algorithm for both PECAN variants is given in Algorithm 1.

Table I illustrates the number of multiplication and addition operations in convolution and fully-connected layers for the traditional CNNs, angle-based and distance-based PECAN during the inference phase. Note that the fully-connected layer can be regarded as a convolution layer when $k = H_{out} = W_{out} = 1$. Instead of using the specialized setting of $D = c_{in}$ and $d = k^2$, we further consider the more general case in Table I where the group number D and dimension of prototypes d satisfy $Dd = c_{in}k^2$. Choosing smaller p and D will reduce the computation complexity for both PECAN-A and PECAN-D. Specifically, in order to limit multiplication complexity in PECAN-A to be

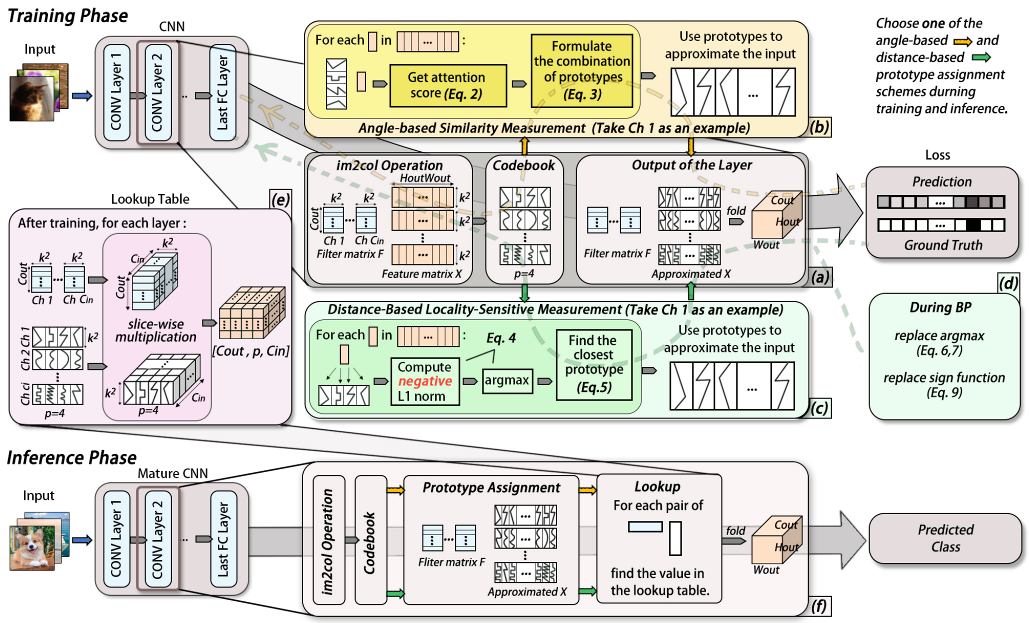

Fig. 2. The proposed PECAN architecture. **(a)** The training phase is mainly composed of template matching for each subvector in the flattened feature map matrices after `im2col` operation. When approximating subvectors with the closest prototypes, PECAN-A and PECAN-D adopt different assignment schemes. **(b)** For PECAN-A, an attention module compares the subvectors with each of the prototypes in the same group. Subsequently, the resulting scores are subjected to the weighted sum to produce the approximate feature matrix. **(c)** For PECAN-D, the similarity is measured with a sign flip l_1-norm and the approximation is selected with `argmax` function. **(d)** Since the `argmax` is not differentiable and the gradient of l_1-norm is discrete $(1, -1, 0)$, we propose Eq. (4, 5) and (7) to do the backpropagation. **(e)** After getting the converged neural network, we calculate the slice-wise product between convolution filters and prototypes, and store the results in the memory. **(f)** In the inference phase, we only need to calculate the distance of feature maps with a small number of prototypes and look up in the stored memory to get the quantized output.

smaller than the baseline, we need $p \leq \min(\lambda c_{out}, (1-\lambda)d)$ with $\lambda \in (0, 1)$. This constraint is also taken into consideration in the experiment section. Note that by design, PECAN-D needs *no multiplication during inference*, thus making it genuinely totally multiplier-less.

IV. Experiments

To demonstrate the effectiveness of PECAN and further benchmark the differences between its two variants (PECAN-A and PECAN-D), we apply PECAN to the classification tasks, taking CIFAR-10 and CIFAR-100 [12] as datasets. The models employed include modified VGG-Small [22], ResNet20 and ResNet32 [8]. We also provide visual results to confirm the approximation capability of the prototypes.

