CU Partition Mode Decision for HEVC Hardwired Intra Encoder Using Convolution Neural Network

Zhenyu Liu, *Member, IEEE*, Xianyu Yu, Yuan Gao, Shaolin Chen, Xiangyang Ji, *Member, IEEE*, and Dongsheng Wang, *Member, IEEE*

***Abstract*— The intensive computation of High Efficiency Video Coding (HEVC) engenders challenges for the hardwired encoder in terms of the hardware overhead and the power dissipation. On the other hand, the constrains in hardwired encoder design seriously degrade the efficiency of software oriented fast coding unit (CU) partition mode decision algorithms. A fast algorithm is attributed as VLSI friendly, when it possesses the following properties. First, the maximum complexity of encoding a coding tree unit (CTU) could be reduced. Second, the parallelism of the hardwired encoder should not be deteriorated. Third, the process engine of the fast algorithm must be of low hardware- and power-overhead. In this paper, we devise the convolution neural network based fast algorithm to decrease no less than two CU partition modes in each CTU for full rate-distortion optimization (RDO) processing, thereby reducing the encoder's hardware complexity. As our algorithm does not depend on the correlations among CU depths or spatially nearby CUs, it is friendly to the parallel processing and does not deteriorate the rhythm of RDO pipelining. Experiments illustrated that, an averaged 61.1% intraencoding time was saved, whereas the Bjøntegaard-Delta bit-rate augment is 2.67%. Capitalizing on the optimal arithmetic representation, we developed the high-speed [714 MHz in the worst conditions (125 °C, 0.9 V)] and low-cost (42.5k gate) accelerator for our fast algorithm by using TSMC 65-nm CMOS technology. One accelerator could support HD1080p at 55 frames/s real-time encoding. The corresponding power dissipation was 16.2 mW at 714 MHz. Finally, our accelerator is provided with good scalability. Four accelerators fulfill the throughput requirements of UltraHD-4K at 55 frames/s.**

***Index Terms*— HEVC, fast CU/PU mode decision, CNN, VLSI, intra encoding.**

Manuscript received December 27, 2015; revised June 5, 2016 and July 17, 2016; accepted August 8, 2016. Date of publication August 18, 2016; date of current version September 13, 2016. This work was supported in part by Huawei Technologies, National Science and Technology Major Project under Grant 2016YFB0200505 and in part by the National Natural Science Foundation of China under Grant 61325003. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Yui-Lam Chan.

Z. Liu and D. Wang are with Tsinghua National Laboratory for Information Science and Technology, Research Institute of Information Technology, Tsinghua University, Beijing 100084, China (e-mail: liuzhenyu73@tsinghua.edu.cn).

X. Yu is with the Institute of Microelectronics, Tsinghua University, Beijing 100084, China.

Y. Gao is with the Department of Computer Science, Tsinghua University, Beijing 100084, China.

X. Ji is with the Department of Automation, Tsinghua University, Beijing 100084, China.

S. Chen is with Huawei Technologies Company, Ltd., Shenzhen 518129, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2016.2601264

I. Introduction

THE state-of-the-art video coding standard, High Efficiency Video Coding (HEVC) [1], [2], was developed by Joint Collaborative Team on Video Coding (JCT-VC). With the equivalent subjective video quality, HEVC can double the compression ratio as compared to the predecessor H.264/AVC, especially when operating on low bit-rate, high-resolution image, and low-delay communication applications [3].

HEVC introduces the quadtree-based coding data structure. The encoder splits one picture into basic square regions. Each region is denoted as one coding tree unit (CTU), of which the luma partition size can be chosen as $2N \times 2N$ ($N \in \{32, 16, 8\}$). The data organization of CTU involves the quadtree structure. For one CU (including CTU), it can be coded as a whole ($2N \times 2N$ mode), or be split into four $N \times N$ sub-CUs. The splitting of CU can be iterated on the basis of the signal features, until the minimum CU size is reached. When $N \geq 8$, the prediction unit (PU) size of Intra CU is identical to its CU size; otherwise, it can adopt either 8×8 or 4×4 PU sizes. In general, when the minimum CU size is set, the larger CTU size always provides better compression efficiency, especially for high-resolution pictures. The experiments in literature [3] showed that, when 8×8 minimum CU size was applied, using 64×64 CTU size reduced the bit rate by 11.0% on average as compared to the 16×16 CTU size. With Class-A test sequences, this performance gap dilated up to 28.2%.

To alleviate the Intra encoding complexity, innumerable algorithms have been developed for fast Intra coding mode decision. The previous methods can be classified into the following categories: The first kind of fast algorithms reduce the complexity of prediction mode rate-distortion optimization (RDO) [4]–[6]. For example, Ma et al. applied the number of L best modes from rough mode decision and the most probable modes (MPMs) from the neighboring coded blocks as the candidates, to undergo the full RDO processing [6]. The methods of the second category, which are the continuation of H.264/AVC low complexity encoding algorithm study [7]–[10], simplify the complexity of rate-distortion (RD)-cost computation [11], [12]. To relieve the complexity induced by CU/PU modes, the algorithms belonging to the third category dynamically skip the unpromising CU/PU depths or early terminate the CU/PU mode RDO procedure [13]–[23]. For instance, literatures [13]–[16] defined the dynamic CU depth range on the basis of the CU depth information

of previously coded slices and CUs. The algorithm in literature [17] terminates the splitting of the current CU when this CU node selected SKIP mode as the best prediction mode. Shen et al. proposed the dynamic CU depth range and early termination methods by exploring the coding information of spatial neighboring and co-located coding blocks [18]. In literature [19], Zhang exploited the early CU split termination algorithm by using the approximate aggregated RD-cost with sub-CUs' RDO results. To improve the coding performance, the machine-learning methods are employed to evaluate the critical parameters in fast CU mode decision [16], [22], [23].

On the other hand, the hardwired HEVC encoders adopt CTU pipeline processing [24]–[26], which incurs the following constraints: First, the process latency for one CTU is fixed (6.6k clock cycles in [25]). Second, the high degree of parallelism, especially in the CU level, is a must to guarantee the required throughput. A fast algorithm is considered as VLSI friendly, only when it possesses the following properties:

- The CTU-grain maximum computational complexity should be reduced. Only with this precondition, the encoder hardware cost, which is designed to satisfy the stipulated process latency under the maximum burden, can be saved. The dynamic CU level skipping and early split termination algorithms cannot contribute to simplifying the encoder's hardware complexity. The same problem exists in the online training stage of machine learning based algorithms [16], [22], [23].
- The parallelism should not be degraded by the fast algorithm. In the high resolution video encoder deigns [24], [25], the 4×4 PU is always being encoded in parallel with other CU modes to improve the throughput as far as possible. If the current CU mode is inferred from the neighboring CUs' coding information, the CU level parallel processing is infeasible.
- The hardware and power costs for implementing the fast algorithm should be low. From the system view, the fast algorithm is a part of the whole encoder. Its overheads directly affect the encoder performance.

For the above reasons, the software oriented fast algorithms are not adopted in the real-time Intra encoders in literatures [25], [26].

In [26], Zhu et al. proposed to skip a certain number of CU/PU modes in full RDO procedure according to the estimated RD-cost from the source image texture investigation. Specifically, the prediction residue of a pixel is first evaluated according to its edge strength, edge direction, and its location in CU. Next, the RD-cost of one CU, with which the fast CU mode decision could be made, is estimated from the prediction residue evaluations. The drawbacks of [26] lie in the empirical feature extractors and the ignorance of the topology of feature points, which degraded the compression efficiency with the averaged BDBR $=+4.53\%$. Convolution neural network (CNN) [27], is one model inspired by the animal visual cortex. The CNN with appropriate architecture can be trained with gradient-based learning algorithms [28] to classify two-dimensional image patterns, such as handwriting recognition [29], image classification [30], image segmentation [31], and so on. In this study, we introduce CNN to circumvent the aforementioned hindrances of [26]: First, the input layer's convolution kernels, which are viewed as the feature extractors, are trained by the samples, instead of a rule of thumb; second, the topology information of obtained features can be exploited by CNN during the classification processing. In addition, the proposed CNN for fast CU/PU mode decision is rectified according to the specific RDO task. For a VLSI friendly algorithm, it is desired that the process engine of the fast algorithm should be of low hardware- and power-costs, as well as high throughput. To this end, we provide the reconfigurable *Propagate Partial Multiply-Accumulate* (*PPMAC*) CNN accelerator by using the optimal floating-point arithmetic. As compared to the fast mode decision engine in [26], the proposed CNN accelerator reduced the chip area by 80.2%.

```
function XCompressCU(*pCurCU)
    M ← FastCUMode(PO,QP)
    if M ≠ SPLIT then
        C_2N ←CheckINTRA(pCurCU)
    else
        C_2N ← ∞
    end if
    if M ≠ HOMO and D_CUR < D_MAX then
        C_N ← 0
        for i = 0 to 3 do
            pSubCU_i ← pointer to SubCU_i
            C_N ← C_N + XCompressCU(pSubCU_i)
        end for
    else
        C_N ← ∞
    end if
    CheckBestMode(C_2N, C_N)
end function
```

Fig. 1. Pseudo codes of CNN oriented XCOMPRESSCU function in HM software ($\mathcal{M}$ denotes the predicted CU mode; pCurCU is the pointer of current CU data structure; **PO** and QP represent Y-component of current CU and quantization parameter, respectively; D_{CUR} is the current CU depth; D_{MAX} is the maximum CU depth; C_{2N} and C_N stand for the RD-costs of modes $2N\times2N$ and $N\times N$, respectively).

The rest of article is organized as follows. Section II briefly introduces the CU encoding procedure integrated with the CNN oriented fast mode decision scheme. The proposed CNN and its training method are described in Section III. The architecture of our CNN accelerator is presented in Section IV. The experimental results are illustrated in Section V, followed by the conclusions in Section VI.

II. CNN Based Fast CU/PU Mode Decision

As a universal approximation of nonlinear systems, CNN has the following virtues: First, the feature extractors in CNN, derived by training, are propitious to recognizing complex singularities, such as stroke end points or corners. Second, CNN could exploit the topology information among the singularities to improve the estimate accuracy. Finally, as compared with the fully connected network, the weight scale of CNN is greatly reduced, which contributes to reduction in hardware . The inherent properties of CNN make it favored in our CU/PU coding mode decision task.

The pseudo-code of XCompressCU function integrated with our CNN based mode prediction algorithm is described in Fig. 1, and the main optimizations to the reference software

```
function FastCUMode(PO,QP)
    P ←AvgSub(PO)
    s ← size of current CU
    Calculate E_T, E_C, E_M, and E_P by using (2)-(5)
    if E_P < 5E_T and E_M ≤ QP^2 then
        M ← HOMO
    else if E_C > 2 and IsBoundary == true then
        M ← SPLIT
    else if s == 64 then
        M ← COMB
    else if E(s)==true then
        M ←CNN_s(P, QP)
    else
        M ← COMB
    end if
    return M
end function
```

Fig. 2. Pseudo codes of FASTCUMODE function (IsBoundary being true indicates the current CU is on the picture boundary; CNN_s is the mode decision CNN for $s \times s$ CU ($s \in \{32, 16, 8\}$); E(s) enables the fast mode decision for $s \times s$ CU level).

have been highlighted. We provide the unitary expression of CU and PU mode decision, because the $8 \times 8/4 \times 4$ PU mode selection can be viewed as a special case of $2N \times 2N/N \times N$ CU mode decision. The function FastCUMode is the essential procedure, in which CNN is adopted to determine the optimal CU mode. When the current CU size is 64×64, the return values of FastCUMode include three cases: HOMO, SPLIT, and COMB. In other cases (the CU size is 32×32, 16×16 or 8×8), if the corresponding fast mode decision is enabled (E(s) == true), FastCUMode merely returns HOMO or SPLIT. If XCompressCU receives HOMO, the RD-cost of $N \times N$ is set as infinite ($C_N \leftarrow \infty$), and the CU splitting test is eliminated. Otherwise (SPLIT is received), C_{2N} is assigned infinite ($C_{2N} \leftarrow \infty$) and $2N \times 2N$ mode search is skipped. In case of COMB, modes $2N \times 2N$ and $N \times N$ are both traversed as the original HM software.

The pseudo codes of FastCUMode are depicted in Fig. 2. The inputs of FastCUMode include the Y-component of the current CU (**PO**) and the quantization parameter (QP). To reduce the computational complexity of CNN, we apply the *local averaging* and *sub-sampling* function AvgSub(**PO**) to derive the 8×8 matrix **P**. In specific, if the size of **PO** is $s \times s$ ($s \in \{8, 16, 32, 64\}$), each entry in **P**, that is, $p_{i,j}$, is derived as

$$p_{i,j} = \frac{1}{(s/8)^2} \sum_{m=\frac{i \cdot s}{8}}^{\frac{(i+1)\cdot s}{8}-1} \sum_{n=\frac{j \cdot s}{8}}^{\frac{(j+1)\cdot s}{8}-1} po_{m,n}, \tag{1}$$

where, $po_{m,n}$ is one entry of the receptive field in **PO**.

