eCNN: A Block-Based and Highly-Parallel CNN Accelerator for Edge Inference

Chao-Tsung Huang
National Tsing Hua University
Taiwan, R.O.C.

Yu-Chun Ding
National Tsing Hua University
Taiwan, R.O.C.

Huan-Ching Wang
National Tsing Hua University
Taiwan, R.O.C.

Chi-Wen Weng
National Tsing Hua University
Taiwan, R.O.C.

Kai-Ping Lin
National Tsing Hua University
Taiwan, R.O.C.

Li-Wei Wang
National Tsing Hua University
Taiwan, R.O.C.

Li-De Chen
National Tsing Hua University
Taiwan, R.O.C.

ABSTRACT

Convolutional neural networks (CNNs) have recently demonstrated superior quality for computational imaging applications. Therefore, they have great potential to revolutionize the image pipelines on cameras and displays. However, it is difficult for conventional CNN accelerators to support ultra-high-resolution videos at the edge due to their considerable DRAM bandwidth and power consumption. Therefore, finding a further memory- and computation-efficient microarchitecture is crucial to speed up this coming revolution.

In this paper, we approach this goal by considering the inference flow, network model, instruction set, and processor design jointly to optimize hardware performance and image quality. We apply a block-based inference flow which can eliminate all the DRAM bandwidth for feature maps and accordingly propose a hardware-oriented network model, ERNet, to optimize image quality based on hardware constraints. Then we devise a coarse-grained instruction set architecture, FBISA, to support power-hungry convolution by massive parallelism. Finally, we implement an embedded processor—eCNN—which accommodates to ERNet and FBISA with a flexible processing architecture. Layout results show that it can support high-quality ERNets for super-resolution and denoising at up to 4K Ultra-HD 30 fps while using only DDR-400 and consuming 6.94W on average. By comparison, the state-of-the-art Diffy uses dual-channel DDR3-2133 and consumes 54.3W to support lower-quality VDSR at Full HD 30 fps. Lastly, we will also present application examples of high-performance style transfer and object recognition to demonstrate the flexibility of eCNN.

CCS CONCEPTS

• **Computer systems organization → Embedded hardware**; • **Computing methodologies → Machine learning**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MICRO-52, October 12–16, 2019, Columbus, OH, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358263

KEYWORDS

convolutional neural network, computational imaging, edge inference, hardware accelerator, ultra-high-definition

ACM Reference Format:
Chao-Tsung Huang, Yu-Chun Ding, Huan-Ching Wang, Chi-Wen Weng, Kai-Ping Lin, Li-Wei Wang, and Li-De Chen. 2019. eCNN: A Block-Based and Highly-Parallel CNN Accelerator for Edge Inference. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3352460.3358263

1 INTRODUCTION

Convolutional neural networks (CNNs) recently draw a lot of attention for their great success in the fields of computer vision and computational imaging. Their hardware accelerators also become an emerging need to enable edge applications. The performance of pixel throughput and inference quality is determined jointly by model structure, processor architecture, and inference flow.

The CNN model structure has evolved mainly for object recognition, e.g. from shallow AlexNet [33] to deep VGGNet with small filters [54] and ResNet with residual connections [21]. Several hardware-oriented variants, like depth-wise convolution [23] and feature squeezing [26, 52], were also proposed to reduce model complexity for edge inference. On the other hand, CNNs have also shown dominant performance for computational imaging applications [39], such as image denoising [18, 62, 63], super-resolution [16, 32, 35, 36], image deblurring [9, 46], and view synthesis [31, 57]. They even can provide novel applications which are hard to achieve using traditional methods, like style transfer [29, 66], DSLR-quality conversion [27], and algorithm mimicking [10]. However, there were seldom discussions on hardware-oriented models for computational imaging despite their potential to enable next-generation image pipelines on edge devices.

On the other hand, several hardware accelerators have been proposed for deep neural networks. For example, DaDianNao [11], Cambricon-ACC [37], TPU [30], and DNPU [53] were designed for general-purpose inference. In contrast, ShiDianNao [17], Eyeriss [12] , and Morph [22] were dedicatedly optimized for classification CNNs. The weight sparsity in these CNNs has been used to reduce computation complexity [3, 48, 64, 65], and the bit sparsity

in activations was deployed in [2]. Another approach for saving complexity is to use low-precision computation, such as dynamic fixed-point format [45, 49] and even binary networks [5]. However, these accelerators are not optimized for computational imaging and also not for high-resolution videos, especially in terms of DRAM bandwidth and computing capability. Recently, Diffy [41] attacked this problem by utilizing the bit sparsity in activation differences to reduce DRAM access and computing power. But many Diffy tiles and high-end DRAM settings are still required for Full-HD videos.

Finally, the inference flow of a given CNN model determines the data reuse scheme for an accelerator and thus its memory access efficiency. A systematic approach to partition CNNs into computation sub-blocks was introduced in [59], and several energy-efficient dataflows were analyzed in [12]. In particular, a line-based flow was considered for layer fusion in [4] which avoids external traffics for feature maps by applying pyramid inference on moving blocks. For the overlapped features between blocks, a reuse scheme was chosen for fusing up to five CNN layers. However, the line buffer size will increase linearly with model depth, image width, and channel number. For example, 9.3MB of SRAM will be required for supporting VDSR [32] in Full HD resolution. Similar trade-offs between the on-chip SRAM size and off-chip DRAM bandwidth have also been widely studied for image processing applications, such as motion estimation [56] and discrete wavelet transform [24].

In this work, we aim to enable high-quality edge inference at up to 4K Ultra-HD (UHD) 30 fps for computational imaging tasks. In particular, we target low-end DRAM settings for cost-effective and power-efficient integration on embedded devices. We found this challenging goal is hard to achieve by directly accelerating state-of-the-art models which mostly have wide features and deep layers. Instead, we expand our design space to consider the inference flow, model structure, instruction set, and processor design jointly for optimizing both hardware performance and image quality.

We first propose a block-based truncated-pyramid inference flow which can eliminate all the DRAM bandwidth for feature maps by storing them in on-chip block buffers. To avoid huge on-chip storage, we choose to recompute the overlapped results between neighboring blocks. The block buffer size is proportional to model width, and the recomputation overhead almost increases quadratically with model depth. As a result, these two factors defy the rule of thumb that simply adds more features and more layers to enhance model quality. Instead, we propose a novel ERNet model to optimize CNNs under these hardware constraints. Then we construct a feature-block instruction set architecture (FBISA) to support highly-parallel convolution. It specifies block-buffer-level operations in the fashion of Single Instruction, Multiple Data (SIMD). In addition, it provides flexibility for programmers and compilers to optimize the computing flow based on different constraints. Finally, we implement an embedded CNN processor—eCNN—which flexibly accommodates to ERNet and FBISA with highly-parallel filters and locally-distributed parameters.

In summary, the main contributions and findings of this work are:

- We propose a block-based flow to enable high-resolution inference with low DRAM bandwidth and also analyze its computation and bandwidth overheads. (Section 3)
- We propose a hardware-aware ERNet to optimize image quality based on hardware constraints and also build training procedures for model optimization and dynamic fixed-point precision. (Section 4)
- We devise a coarse-grained FBISA with parallel parameter bitstreams to provide massive computation parallelism efficiently and flexibly. (Section 5)
- We design an embedded processor, eCNN, to support FBISA with highly-parallel convolution using 81,920 multipliers. (Section 6)
- We train ERNets for image super-resolution (SR) and denoising with 8-bit precision. In particular, the quality for four-times SR can outperform VDSR [32] by 0.57 dB and 0.44 dB in PSNR when eCNN delivers Full HD and 4K UHD 30 fps respectively. (Section 7.1)
- Layout results show that eCNN can achieve 41 TOPS (tera operations per second) on 40 nm technology. It supports high-quality ERNets at up to 4K UHD 30 fps while consuming 6.94W and using DDR-400. By comparison, Diffy consumes 54.3W and uses dual-channel DDR3-2133 for VDSR at Full HD 30 fps. (Section 7.2)
- Computer-vision tasks can also be well supported by FBISA-compatible models, such as style transfer and object recognition. (Section 7.3)

2 MOTIVATION

Recent research on CNN accelerators mainly focuses on object recognition/detection networks. Therefore, two specific features for computational imaging networks are not considered for optimization: 1) the spatial resolution of feature maps is not aggressively downsampled and 2) the models are not very sparse. The former results in a dramatically-high amount of memory bandwidth, and the latter introduces an extremely-high demand of computing power.

The aggressive downsampling for object recognition is as shown in Fig. 1(a). It can extract high-level features and also reduce the data amount for feature maps (volume of cuboids) in deeper layers. Most of conventional accelerators can thus apply a frame-based inference flow to perform convolution layer-by-layer with limited DRAM bandwidth. However, this flow will induce a huge amount of DRAM bandwidth for computational imaging networks. It is because high-resolution feature maps are required to generate texture details and only mild downsampling is allowed [39]. Take the plain network without downsampling in Fig. 1(b) as an example, its corresponding DRAM bandwidth for feature maps, except input and output images, can be derived as

$$H \times W \times C \times (D-1) \times fR \times L \times 2, \tag{1}$$

where H stands for image height, W for image width, C for the number of feature channels (model width), D for model depth, fR for frame rate, L for the bit length of each feature, and the factor 2 for writing per-layer feature maps into DRAM and then loading back for the next layer. Accordingly, the 20-layer 64-channel VDSR will require 303 GB/s of memory bandwidth for Full HD 30 fps when using 16-bit features. Even with the state-of-the-art compression, Diffy still requires dual-channel DDR3-2133 (34 GB/s) to meet this Full-HD specification. When the resolution is raised to 4K UHD, the

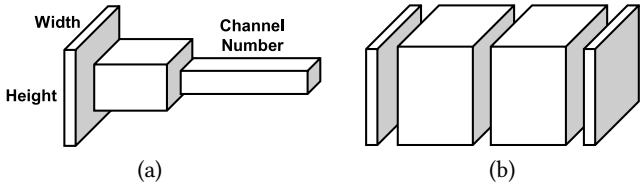


Figure 1: CNN examples in per-layer feature maps for (a) object recognition and (b) computational imaging.