Implementation Details. To implement the PECAN framework for the CIFAR-10 and CIFAR-100 tasks, we use the co-optimization strategy that update the prototypes and weights together. We set the training epochs for PECAN-A and PECAN-D as 150 and 300, respectively. The learning rate for PECAN-A is set to 0.01 initially, decaying every 50 epoch, while that of PECAN-D is initialized as 0.001, decaying at epoch 200. For both datasets, we employ `softmax` function and set the temperature τ at 1 and 0.5 for PECAN-A and PECAN-D, respectively. We set the batch size to 64, and use cross-entropy as the loss function, which is optimized by Adam. All experiments are run on a machine equipped with four NVIDIA Tesla V100 GPU with 24GB frame buffer, and all codes are implemented by PyTorch.

Algorithm 1 Inference Algorithm of PECAN

Input: Codebook $C \in \mathbb{R}^{c_{in}k^2 \times p}$, 4-D learned kernel tensor $\mathcal{K} \in \mathbb{R}^{c_{out} \times c_{in} \times k \times k}$, unfolded features $X \in \mathbb{R}^{c_{in}k^2 \times H_{out}W_{out}}$.

Output: The approximated convolution output $\tilde{Y} \in \mathbb{R}^{c_{out} \times H_{out}W_{out}}$

Permute and reshape weights to $W_1 \in \mathbb{R}^{D \times c_{out} \times d}$, codebooks to $C_1 \in \mathbb{R}^{D \times d \times p}$
for j **in** $\{1, 2, \cdots, D\}$ **do**
  $Y^{(j)} = {W_1}^{(j)} {C_1}^{(j)} \in \mathbb{R}^{c_{out} \times p}$
end for
for i **in** $\{1, 2, \cdots, H_{out}W_{out}\}$ **do**
  if PECAN-A **then**
    $\tilde{Y}_i = \sum_{j=1}^{D} Y^{(j)} \text{softmax}(C^{(j)T} X_i^{(j)})$
  end if
  if PECAN-D **then**
    $k_i^{(j)} = \arg\max_m -\|X_i^{(j)} - C_m^{(j)}\|_1$
    $\tilde{Y}_i = \sum_{j=1}^{D} Y^{(j)}_{k_i^{(j)}}$
  end if
end for
return Concatenate $(\tilde{Y}_1, \tilde{Y}_2, \cdots, \tilde{Y}_{H_{out}W_{out}})$

A. VGG and ResNet on CIFAR-10/100

We evaluate our proposed PECAN using VGG-Small and ResNet20/32 on CIFAR-10 and CIFAR-100. VGG-Small is a simplified VGGNet [17] with only one fully-connected layer. The size of the output feature maps and the corresponding codebook information for each layer are provided in Table II. We remark that the bottom row of each block in the table represents the FC layer, while the rows above represent the CONV layers. For the codebook settings, it is seen that the number of prototypes p used in PECAN-A is much fewer than that of PECAN-D for all five layers. We adopt this setting considering the gaps between the representation capabilities of PECAN-A and PECAN-D. By adjusting the weights assigned to prototypes, PECAN-A is expected to better approximate the features with limited choices, i.e., a smaller p. The number of required addition and multiplication operations and the accuracy of the models are summarized in Table III, where the VGG-Small baseline has $0.61G$ multiplication and addition operations with 91.21% accuracy on CIFAR10. Since batch normalization can be folded into convolution layers in the inference stage, we do not count FLOPs for both baseline and PECAN. Focusing on the third and fourth columns, it is noticeable that PECAN-A has fewer multiplications and additions compared with the baseline, and PECAN-D needs no multiplication at all. We find that PECAN-A only performs $0.54G$ multiplications while reaching 91.82% accuracy on CIFAR-10, which is even higher than the baseline, similar performance can be obtained on CIFAR-100. A possible reason is that PECAN experiences less information loss for shallower CNNs, and bigger input channels allow more groups of prototypes to improve the representation capability. This assumption is also validated by the experiments on ResNet20/32 that are deeper than VGG-Small but with smaller input channels.

TABLE II
THE SETTINGS OF PROTOTYPE NUMBERS AND DIMENSIONS FOR EACH LAYER IN DIFFERENT MODELS FOR PECAN ON CIFAR10 AND CIFAR100
.