FastCUMode first carries out the coarse edge strength analysis to detect two special cases, i.e., the homogeneous blocks and the strong edge blocks on picture boundaries. The edge at (i, j) in **P** is denoted as $\vec{\Delta}_{i,j}$, where $i, j \in [0, 6]$. $\vec{\Delta}_{i,j}$ is a vector with horizontal component $\delta x_{i,j}$ and vertical component $\delta y_{i,j}$, which are written as

$$\begin{cases} \delta x_{i,j} = p_{i,j} + p_{i+1,j} - p_{i,j+1} - p_{i+1,j+1} \\ \delta y_{i,j} = p_{i,j} + p_{i,j+1} - p_{i+1,j} - p_{i+1,j+1}. \end{cases} \tag{2}$$

We further define a threshold E_T as

$$E_T = \max(QP^2, Q^2). \tag{3}$$

Q is quantization step that depends on QP,

$$Q = MF(QP\%6) \cdot 2^{\lfloor QP/6 \rfloor},$$

in which, $MF = [0.625, 0.7031, 0.7969, 0.8906, 1, 1.125]$. With (2) and (3), three auxiliary parameters, i.e., E_C, E_M, and E_P are devised. E_C has the form

$$E_C = \aleph\left(\left\{\vec{\Delta}_{i,j} | \delta x_{i,j} > E_T \text{ and } \delta y_{i,j} > E_T\right\}\right), \tag{4}$$

where, $\aleph$ represents the cardinal number (what is normally referred to as a counting number) of a set. E_M is

$$E_M = \max_{i,j}\left(\delta x_{i,j}^2 + \delta y_{i,j}^2\right). \tag{5}$$

And, E_P is the power of all edges, i.e.,

$$E_P = \sum_{i,j}\left(\delta x_{i,j}^2 + \delta y_{i,j}^2\right). \tag{6}$$

When $E_P < 5E_T$ and $E_M \leq QP^2$, the current CU is identified as homogeneous and the return value is HOMO. On the other hand, when the current CU is on the picture boundary (IsBoundary==true) and possesses the strong edges ($E_C > 2$), the return value is SPLIT. The coarse analysis contributes from two aspects: First, the simple coarse analysis relives the burden of CNN; second, the homogeneous samples, which trap the CNN in ill-conditions, could be filtered out. The coding performance loss of coarse analysis is BDBR=+0.42%. When the flag E(s) is set, the CU, which is not identified by the coarse analysis, is dispatched to the dedicated neural network (CNN_{32} for 32×32 CU, CNN_{16} for 16×16 CU, and CNN_8 for 8×8 CU) to determine the proper mode from **P** and QP. The return value of CNN_s is either HOMO or SPLIT. That is, CNN just chooses one from the two CU candidate modes for the following RDO processing. E(s) makes the tradeoff between coding quality and computational complexity, which will be described in Section V.

III. CNN Based Fast CU Mode Decision Algorithm

The block diagram of CTU pipelined HEVC Intra encoder integrated with the CNN based CU mode decision engine is illustrated in Fig. 3(a), which is inherent from our previous study [26]. Similar to the typical HEVC encoders [24], [25], a two-stage CTU pipeline is applied in our encoder. With the aforementioned methods, the first stage decides the promising CU/PU competitors that will be dispatched to the second stage for full RDO processing. The second stage constitutes the reconfigurable predictor, $8 \times 8/4 \times 4$ RDO engine, $32 \times 32/16 \times 16$ RDO engine, and reconstruction datapath. For one 8×8 CU, it is fed into $8 \times 8/4 \times 4$ RDO engine with the assigned PU mode ($2N \times 2N$ or $N \times N$) to calculate its best RD-cost. Other large CUs (16×16, 32×32 and 64×64) enter $32 \times 32/16 \times 16$ RDO engine. After deriving the optimal coding configuration, the reconstruction datapath produces the coding coefficients and the reconstructed pixels. Two successive CTUs could be processed

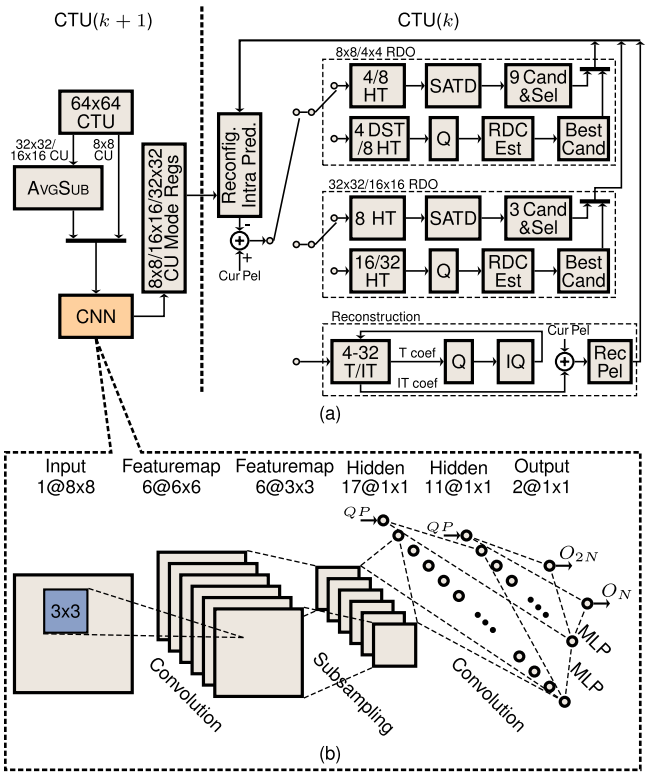


Fig. 3. Architecture of intra encoder embedded with CNN based fast CU mode decision engine. (a) Intra Encoder Top Block Diagram. (b) CNN Architecture.

simultaneously in two stages. Specifically, if the $(k+1)$th CTU (CTU$(k+1)$) is undergoing the CNN based mode decision in the first stage, the previous one (CTU(k)) is carrying out RDO and reconstruction in the second stage. For the VLSI implementation, our CNN based CU/PU pre-decision engine does not degrade the parallelism of the RDO processing in the second stage, even as reducing at least two CU/PU modes.

In the CNN based CU mode decision unit, CNN_{32}, CNN_{16}, and CNN_8 adopt the same architecture, as depicted in Fig. 3(b), to share the hardwired accelerator. The proposed CNN is composed of the lower alternating convolution and max-pooling layers, and the upper full-connected Multilayer Perceptron (MLP). Counting the input, our CNN comprises six layers, which are explained as follows:

- The input is an 8×8 matrix $\mathbf{P}$, which is derived as (1).
- The first hidden layer is a convolution layer with 6 feature maps. Each neuron is connected to a 3×3 receptive field in the input. The size of each feature map is 6×6, to prevent the convolution from falling off the boundary. The kernels in this layer are regarded as feature extractors. There are 60 trainable parameters, which is composed of six 3×3-kernels and six biases.
- The second hidden layer performs the local maxing and sub-sampling. There is no trainable parameter.
- The third hidden layer implements the second convolution. There are seventeen 1×1 output neurons, of which one input is QP. As the kernel size is 3×3, the trainable parameter number is $16 \times (6 \times 3 \times 3 + 1) = 880$.
- The last two layers are fully connected MLP. The fourth hidden layer consists of 11 units, and the output layer contains 2 output units. Including biases, the trainable parameter numbers in the fourth hidden layer and the output layer are 180 and 24, respectively.

Gradient-based learning algorithms [28] is adopted in our CNN training phase. To improve the CNN performance, the training strategy is rectified from two aspects, i.e., sample selection and target value definition, which will be descried in the following sub-sections.

A. Training Sample Selection

In the training sample selection stage, we introduce the following techniques.

- Firstly, the samples must not belong to the homogeneous type, which are explained in Fig. 2. We define the parameter γ as follows to indicate edge strength in one CU,

$$\gamma = \frac{\sum_{i,j}(\delta x_{i,j}^2 + \delta y_{i,j}^2)}{49 \cdot Q^2}$$

  where, $\delta x_{i,j}$ and $\delta y_{i,j}$ are defined as (2). In the CNN training phase, it is desired that the samples be evenly distributed in all output categories [32]. When the value of γ is small, there is a high probability to use $2N \times 2N$ mode. To prevent too many homogeneous samples in training, for 32×32 and 16×16 CUs, we use the samples with $\gamma > 0.1$; for 8×8 CU, the samples with $\gamma > 1.3$ are adopted.
- To capitalize on the edge strength information, we did not normalize the input signals. The vanishing gradient problem (the gradient of active function approaches 0 when its input amplitude is large enough) can be avoided by using the modified sigmoidal activation function.
- We eliminate such samples, in which the RD-cost difference between $2N \times 2N$ mode and $N \times N$ mode is too small. The parameter Δ_{RD} is defined as

$$\Delta_{RD} = \frac{C_{2N} - C_N}{C_{2N} + C_N},$$

  where, C_{2N} and C_N denominate the RD-costs of mode $2N \times 2N$ and mode $N \times N$, respectively. The samples with $|\Delta_{RD}| \leq 0.02$ are discarded. This scheme will improve the compression efficiency by BDBR=-0.20% averagely.
- Finally, six typical video sequences, including PeopleOnStreet, BasketballDrive, BQTerrace, Cactus, Vidyo3, and Johnny, are adopted to generate the training samples.

B. Target Value Definition

We will derive the target output values for nodes O_{2N} and O_N (as shown in Fig.3(b)) during the training phase. Let vector $\vec{e}$ denote the singularities in one CU. The variance of residual transform coefficient (σ_c) is a function of singularities, i.e., $\sigma_c(\vec{e})$. From the distortion model provided in [33], we can see that the nonlinear relation between the distortion D and Q^2 could be formulated as

$$D = \Phi(\vec{e})Q^2, \tag{7}$$

in which, $\Phi(\vec{e})$ is a nonlinear function of variable $\sigma_c(\vec{e})$. On the basis of RD relation [34], [35]

$$R = \frac{1}{2}\ln\frac{\sigma_c^2(\vec{e})}{D},$$

and the distortion model of (7), the rate has the form

$$R = \frac{1}{2}\ln\frac{\Psi(\vec{e})}{Q^2}, \tag{8}$$

where, $\Psi(\vec{e}) = \sigma_c^2(\vec{e})/\Phi(\vec{e})$.

With (7) and (8), for the $2N \times 2N$ mode, its distortion and rate are both nonlinear functions of $\vec{e}$, expressed as

$$\begin{cases} D_{2N} = \Phi_{2N}(\vec{e})Q^2 \\ R_{2N} = \dfrac{1}{2}\ln\dfrac{\Psi_{2N}(\vec{e})}{Q^2}, \end{cases} \tag{9}$$

where, $\Phi_{2N}(\vec{e})$ and $\Psi_{2N}(\vec{e})$ are nonlinear functions to be approximated. Consequently, we have

$$C_{2N} = D_{2N} + \lambda_{2N} R_{2N}. \tag{10}$$

The relationships of λ_{2N}, Q and QP are expressed as

$$\lambda_{2N} = a_{2N} 2^{QP/3}, \tag{11}$$

and

$$Q \approx b 2^{QP/6}. \tag{12}$$

Substituting (9), (11) and (12) to (10), we deduce that the logarithm of C_{2N} can be formulated as

$$\begin{aligned} \ln C_{2N} = {} & \mu_0 + \mu_1 QP + \ln\{\mu_2 \Phi_{2N}(\vec{e}) \\ & + \mu_3 \ln[\Psi_{2N}(\vec{e})] + \mu_4 QP + \mu_5\}, \end{aligned} \tag{13}$$

in which, $\{\mu_i | i = 0, 1, \ldots, 5\}$ represent the coefficients derived from the parameters a_{2N} and b. The expression of $\ln C_N$ can be traced by analogy. That is,

$$\begin{aligned} C_N &= D_N + \lambda_N R_N. \\ &= \Phi_N(\vec{e})Q^2 + \frac{\lambda_N}{2}\ln\frac{\Psi_N(\vec{e})}{Q^2} \\ &= \Phi_N(\vec{e})Q^2 + \frac{a_N 2^{QP/3}}{2}\ln\frac{\Psi_N(\vec{e})}{Q^2} \end{aligned}$$

It should be noticed that, because of the changed residue distribution, the Lagrange multipliers of mode $2N$ and mode N are different, $\lambda_{2N} \neq \lambda_N$ [36]. Then, we have

$$\begin{aligned} \ln C_N = {} & \nu_0 + \nu_1 QP + \ln\{\nu_2 \Phi_N(\vec{e}) \\ & + \nu_3 \ln[\Psi_N(\vec{e})] + \nu_4 QP + \nu_5\} \end{aligned} \tag{14}$$

in which, $\{\nu_i | i = 0, 1, \ldots, 5\}$ represent the coefficients derived from the parameters a_N and b.