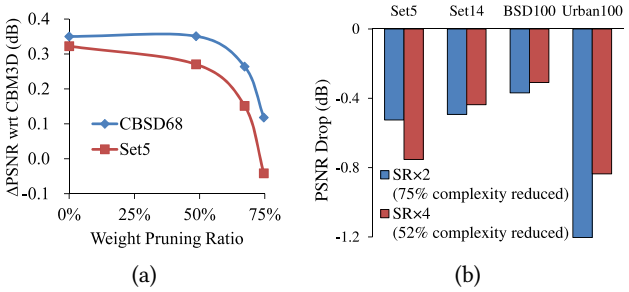


Figure 2: Quality degradation of computational imaging networks for using sparsity techniques. (a) Weight pruning for DnERNet-B16R1N0. (b) Depth-wise convolution in residual blocks for EDSR-baseline (SR×2 and SR×4).

DRAM bandwidth will be four times larger and thus unaffordable for small-form-factor and power-limited edge devices.

On the other hand, the sparsity of object recognition models has been deployed to develop many complexity-saving techniques, such as weight pruning [20] and depth-wise convolution [23]. However, computational imaging networks rely on the variety of parameters to extract local features and generate fine textures. Their image quality is highly related to the model size, so the sparsity techniques could result in significant degradation. Two such examples are shown in Fig. 2. One is pruning weights for a denoising DnERNet model (Section 7). When pruning 75% of weights away, its PSNR gain over the benchmark CBM3D [14] drops by 0.2-0.4 dB for two datasets (CBSD68 [43] and Set5 [7]) and could even become negative. Another example is using depth-wise convolution for EDSR-baseline models [36]. Although 52-75% of complexity can be saved, the quality drop is 0.3-1.2 dB for four datasets (Set5, Set14 [61], BSD100 [43], and Urban100 [25]) and thus makes the saving unjustified. Therefore, we need to confront the computation demand for computational imaging CNNs. Furthermore, high-resolution image generation will make this demand more challenging. For example, VDSR already demands as high as 83 TOPS for Full HD real-time applications and will require 332 TOPS for 4K UHD.

The issues of huge DRAM bandwidth and computing power motivate us to find a novel approach for ultra-high-resolution CNN acceleration. In the following, we will propose the block-based inference flow and the hardware-oriented ERNet to resolve the memory issue. And the computation issue will be addressed by the coarse-grained FBISA and the corresponding highly-parallel eCNN processor.

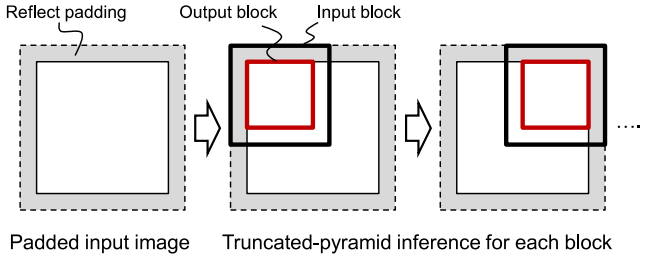
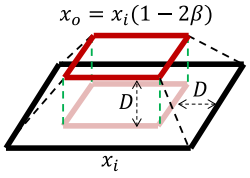


Figure 3: Proposed block-based inference flow.



Filter:	3×3
Depth:	D
Receptive field:	$(2D+1) \times (2D+1)$
Input block:	$x_i \times x_i$
Output block:	$x_o \times x_o$
Depth-input ratio:	$\beta = D/x_i$
Block buffer size:	$CLX_i^2,\ X_i = \max x_i$

Figure 4: Truncated-pyramid inference for a plain CNN.

3 BLOCK-BASED INFERENCE FLOW

The proposed block-based flow is shown in Fig. 3. An input image is partitioned into several blocks which can be processed independently, and all the output blocks are then stitched to form a final output image. In contrast to layer fusion [4], we recompute block-overlapped features to avoid huge on-chip SRAM. However, this induces additional bandwidth and computation. To reduce these overheads, we then propose the truncated-pyramid inference (output block larger than one pixel) as detailed in the following.

To analyze the overheads of this inference flow, we use the plain network in Fig. 4 as an example. It consists of only CONV3×3 layers, and the receptive field is thus linked directly to the depth D. As the convolution goes to deeper (upper) layers, the effective region will become smaller as a truncated pyramid. For example, an $x_i \times x_i$ input block will generate an $x_o \times x_o$ output block where $x_o = x_i - 2D$. When the depth (receptive field) is increased, more input blocks and thus more DRAM bandwidth will be required because fewer output pixels are generated for the same input block size. This bandwidth overhead can be evaluated by a normalized bandwidth ratio (NBR) which is equal to the bandwidth for all input and output blocks over that for an output image:

$$NBR = \frac{3x_o^2 + 3x_i^2}{3x_o^2} = 1 + \frac{1}{(1-2\beta)^2}, \tag{2}$$

where RGB images are considered and β is a depth-input ratio, D/x_i. Similarly, there are also computation overheads for the recomputed features among neighboring blocks. It can be evaluated by a normalized computation ratio (NCR) which represents the computation complexity of this block-based flow over that of the frame-based one (intrinsic):

$$NCR = \frac{\text{volume of truncated pyramid}}{\text{volume of center cuboid}} = \frac{1}{3} + \frac{2}{3}\frac{1-\beta}{(1-2\beta)^2}, \tag{3}$$

where the volume (in Fig. 4) is proportional to the amount of features and therefore that of computing operations.

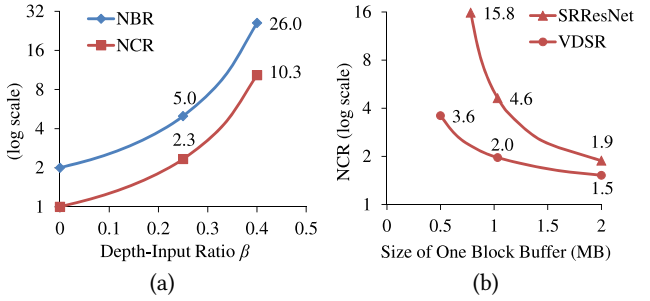


Figure 5: Bandwidth and computation overheads for truncated-pyramid inference. (a) NBR and NCR versus depth-input ratio for the plain network. (b) NCR versus block buffer size for 20-layer VDSR and 37-layer SRResNet. ($L = 16$)

These two ratios both grow rapidly with respect to the depth-input ratio β as shown in Fig. 5(a), and they eventually go to infinity when $\beta = 0.5$ for $x_o = 0$, i.e. no valid output pixels. If β is not too close to 0.5, the induced bandwidth overhead is generally acceptable compared to the bandwidth we save. For example, the NBR is 26× for a large $\beta = 0.4$ while the bandwidth overhead of the frame-based flow can be derived as $\frac{2C(D-1)}{3}$ based on (1) and is as high as 811× for VDSR. However, the computation overhead does become the main side effect of the block-based flow. We will spend 90% of the computing power for feature recomputation when β approaches 0.4. To avoid this situation, we will need to adopt larger block buffers to reduce β for deeper networks.

The evaluation of the above plain example can be extended to more complicated models, and the conclusions on bandwidth and computation overheads are similar. Most of state-of-the-art networks have deep layers of 3×3 (or larger) filters and wide channels of feature maps, e.g. $C \geq 64$. Therefore, they will either require huge block buffers of size CLX_i^2 to reduce NCR or suffer significant computation overheads for using small buffers to save area. Note that usually more than one block buffer will be required for switching between input and output layers, which makes the area cost more severe.

Fig. 5(b) shows this trade-off between the NCR and block buffer size for VDSR and also a state-of-the-art SRResNet [35] which outperforms VDSR by 0.6 dB [36]. The NCR for the 20-layer VDSR is well controlled as 2× using 1MB block buffers. But the 37-layer SRResNet needs around 2MB to have a similar NCR. Using smaller block buffers to save area for SRResNet will make the NCR skyrocket quickly. Therefore, with the block-based flow it is difficult to have a low NCR and use small buffers simultaneously for deep high-quality networks. In the following, we will achieve this goal by considering these hardware constraints as early as model construction and accordingly introduce the ERNet.

4 ERNet

We will first introduce the basic building modules of ERNet and then present a procedure for model optimization. The quantization method we adopt for dynamic fixed-point precision will also be discussed.

4.1 Model Structure

Consider a thin network which can use small block buffers. To increase its capacity without enlarging the NCR and buffer area, we explore another direction of model construction by temporarily expanding the model width. This is achieved by the ERModule shown in Fig. 6(a). It uses a CONV3×3 layer to expand the model width by R_m times and a following CONV1×1 to reduce it back. A residual connection is added for robust training. All the operations are performed internally without accessing to block buffers. Therefore, we can pump complexity into ERModule to improve image quality with the same block buffer size and model depth.