Model	#Layers	Output map size	p/d (PECAN-A)	p/d (PECAN-D)
VGG-Small	2	32×32	$16/9$	$32/3$
	2	16×16	$16/32$	$32/3$
	2	8×8	$16/32$	$32/3$
	1	1×1	$16/16$	$32/16$
ResNet20	1	32×32	$8/9$	$128/3$
	6	32×32	$8/9$	$64/3$
	6	16×16	$8/16$	$64/3$
	6	8×8	$8/16$	$64/3$
	1	1×1	$8/16$	$64/4$
ResNet32	1	32×32	$8/9$	$128/3$
	10	32×32	$8/9$	$64/3$
	10	16×16	$8/16$	$64/3$
	10	8×8	$8/16$	$64/3$
	1	1×1	$8/16$	$64/4$

B. Comparison with AdderNet

We compare PECAN-D with AdderNet on VGG-Small in Table IV. It should be emphasized that batch normalization is not taken into consideration in this table, it can not be folded into AdderNet layer so multiplication is indispensable. For VGG-Small, the memory cost is so high that even four NVIDIA Tesla V100 GPUs are not able to train successfully. As shown in the table, the proposed PECAN-D with only 0.37G additions achieves a 90.19% accuracy on VGG-Small.

TABLE III
EXPERIMENT RESULTS ON CIFAR10 AND CIFAR100.

Model	Method	#Add.	#Mul.	Accuracy (CIFAR10/100)
VGG-Small	Baseline	$0.61G$	$0.61G$	91.21% / 67.84%
	PECAN-A	$0.54G$	$0.54G$	91.82% / 69.21%
	PECAN-D	$0.37G$	0	90.19% / 60.43%
ResNet20	Baseline	$40.56M$	$40.56M$	92.55% / 69.55%
	PECAN-A	$38.12M$	$38.12M$	90.32% / 63.15%
	PECAN-D	$211.71M$	0	87.88% / 58.01%
ResNet32	Baseline	$68.86M$	$68.86M$	92.85% / 70.57%
	PECAN-A	$64.20M$	$64.20M$	90.53% / 64.13%
	PECAN-D	$353.27M$	0	88.46% / 58.26%

Here we showcase the efficacy of PECAN on larger models from a hardware perspective. In the Intel VIA Nano 2000 CPU (used in the AdderNet paper), the latency cycles of float multiplication and addition are 4 and 2, respectively. PECAN-D of VGG-Small model will incur ∼720M(cycles) while that of a CNN is ∼3660M. The power consumption ratio of 32bit multiplication and addition units is 4:1. Power-wise and latency-wise, PECAN-D network is more efficient than both AdderNet and regular CNN.

TABLE IV
COMPARISON WITH ADDERNET.

Model	Method	# Mul.	# Add.	Accuracy (%)	Normalized Power	Latency(cycles)
VGG-Small	CNN	$0.61G$	$0.61G$	93.80	8.24	$3.66G$
	AdderNet	0	$1.22G$	N.A.	3.30	$2.44G$
	PECAN-D	0	$0.37G$	90.19	1	$0.72G$

C. Visualization of Prototypes

To inspect the effectiveness of PECAN-D in CNNs, we take the intermediate convolution layers of VGG-Small and plot the patterns of the feature maps before and after replacement. In Fig. 3, we select the first channel of the flattened feature maps and visualize the matrices. The dimension of all subvectors is set as $k^2 = 9$. As can be seen, though the number of prototypes is limited for each convolution layer, the quantized feature maps still preserve the basic patterns after training.

V. CONCLUSION

A brand new DNN architecture called PECAN is proposed which transcends the regular DNN linear transform, and replaces it by product quantization and table lookup. Both angle- and distance-based measures are developed for similarity matching of prototypes in product quantization for different complexity-accuracy tradeoffs. The distance-based PECAN, to our knowledge, is the *first* neural network that is multiplier-less and uses only adders all over. PECAN is end-to-end trainable and infers only through a content addressable memory (CAM)-like, similarity search protocol. It facilitates a lightweight and hardware-generic solution favorable for edge AI, and fits perfectly into the in-memory-computing regime. Experiments have shown that PECAN exhibits accuracies on par with multi-bit networks even without using multipliers. We expect more advancement on top of this interesting PECAN framework will follow after this debut.