Because the sigmoidal like activation function in our CNN is an odd-function, it is desired that the mean of two output nodes is zero. Consequently, the teaching output value for O_N is set as

$$O_N = \ln C_{2N} - \ln C_N,$$

and the corresponding teaching output value for O_{2N} is negative of O_N. Considering the effects of QP in (13) and (14), QP is adopted as the inputs to the third and the fourth hidden layers, as shown in Fig3(b). Introducing QP as the input of CNN is the prominent improvement as compared with our previous study [32], up to BDBR=−2.04% rate reduction could be achieved for the test of Kimono.

As $|\ln C_{2N} - \ln C_N|$ always increases with the magnitudes of QP and $\vec{e}$, the output of activation function in our design should be not constrained. Therefore, the active function is our CNN has the form

$$\begin{cases} 1.716\tanh(0.667x) & |x| < \tau \\ 1.716[\tanh(2\tau/3) + \tanh'(2\tau/3)(x-\tau)] & x \geq \tau \\ 1.716[\tanh(-2\tau/3) + \tanh'(-2\tau/3)(x+\tau)] & x \leq -\tau. \end{cases}$$

The values of τ on different CNN layers are derived by experiments. Let τ_i represent the ith layer's threshold. The candidate value set of τ_i is defined as $\tau_i \in \{1.0, 1.1, 1.2, \cdots, 3.5\}$. We test the performance of various combinations of τ_i, and find the optimal one. The threshold values in our design are provided in Table I.

The kernels in the first convolution layers of CNN_{32} and CNN_8 are illustrated in Fig. 4. The white square represents the positive pole and the black one denotes the negative pole. It is obvious that, these kernels are efficient to detect singularities, such as end points and corners. For example, Kernel#0 of CNN_{32} is composed of a negative pole on the first row and a positive one on the second row; One corner is included in Kernel#0 of CNN_8. The computational complexity of the proposed CNN is summarized as Table II. Totally, the CNN processing consumes 3000 multiplications, 3000 additions/subtractions and 244 pseudo sigmoidal functions. The experiments show that FastCUMode function accounts for 2-3% of the HM-12.0 overall Intra encoding time.

TABLE I
DEFINITION OF THRESHOLD τ

CU Size	CNN layer			
	Conv1	Conv2	MLP1	MLP2
32×32	1.3	2.1	2.5	1.7
16×16	1.3	1.7	2.5	2.1
8×8	1.7	1.7	3.0	1.3

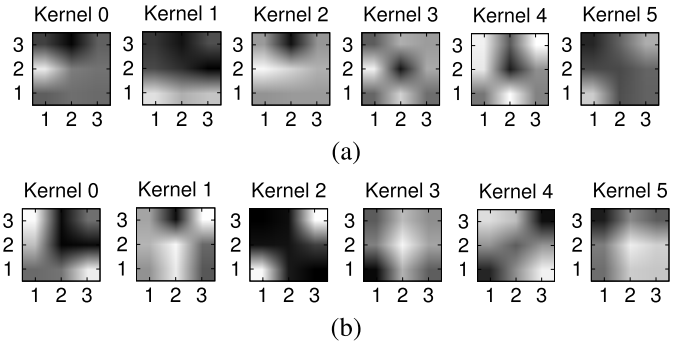


Fig. 4. Feature extraction kernels of the first hidden layer in 32×32/16×16 and 8×8/4×4 CU mode decision CNN. (a) Kernels of CNN_{32}. (b) Kernels of CNN_8.

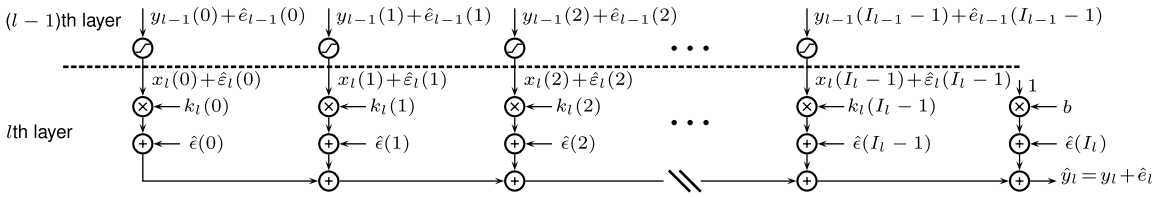


Fig. 5. Noisy fixed-point convolution operation $(y_{l-1}(i) + \hat{e}_{l-1}(i)$ represents the output signal and the additive roundoff error in the $(l-1)$th layer's convolution result; $x_l(i)$ and $\hat{\varepsilon}_l(i)$ coming from the activation function are input signal and additive noise of the lth layer; The kernel coefficients and bias are $k(i)$ and b; $\hat{\epsilon}(i)$ is the additive rounding error from the multiplication operation; $\hat{y}_l$, which is composed of the signal y_l and the noise $\hat{e}_l$, indicates the convolution result of the lth layer).

TABLE II
LAYER-WISE COMPUTATIONAL COMPLEXITY OF PROPOSED CNN

layer	Mult #	Add/Sub #	Act #
Convolution 1	1944	1944	216
Convolution 2	864	864	16
MLP 1	170	170	10
MLP 2	22	22	2
Total	3000	3000	244

IV. VLSI IMPLEMENTATION OF CNN ACCELERATOR

Our analytical models reveal that, when the CNN scale is small, the hardware and power costs of CNN accelerator can be greatly reduced by using the optimal arithmetic and the associated representation format. Based on the optimal arithmetic, a reconfigurable hardwired accelerator is devised to speed up the computation.

A. Effect of Finite Bit-Depth

In this section, we will reveal the impacts of finite bit-depth and network scale to the output noise-to-signal-ratio (NSR) for fixed-point and floating-point, respectively.

Let's build the data-flow graph for computation error analysis. In the convolution procedure, the feature maps in the current layer are obtained from the neuron outputs of the previous layer. This procedure is formulated as

$$\begin{cases} y_l(m,n) = \sum_{t=0}^{T-1} \sum_{p=0}^{K_l-1} \sum_{q=0}^{K_l-1} x_l(t, m+p, n+q) k_l(t,p,q) + b \\ x_{l+1}(m,n) = \text{sigmoid}(y_l(m,n)), \end{cases} \tag{15}$$

in which, $x_l(t, m+p, n+q)$ denotes the pixel (neuron output) from the tth feature map in the previous layer, $k_l(t,p,q)$ and b denote the kernel coefficients and the bias, respectively. Equation (15) is composed of two stages: the biased sum of weighted input pixels (calculating $y_l(m,n)$), and the limitation of $y_l(m,n)$ by activation function, i.e., sigmoid$(y_l(m,n))$. $x_{l+1}(m,n)$ is the input pixel for the following the layer.

Each pixel $y_l(m,n)$ can be viewed as the biased weighted average of all pixels in the receptive field. Without loss of generality, it is assumed that the input signals x_l are independent identically distributed (i.i.d.) random variables (r.v.) [37], [38]. Consequently, the statistical properties of y_l are invariant with respect to the indices m and n. In our error analysis, (15) is simplified with 1-D notation, written as

$$y_l = \sum_{i=0}^{I_l-1} x_l(i) k_l(i) + b, \tag{16}$$

where, $i = t \cdot K_l^2 + p \cdot K_l + q$ and $I_l = T \cdot K_l^2$.

For the fixed-point, a $(1 + \hat{\mathcal{A}} + \hat{\mathcal{B}})$-bit number is composed of 1-bit sign, $\hat{\mathcal{A}}$-bits integer, and $\hat{\mathcal{B}}$-bit fraction. That is

$$\underbrace{bS_0}_{\text{sign}} \underbrace{bI_{\hat{\mathcal{A}}-1} \ldots bI_0}_{\text{integer}} \Delta \underbrace{bF_{\hat{\mathcal{B}}-1} \ldots bF_0}_{\text{fraction}},$$

in which Δ denotes the binary point.

$\hat{\mathcal{A}}$ is devised to prevent overflow when representing the maximum amplitude of CNN input and output signals. The bit-depth of fractional part ($\hat{\mathcal{B}}$) affects the energy of roundoff noises. With no overflow, the additive roundoff noises in the fixed-point CNN merely come from the multiplications in (16). The error analysis model of two successive layers is illustrated in Fig.5. As mentioned in the literatures [37] and [38], it is assumed that all signals and noises are zero-mean independent variables, and the variance of error $\hat{\epsilon}(i)$ is determined by $\hat{\mathcal{B}}$, i.e.,

$$\begin{cases} \mathrm{E}\left[\hat{\epsilon}(i) \cdot \hat{\epsilon}(j)\right] = 0 & \text{when } i \neq j \\ \mathrm{E}\left[\hat{\epsilon}^2(i)\right] = 2^{-2\hat{\mathcal{B}}}/12 & \\ \mathrm{E}\left[x_l(i) \cdot x_l(j)\right] = 0 & \text{when } i \neq j \\ \mathrm{E}\left[x_l^2(i)\right] = \sigma_{x_l}^2 & \\ \mathrm{E}\left[x_l(i) \cdot \hat{\epsilon}(j)\right] = 0 & \forall i \text{ and } \forall j \end{cases}$$

in which, $\mathrm{E}[\cdot]$ denotes the expectation of its input variable.

In general, the activation function in CNN is non-linear. We need to investigate the affects of the activation function to NSR. From Appendix A, we derive lemma 1. Because the traditional sigmoid and Relu functions, and the pseudo sigmoid functions adopted in this article, always satisfy the premises in lemma 1, it is concluded that the output NSR of activation function is less than its input NSR.

Lemma 1: For an activation function $\Theta(y)$, if it possesses the properties of $\Theta(0) = 0$, the sign of $\Theta'(\cdot)$ is invariant, and $\left|\Theta'(y_2)\right| \leq \left|\Theta'(y_1)\right|$ when $|y_2| > |y_1|$, the output NSR is always not greater than the input NSR.

Let NSR_l denote the local NSR of the lth layer. In specific, if $\sigma_{\hat{\epsilon}_l}$ denotes the variance of the noises stemming from the

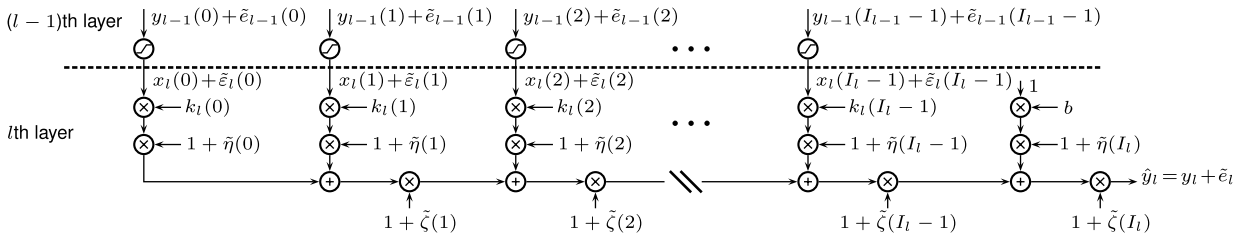


Fig. 6. Noisy floating-point convolution operation ($y_{l-1}(i)+\tilde{e}_{l-1}(i)$ denotes the output signal and the associated roundoff error in the $(l-1)$th layer's convolution result; The kernel coefficients and bias are $k(i)$ and b; $\tilde{\eta}(i)$ and $\tilde{\zeta}(i)$ represent the multiplicative roundoff errors of the floating-point multiplication and addition operations, respectively; $\tilde{y}_l$ being composed of the signal y_l and the noise $\tilde{e}_l$, indicates the convolution result of the lth layer).

lth layer's multiplications, and σ_{y_l} represents the variance of y_l, NSR_l is defined as

$$\widehat{NSR}_l = \frac{\sigma_{\tilde{e}_l}^2}{\sigma_{y_l}^2}.$$

From the analysis in Appendix B, NSR_l is formulated as

$$\widehat{NSR}_l = \frac{2^{-2\hat{\mathcal{B}}}(I_l+1)}{12\sigma_{y_l}^2}. \tag{17}$$

$\widehat{NSR}_l$ always increases linearly with the number of noise sources (I_l), and is inversely proportional to the power of output signal ($\sigma_{y_l}^2$).

Theorem 1: For one fixed-point CNN, of which the activation functions meet the requirements provided in lemma 1, the NSR upper bound of the lth layer output is equal to the sum of all $\widehat{NSR}_i$ $(0 \le i \le l)$, i.e.,

$$\frac{\sigma_{\tilde{e}_l}^2}{\sigma_{y_l}^2} \le \sum_{i=0}^{l} \widehat{NSR}_i \tag{18}$$

The detailed analysis of Theorem 1 is given in Appendix B. From (17) and (18), the final output NSR always increases with the network scale, which is determined by the parameters I_i and l.