We only consider integer expansion ratios for R_m to guarantee high hardware utilization. To increase model flexibility, we further construct a larger building block by connecting B ERModules as shown in Fig. 6(b). The first N modules can be assigned an incremented $R_m = R + 1$ to make the overall expansion ratio R_E as a fraction $R\frac{N}{B}$. Accordingly, we now have two model hyper-parameters to build networks: B for increasing depth and R_E for pumping complexity.

Fig. 7 shows a model example, SR4ERNet, for performing four-times SR. It basically replaces the residual blocks in SRResNet [35] or EDSR-baseline [36] by ERModules. It starts with small images of 1/4 size in width and height and uses two pixel-shuffle upsamplers to restore full resolution. In addition, we reduce the channel number from 64 to 32 for saving area for block buffers.

4.2 Model Optimization

The major hardware constraint considered here is the overall computation complexity, i.e. NCR×(intrinsic complexity), since the bandwidth overhead is usually small. Each complexity target will correspond to a real-time throughput, and we aim to optimize image quality under such constraints. Our model selection procedure can be illustrated using SR4ERNet as shown in Fig. 8. We assume the size of input blocks is 128×128 and consider three computation constraints: 164, 328, and 655 KOP/pixel (thousand operations per output pixel).

First of all, we derive the largest possible expansion ratio R_E for each module number B under each constraint, and we choose $R_E \leq 4$ as a system upper bound. As shown at the top of Fig. 8, R_E will decrease quickly as the model depth grows with B because of the fast-increasing NCR. In the case of 655 KOP/pixel, NCR can be as high as 2.8-5.9×, and the corresponding intrinsic complexity is as low as 223-107 KOP/pixel. Note that deeper networks do not necessarily perform better now because of their lower intrinsic complexity. Then we scan all of these candidate models with a lightweight training setting, e.g. using smaller patches and fewer mini-batches. After that, we test their image quality using validation datasets and pick the best model for each constraint as shown at the bottom of Fig. 8. Finally, we will further polish the best models by retraining them with a full setting. In this example, the highest-quality SR4ERNet-B34R4N0 can even outperform SRResNet by 0.04 dB using a thinner and less-complex network.

4.3 Dynamic Fixed-Point Precision

We further quantize the polished models for saving computation logics and on-chip memory. The multiplications and block buffers

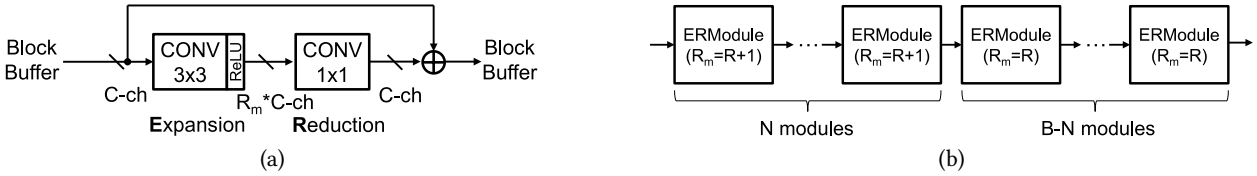


Figure 6: Basic building modules of ERNet: (a) ERModule and (b) connected ERModules ($R_E = R\frac{N}{B}$).

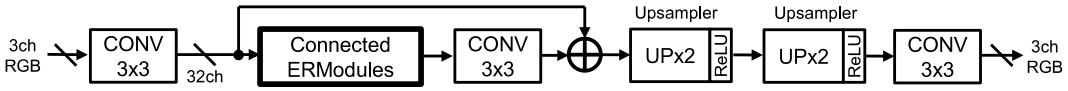


Figure 7: SR4ERNet for four-times SR.

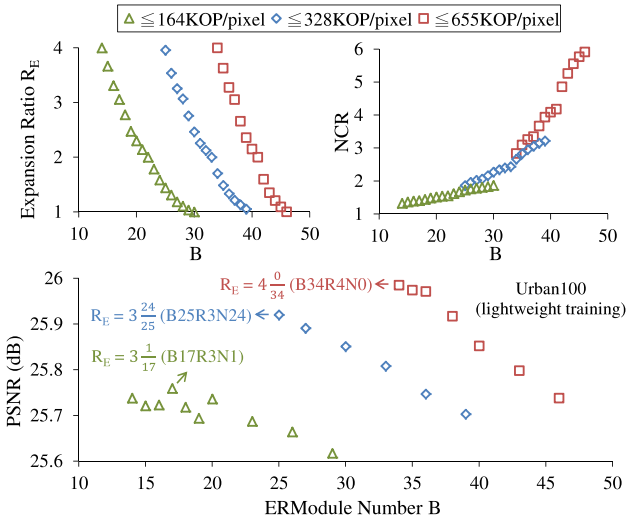


Figure 8: Model scanning of SR4ERNet for three computation constraints with $x_i = 128$.

Figure 9: Examples of 8-bit Q-formats: Qn and UQn.

are both considered in 8-bit precision while the internal partial sums are accumulated in full precision to preserve quality. We adopt the fixed-point Q-format for decimals as illustrated in Fig. 9. Qn and UQn stand for signed and unsigned values respectively, and n is the fractional position of the last effective bit. We apply dynamic fixed-point precision to optimize image quality, so each convolution layer has its own Q-formats for weights, biases, and feature outputs, respectively. We then build a two-stage procedure, quantization and fine-tuning, based on [6, 19, 49].

The quantization stage is to determine the best fractional precision $\hat{n}$ of each Q-format. With a collection Ω of the corresponding floating-point values, we can use either L1-norm [49] or L2-norm [6] errors for the optimization:

$$\hat{n}_l = \arg\min_n \sum_{x\in\Omega} |x - Q_n(x)|^l, \quad l \in \{1, 2\}, \tag{4}$$

where the quantization function $Q_n(\cdot)$ performs clipping and rounding for precision n. The value distributions of parameters are directly derived from the floating-point model, and those of feature maps are collected by inferencing on the training dataset. Since this 8-bit quantization induces up to 3.69 dB of PSNR loss for denoising and SR, we then use the fine-tuning method in [19] to refine these quantized parameters. For calculating gradients more accurately, we add clipped ReLU (rectified linear unit) functions to the model for the clipping behavior of $Q_n(\cdot)$. As a result, the fixed-point ERNet has only 0.08 dB of PSNR degradation on average.

5 FBISA

We design the SIMD instruction set, FBISA, to support the truncated-pyramid inference for fully convolutional networks. To increase its flexibility, we also include a zero-padded inference type and a variant of upsamplers and downsamplers. FBISA provides massive parallelism by coarse-grained instructions between feature blocks and internal accessing of parameter memories which will be introduced sequentially.

5.1 Instruction Set

Fig. 10 shows the instruction format. An opcode can specify a convolution task with specific attributes, e.g. inference type and block size. There are two kinds of operands for features and parameters, respectively, and they also have their own attributes, in particular for Q-formats. For the feature operands, there are two mandatory types to inform the source (*src*) and destination (*dst*) of the convolution. In addition, two supplementary ones (*srcS*/*dstS*) are designed to support feature accumulation among instructions, such as skip/residual connection or partial sums. Finally, the parameter operand specifies where to access the corresponding weights and biases in parameter memories for the opcode. For these operands, we use named expressions, instead of the conventional ordered ones, to improve readability.

Table 1 provides an overview of the instruction set. The smallest computing task in FBISA is called a leaf-module, and it performs a 32ch-to-32ch CONV3×3 filter on one feature block. Each opcode

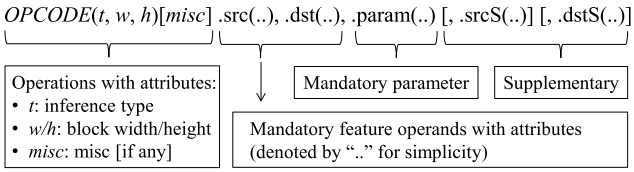


Figure 10: FBISA instruction format.

OPCODE	Description	Leaf Num
ER	32ch ERModule (R_m=1-4)	1-4
CONV3X3	32ch CONV3×3 (†)	1/2/4
UPX2	32ch upsampler (pixel-shuffle)	4
DNX2	32ch downsampler	1
DNX2_DI	32ch downsampler (DI > 128×128)	1
DNX2_CHX2	Downsampler for doubling width (†)	2/4
UPX2_CHD2	Upsampler for halving width (†)	2/4
Operand Type	**Description**	**Operand**
src	Soure of feature input	BB[#]/DI
dst	Destination of feature output	BB[#]/DO
srcS	Source for accumulation	BB[#]
dstS	Destination for copying src	BB[#]
param	Parameter memory (PM) setting	PM

† Basic leaf-modules for building wider filters for 64ch/128ch/256ch/512ch output.

Table 1: FBISA instruction overview.

can contain up to four leaf-modules based on its attribute. The opcodes mainly differ on their usage purposes and thus on the post-processing of outputs. For example, the opcode *UPX2* shuffles pixels for spatial upsampling while *DNX2* performs strided- or max-pooling for downsampling. In particular, *ER* is devised specifically for ERModule and its leaf-module has an additional 32ch CONV1×1 for feature reduction. If wider filters are required for CONV3×3, they can be constructed by using 32ch-based opcodes and accumulating partial sums via *srcS*.

Regarding the feature operands, we apply two strategies to provide efficient data movement for highly-parallel convolution. First, they are specified on the basis of block buffers (BBs), instead of conventional small registers or vectors. Therefore, the internal partial sums for one instruction can be accumulated inside hardware-optimized datapaths without accessing to large SRAM or DRAM which is mostly bandwidth-limited and power-hungry. Another advantage is that we can have small-sized programs and avoid complex compilers. For example, the high-quality SR4ERNet-B34R4N0 uses only 45 lines of instructions.