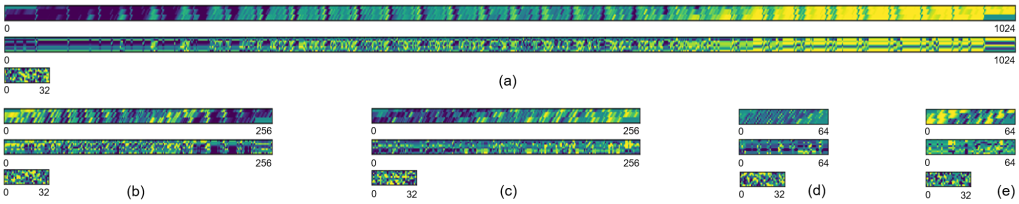

Fig. 3. The flattened features and codebooks for five different layers in VGG-Small, (a)-(e) for conv1-conv5. For each subfigure, the upper image is the input feature after `im2col` operation, the second image shows the approximation matrix after substitution with PECAN-D which is composed of the corresponding codebook shown in the third row. The y-axis is the dimension of each subvector k^2. The x-axis represents the size of output feature maps $H_{out}W_{out}$ for the first two rows, and denotes the number of prototypes for the third row.

ACKNOWLEDGEMENT

This work is supported in part by the Research Grants Council of the Hong Kong Special Administrative Region, China, under the General Research Fund (GRF) projects 17206020 and 17209721.

REFERENCES

[1] Davis Blalock and John Guttag. Multiplying matrices without multiplying. *arXiv preprint arXiv:2106.10860*, 2021.

[2] Hanting Chen, Yunhe Wang, Chunjing Xu, Boxin Shi, Chao Xu, Qi Tian, and Chang Xu. Addernet: Do we really need multiplications in deep learning? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1468–1477, 2020.

[3] Ting Chen, Lala Li, and Yizhou Sun. Differentiable product quantization for end-to-end embedding compression. In *International Conference on Machine Learning*, pages 1617–1626. PMLR, 2020.

[4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[5] Minjing Dong, Yunhe Wang, Xinghao Chen, and Chang Xu. Towards stable and robust addernets. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[6] Mostafa Elhoushi, Zihao Chen, Farhan Shafiq, Ye Henry Tian, and Joey Yiwei Li. Deepshift: Towards multiplication-less neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2359–2368, 2021.

[7] Denis A Gudovskiy and Luca Rigazio. Shiftcnn: Generalized low-precision architecture for inference of convolutional neural networks. *arXiv preprint arXiv:1706.02393*, 2017.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[10] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[11] Geethan Karunaratne, Manuel Schmuck, Manuel Le Gallo, Giovanni Cherubini, Luca Benini, Abu Sebastian, and Abbas Rahimi. Robust high-dimensional memory-augmented neural networks. *Nat. Commun.*, 12, 2021.

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[13] Yudong Liu, Yongtao Wang, Siwei Wang, TingTing Liang, Qijie Zhao, Zhi Tang, and Haibin Ling. Cbnet: A novel composite backbone network architecture for object detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11653–11660, 2020.

[14] Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *NIPS*, volume 3, page 5. Citeseer, 2007.

[15] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[16] Yuan Ren, Rui Lin, Jie Ran, Chang Liu, Chaofan Tao, Zhongrui Wang, Can Li, and Ngai Wong. Batmann: A binarized-all-through memory-augmented neural network for efficient in-memory computing. In *2021 IEEE 14th International Conference on ASIC (ASICON)*, pages 1–4, 2021. doi: 10.1109/ASICON52560.2021.9620292.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[18] Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics*, 50(9):3840–3854, 2020.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[20] Yixing Xu, Chang Xu, Xinghao Chen, Wei Zhang, Chunjing Xu, and Yunhe Wang. Kernel based progressive distillation for adder neural networks. *arXiv preprint arXiv:2009.13044*, 2020.

[21] Yixing Xu, Kai Han, Chang Xu, Yehui Tang, Chunjing Xu, and Yunhe Wang. Learning frequency domain approximation for binary neural networks. *arXiv preprint arXiv:2103.00841*, 2021.

[22] Ping Xue, Yang Lu, Jingfei Chang, Xing Wei, and Zhen Wei. Self-distribution binary neural networks. *arXiv preprint arXiv:2103.02394*, 2021.

[23] Haoran You, Xiaohan Chen, Yongan Zhang, Chaojian Li, Sicheng Li, Zihao Liu, Zhangyang Wang, and Yingyan Lin. Shiftaddnet: A hardware-inspired deep network. *arXiv preprint arXiv:2010.12785*, 2020.

[24] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6881–6890, 2021.