For the floating-point arithmetic, a $(1+\tilde{\mathcal{A}}+\tilde{\mathcal{B}})$-bit number is composed of three sections, i.e., 1-bit sign, $\tilde{\mathcal{A}}$-bit exponent, and $\tilde{\mathcal{B}}$-bit mantissa, which is described as

$$\underbrace{bS_0}_{\text{sign}}\ \underbrace{bE_{\tilde{\mathcal{A}}-1}\ldots bE_0}_{\text{exponent}}\ \underbrace{bM_{\tilde{\mathcal{B}}-1}\ldots bM_0}_{\text{mantissa}}. \tag{19}$$

The value of a number x represented by (19) is

$$x = (-1)^{bS_0} \times \left(1 + \sum_{i=0}^{\tilde{\mathcal{B}}-1} bM_i \cdot 2^{i-\tilde{\mathcal{B}}}\right) \times \exp\left(2, \sum_{i=0}^{\tilde{\mathcal{A}}-1} bE_i \cdot 2^i - E_{bias}\right)$$

where, E_{bias} is the exponent bias. In our design, E_{bias} is set as $2^{\tilde{\mathcal{A}}-1}$.

The data flow of noisy floating-point operation based CNN is depicted in Fig. 6. Different from the fixed-point counterpart, the roundoff errors in floating-point operations ($\tilde{\eta}(\cdot)$ and $\tilde{\zeta}(\cdot)$ in Fig. 6) are multiplicative. For example, if we perform the first floating-point multiplication in (16), except for the signal $x_l(0)k_l(0)$, the noise $x_l(0)k_l(0)\tilde{\eta}(0)$ is also generated. In addition, all multiplications and additions in floating-point CNN will generate the noises.

As the investigation of fixed-point CNN, it is assumed that $\tilde{\eta}(i)$ and $\tilde{\zeta}(i)$ are zero-mean i.i.d. r.v. [37]. If $\tilde{\eta}(i)$ and $\tilde{\zeta}(i)$ are both uniformly distributed in the range of $[-2^{-(\tilde{\mathcal{B}}+1)}, 2^{-(\tilde{\mathcal{B}}+1)}]$, the variances of $\tilde{\eta}(i)$ and $\tilde{\zeta}(i)$ can be expressed as

$$\sigma_{\tilde{\eta}}^2 = \sigma_{\tilde{\zeta}}^2 = \frac{2^{-2\tilde{\mathcal{B}}}}{28}.$$

From the above properties, we could derive the local NSR, i.e., $\widetilde{NSR}_l$, which is produced by the floating-point operations in the lth layer. The mathematical analysis in Appendix C yields the expression of $\widetilde{NSR}_l$ as

$$\widetilde{NSR}_l = \frac{2^{-2\tilde{\mathcal{B}}}(I_l+2)}{28} \cdot \frac{\sum_{i=0}^{I_l-1}\left(1-\frac{i}{I_l+2}\right)k_l^2(i)\sigma_{x_l}^2(i)}{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{x_l}^2(i)}.$$

In consequent, the NSR upper bound for multiple-layer floating-point CNN is given by Theorem 2. The associated investigation can be referred in Appendix C.

Theorem 2: For one floating-point CNN, of which the activation functions meet the requirements provided in lemma 1, the NSR upper bound of the lth layer output is

$$\frac{\sigma_{\tilde{e}_l}^2}{\sigma_{y_l}^2} \le \left(\sum_{i=0}^{l} I_i + 2l\right)\frac{2^{-2\tilde{\mathcal{B}}}}{28}$$

From Theorem 1 and Theorem 2, it is concluded that the NSR upper bound is linear with the network scale. Namely, for the small scale CNN, we can reduce the bit-depth of fixed-point fraction part and that of floating-point mantissa part, while maintaining the desired precision. As to be illustrated in Section V, 12-bit fixed-point ($\hat{A}=5$ and $\hat{B}=6$) and 10-bit floating-point ($\tilde{A}=4$ and $\tilde{B}=5$) both fulfill the precision requirements.

As to be discussed in Section IV-B, the primary arithmetic modules in our CNN accelerator include nine multipliers and three four-operand adder-trees. As compared to the fixed-point arithmetic, because of the bit-depth reduction, the hardware saving gains of the floating-point multipliers outweigh the losses arising from the floating-point adder-trees, which finally leads to the averaged 31% gate count reduction for the arithmetic operations. Therefore, we adopted the floating-point arithmetic in our CNN accelerator design.

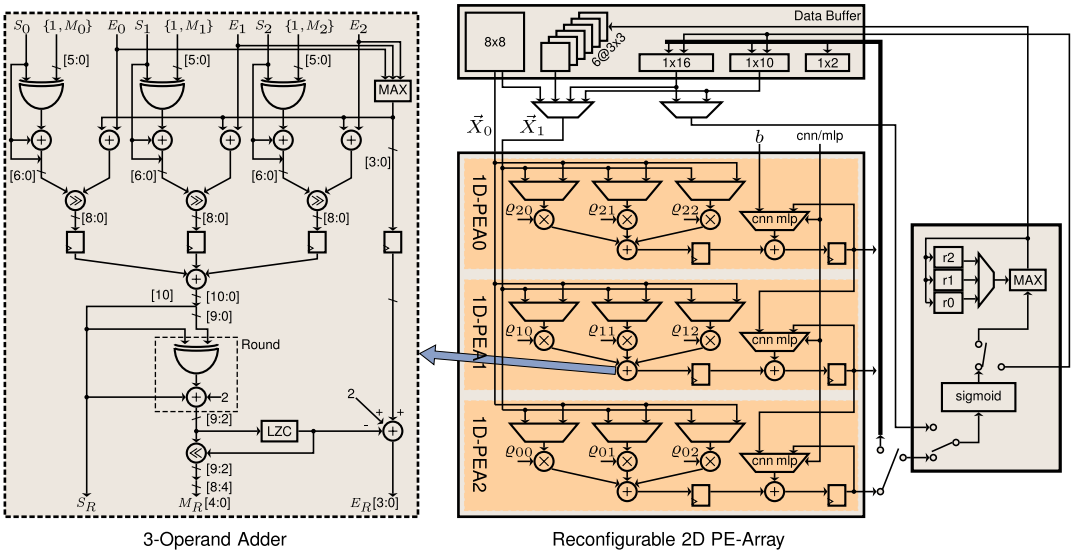


Fig. 7. Reconfigurable *PPMAC* Architecture of CNN Accelerator Design (LZC: leading zero counter; sigmoid: sigmoid function shared by convolution and MLP layers; S_i, E_i and M_i ($i \in \{0, 1, 2\}$): sign, exponent and mantissa of the ith operand to 3-Operand Adder; S_R, E_R and M_R: sign, exponent and mantissa of 3-Operand Adder result; ϱ_{ij} is the kernel coefficient in convolutional layers and the weight in MLP layers).

TABLE III
DATA FLOW OF PROPOSED *PPMAC* IN CONVOLUTION PROCESSING

cycle	$\vec{X}_0$	$\vec{X}_1$	1D-PEA0	1D-PEA1	1D-PEA2	Derived MAC
0	$\{x_{00},x_{10},x_{20}\}$	-	-	-	-	-
1	$\{x_{01},x_{11},x_{21}\}$	-	-	-	-	-
2	$\{x_{02},x_{12},x_{22}\}$	-	-	-	-	-
3	$\{x_{03},x_{13},x_{23}\}$	-	$\mathrm{MAC}_{\mathrm{row}0}(0,0)$	-	-	-
4	$\{x_{04},x_{14},x_{24}\}$	-	$\mathrm{MAC}_{\mathrm{row}0}(0,1)$	$\mathrm{MAC}_{\mathrm{row}1}(0,0)$	-	-
5	$\{x_{05},x_{15},x_{25}\}$	-	$\mathrm{MAC}_{\mathrm{row}0}(0,2)$	$\mathrm{MAC}_{\mathrm{row}1}(0,1)$	$\mathrm{MAC}_{\mathrm{row}2}(0,0)$	$\mathrm{MAC}(0,0)$
6	$\{x_{10},x_{20},x_{30}\}$	$\{x_{06},x_{16},x_{26}\}$	$\mathrm{MAC}_{\mathrm{row}0}(0,3)$	$\mathrm{MAC}_{\mathrm{row}1}(0,2)$	$\mathrm{MAC}_{\mathrm{row}2}(0,1)$	$\mathrm{MAC}(0,1)$
7	$\{x_{11},x_{21},x_{31}\}$	$\{x_{07},x_{17},x_{27}\}$	$\mathrm{MAC}_{\mathrm{row}0}(0,4)$	$\mathrm{MAC}_{\mathrm{row}1}(0,3)$	$\mathrm{MAC}_{\mathrm{row}2}(0,2)$	$\mathrm{MAC}(0,2)$
8	$\{x_{12},x_{22},x_{32}\}$	-	$\mathrm{MAC}_{\mathrm{row}0}(0,5)$	$\mathrm{MAC}_{\mathrm{row}1}(0,4)$	$\mathrm{MAC}_{\mathrm{row}2}(0,3)$	$\mathrm{MAC}(0,3)$
9	$\{x_{13},x_{23},x_{33}\}$	-	$\mathrm{MAC}_{\mathrm{row}0}(1,0)$	$\mathrm{MAC}_{\mathrm{row}1}(0,5)$	$\mathrm{MAC}_{\mathrm{row}2}(0,4)$	$\mathrm{MAC}(0,4)$
10	$\{x_{14},x_{24},x_{34}\}$	-	$\mathrm{MAC}_{\mathrm{row}0}(1,1)$	$\mathrm{MAC}_{\mathrm{row}1}(1,0)$	$\mathrm{MAC}_{\mathrm{row}2}(0,5)$	$\mathrm{MAC}(0,5)$
11	$\{x_{15},x_{25},x_{35}\}$	-	$\mathrm{MAC}_{\mathrm{row}0}(1,2)$	$\mathrm{MAC}_{\mathrm{row}1}(1,1)$	$\mathrm{MAC}_{\mathrm{row}2}(1,0)$	$\mathrm{MAC}(1,0)$
12	$\{x_{20},x_{30},x_{40}\}$	$\{x_{16},x_{26},x_{36}\}$	$\mathrm{MAC}_{\mathrm{row}0}(1,3)$	$\mathrm{MAC}_{\mathrm{row}1}(1,2)$	$\mathrm{MAC}_{\mathrm{row}2}(1,1)$	$\mathrm{MAC}(1,1)$
⋮	⋮		⋮		⋮	⋮

$x_{m,n}$ is the entry of input matrix; $\mathrm{MAC}_{\mathrm{row}\,i}(m,n)$ represents the partial MAC of the ith row at convolution candidate (m,n); m is the index of abscissa, and n is the index of ordinate

B. Propagate Partial Multiply-Accumulate CNN Accelerator

To accelerate 2D convolution and MLP processing, we proposed the reconfigurable *PPMAC* architecture, as shown in Fig. 7. The data flow of *PPMAC* is inspired by the *Propagate Partial SAD* architecture of motion estimation accelerator in H.264/AVC video encoder [39]. The architecture is composed of three row-wise 1D PE arrays (1D-PEAi, $i \in \{0, 1, 2\}$). 1D-PEAi is of 4-stage pipeline, including 1-stage multiplier, 2-stage 3-operand adder, and the last stage 2-operand adder.

For the convolution, the last stage adder of 1D-PEAi ($i \in \{1, 2\}$) is configured to fetch the output of 1D-PEA$i-1$ as the operand. This operand of 1D-PEA0 is the bias, i.e., b in (15). The row-wise input vectors of 1D-PEAi are $\vec{X}_0$ and $\vec{X}_1$, which are both of the size of 1×3 (one row by three columns). The data flow of *PPMAC* is described in Table III. With the 4-stage pipeline 1D-PEAi, our design achieved 714MHz clock speed even at the worst working conditions (0.9v, 125°C). It should be emphasized that, our design needs not be compliant with IEEE-754 specifications, which provides more design space for hardware optimizations. For example, the 3-operand adder in our design applies the 2-stage uniform aligning based addition architecture, as shown in Fig. 7, to reduce the circuits complexity. To maintain the precision, after the uniform aligning, we extend two least significant bits. The rounding operation is a must. Otherwise, the result amplitude is always decreased. With the above optimizations, 6.4-38.9% chip area could be saved for the 3-operand adder. Because the last stage adder in 1D-PEAi is used as the accumulator in MLP, it cannot be divided into more pipeline stages. For this adder, we apply the 1-stage dual-path approach [40]. Except the two initial cycles, the hardware utilization of 2D PE-Array is 100% during the first convolution layer.