The second strategy is *not* using conventional load-store instructions for external feature reading and writing. Instead, we devise operands *DI* and *DO* as virtual block buffers for data input and output respectively. They can be implemented by FIFO interfaces and stream data in the same way as normal block buffers. Therefore, the processor pipeline can be fully optimized for a computation-only instruction set. This strategy also decouples FBISA from the data structure in the main memory for better system integration portability.

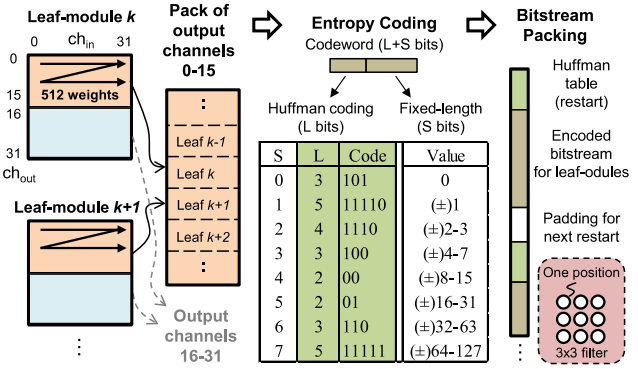


Figure 11: Weight bitstream for one filter position.

5.2 Parameter Format

The data movement of parameters is also of paramount importance for CNN acceleration. To avoid retransmitting parameters for each block, we keep them in internal parameter memories for reuse. This is feasible thanks to the small-sized computational imaging networks. For example, the numbers of parameters in VDSR and SRResNet are 651K and 1479K respectively while it is 11M for ResNet-18 [21].

In FBISA, we split the filter weights into 20 bistreams to enable parallel loading and distribution of them in the processor: 18 for CONV3×3 and two for CONV1×1. Each spatial filter position corresponds to two bitstreams for its first and second halves of output channels in leaf-modules. Fig. 11 shows the format of one such bitstream. The weights are compressed to increase the supported model size, and we adopt the DC Huffman coding in JPEG [28]. This simple coding algorithm enables fast and parallel decoding with small hardware overheads. We also found that the weights are mostly uncorrelated, so differential encoding is unnecessary. On the other hand, the filter biases are gathered in another one bistream and compressed in the same way.

A decoding restart mechanism is further devised to enable parameter reuse between different instructions. In this case, a byte-aligned address referred to the bias bitstream should be specified as the restart attribute in the parameter operand. The Huffman table will be placed first and followed by the encoded bitstream. For the 20 weight bitstreams, their restart addresses will be synchronized to 8× of the restart attribute. It is because each of them contains 512 coefficients for one leaf-module but the bias bistream only has 64. Finally, the 21 bistreams are synchronized for each restart segment by padding shorter ones. Regarding compression efficiency, we found that one Huffman table for each restart segment in each bitstream is sufficient since the 8-bit quantized parameters have similar distributions. As a result, the compression ratio is around 1.1-1.5× for denoising and SR ERNets.

6 eCNN

This embedded processor is implemented to support FBISA with highly-parallel convolution for high-performance and also power-efficient computing. In the following, we will first present its system architecture for top control and embedded integration. Then we will

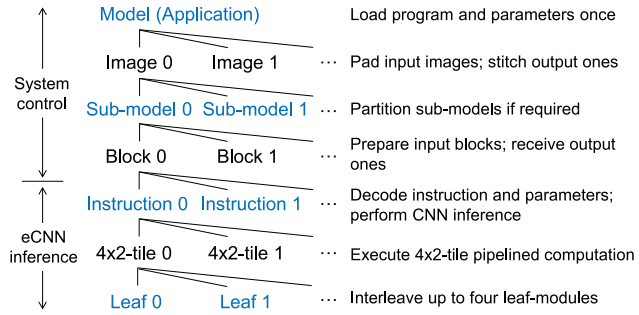


Figure 12: Processing flow for system control and eCNN.

introduce its two main functional units for distributing parameters and performing convolution, respectively.

6.1 System Architecture

6.1.1 Processing Flow. For one target model, its program and parameters are loaded into eCNN only once. Then the inference for each image is performed based on a model hierarchy: sub-model(s), instructions, and leaf-modules. On the other head, the image is also processed with a pixel-grouping hierarchy: blocks and tiles (4×2 32ch features). These two hierarchies are interleaved to form a flexible processing flow shown in Fig. 12. In particular, a deep model can be partitioned into few shallower sub-models to reduce computation overheads. However, the intermediate features between them may increase DRAM bandwidth sharply, which is a performance tradeoff.

The eCNN is designed for embedded integration, e.g. controlled by a main processor and connected to a DMA controller via FIFO interfaces. These system transactions can be handled on a block basis without inducing heavy system burdens. The eCNN then accelerates the computation for each block in an instruction-by-instruction fashion. For each instruction, it calculates one 32ch leaf-module for a 4×2-tile in one cycle. And, for each tile, the leaf-modules of this same instruction are calculated consecutively to accumulate their partial sums on-the-fly without precision loss and SRAM access. After all of the specified 4×2-tiles are processed, the eCNN will repeat similar acceleration for the next instruction.

6.1.2 Block Diagram. Fig. 13 shows the system block diagram to implement the above-mentioned processing flow. It consists of two functional units: information decode unit (IDU) and CNN inference unit (CIU). The IDU is responsible to decode instructions and parameters, and the CIU computes the corresponding convolution. To enable highly-parallel computing, we deploy a massive amount of multipliers in two convolution engines in CIU: LCONV3×3 and LCONV1×1. They perform the 32ch CONV3×3 and CONV1×1, respectively, in each leaf-module, and the latter is used for ERModule.

We keep all of the model parameters in IDU to avoid excessive external bandwidth for parameter retransmission. However, the multipliers need to access up to 10,240 weights for each leaf-module in one single cycle. The throughput is much higher than the affordable bandwidth of the parameter memories. Therefore, we devise an instruction pipelining scheme to distribute parameters efficiently. As a result, the IDU has a whole pipeline stage to progressively decode the parameters for one instruction. Meanwhile, they are sequentially sent to the locally-distributed registers inside the multipliers of CIU and will be used for the convolution in the next pipeline stage. In the following, we will introduce the implementation details for the IDU and CIU.

6.2 Information Decode Unit (IDU)

For each instruction, the IDU will first decode its opcode and operands and then trigger a parameter decompression procedure. The 21 parameter bitstreams mentioned in Section 5.2 are stored in 21 corresponding memories and decoded by 21 parallel decoders. For each leaf-module, a weight decoder is responsible to decode 512 weights while the bias one generates at most 64 biases. All the parameters follow a ping-pong distribution scheme between the IDU and CIU.

Fig. 14 shows the distribution scheme for the weights in the first half of output channels in CONV3×3, and the other cases are similar. Nine decoders are deployed for nine filter positions, and each one decodes two weights in one cycle for two input channels. The decoded weights are then distributed through a two-stage network: the first and second stages are for output and input channels respectively. In particular, there are 16×32 local register files (Weight 2D 3×3) accompanied with their corresponding 2D filters (Filter 2D 3×3). Each register file will keep the decoded parameters in a ping-pong fashion for the CIU convolution in the next instruction pipeline. Also, it can switch between four leaf-modules for the consecutive computation in one instruction. In most cases, the IDU decodes one leaf-module in 256 cycles and completes one instruction faster than the CIU of which the run time is proportional to the number of 4×2-tiles.

6.3 CNN Inference Unit (CIU)

All of the computation in the CIU goes through an inference datapath which is closely coupled to the two convolution engines and three block buffers (BBs) as shown in Fig. 13. The details of these designs are discussed as follows.

6.3.1 Tile-Pipelined Inference Datapath Engine. The highly-parallel convolution is prone to inefficiency of data movement and inflexibility of model supporting. Thus we carefully designed the inference datapath engine to alleviate these issues. It follows a 32ch 4×2-tile pipeline as shown in Fig. 15 and mainly consists of two functions: input preparation for the LCONV3×3 engine and output post-processing for different opcodes and operands.

The first function prepares input 6×4-tiles for 3×3 filtering. However, a 6×4-tile is three times as large as a 4×2-tile and thus induces heavy bandwidth for block buffers. To reduce the bandwidth, we only read 4×2-tiles from block buffers and store them in a line FIFO buffer. Then for each leaf-module the corresponding 6×4-tile can be rearranged in a register file (RF6×4), and up to four leaf-modules are supported. In addition, a data reordering circuit (Src Reorder) is used to address a tile misalignment issue (Section 6.3.3).

The second function, output post-processing, provides model flexibility and there are five sub-functions supported: 1) ERModule through the LCONV1×1 engine; 2) Accumulation which uses an adder (ADDE) for calculating cross-instruction partial sums and another adder (ACCI) for internal ones; 3) Upsampling which writes

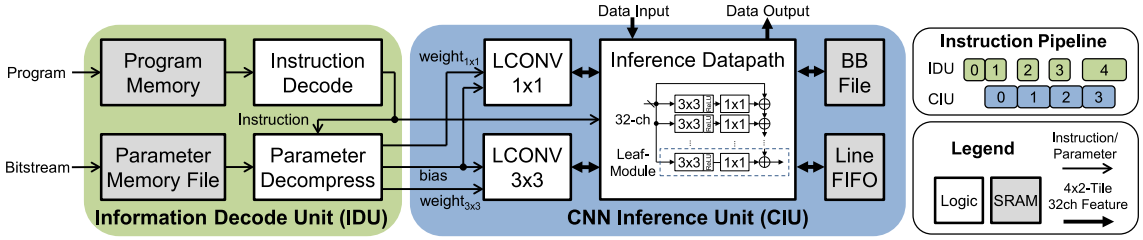


Figure 13: eCNN system block diagram and instruction pipelining scheme.