In the first layer convolution, the output of 2D PE-Array is directly dispatched to sigmoid module, which then feeds its output to the 2×2 max-pooling stage. Because the convolution

TABLE IV
DATA FLOW OF *PPMAC* IN MLP PROCESSING

cycle	$\vec{X}_1$	1D-PEA0	1D-PEA1	1D-PEA2
0	$\{x_0, x_1, x_2\}$	-	-	-
1	$\{x_3, x_4, x_5\}$	-	-	-
2	$\{x_6, x_7, x_8\}$	b_0	b_1	b_2
3	$\{x_9, x_{10}, x_{11}\}$	$b_0+\sum_{i=0}^{2} x_i w_{i0}$	$b_1+\sum_{i=0}^{2} x_i w_{i1}$	$b_2+\sum_{i=0}^{2} x_i w_{i2}$
4	$\{x_{12}, x_{13}, x_{14}\}$	$b_0+\sum_{i=0}^{5} x_i w_{i0}$	$b_1+\sum_{i=0}^{5} x_i w_{i1}$	$b_2+\sum_{i=0}^{5} x_i w_{i2}$
5	$\{x_{15}, x_{16}, 0\}$	$b_0+\sum_{i=0}^{8} x_i w_{i0}$	$b_1+\sum_{i=0}^{8} x_i w_{i1}$	$b_2+\sum_{i=0}^{8} x_i w_{i2}$
6	-	$b_0+\sum_{i=0}^{11} x_i w_{i0}$	$b_1+\sum_{i=0}^{11} x_i w_{i1}$	$b_2+\sum_{i=0}^{11} x_i w_{i2}$
7	$\{x_0, x_1, x_2\}$	$b_0+\sum_{i=0}^{14} x_i w_{i0}$	$b_1+\sum_{i=0}^{14} x_i w_{i1}$	$b_2+\sum_{i=0}^{14} x_i w_{i2}$
8	$\{x_3, x_4, x_5\}$	$b_0+\sum_{i=0}^{15} x_i w_{i0}$	$b_1+\sum_{i=0}^{15} x_i w_{i1}$	$b_2+\sum_{i=0}^{15} x_i w_{i2}$
9	$\{x_6, x_7, x_8\}$	b_3	b_4	b_5
10	$\{x_9, x_{10}, x_{11}\}$	$b_3+\sum_{i=0}^{2} x_i w_{i3}$	$b_4+\sum_{i=0}^{2} x_i w_{i4}$	$b_5+\sum_{i=0}^{2} x_i w_{i5}$
11	$\{x_{12}, x_{13}, x_{14}\}$	$b_3+\sum_{i=0}^{5} x_i w_{i3}$	$b_4+\sum_{i=0}^{5} x_i w_{i4}$	$b_5+\sum_{i=0}^{5} x_i w_{i5}$
⋮	⋮		⋮	

is column-wise and one pixel is generated in each cycle, three pixel registers (r0, r1, and r2) are equipped to store the local maximum pixels in the corresponding 2×2 receptive fields.

For the MLP layer, each neuron y_j in the current layer first forms the biased inner product of its weight vector (w_{ij}) and the output vector of the previous layer (x_i), which is formulated as

$$y'_j = \sum_{i=0}^{I-1} w_{ij} x_i + b_j, \tag{20}$$

and then, emits

$$y_j = \text{sigmoid}(y'_j). \tag{21}$$

In MLP computation, each 1D-PEA is configured to work independently. In specific, the final adder in 1D-PE*Ai* fetches one operand from the last stage accumulation registers. The 3-pixel inputs of 1D-PE*Ai* all come from $\vec{X}_1$. The outputs of 1D-PE*Ai* belong to different neurons in the output layer. The initial value of accumulation registers is set as the corresponding bias. The data flow of Hidden 17@1×1 layer to Hidden 11@1×1 layer processing is described as Table IV. It should be noticed that the Feature maps 6@3×3 to Hidden 17@1×1 convolution operations also adopt the MLP alike data flow, instead of the one in Table III. This is mainly because of that, the output number for each input feature map is one, considering the bubbles in initialization, the hardware utilization of 2D-PE Array is merely 33% with the convolution data flow. In contrast, using MLP data flow, this metric is improved up to 84%. In MLP procedure, 1D-PE*Ai* implements the MAC operations in (20), and the intermediate results are saved in Data Buffer, as shown in Fig. 7. The sigmoid module fetches the intermediate MAC results and carries out the activation processing (expressed as (21)), which does not disturb the pipeline scheduling of 1D-PE*Ai*.

V. EXPERIMENTS

In this section, we first evaluate the coding performance of the proposed fast CU/PU mode decision algorithm. Thereafter, we analyze the effects of finite precision arithmetic to the coding quality. Finally, we investigate the throughput, the hardware cost, and the power dissipation of our CNN accelerator in detail.

A. Performance Analysis of Fast CU/PU Mode Decision

The proposed method is conducted on HEVC reference test model HM12.0. The test platform is Huawei RH5885, which combines Intel® Xeon™ E7-4830-v2 2.20GHz processor and 128.0GB RAM. Twenty-seven typical open test sequences, including Classes A-F, were tested with QP ={22, 27, 32, 37} and *intra_main* configuration [41]. To verify the robustness of our algorithm, we introduced additional six HD1080p@25fps in-house sequences provided by Huawei®. These videos cover the representative surveillance scenarios, such as the building internal environment (B2Inside), the open space with moving peoples (3PeopleD4, GrassTreeD4), the traffic scenes (RoadB, RoadCar), and the surveillance under weak light (LowlightNight).

The coding performance results are shown in Table V, in which the original HM12.0 is the standard benchmark. BP and BR in Table V stand for the average PSNR difference (BDPSNR) and the average bit rates difference (BDBR) [42], respectively. ΔT represents the encoding time reduction, which is defined as

$$\Delta T = (T_{HM} - T_{CNN})/T_{HM} \times 100\%,$$

with T_{HM} and T_{CNN} denoting the encoding time of original HM12.0 and the CNN based counterpart, respectively. To recall the pseudo codes in Fig.2, we can indicate the specific CU/PU depth, in which the fast mode decision is performed. In our evaluations, four configurations, i.e., [E(32),E(16),E(8)]={[0,0,1], [0,1,1], [1,0,1], [1,1,1]} were tested. Obviously, our algorithm possessed the computational scalability. That is, we could reduce more complexity by scarifying the compression efficiency. It was observed that the configuration [0,0,1] achieved the best coding quality, i.e., on average BDBR=+1.54%, while the encoding time reduction was merely 43.7%. In contrast, the time saving of [1,1,1] was 72.0%. Meanwhile, its coding performance was degraded with BDBR=+4.79%. The configuration of [1,0,1] achieved the good balance between the coding efficiency and the computational reduction. In this context, the BDBR augment was +2.67%, while 61.1% encoding complexity was saved. As compared with the counterpart [0,1,1], the setting of [1,0,1] had the advantages of both coding quality and processing speed.

The performance comparisons of the proposed algorithm and other recent works [14], [16], [19]–[21], [26], [32] are illustrated in Table VI. ΔT_C and ΔT_P stand for the encoding time reductions from fast CU/PU mode and prediction mode methods, respectively. BR_{MAX} represents the maximum BDBR increase. VLSI indicates the algorithm's friendship to

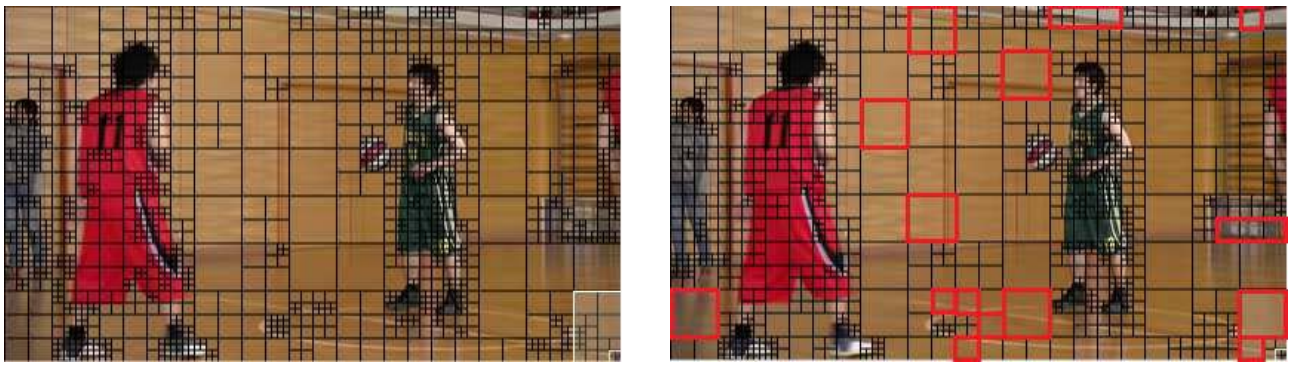
TABLE V
CODING PERFORMANCE ANALYSIS OF OUR FAST CU/PU MODE DECISION UNDER VARIOUS CONFIGURATIONS

Class	Sequence	Proposed Configurations											
		[1,0,1]			[0,0,1]			[0,1,1]			[1,1,1]		
		BP	BR	ΔT	BP	BR	ΔT	BP	BR	ΔT	BP	BR	ΔT
A	PeopleOnStreet	-0.11	2.27	61.8	-0.08	1.62	45.6	-0.21	4.10	63.4	-0.26	5.24	74.6
	Traffic	-0.11	2.35	60.7	-0.06	1.27	43.7	-0.16	3.38	60.4	-0.24	5.01	73.4
Class A Average		-0.11	2.31	61.3	-0.07	1.45	44.7	-0.19	3.74	61.9	-0.25	5.13	74.0
B	BasketballDrive	-0.09	3.49	70.9	-0.02	0.94	50.5	-0.10	3.82	65.5	-0.14	5.52	76.1
	BQTerrace	-0.11	2.26	62.4	-0.08	1.54	46.9	-0.17	3.33	60.8	-0.20	4.03	72.3
	Cactus	-0.09	2.59	60.4	-0.05	1.43	43.9	-0.11	3.48	60.4	-0.16	4.72	72.8
	Kimono	-0.08	2.47	62.6	-0.00	0.01	45.1	-0.03	1.02	64.5	-0.12	3.64	77.5
	ParkScene	-0.07	1.86	60.3	-0.04	0.87	44.1	-0.10	2.52	60.3	-0.16	3.97	72.0
	Tennis	-0.09	3.00	65.5	-0.02	0.68	46.6	-0.08	2.69	63.4	-0.14	4.92	75.5
	3PeopleD4	-0.09	0.98	57.6	-0.07	0.79	42.3	-0.15	1.58	55.6	-0.18	1.90	66.2
	B2Inside	-0.14	2.42	60.9	-0.09	1.47	43.5	-0.22	3.66	59.3	-0.25	4.23	70.4
	GrassTreeD4	-0.11	1.25	57.0	-0.08	0.92	40.7	-0.17	1.88	54.5	-0.19	2.16	66.3
	LowlightNight	-0.05	1.56	69.5	-0.03	0.76	56.5	-0.08	2.40	69.1	-0.13	3.70	77.5
	RoadB	-0.08	1.29	55.4	-0.03	0.48	40.1	-0.13	2.08	55.0	-0.25	3.93	67.6
	RoadCar	-0.09	1.51	57.6	-0.05	0.75	42.9	-0.15	2.45	58.0	-0.28	4.53	69.7
Class B Average		-0.09	2.06	61.7	-0.05	0.89	45.3	-0.12	2.58	60.5	-0.18	3.94	72.0
C	BQMall	-0.16	2.93	57.8	-0.11	2.01	39.3	-0.21	3.97	57.1	-0.26	4.97	71.5
	BasketballDrill	-0.19	4.26	56.9	-0.12	2.65	38.5	-0.27	5.91	56.1	-0.33	7.26	70.2
	PartyScene	-0.15	2.19	51.1	-0.03	0.86	33.5	-0.20	2.97	50.8	-0.22	3.17	65.8
	RacehorsesC	-0.12	2.08	56.2	-0.07	1.23	37.7	-0.18	2.98	55.2	-0.25	4.23	69.7
Class C Average		-0.16	2.87	55.5	-0.08	1.69	37.3	-0.22	3.96	54.8	-0.27	4.91	69.3
D	BasketballPass	-0.16	2.89	59.3	-0.11	2.04	37.9	-0.21	3.86	59.3	-0.25	4.56	71.4
	BlowingBubbles	-0.14	2.54	53.6	-0.13	2.34	34.5	-0.21	3.91	54.6	-0.23	4.21	69.0
	BQSquare	-0.11	1.48	52.6	-0.08	1.26	33.0	-0.13	1.78	51.4	-0.14	1.86	66.4
	RaceHorses	-0.14	2.43	53.6	-0.12	2.01	36.6	-0.22	3.80	55.4	-0.27	4.53	69.3
	Keiba	-0.12	2.06	65.2	-0.08	1.33	39.5	-0.21	3.57	57.8	-0.31	5.28	69.0
Class D Average		-0.13	2.28	56.9	-0.10	1.80	36.3	-0.20	3.38	55.7	-0.24	4.09	69.0
E	Vidyo1	-0.15	3.23	68.7	-0.07	1.64	51.0	-0.21	4.72	66.6	-0.27	5.97	76.9
	Vidyo3	-0.25	5.01	63.9	-0.07	1.47	45.8	-0.23	4.69	63.0	-0.45	9.37	73.4
	Vidyo4	-0.12	2.87	68.1	-0.05	1.13	47.4	-0.14	3.48	65.8	-0.22	5.31	76.2
	Johnny	-0.17	4.42	72.2	-0.07	1.89	60.2	-0.22	5.63	71.6	-0.29	7.58	79.3
	KristenAndSara	-0.15	3.12	69.3	-0.09	1.83	54.5	-0.20	4.18	67.6	-0.25	5.36	76.9
	FourPeople	-0.16	3.05	60.0	-0.12	2.19	47.0	-0.25	4.79	63.1	-0.32	6.18	72.4
Class E Average		-0.17	3.62	67.0	-0.08	1.69	51.0	-0.21	4.58	66.3	-0.30	6.63	75.9
F	SlideEditing	-0.38	2.69	54.8	-0.34	2.42	33.9	-0.41	2.91	52.8	-0.50	3.52	67.4
	ChinaSpeed	-0.31	3.74	62.8	-0.22	2.60	44.3	-0.33	3.94	59.5	-0.39	4.66	72.3
	SlideShow	-0.34	4.02	70.1	-0.30	3.50	57.5	-0.46	5.45	69.2	-0.52	6.17	76.3
	BasketballDrilltext	-0.20	3.91	56.3	-0.14	2.75	37.3	-0.28	5.60	55.4	-0.33	6.54	69.7
Class F Average		-0.31	3.59	61.0	-0.25	2.82	43.25	-0.37	4.48	59.2	-0.44	5.22	71.4
Overall Average		-0.15	2.67	61.1	-0.09	1.54	43.7	-0.19	3.53	60.1	-0.26	4.79	72.0

the hardwired encoder design. All performance statistics of previous works are cited from the references.

The software oriented fast algorithms, i.e., literatures [14], [16], [19], [21], provide the superior coding quality than ours. However, for the hardwired encoder design, the above methods have the following hindrances: First, the CTU-grain maximum encoding complexity is not pruned, which impedes the fast algorithm contributing to the optimization of the encoder's hardware cost. For instance, to maintain the coding quality, literatures [19], [21] define the conservative thresholds. Only when one CU satisfies these thresholds, its CU splitting trial is terminated; Otherwise, the CU mode decision is carried out as the original full RDO procedure. We can see that the fast CU mode method in [19] merely reduced 26% of the overall encoding time. The algorithm of literature [14] has the similar problem: the maximum encoding complexity of one CTU, which occurs in the parameter estimation phase, does not diminish. Second, the CU level data dependency in [16], which uses the current CU depth coding information to prune the RDO processing of the deeper CU levels, incurs the cumbersome to encoding parallelism. Namely, if this algorithm is adopted, the encoding throughput will be demoted.

Literatures [20], [26], [32] are VLSI friendly, because these methods merely employ the source image texture analysis to predict the promising CU mode competitors. As depicted in Fig. 3, for the CU mode pre-decision engine can be allocated in the advanced CTU pipeline stage, the rhythm of following RDO based CTU encoding will not be hindered. However, because [20] and [26] did not use the topology information of feature points, their coding efficiency losses are obvious, i.e., BDBR=+5.10% and BDBR=+4.53%, respectively. In the



(a) (b)

Fig. 8. CU/PU partition comparisons between HM-12.0 and our fast algorithm (BasketballPass when QP=22). (a) HM-12.0. (b) Proposed Algorithm.

TABLE VI

PERFORMANCE COMPARISONS BETWEEN PROPOSED SOLUTION AND EXISTING ALGORITHMS

Algorithm	ΔT_{C} [%]	ΔT_{P} [%]	ΔT [%]	$\mathrm{BR}_{\mathrm{MAX}}$ [%]	BR [%]	VLSI
[14]	50-α	α	50	1.04	0.71	No
[16]	47	0	47	3.40	1.08	No
[19]†	26	45	60	1.80	1.06	No
[20]†	52	5	57	7.00	5.10	Yes
[21]	52	0	52	3.64	0.82	No
[26]†	62	0	62	6.73	4.53	Yes
[32]	61	0	61	5.72	3.39	Yes
Proposed	61	0	61	5.01	2.67	Yes

† : class F sequences were not tested.
$\mathrm{BR}_{\mathrm{MAX}}$: Maximum BDBR augment in tested sequences.
VLSI : abbreviation of VLSI friendly, indicates whether the algorithm possesses the properties provided in Section I .

TABLE VII

CODING QUALITY ANALYSIS WITH DIFFERENT REPRESENTATION FORMATS (BDBR UNIT:%)

Floating-Point					Fixed-Point				
$\tilde{B}$	$\tilde{A}$				$\hat{B}$	$\hat{A}$			
	3	4	5	6		3	4	5	6
4	3.67	2.81	2.81	2.81	4	4.40	3.63	3.49	3.47
5	3.84	2.66	2.66	2.66	5	4.23	3.19	2.96	2.93
6	3.89	2.63	2.63	2.63	6	4.28	2.98	2.68	2.65
7	4.00	2.64	2.65	2.65	7	4.99	3.05	2.68	2.64
8	4.04	2.66	2.66	2.66	8	4.81	3.09	2.69	2.65

work [32], we adopted CNN to resolve the above problem, which ameliorates the coding loss to BDBR=+3.39%. As compared to [32], the primary improvements of our most recent study include the introduction of QP in CNN, and the refinement of training strategies. Consequently, the averaged BDBR is merely +2.67%. The worst case comes from Vidyo3, of which the BDBR augments of this study and the counterpart [32] are +5.01% and +5.72%, respectively.

The visualized CU/PU partition comparisons between the HM12.0 benchmark and ours (with the configuration [E(32),E(16),E(8)]=[1,0,1]) are illustrated in Fig.8. Our results of CU blocks with strong singularities, such as the blocks on object outlines, are consistent well with benchmarks. Whereas, as high-lighted with the red borders, for the blocks being lack of features, especially on picture boundaries, the probability of false prediction increases.

B. Effects of Finite Bit-Depth Arithmetic

We analyze the effects of finite precision to the video coding quality, including both of fixed-point and floating-point. As aforementioned, the computational precision is improved by increasing the bit-depth. On the other hand, the CNN hardware cost also rises proportionally to the bit-depth. Our target is to find the minimum bit-depth that satisfies the coding quality requirements. Twenty representation formats were tested for floating-point and fixed-point, respectively, as shown by Table VII. With 4-bit exponent and 5-bit mantissa, the floating-point arithmetic achieved the competitive compression ratio as the traditional 32-bit floating-point computation. The sequence-wise statistics are provided in Table VIII. As compared with Table V, it was illustrated that the fluctuations in performance incurred by the short bit-depth is less than BDBR=0.6% in sequence Kimono. For fixed-point, the comparative coding quality is obtained with 12-bit representation format, in which $\hat{A} = 5$ and $\hat{B} = 6$. Experiments revealed that the 10-bit floating-point CNN accelerator could save 21% hardware cost as compared with the 12-bit fixed-point counterpart.

C. Performance Analysis of CNN Accelerator

With TSMC 65nm CMOS technology, our CNN accelerator was described with Verilog-HDL and was synthesized with SYNOPSYS Design Compiler. IC-Compiler was adopted to implement the place-and-route job.

The synaptic weights storage is one hardware consuming component for CNN VLSI implementation. In literature [43], EDRAM is applied in the general purpose CNN processor to hold all synapses values on-chip. In our design, if we use the register file to buffer the weights, the corresponding hardware cost is 23.2k-gate. As the weight coefficients were trained off-line, we adopted combinational logic to realize the synaptic weight storage, which merely accounted for 8.9k-gate. As compared with the traditional on-chip register-file approach, 14.3k gates were saved.

TABLE VIII
CODING QUALITY ANALYSIS OF FLOATING-POINT WITH $\tilde{A}=4$, $\tilde{B}=5$ AND [E(32),E(16),E(8)]=[1,0,1]

Class	Sequence	BP[dB]	BR[%]	ΔT[%]
A	PeopleOnStreet	-0.12	2.31	62.3
	Traffic	-0.11	2.35	60.8
Class A Average		-0.11	2.33	61.6
B	BasketballDrive	-0.09	3.67	71.0
	BQTerrace	-0.12	2.32	62.4
	Cactus	-0.09	2.57	59.9
	Kimono	-0.10	3.07	59.8
	ParkScene	-0.07	1.82	59.3
	Tennis	-0.09	3.23	64.7
	3PeopleD4	-0.09	0.98	57.6
	B2Inside	-0.14	2.38	61.1
	GrassTreeD4	-0.11	1.22	56.9
	LowlightNight	-0.06	1.64	69.6
	RoadB	-0.08	1.30	54.0
	RoadCar	-0.10	1.54	57.4
Class B Average		-0.10	2.15	61.1
C	BQMall	-0.15	2.77	57.3
	BasketballDrill	-0.18	3.98	56.6
	PartyScene	-0.13	1.95	50.6
	RacehorsesC	-0.12	2.03	54.2
Class C Average		-0.15	2.68	54.7
D	BasketballPass	-0.16	2.92	59.6
	BlowingBubbles	-0.13	2.44	53.0
	BQSquare	-0.11	1.39	53.4
	RaceHorses	-0.14	2.41	52.7
	Keiba	-0.12	2.03	65.3
Class D Average		-0.13	2.24	56.8
E	Vidyo1	-0.16	3.46	68.0
	Vidyo3	-0.25	4.97	63.9
	Vidyo4	-0.13	3.04	67.8
	Johnny	-0.18	4.58	72.7
	KristenAndSara	-0.15	3.10	69.3
	FourPeople	-0.16	3.06	59.6
Class E Average		-0.17	3.70	66.9
F	SlideEditing	-0.33	2.36	54.6
	ChinaSpeed	-0.28	3.38	62.5
	SlideShow	-0.33	3.88	70.6
	BasketballDrilltext	-0.19	3.74	55.7
Class F Average		-0.28	3.34	60.9
Overall Average		-0.14	2.66	60.7

BR_{MAX}: 4.97% for sequence Vidyo3.

TABLE IX
HARDWARE COST AND POWER CONSUMPTION ANALYSIS OF PROPOSED CNN ACCELERATOR (TSMC 65nm CMOS TECHNOLOGY, WORST CONDITIONS: 125°C, 0.9V)

HW&PWR		Freq.(MHz)					
		300	400	500	600	700	714
HW (kgate)	Combinational	18.6	20.2	23.3	26.5	29.7	32.7
	Sequential	9.6	9.6	9.6	9.7	9.7	9.8
	Total	28.2	29.8	32.9	36.2	39.4	42.5
PWR(mW)		3.1	4.6	6.8	9.7	14.5	16.2

The hardware cost and power consumption statistics of the proposed CNN accelerator design under typical working frequencies are illustrated in Table IX. Under the worst work conditions (0.9v, 125°C), the peak clock speed is 714MHz and the associated gate count is 42.5-k. The CNN accelerator is merely 2.1-3.9% of the overall encoder costs, which are 2055k-gate and 1086k-gate in [25] and [26], respectively. The power consumption of our design is 16.2mW@714MHz. It takes 372 cycles to process one 8×8 image block. One CNN accelerator can fulfill the throughput of HD1080p@55fps real-time encoding. Because our algorithm is based on the source texture analysis, multiple 8×8 image blocks could be processed in parallel. When facing the higher resolution specifications, the throughput requirements are met by using parallelism. For example, the real-time processing of 4K@55fps videos can be handled by using four proposed CNN accelerators.

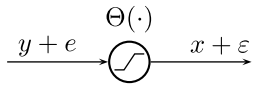


Fig. 9. Data flow of the noisy signal propagating the activation function.