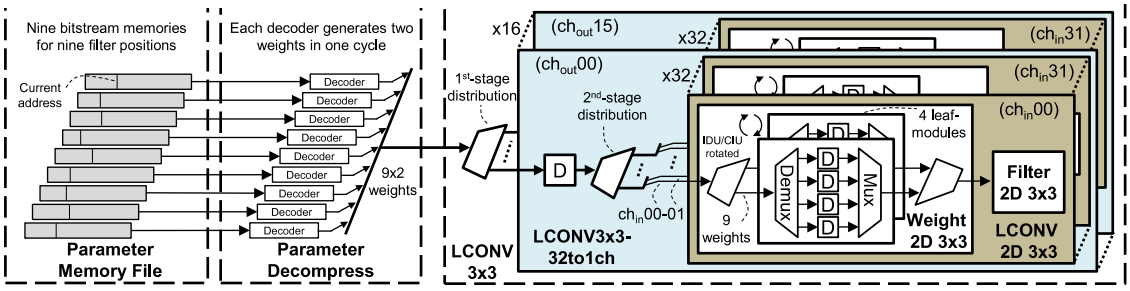


Figure 14: Weight distribution scheme for the first half of output channels in the LCONV3×3 engine.

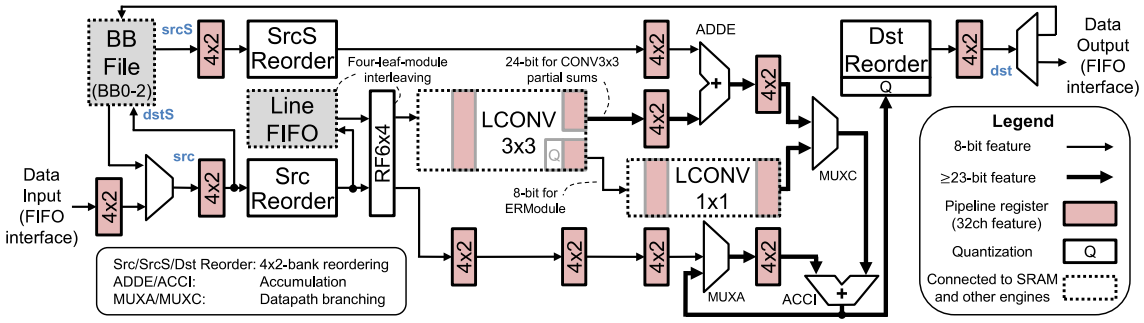


Figure 15: Inference datapath engine.

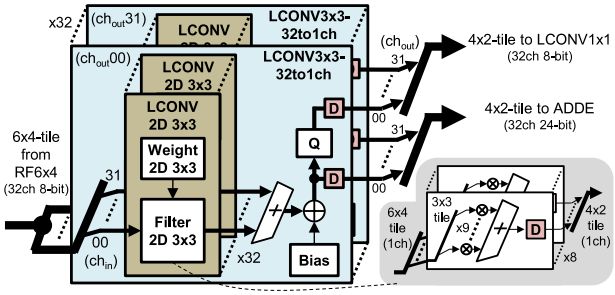


Figure 16: LCONV3×3 engine.

data in pixel-shuffle order (by Dst Reorder); 4) Downsampling for strided- or max-pooling (also by Dst Reorder); 5) Quantization which quantizes features or partial sums to their 8-bit Q-format before going to block buffers or output FIFO interface. In addition, one 8-bit quantization circuit is used inside LCONV3×3 to reduce the input bitwidth of LCONV1×1 for saving area.

6.3.2 Highly-Parallel Convolution Engine. The LCONV3×3 and LCONV1×1 engines serve a 4×2-tile in one cycle for 3×3 and 1×1 filtering respectively. They employ the weight-stationary strategy [12] to optimize data reuse for the block-based inference. Also, compared to the conventional accelerators with much fewer multipliers, their massive parallelism enables power-efficient accumulation of internal partial sums. It is because their communications are all locally hardwired without going through additional register files, SRAM, or DRAM. For example, the LCONV3×3 engine contains 32×32 2D filters (Filter 2D 3×3) as shown in Fig. 16. And each 2D filter shares the same 3×3 weights for a 4×2-tile and reuses them for a block.

6.3.3 Eight-Bank Block Buffer Mapping. The highly-parallel data movement also brings a misalignment issue for the block buffers: the features are stored in 4×2-tiles but their accesses are not always

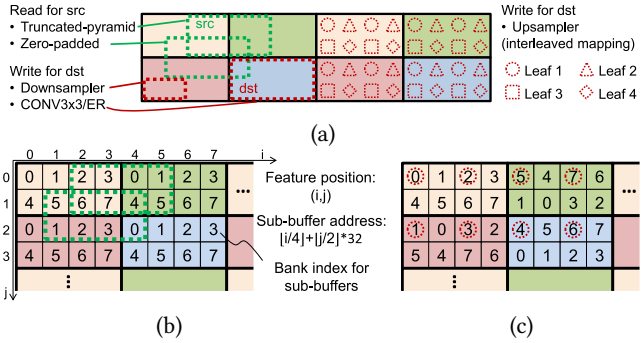


Figure 17: Eight-bank BB implementation. (a) Access patterns for reading and writing 4×2-tiles in a block. (b) Normal bank mapping. (c) Interleaved bank mapping.

System		On-Chip Memory		Real-Time Computation	
Technology	40 nm	Program	6KB	UHD30	4K UHD 30fps (164KOP/pixel)
Frequency	250 MHz	Parameter	1288KB		
Performance	41 TOPS	Block Buffer File	1536KB (3 BBs)	HD60	Full HD 60fps (328KOP/pixel)
MAC Number	81,920				
Q-format	8-bit	Line FIFO	34KB (4-leaf)	HD30	Full HD 30fps (655KOP/pixel)
BB size (32ch)	128×128				

Table 2: eCNN configurations.

Stage	Model	Dataset	Patch	Batch	Batch #	Learning Rate
Model Scanning	SR/Dn	DIV2K	48×48	16 Patches	300K	$2^{-k}\times10^{-4}$, k=0-1
Model Polishment	SR	DIV2K	96×96	16 Patches	600K	$2^{-k}\times10^{-4}$, k=0-4
	Dn	Waterloo			1200K	$10^{-k}\times10^{-3}$, k=0-2
Fixed-Point Fine-Tuning	SR	DIV2K	96×96	16 Patches	100K	$2^{-k}\times10^{-4}$, k=5-6
	Dn	Waterloo			800K	$2^{-k}\times10^{-5}$, k=0-2

Table 3: ERNet training settings.

Floating-Point Model/Method	KOP/pixel	PSNR (dB)				
SR×4	Intrinsic	Set5	Set14	BSD100	Urban100	**Avg**
VDSR (20 layers, 64ch)	1334	31.35	28.01	27.29	25.18	27.96
SRResNet (37 layers, 64ch)	248	32.05	28.53	27.57	26.07	28.56
SR4ERNet-B17R3N1 (UHD30)	115	31.97	28.48	27.51	25.84	28.45
SR4ERNet-B25R3N24 (HD60)	175	32.08	28.57	27.57	26.03	28.56
SR4ERNet-B34R4N0 (HD30)	223	32.11	28.61	27.59	26.11	28.60
SR×2	Intrinsic	Set5	Set14	BSD100	Urban100	**Avg**
VDSR (20 layers, 64ch)	1334	37.53	33.03	31.90	30.76	33.31
SR2ERNet-B9R1N6 (UHD30)	128	37.62	33.17	31.93	30.60	33.33
SR2ERNet-B12R3N0 (HD60)	235	37.80	33.36	32.05	31.04	33.56
SR2ERNet-B19R3N8 (HD30)	384	37.89	33.48	32.13	31.29	33.70
Denoise ($\sigma = 25$)	Intrinsic	Set5	Set14	CBSD68	**Avg**	
CBM3D	N/A	31.63	30.26	30.70	30.86	
FFDNet (12 layers, 96ch)	426	N/A	N/A	31.21	N/A	
DnERNet-B3R1N0 (UHD30)	123	31.52	29.97	30.66	30.72	
DnERNet-B9R1N0 (HD60)	246	31.97	30.45	31.08	31.17	
DnERNet-B12R1N7 (HD30)	450	32.11	30.62	31.21	31.31	

Table 4: PSNR performance of polished ERNet models.

tile-aligned. To address this issue, we implement each block buffer using eight sub-buffer banks as shown in Fig. 17. A normal mapping is sufficient for all cases except for pixel-shuffle upsampling which causes sub-buffer conflicts; therefore, another interleaved mapping is devised to resolve this issue.

7 EVALUATION

The configurations of our implementation are listed in Table 2. The three computation constraints used for model optimization correspond to three real-time specifications: 4K UHD 30fps (UHD30), Full HD 60fps (HD60), and Full HD 30fps (HD30). In the following, we will first present the results for ERNet models and then the layout performance for the eCNN processor. We will also introduce two application examples in computer vision, style transfer and object recognition, to demonstrate the flexibility of our approach.

7.1 ERNet Models

Model structure. In addition to SR4ERNet for four-times SR, we also implement SR2ERNet and DnERNet for two-times SR and denoising respectively. Their models are derived by accordingly removing one and two upsamplers from the SR4ERNet in Fig. 7. We do not use batch normalization layers as suggested in [36]. Also, we pad 29 zero-valued channels for RGB images to form 32ch inputs for eCNN.

Training. The hyper-parameters are listed in Table 3 for the three stages of our training procedure: model scanning, polishment, and fine-tuning for quantization. The scanning uses lightweight settings for speeding up the process, and then the other two apply heavy settings for improving image quality. We train on two datasets, DIV2K [1] and Waterloo Exploration [40], for the following comparison with the state-of-the-art networks of SR and denoising, respectively.

Polished models. The PSNR performance of the picked models and their polished results is shown in Table 4. For comparison, we include VDSR and SRResNet (implementation in [36]) for SR and also CBM3D and FFDNet [63] for denoising. For the HD30 specification, the hardware-constrained ERNet models can achieve similar quality compared to the state-of-the-art SRResNet and FFDNet. When we increase the specification, the PSNR performance will drop as the intrinsic complexity goes down. However, for UHD30 the SR4ERNet can still outperform VDSR by 0.49 dB while the SR2ERNet and DnERNet are comparable to the benchmark VDSR and CBM3D respectively.

Fixed-point precision and entropy coding. We tested both the L1-norm and L2-norm quantization on the polished models, and the results are shown in Table 5. For the case of SR4ERNet for HD30, the bitstreams will exceed the capacity of the parameter memory in Table 2 using 8-bit precision. Therefore, we further perform 7-bit quantization on some selected parameter groups to match the capacity. In general, the L1-norm causes more quality degradation at first because more large values are cropped, but it can be well recovered after fine-tuning. We chose to use the L1-optimized models for their better PSNR quality despite their higher entropy for larger dynamic range, and the compression ratio is around 1.1-1.5×. As a result, the PSNR drops are well limited between 0.05 to 0.14 dB for using dynamic 8-bit precision and

Model		SR4ERNet			SR2ERNet			DnERNet		
Specification		UHD30	HD60	HD30*	UHD30	HD60	HD30	UHD30	HD60	HD30
PSNR drop (avg. dB)	L2-Q	-0.08	-0.09	-0.16	-0.17	-0.13	-0.16	-0.12	-0.24	-0.30
	L2-FT	-0.06	-0.10	-0.11	-0.14	-0.13	-0.14	-0.06	-0.12	-0.09
	L1-Q	-1.55	-1.83	-1.81	-2.15	-3.00	-3.69	-1.11	-2.21	-1.76
	L1-FT	**-0.05**	**-0.07**	**-0.07**	**-0.13**	**-0.12**	**-0.14**	**-0.05**	**-0.06**	**-0.06**
Entropy (bit/param.)	L2-SE	5.89	6.30	6.30	5.93	6.09	6.28	4.79	5.48	5.72
	L2-CE	5.98	6.38	6.38	6.02	6.19	6.38	4.92	5.60	5.81
	L1-SE	6.83	6.87	5.94	6.65	6.84	6.93	5.20	6.04	6.40
	L1-CE	**6.89**	**6.94**	**6.03**	**6.72**	**6.91**	**7.00**	**5.30**	**6.13**	**6.47**
CR (L1)		1.16	1.15	1.33	1.19	1.16	1.14	1.51	1.30	1.24
PM Usage (L1)		42%	75%	87%	14%	29%	49%	3%	7%	14%

Note - Q: Quantization; FT: Fine-Tune; SE: Shannon Entropy; CE: Cross Entropy; CR: Compression Ratio; PM: Parameter Memory.

Table 5: Model quantization and entropy coding.

```
CONV3X3(TP,31,62)  .src(DI,32,Q7),.dst(BB0,32,Q6),.param(Q8,Q10,0)
ER(TP,30,61)(0,UQ4) .src(BB0,32,Q6),.dst(BB1,32,Q5),.param(Q6,Q5,Q7,Q7)
ER(TP,30,60)(0,UQ4) .src(BB1,32,Q5),.dst(BB2,32,Q6),.param(Q5,Q5,Q7,Q8)
ER(TP,29,59)(0,UQ4) .src(BB2,32,Q6),.dst(BB1,32,Q6),.param(Q6,Q6,Q7,Q6)
CONV3X3(TP,29,58)  .src(BB1,32,Q6),.dst(BB2,32,Q5),.param(Q8,Q10),.srcS(BB0,Q6,4)
CONV3X3(TP,28,57)  .src(BB2,32,Q5),.dst(DO,32,Q7),.param(Q9,Q9)
```

Figure 18: Program of DnERNet-B3R1N0 (UHD30).

1,288KB of parameter memory. In addition, the values of cross entropy are close to the Shannon limits, which justifies the usage of the simple encoding method.

Program. The coarse-grained FBISA instructions result in concise programs. Fig. 18 shows a six-line program for the six-layer DnERNet for UHD30. The attributes of the opcodes specify the sizes of output blocks in terms of 4×2-tiles, and most of other fields identify the dynamic Q-formats.

7.2 eCNN Performance

Implementation. We implemented eCNN in Verilog HDL for TSMC 40nm technology and used ARM memory compilers to generate all SRAM macros. We used Synopsys IC Compiler for placement and routing. We performed layouts for five essential and well-pipelined macro circuits which constitute the eCNN in a collectively exhaustive way. For fast and accurate power estimation, we ran RTL simulation to generate signal activity waveforms and then propagated them to post-layout netlists with extracted parasitics.

Layout performance. The eCNN processor can run at 250MHz and achieves up to 41 TOPS of inference performance. The total area is 55.23mm^2 and the average power consumption is 6.94W at 0.9V. The details are summarized in Table 6. The LCONV3×3 engine delivers 90% of inference performance and thus occupies the most resource, i.e. 65.8% of area and 87.4% of power. And the LCONV1×1 engine is responsible for the rest 10% of performance and uses another 7.0% of area and 6.6% of power. On the other hand, the three block buffers (1536KB) and the parameter memory (1288KB) contributes 11.3% and 7.9% of area for storing feature maps and parameters respectively. But they consume only 3.9% of power in total thanks to well-constrained word depths and highly-optimized SRAM macros. The computation for ERNets is profiled in Fig. 19 where the inference time indicates real-time capability and NCR

Functional Unit	Computing Engine/ On-Chip SRAM	Area		Power	
		mm^2	%	W	%
IDU	Instruction Decode	0.02	0.0	0.00	0.0
	Parameter Decompress	1.46	2.6	0.01	0.2
	Program Memory	0.02	0.0	0.00	0.0
	Parameter Memory	4.38	7.9	0.06	0.9
CIU	**LCONV3×3**	36.32	65.8	6.07	87.4
	LCONV1×1	3.88	7.0	0.46	6.6
	Inference Datapath	2.26	4.1	0.10	1.4
	Block Buffer File	6.24	11.3	0.21	3.0
	Line FIFO	0.65	1.2	0.04	0.5
Total (eCNN)		**55.23**	100.0	**6.94**	100.0

Table 6: Area and power consumption of eCNN.

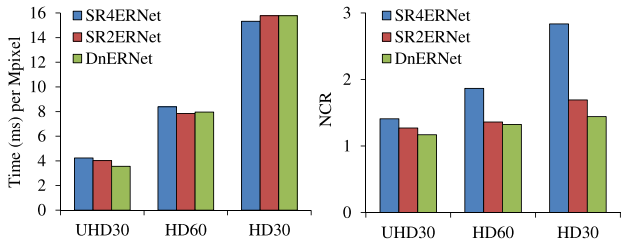


Figure 19: Inference time (left) and NCR (right).

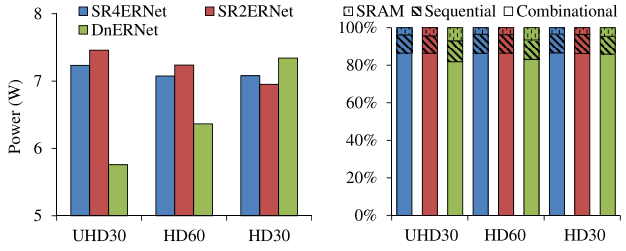


Figure 20: Power consumption breakdown on ERNet models (left) and circuit types (right).

shows computing overheads. Note that there is a tradeoff between inference time and image quality as shown in Table 4.

Power breakdown. The power consumption for each ERNet model is shown in Fig. 20. We found that the variation between different specifications is related to the quality difference. For example, DnERNets have the largest power variation, 1.58W, while they have 0.58 dB of PSNR drop from HD30 to UHD30. In contrast, SR4ERNets have the smallest variations in both power consumption and PSNR. Fig. 20 also shows the breakdown for three circuit types. The combinational circuits contribute 82-87% of power consumption for the highly-parallel convolution. The sequential circuits constantly occupy about 10% for the locally-distributed parameter registers, 4×2-tile pipeline registers, and clock tree. The rest 3-7% is then consumed by SRAMs.

DRAM bandwidth and power. The DRAM access via the data input and output FIFOs is highly regular and can be optimized in a deterministic way. The DRAM bandwidth and dynamic power consumption for each ERNet model are shown in Fig. 21. The DnERNets

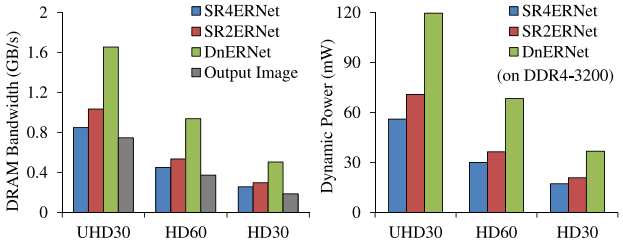


Figure 21: DRAM bandwidth and dynamic power.