VI. CONCLUSION

This paper presents the CNN based fast CU/PU mode decision to reduce the maximum Intra coding complexity of one CTU for hardwired HEVC encoder design. Specifically, CNN investigates the textures of one CU, and then determines the promising candidate in each of $32\times32/16\times16$ and $8\times8/4\times4$ CU/PU mode pairs. The contributions of our proposals include: (1) Because the maximum number of CU/PU candidate mode in one CTU is reduced, the corresponding VLSI encoder hardware complexity is ameliorated; (2) With the CTU pipeline architecture, the parallelism of the critical RDO processing will not be deteriorated by our fast algorithm; (3) On the basis of the theoretical analysis, a reconfigurable CNN accelerator is developed using the optimal floating-point arithmetic, which greatly reduces the hardware overhead of our algorithm. The experiments demonstrate that, on average, 61.1% Intra encoding complexity is reduced, whereas the incurred compression loss is merely BDBR=+2.67%, or equivalently BDPSNR=−0.15dB quality loss. Using TSMC 65nm CMOS techniques, one hardwired CNN accelerator, which achieved 714MHz clock speed in the worst conditions, is implemented with 42.5-k logic gates. The power dissipation of our accelerator is 16.2mW@714MHz. One CNN accelerator fulfills the throughput requirement of HD1080p@55fps real-time encoding, and the higher performance can be achieved by applying parallelism.

APPENDIX A
NSR PROPAGATION PROPERTY OF ACTIVATION FUNCTION

As shown in Fig.9, if the input signal y and the additive noise e are independent zero-mean r.v., the output signal and the corresponding noise of the activation function are labeled as x and ε, respectively. It is reasonable to assume that the variance of e, i.e., σ_e, is much less than that of input signal, i.e., σ_y. With the aid of Taylor's theorem, we could derive that

$$\begin{cases} x = \Theta(y) \\ \varepsilon = \Theta'(y)\cdot e. \end{cases} \tag{A.1}$$

For any y, $x = \Theta(y)$ can be formulated as

$$x = \Theta(0) + \Sigma_{k=0}^{N-1} \left(\Theta'(k\Delta) \cdot \Delta\right)$$

where, $\Delta = y/N$ and $N \to \infty$. Using the properties of $\Theta(\cdot)$ claimed in lemma1, we deduce

$$|x| \geq \left|\Theta'(y)\Sigma_{k=0}^{N-1}\Delta\right| = \left|\Theta'(y)y\right|. \tag{A.2}$$

From (A.1) and (A.2), we obtain that

$$\begin{aligned} \mathrm{E}\left[\frac{\varepsilon^2}{x^2}\right] &= \mathrm{E}\left[\frac{e^2(\Theta'(y))^2}{\Theta^2(y)}\right] \\ &\leq \mathrm{E}\left[\frac{e^2(\Theta'(y))^2}{y^2(\Theta'(y))^2}\right] \\ &= \quad \mathrm{E}\left[\frac{e^2}{y^2}\right]. \end{aligned}$$

Appendix B
Rounding Error Analysis of Fixed-Point CNN

Let the input noise signals $\varepsilon_l(i)$ are i.i.d. r.v. with the variance σ_{ε_l}. From the data flow in Fig.5, we have

$$\sigma_{y_l}^2 = \mathrm{E}\left[\left(\sum_{i=0}^{I_l-1} x_l(i)k_l(i)\right)^2\right]. \tag{B.1}$$

Using $\mathrm{E}[x_l^2(i)] = \sigma_{x_l}^2$ and $\mathrm{E}[x_l(i)\cdot x_l(j)] = 0$ (if $i \neq j$), (B.1) is recast to

$$\sigma_{y_l}^2 = \sigma_{x_l}^2 \cdot \parallel \vec{k}_l \parallel^2, \tag{B.2}$$

where, $\parallel \vec{k}_l \parallel^2 = \sum_{i=0}^{I_l-1} k_l^2(i)$. Assuming the roundoff error $\hat{\epsilon}(i)$ is uniformly distributed in $[-2^{-(\hat{B}+1)}, 2^{-(\hat{B}+1)}]$, $\mathrm{E}\left[\hat{\epsilon}^2(i)\right]$ is equal to $\frac{2^{-2\hat{\mathcal{B}}}}{12}$. The variance of $\hat{e}_l$ is formulated as

$$\begin{aligned} \sigma_{\hat{e}_l}^2 = &\ \mathrm{E}\left[\left(\textstyle\sum_{i=0}^{I_l}\hat{\epsilon}(i) + \sum_{i=0}^{I_l-1}\hat{\varepsilon}_l(i)k_l(i)\right)^2\right] \\ = &\ \textstyle\sum_{i=0}^{I_l}\mathrm{E}\left[\hat{\epsilon}^2(i)\right] + \sum_{i=0}^{I_l-1}\mathrm{E}\left[\hat{\varepsilon}^2(i)k_l^2(i)\right] \\ = &\quad \frac{2^{-2\hat{\mathcal{B}}}}{12}(I_l+1) + \sigma_{\hat{\varepsilon}_l}^2 \parallel \vec{k}_l \parallel^2. \end{aligned} \tag{B.3}$$

The output NSR of the lth layer is defined as

$$\frac{\sigma_{\hat{e}_l}^2}{\sigma_{y_l}^2}. \tag{B.4}$$

Substituting (B.2) and (B.3) into (B.4), we obtain

$$\frac{\sigma_{\hat{e}_l}^2}{\sigma_{y_l}^2} = \frac{2^{-2\hat{\mathcal{B}}}(I_l+1)}{12\sigma_{y_l}^2} + \frac{\sigma_{\hat{\varepsilon}_l}^2}{\sigma_{x_l}^2}. \tag{B.5}$$

It should be noted the first term in (B.5) is the NSR generated by the local noises. In addition, with the conclusion of lemma 1, we have that the second term in (B.5) is always less than the output NSR of the $(l-1)$th layer, i.e.,

$$\frac{\sigma_{\hat{\varepsilon}_l}^2}{\sigma_{x_l}^2} \leq \frac{\sigma_{\hat{e}_{l-1}}^2}{\sigma_{y_{l-1}}^2}$$

In consequence, the upper bound of lth layer's NSR is derived as

$$\frac{\sigma_{\hat{e}_l}^2}{\sigma_{y_l}^2} \leq \widehat{NSR}_l + \frac{\sigma_{\hat{e}_{l-1}}^2}{\sigma_{y_{l-1}}^2}, \tag{B.6}$$

in which, the variable $NSR_l = \frac{2^{-2\hat{\mathcal{B}}}(I_l+1)}{12\sigma_{y_l}^2}$ is the local NSR. Using (B.6) and noticing $\frac{\sigma_{\hat{e}_0}^2}{\sigma_{y_0}^2} = NSR_0$, the NSR upper bound of the lth layer has the simple expression as

$$\begin{aligned} \frac{\sigma_{\hat{e}_l}^2}{\sigma_{y_l}^2} &\leq \sum_{i=0}^{l}\frac{2^{-2\hat{\mathcal{B}}}(I_i+1)}{12\sigma_{y_i}^2} \\ &= \sum_{i=0}^{l}\widehat{NSR}_i \end{aligned}$$

Appendix C
Rounding Error Analysis of Floating-Point CNN

From the data flow depicted in Fig.6, we have

$$\tilde{y}_l = \sum_{i=0}^{I_l-1}\left\{\Gamma(i)k_l(i)\left[x(i)+\tilde{\varepsilon}(i)\right]\right\} + \Gamma(I_l)b,$$

where, $\Gamma(\cdot)$ is defined as

$$\begin{cases} \Gamma(0) = \left[1+\tilde{\eta}(0)\right]\Pi_{j=1}^{I_l}\left[1+\tilde{\zeta}(j)\right] & \text{For } i=0 \\ \Gamma(i) = \left[1+\tilde{\eta}(i)\right]\Pi_{j=i}^{I_l}\left[1+\tilde{\zeta}(j)\right] & \text{Otherwise.} \end{cases}$$

Then, it is straight forward to obtain

$$\begin{cases} \tilde{e}_l = \sum\limits_{i=0}^{I_l-1}\left(\Gamma(i)-1\right)k_l(i)x_l(i) \\ \quad + \sum\limits_{i=0}^{I_l-1}k_l(i)\tilde{\varepsilon}_l(i) \\ \quad + \sum\limits_{i=0}^{I_l-1}\left(\Gamma(i)-1\right)k_l(i)\tilde{\varepsilon}_l(i) \\ \quad + \left(\Gamma(I_l)-1\right)b. \end{cases} \tag{C.1}$$

From the zero-mean and i.i.d. properties of $\tilde{\eta}(i)$ and $\tilde{\zeta}(i)$, we have

$$\begin{cases} \mathrm{E}\left[\Gamma(i)\right] = 1 \\ \mathrm{E}\left[(\Gamma(i)-1)^2\right] = \mathrm{E}\left[\Gamma^2(i)\right] - 1 \end{cases} \tag{C.2}$$

As mentioned in Section IV-A, because the mantissa and the associated roundoff error are assumed to possess the uniform distributions in $[1,2]$ and $[-2^{-(\tilde{\mathcal{B}}+1)}, 2^{-(\tilde{\mathcal{B}}+1)}]$ respectively, the variances of $\tilde{\eta}(i)$ and $\tilde{\zeta}(i)$ are same as

$$\begin{aligned} \mathrm{E}\left[\tilde{\eta}^2(i)\right] = &\quad \mathrm{E}\left[\tilde{\zeta}^2(i)\right] \\ = &\ \frac{\frac{1}{2^{-\tilde{B}}}\int_{-2^{-(\tilde{B}+1)}}^{2^{-(\tilde{B}+1)}}x^2dx}{\int_1^2 x^2dx} \\ = &\quad \frac{2^{-2\tilde{B}}}{28} \end{aligned}$$

In consequence, we obtain

$$\begin{cases} \mathrm{E}\left[\Gamma^2(0)\right] = \left(1+\dfrac{2^{-2\tilde{\mathcal{B}}}}{28}\right)^{I_l+1} & \text{For } i=0 \\ \mathrm{E}\left[\Gamma^2(i)\right] = \left(1+\dfrac{2^{-2\tilde{\mathcal{B}}}}{28}\right)^{I_l+2-i} & \text{Otherwise.} \end{cases} \tag{C.3}$$

When $I_l \gg 1$, we can approximate $\mathrm{E}\left[\Gamma^2(0)\right]$ with the general expression $\mathrm{E}\left[\Gamma^2(i)\right]$.

From (C.1) and (C.2), we can see that $\tilde{e}_l$ is zero-mean and its variance is expressed as

$$\begin{aligned}\sigma_{\tilde{e}_l}^2 = &\sum_{i=0}^{I_l-1}\left\{\mathrm{E}\left[\Gamma^2(i)\right]-1\right\}k_l^2(i)\sigma_{x_l}^2(i)+\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{\tilde{\varepsilon}_l}^2(i)\\&+\sum_{i=0}^{I_l-1}\left\{\mathrm{E}\left[\Gamma^2(i)\right]-1\right\}k_l^2(i)\sigma_{\tilde{\varepsilon}_l}^2(i)\\&+\left\{\mathrm{E}\left[\Gamma^2(I_l)\right]-1\right\}b^2 \qquad \text{(C.4)}\end{aligned}$$

In addition, when $\frac{2^{-2\tilde{B}}}{28} \ll 1$, which is always hold with $\tilde{B} > 1$, (C.3) can be approximated by discarding the high order term of $\frac{2^{-2\tilde{B}}}{28}$. That is

$$\mathrm{E}\left[\Gamma^2(i)\right] \approx 1 + (I_l + 2 - i)\,\frac{2^{-2\tilde{B}}}{28}. \qquad \text{(C.5)}$$

Substituting (C.5) in (C.4) yields

$$\begin{aligned}\sigma_{\tilde{e}_l}^2 = &\sum_{i=0}^{I_l-1}\frac{2^{-2\tilde{B}}}{28}(I_l+2-i)\,k_l^2(i)\sigma_{x_l}^2(i)\\&+\frac{2^{-2\tilde{B}}}{28}(I_l+2-I_l)\,b^2+\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{\tilde{\varepsilon}_l}^2(i)\\&+\sum_{i=0}^{I_l-1}\frac{2^{-2\tilde{B}}}{28}(I_l+2-i)\,k_l^2(i)\sigma_{\tilde{\varepsilon}_l}^2(i). \qquad \text{(C.6)}\end{aligned}$$

Because $\frac{2^{-2\tilde{B}}}{28} \ll 1$ and $\sigma_{\tilde{\varepsilon}_l}^2(i) \ll \sigma_{x_l}^2(i)$, we discard the last term in (C.6) to get the expression of the lth layer's NSR as

$$\begin{aligned}\frac{\sigma_{\tilde{e}_l}^2}{\sigma_{y_l}^2} = &\frac{2^{-2\tilde{B}}(I_l+2)}{28}\frac{\sum_{i=0}^{I_l-1}\left(1-\frac{i}{I_l+2}\right)k_l^2(i)\sigma_{x_l}^2(i)+\frac{2b^2}{I_l+2}}{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{x_l}^2(i)}\\&+\frac{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{\tilde{\varepsilon}_l}^2(i)}{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{x_l}^2(i)} \qquad \text{(C.7)}\end{aligned}$$