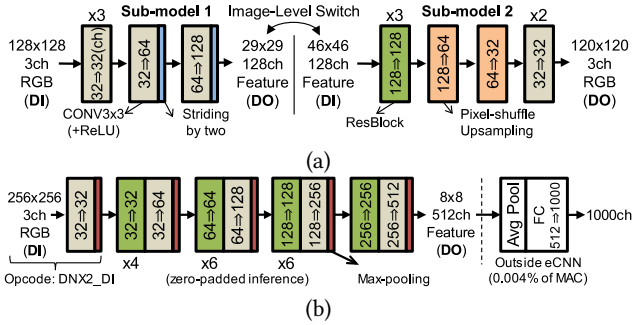


Figure 22: Computer vision models based on FBISA: (a) style transfer and (b) object recognition.

require the most bandwidth for each specification: 1.66GB/s for UHD30, 0.94GB/s for HD60, and 0.5GB/s for HD30. But the NBRs are still only 2.2×, 2.5×, and 2.7×, respectively. Therefore, the eCNN can support high-end applications with low-end DRAM configurations. For example, DDR-400 (3.2GB/s), DDR-266 (2.1GB/s), and DDR-200 (1.6GB/s) are sufficient for UHD30, HD60, and HD30 respectively. Regarding power consumption, we use Micron DDR4 SDRAM System-Power Calculator [44] for evaluation on DDR4-3200. The small bandwidth of eCNN consumes only less than 120mW of dynamic power (activation/read/write) while the leakage power consumes 267mW.

Comparison. Table 7 compares the eCNN with two state-of-the-art processors for computational imaging applications: IDEAL [42] for BM3D and Diffy [41] for CNN. Both of them can only support the HD30 specification and already require high-end DRAM settings, i.e. dual-channel DDR3-1333 to DDR3-2133. However, eCNN can deliver up to UHD30 performance using only DDR-400. Another advantage of eCNN is its constant pixel throughput to facilitate real-time applications. In contrast, the performance of IDEAL and Diffy highly varies with input images since statistical properties are deployed for acceleration.

To compare power consumption, we list the reported numbers from [42] and [41] for IDEAL and Diffy on the right of Table 7. Note that these numbers cannot be used to determine superiority directly because they are highly related to technology nodes, implementation details, and deployed models/algorithms. For denoising at HD30, IDEAL needs 12.05W for BM3D and Diffy demands 27.16W (8 tiles) for FFDNet; however, eCNN consumes only 7.34W for DnERNet which is 0.39 dB better than CBM3D and comparable to FFDNet. For four-times SR at HD30, Diffy demands 54.32W (16 tiles) for VDSR while eCNN consumes only 7.08W for SR4ERNet (0.57 dB better than VDSR).

We also used a CNN accelerator simulator, SCALE-Sim [51], to simulate the performance of ERNets with the same processor configuration as the classical TPU [30]. Note that TPU is a high-performance 28nm processor which provides 92 TOPS at 40 W and has 28 MB of SRAM to store feature maps and parameters for data reuse. The simulation shows that 4K UHD 21.9 fps and Full HD 55.3 fps are achieved for SR4ERNet-B17R3N1 and SR4ERNet-B34R4N0 respectively. And the required DRAM bandwidths are 12.2 GB/s and 8.3 GB/s. As a result, eCNN provides 3.1× and 1.2× of throughput efficiency (fps/TOPS) and, in particular, 6.4× and 14.4× of arithmetic intensity (TOPS/GB/s), respectively, for these two models. This also demonstrates the advantage of our joint-design approach for computational imaging tasks.

7.3 Computer Vision Applications

Model structure. To show the model flexibility of our approach, we built FBISA-compatible models for style transfer and object recognition as shown in Fig. 22. They differ from the ones for computational imaging mainly in three respects: spatial downsampling, wider channels, and batch normalization layers. The first two are supported in FBISA by concatenating 32ch leaf-modules. The last one is used to stabilize model pre-training and will be merged into convolutional layers for quantization and inference.

Style transfer. We used two downsamplers to increase the receptive field as suggested in [29]. Since this will increase NCR significantly, we split the model into two sub-models as shown in Fig. 22(a) to reduce computing overheads. After training and quantization, our model can deliver similar style transfer effects with [29]. The performance on eCNN is Full HD 29.5 fps for this model while Nvidia Titan X GPU is used in [29] and only generates 512×512 20fps. In addition, the DRAM bandwidth of our approach is only 1.91GB/s, which can enable this advanced application on embedded devices.

Object recognition. We devised a 40-layer residual network in Fig. 22(b) for eCNN to perform object recognition. To reduce the amount of parameters, we avoided 512ch ResBlocks and instead put more computation in thinner layers. The final 8-bit model achieves 69.7% top-1 accuracy for ImageNet [15] with 5M parameters. The performance is comparable to ResNet-18 (69.6%; 11M) and VGG-16 (71.5%; 138M) [54]. To support this model, we need to increase the size of parameter memory by three times, and the area of eCNN would become 63.99 mm^2. Then the performance achieves 1344 fps (0.74 ms per image) with 308 MB/s of DRAM bandwidth. For each image, eCNN only consumes 5.25 mJ of energy and 231 KB of DRAM access. For comparison, the Eyeriss [13], which has 12.25 mm^2 of core area with 65nm technology, delivers 0.7 fps (4.3 s for a batch of three images) with 236mW of power consumption and 74 MB/s of DRAM bandwidth for VGG-16. Thus it demands as high as 337 mJ of energy and 106 MB of DRAM access for one image. In this case study, we demonstrate the model flexibility of eCNN and also show that our joint hardware-model approach can benefit object recognition tasks as well.

Processor	Application	Model	Specification	Throughput	DRAM Setting	Process	Frequency	Area (mm^2)	Power (W)
$IDEAL_{MR}$ [42]	Denoising	CBM3D	HD30	Variant/ Statistical	DDR3-1333 ×2 (21.3GB/s)	65nm	1GHz	23.08	12.05
Diffy [41]	Denoising	FFDNet	HD30		DDR3-1600 ×2 (25.6GB/s)	65nm	1GHz	58.44	27.16
	SR	VDSR	HD30		DDR3-2133 ×2 (34.2GB/s)			116.88	54.32
eCNN	SR/ Denoising	SR4ERNet/ SR2ERNet/ DnERNet	HD30	Constant	DDR-200 (1.6GB/s)	40nm†	250MHz	55.23	7.08/6.95/7.34
			HD60		DDR-266 (2.1GB/s)				7.07/7.24/6.36
			UHD30		DDR-400 (3.2GB/s)				7.23/7.46/5.76

† The 40nm technology outperforms its 65nm counterpart at half of the power consumption under the same operation speed [55].

Table 7: Comparison of computational imaging processors.

8 RELATED WORK

Instruction set. Previous SIMD works usually devised load instructions for parameters and adopted medium-grained operands for features, such as vector/matrix [37], 2D tile [49], and compute tile [47], for providing flexibility. In contrast, we apply a parameter-inside approach and large-grained feature operands for optimizing power consumption and computing capability for highly-parallel convolution.

Model structure. Most of previous hardware-oriented models aim to reduce complexity. In particular, SqueezeNet [26] temporarily reduces model width and then expands back for residual connections. And MobileNetV2 [52] moves the connections to thinner layers to reduce storage. This results in an expansion-reduction structure similar to ERNet. However, our goal is to increase complexity under hardware constraints, and thus the implementation details are quite different.

Winograd convolution. It is an efficient algorithm to reduce multipliers for CONV3×3 and recently shows advantages on GPU [34], FPGA [38], and embedded processor [58]. However, it increases 23.5% of area in our case because the overheads of long internal bitwidths and additional pre-/post-processing become significant for our 8-bit implementation. Therefore, we used a direct implementation instead.

Cross-frame optimization. It is a new research direction to reduce CNN computation by exploiting temporal redundancy [8, 67] or input similarity [50] across video or audio frames. Moreover, this concept has also been applied to compensate the unreliability brought by pruning [60]. This direction is complementary to our approach and can be used to further enhance the performance of eCNN.

9 CONCLUSION

In this paper, we investigate a hardware-first framework to support computational imaging CNNs for up to 4K Ultra-HD applications on edge devices. Instead of accelerating existing models, we devise the hardware-oriented ERNets for the adopted block-based inference flow which can eliminate DRAM bandwidth for feature maps. For providing high computing capability, we construct the coarse-grained FBISA to enable highly-parallel convolution. Finally, we implement the high-performance eCNN processor to incorporate ERNet and FBISA. The training and layout results show that this framework can provide superior hardware performance and image quality at the same time. In addition, its flexibility is demonstrated by the usage examples of style transfer and object recognition. Our future work is to include more CNN variants for different applications and unleash their power on edge devices.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Science and Technology, Taiwan, R.O.C. under Grant no. MOST 107-2218-E-007-029.

REFERENCES

[1] Eirikur Agustsson and Radu Timofte. 2017. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

[2] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. 2017. Bit-Pragmatic Deep Neural Network Computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[3] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In *Proceedings of the 43rd Annual ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

[4] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[5] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. 2018. YodaNN: An Architecture for Ultralow Power Binary-Weight CNN Acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37 (2018).

[6] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2015. Fixed point optimization of deep convolutional neural networks for object recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

[7] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie-Line Alberi Morel. 2012. Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding. In *British Machine Vision Conference (BMVC)*.

[8] Mark Buckler, Philip Bedoukian, Suren Jayasuriya, and Adrian Sampson. 2018. EVA^2: Exploiting Temporal Redundancy in Live Computer Vision. In *Proceedings of the 45th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

[9] Ayan Chakrabarti. 2016. A Neural Approach to Blind Motion Deblurring. In *European Conference on Computer Vision (ECCV)*.

[10] Qifeng Chen, Jia Xu, and Vladlen Koltun. 2017. Fast Image Processing with Fully-Convolutional Networks. In *IEEE International Conference on Computer Vision (ICCV)*.

[11] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Ja Wang, and Ling Li. 2014. DaDianNao: a Machine-Learing Supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[12] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: a Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *Proceedings of the 43rd Annual ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

[13] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: an Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52 (2017).

[14] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. 2007. Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering. *IEEE Transactions on Image Processing* 16 (2007).

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[16] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2014. Learning a Deep Convolutional Network for Image Super-Resolution. In *European Conference*

on Computer Vision (ECCV).

[17] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual ACM/IEEE International Symposium on Computer Architecture (ISCA).*

[18] Michaël Gharbi, Gaurav Chaurasia, Sylvain Paris, and Frédo Durand. 2016. Deep joint demosaicking and denoising. *ACM Transactions on Graphics (TOG)* 35 (2016).

[19] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi. 2016. Hardware-oriented Approximation of Convolutional Neural Networks. *arXiv:1604.03168* (2016).

[20] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations (ICLR).*

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[22] Kartik Hegde, Rohit Agrawal, Yulun Yao, and Christopher W. Fletcher. 2018. Morph: Flexible Acceleration for 3D CNN-based Video Understanding. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).*

[23] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861* (2017).

[24] Chao-Tsung Huang, Po-Chih Tseng, and Liang-Gee Chen. 2005. Analysis and VLSI architecture for 1-D and 2-D discrete wavelet transform. *IEEE Transactions on Signal Processing* 53 (2005).

[25] J.-B. Huang, A. Singh, and N. Ahuja. 2015. Single image super-resolution from tranformed self-exemplars. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[26] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2017. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360* (2017).

[27] Andrey Ignatov, Nikolay Kobyshev, Radu Timofte, Kenneth Vanhoey, and Luc Van Gool. 2017. DSLR-Quality Photos on Mobile Devices with Deep Convolutional Networks. In *IEEE International Conference on Computer Vision (ICCV).*

[28] International Organization for Standardization. 1994. *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines.* ISO/IEC 10918-1:1994.

[29] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *European Conference on Computer Vision (ECCV).*

[30] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA).*

[31] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi. 2016. Learning-based view synthesis for light field cameras. *ACM Transactions on Graphics (TOG)* 35 (2016).

[32] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Accurate Image Super-Resolution Using Very Deep Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS).*

[34] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[35] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2017. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[36] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced Deep Residual Networks for Single Image Super-Resolution. *arXiv:1707.02921* (2017).

[37] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. 2016. Cambricon: An Instruction Set Architecture for Neural Networks. In *Proceedings of the 43rd Annual ACM/IEEE International Symposium on Computer Architecture (ISCA).*

[38] Liqiang Lu and Yun Liang. 2018. SpWA: an efficient sparse Winograd convolutional neural networks accelerator on FPGAs. In *ACM/ESDA/IEEE Design Automation Conference (DAC).*

[39] Alice Lucas, Michael Iliadis, Rafael Molina, and Aggelos K. Katsaggelos. 2018. Using Deep Neural Networks for Inverse Problems in Imaging: Beyond Analytical Methods. *IEEE Signal Processing Magazine* 35 (2018).

[40] Kede Ma, Zhengfang Duanmu, Qingbo Wu, Zhou Wang, Hongwei Yong, Hongliang Li, and Lei Zhang. 2017. Waterloo Exploration Database: New Challenges for Image Quality Assessment Models. *IEEE Transactions on Image Processing* 26 (2017).

[41] Mostafa Mahmoud, Kevin Su, and Andreas Moshovos. 2018. Diffy: a Déjà vu-Free Differential Deep Neural Network Accelerator. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).*

[42] Mostafa Mahmoud, Bojian Zheng, Alberto Delmás Lascorz, Felix Heide, Jonathan Assouline, Paul Boucher, Emmanuel Onzon, and Andreas Moshovos. 2017. IDEAL: Image DEnoising AcceLerator. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).*

[43] D. Martin, C. Fowlkes, D. Tal, and J. Malik. 2001. A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. In *IEEE International Conference on Computer Vision (ICCV).*

[44] Micron. Retrieved 14 August 2019. *DDR4 SDRAM System-Power Calculator.* [Online]. Available: https://www.micron.com/support/tools-and-utilities/power-calc.

[45] Bert Moons and Marian Verhelst. 2016. A 0.3-2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets. In *2016 IEEE Symposium on VLSI Circuits (VLSIC).*

[46] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. 2017. Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[47] Ravi Narayanaswami, Dong Hyuk Woo, Olivier Temam, and Harshit Khaitan. 2017. Neural network instruction set architecture. U.S. Patent 9836691.

[48] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA).*

[49] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA).*

[50] Marc Riera, Jose-Maria Arnau, and Antonio González. 2018. Computation Reuse in DNNs by Exploiting Input Similarity. In *Proceedings of the 45th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA).*

[51] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN Accelerator Simulator. *arXiv:1811.02883* (2018).

[52] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

[53] Dongjoo Shin, Jinmook Lee, Jinsu Lee, and Hoi-Jun Yoo. 2017. DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In *IEEE International Solid-State Circuits Conference (ISSCC).*

[54] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR).*

[55] TSMC. Retrieved 14 August 2019. *40nm Technology.* [Online]. https://www.tsmc.com/english/dedicatedFoundry/technology/40nm.htm.

[56] Jen-Chieh Tuan, Tian-Sheuan Chang, and Chein-Wei Jen. 2002. On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture. *IEEE Transactions on Circuits and Systems for Video Technology* 12 (2002).

[57] Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep Convolutional Neural Networks. In *European Conference on Computer Vision (ECCV).*

[58] Athanasios Xygkis, Lazaros Papadopoulos, and David Moloney. 2018. Efficient Winograd-based convolution kernel implementation on edge devices. In *ACM/ESDA/IEEE Design Automation Conference (DAC).*

[59] Xuan Yang, Jing Pu, Blaine Burton Rister, Nikhil Bhagdikar, Stephen Richardson, Shahar Kvatinsky, Jonathan Ragan-Kelley, Ardavan Pedram, and Mark Horowitz. 2016. A Systematic Approach to Blocking Convolutional Neural Networks. *arXiv:1606.04209* (2016).

[60] Reza Yazdani, Marc Riera, Jose-Maria Arnau, and Antonio González. 2018. The Dark Side of DNN Pruning. In *Proceedings of the 45th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA)*.
[61] R. Zeyde, M. Elad, and M. Protter. 2010. On single image scale-up using sparse-representations. In *Proceedings of the International Conference on Curves and Surfaces*.
[62] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. 2017. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Transactions on Image Processing* 26 (2017).
[63] Kai Zhang, Wangmeng Zuo, and Lei Zhang. 2018. FFDNet: Toward a Fast and Flexible Solution for CNN-Based Image Denoising. *IEEE Transactions on Image Processing* 27 (2018).
[64] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.
[65] Xuda Zhou, Zidong Du, Qi Guo, Shaoli Liu, Chengsi Liu, Chao Wang, Xuehai Zhou, Ling Li, Tianshi Chen, and Yunji Chen. 2018. Cambricon-S: Addressing Irregularity in Sparse Neural Networks through A Cooperative Software/Hardware Approach. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.
[66] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. In *IEEE International Conference on Computer Vision (ICCV)*.
[67] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. 2018. Euphrates: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision. In *Proceedings of the 45th Annual ACM/IEEE International Symposium on Computer Architecture (ISCA)*.

A MODEL VARIANTS FOR DENOISING

The image quality of DnERNets drops quickly for higher specifications due to shallower layers. Here, under the eCNN framework, we show how to improve their quality using the same downsampling strategy of FFDNet. As shown in Fig. A.1, we can perform pixel unshuffle, which packs 2×2 3-ch RGB pixels into a 12-ch one, for the input image. Then we construct DnERNet-12ch models as DNERNets, and the only difference is their input and output channels both become twelve, instead of three. Finally, the output image can be obtained by performing pixel shuffle.

The PSNR performance of the picked and then polished models is show in Table A.1. For the UHD30 specification, the DnERNet-12ch-B8R2N5 outperforms the DnERNet-B3R1N0 by 0.54 dB and now provides FFDNet-level quality. On the other hand, the DnERNet-12ch-B19R3N15 for HD30 even surpasses FFDNet by 0.15 dB using smaller intrinsic complexity. Lastly, the quantized 8-bit models have 0.09 dB drop on average, which are similar to ordinary ERNets, and the DRAM bandwidth is only at most 1.8GB/s.

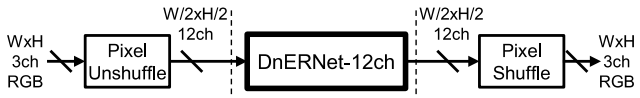


Figure A.1: DnERNet-12ch for denoising.

Floating-Point Model/Method	KOP/pixel	PSNR (dB)			
Denoise ($\sigma = 25$)	Intrinsic	Set5	Set14	CBSD68	**Avg**
CBM3D	N/A	31.63	30.26	30.70	30.86
FFDNet (12 layers, 96ch)	426	N/A	N/A	31.21	N/A
DnERNet-12ch-B8R2N5 (UHD30)	123	32.07	30.52	31.19	31.26
DnERNet-12ch-B11R4N0 (HD60)	241	32.19	30.66	31.29	31.38
DnERNet-12ch-B19R3N15 (HD30)	384	32.27	30.75	31.36	31.46

Table A.1: PSNR of polished DnERNet-12ch models.