The first term in (C.7), which is labelled as $\widetilde{NSR}_l$, represents the effect of local arithmetic operations. When $I_l \gg 1$, the effect of b^2 can be neglected. Therefore, (C.7) is recast to

$$\begin{aligned}\frac{\sigma_{\tilde{e}_l}^2}{\sigma_{y_l}^2} &= \widetilde{NSR}_l+\frac{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{\varepsilon_l}^2(i)}{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{x_l}^2(i)}\\&=\frac{2^{-2\tilde{B}}(I_l+2)}{28}\frac{\sum_{i=0}^{I_l-1}\left(1-\frac{i}{I_l+2}\right)k_l^2(i)\sigma_{x_l}^2(i)}{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{x_l}^2(i)}\\&\quad+\frac{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{\tilde{\varepsilon}_l}^2(i)}{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{x_l}^2(i)} \qquad \text{(C.8)}\end{aligned}$$

It should be emphasized that the second fraction in $\widetilde{NSR}_l$ must be less than 1. Therefore, the upper bound of $\frac{\sigma_{\tilde{e}_l}^2}{\sigma_{y_l}^2}$ is

$$\frac{\sigma_{\tilde{e}_l}^2}{\sigma_{y_l}^2} \le \frac{2^{-2\tilde{B}}(I_l+2)}{28}+\frac{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{\tilde{\varepsilon}_l}^2(i)}{\sum_{i=0}^{I_l-1}k_l^2(i)\sigma_{x_l}^2(i)}. \qquad \text{(C.9)}$$

By mathematical induction, we can derive the concise expression of the output NSR upper bound as

$$\frac{\sigma_{\tilde{e}_l}^2}{\sigma_{y_l}^2} \le \frac{2^{-2\tilde{B}}}{28}\left(\sum_{i=0}^{l}I_i+2l\right). \qquad \text{(C.10)}$$

Proof: When $l = 0$, because $\sigma_{\tilde{\varepsilon}_l}^2(i) = 0$, the theorem (C.10) is true. If it is assumed that, (C.10) holds when $l = t-1$, we just need to demonstrate the truth of (C.10) when $l = t$.

From lemma 1 and the assumption for $l = t-1$, we have

$$\begin{aligned}\sigma_{\tilde{\varepsilon}_t}^2(i) &\le \sigma_{x_t}^2(i)\frac{\sigma_{\tilde{e}_{t-1}}^2}{\sigma_{y_{t-1}}^2}\\&\le \sigma_{x_t}^2(i)\frac{2^{-2\tilde{B}}}{28}\left(\sum_{i=0}^{t-1}I_i+2(t-1)\right) \qquad \text{(C.11)}\end{aligned}$$

With (C.11) and (C.9), when $l = t$, we have

$$\begin{aligned}\frac{\sigma_{\tilde{e}_t}^2}{\sigma_{y_t}^2} &\le \frac{2^{-2\tilde{B}}(I_t+2)}{28}+\frac{2^{-2\tilde{B}}}{28}\left(\sum_{i=0}^{t-1}I_i+2(t-1)\right)\\&=\frac{2^{-2\tilde{B}}}{28}\left(\sum_{i=0}^{t}I_i+2t\right)\end{aligned}$$

Therefore, the theorem is proofed. ■

REFERENCES

[1] B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, Y.-K. Wang, and T. Wiegand, *High Efficiency Video Coding (HEVC) Text Specification Draft 10*, document JCTVC-L1003, Geneva, CH, 2013.

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[3] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards—Including high efficiency video coding (HEVC)," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1669–1684, Dec. 2012.

[4] Y. Piao, J. Min, and J. Chen, *Encoder Improvement of Unified Intra Prediction*, document JCTVC-C207, Guangzhou, CN, 2010.

[5] L. Zhao, L. Zhang, X. Zhao, S. Ma, D. Zhao, and W. Gao, *Further Encoder Improvement of Intra Mode Decision*, document JCTVC-D283, Daegu, South Korea, 2011.

[6] S. Ma, S. Wang, S. Wang, L. Zhao, Q. Yu, and W. Gao, "Low complexity rate distortion optimization for HEVC," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2013, pp. 73–82.

[7] Q. Chen and Y. He, "A fast bits estimation method for rate-distortion optimization in H.264/AVC," in *Proc. Picture Coding Symp. (PCS)*, Dec. 2004, pp. 133–134.

[8] Y.-K. Tu, J.-F. Yang, and M.-T. Sun, "Efficient rate-distortion estimation for H.264/AVC coders," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 5, pp. 600–611, May 2006.

[9] M. G. Sarwer and L.-M. Po, "Fast bit rate estimation for mode decision of H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 10, pp. 1402–1407, Oct. 2007.

[10] X. Zhao, J. Sun, S. Ma, and W. Gao, "Novel statistical modeling, analysis and implementation of rate-distortion estimation for H.264/AVC coders," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 5, pp. 647–660, May 2010.

[11] J. Zhu, Z. Liu, D. Wang, Q. Han, and Y. Song, "Fast prediction mode decision with Hadamard transform based rate-distortion cost estimation for HEVC intra coding," in *Proc. 20th IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2013, pp. 1977–1981.

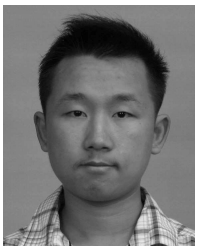
[12] Z. Liu, S. Guo, and D. Wang, "Binary classification based linear rate estimation model for HEVC RDO," in *Proc. 21st IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2014, pp. 3676–3680.

[13] X. Li, J. An, X. Guo, and S. Lei, *Adaptive CU Depth Range*, document JCTVC-E090, Geneva, CH, 2011.

[14] N. Hu and E.-H. Yang, "Fast mode selection for HEVC intra-frame coding with entropy coding refinement based on a transparent composite model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 9, pp. 1521–1532, Sep. 2015.
[15] L. Shen, Z. Zhang, and P. An, "Fast CU size decision and mode decision algorithm for HEVC intra coding," *IEEE Trans. Consum. Electron.*, vol. 59, no. 1, pp. 207–213, Feb. 2013.
[16] L. Shen, Z. Zhang, and Z. Liu, "Effective CU size decision for HEVC intracoding," *IEEE Trans. Image Process.*, vol. 23, no. 10, pp. 4232–4241, Oct. 2014.
[17] K. Choi, S.-H. Park, and E. S. Jang, *Coding Tree Pruning Based CU Early Termination*, document JCTVC-F092, Torino, IT, 2011.
[18] L. Shen, Z. Liu, X. Zhang, W. Zhao, and Z. Zhang, "An effective CU size decision method for HEVC encoders," *IEEE Trans. Multimedia*, vol. 15, no. 2, pp. 465–470, Feb. 2013.
[19] H. Zhang and Z. Ma, "Fast intra mode decision for high efficiency video coding (HEVC)," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 4, pp. 660–668, Apr. 2014.
[20] Y. Zhang, Z. Li, and B. Li, "Gradient-based fast decision for intra prediction in HEVC," in *Proc. Vis. Commun. Image Process.*, Nov. 2012, pp. 1–6.
[21] B. Min and R. C. C. Cheung, "A fast CU size decision algorithm for the HEVC intra encoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 5, pp. 892–896, May 2015.
[22] S. Cho and M. Kim, "Fast CU splitting and pruning for suboptimal CU partitioning in HEVC intra coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 9, pp. 1555–1564, Sep. 2013.
[23] Q. Hu, X. Zhang, Z. Shi, and Z. Gao, "Neyman–Pearson-based early mode decision for HEVC encoding," *IEEE Trans. Multimedia*, vol. 18, no. 3, pp. 379–391, Mar. 2016.
[24] V. Sze, M. Budagavi, and G. J. Sullivan, Eds., *High Efficiency Video Coding (HEVC): Algorithms and Architectures*. New York, NY, USA: Springer-Verlag, Jul. 2014, pp. 343–375.
[25] G. Pastuszak and A. Abramowski, "Algorithm and architecture design of the H.265/HEVC intra encoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 210–222, Jan. 2016.
[26] J. Zhu, Z. Liu, D. Wang, Q. Han, and Y. Song, "HDTV1080p HEVC intra encoder with source texture based CU/PU mode pre-decision," in *Proc. 19th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2014, pp. 367–372.
[27] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Networks*, M. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1995, pp. 255–258.
[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[29] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Convolutional neural network committees for handwritten character classification," in *Proc. 11th IEEE Int. Conf. Document Anal. Recognit. (ICDAR)*, Sep. 2011, pp. 1135–1139.
[30] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. 25th IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2012, pp. 3642–3649.
[31] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3431–3440.
[32] X. Yu, Z. Liu, J. Liu, Y. Gao, and D. Wang, "VLSI friendly fast CU/PU mode decision for HEVC intra encoding: Leveraging convolution neural network," in *Proc. 22nd IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2015, pp. 1285–1289.
[33] Z. Liu, D. Wang, J. Zhou, and T. Ikenaga, "Lagrangian multiplier optimization using correlations in residues," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Mar. 2012, pp. 1185–1188.
[34] T. Berger, *Rate-Distortion Theory*, T. Kailath, Ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 1971.
[35] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Process. Mag.*, vol. 15, no. 6, pp. 74–90, Nov. 1998.
[36] X. Li, N. Oertel, A. Hutter, and A. Kaup, "Laplace distribution based Lagrangian rate distortion optimization for hybrid video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 2, pp. 193–205, Feb. 2009.
[37] A. V. Oppenheim and C. J. Weinstein, "Effects of finite register length in digital filtering and the fast Fourier transform," *Proc. IEEE*, vol. 60, no. 8, pp. 957–976, Aug. 1972.
[38] A. V. Oppenheim, R. W. Schafer, M. T. Yoder, and W. T. Padgett, *Discrete-Time Signal Processing*, vol. 2. Englewood Cliffs, NJ, USA: Prentice-Hall, 1989.
[39] C.-Y. Chen, S.-Y. Chien, Y.-W. Huang, T.-C. Chen, T.-C. Wang, and L.-G. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 3, pp. 578–593, Mar. 2006.
[40] P. M. Farmwald, "On the design of high performance digital arithmetic units," Ph.D. dissertation, Stanford Univ., Stanford, CA, USA, Aug. 1981.
[41] F. Bossen, *Common HM Test Conditions and Software Reference Configurations*, document JCTVC-I1100, Geneva, CH, 2012.
[42] G. Bjøntegaard, *Calculation of Average PSNR Differences Between RD-Curves*, document VCEG-M33, Austin, TX, USA, Apr. 2001.
[43] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitect. (MICRO)*, Dec. 2014, pp. 609–622.

Zhenyu Liu (M'07) received the B.E., M.E., and Ph.D. degrees from the Beijing Institute of Technology, China, in 1996, 1999, and 2002, respectively, all in electrical engineering. From 2002 to 2004, he held a post-doctoral position with Tsinghua University, China, where he was involved in the embedded processor architecture design. From 2004 to 2009, he was a Visiting Researcher with the Graduate School of IPS, Waseda University, Japan. He joined Tsinghua University, China, in 2009, where he is currently an Associate Professor with RIIT&TNList. His research interests include signal processing, energy-efficient real-time video encoding and application specific processor.

Xianyu Yu was born in 1989. He received the B.E. degree in automation from Anhui University, China, in 2012, and the M.E. degree in integrated circuit engineering from Tsinghua University, China, in 2016. He is currently with Huawei Corporation. His research interests include embedded system and Android framework associated with multimedia processing (audio/video/graphics).



Yuan Gao was born in 1991. He received the B.E. degree in electrical engineering from the Beijing Institute of Technology, China, in 2012. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Tsinghua University. His research interests include neural network algorithm and associated very large scale integration architecture design.

Shaolin Chen received the B.S. degree in automation from Anhui University, Hefei, China, 2005, the M.S. degree in geodynamics from the College of Earth Science, Graduation University of Chinese Academy of Sciences, Beijing, China, in 2008, and the Ph.D. degree in pattern recognition and intelligent system from the Institute of Automation, Chinese Academy of Sciences in 2012. He is currently a Senior Algorithm Engineer with Huawei Technologies Company, Ltd. His research interests include video encoding and image enhancement.

Xiangyang Ji (M'10) received the B.S. degree in materials science and the M.S. degree in computer science from the Harbin Institute of Technology, Harbin, China, in 1999 and 2001, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. He joined Tsinghua University, Beijing, in 2008, where he is currently a Professor with the Department of Automation, School of Information Science and Technology. He has authored over 100 referred conference and journal papers. His current research interests include signal processing, image/video compression and communication, and intelligent imaging.

Dongsheng Wang (M'09) was born in China in 1966. He received the B.E., M.E., and Ph.D. degrees from the Harbin Institute of Technology, China, in 1989, 1992 and 1995, respectively, all in computer science. He is currently a Professor with RIIT & TNList, Tsinghua University. His research areas include many-core computer architecture, real-time application oriented SoCs, disaster recovery, and high availability computing.