Efficient Layout Hotspot Detection via Binarized Residual Neural Network Ensemble

Yiyang Jiang, Fan Yang, *Member, IEEE*, Bei Yu, *Member, IEEE*, Dian Zhou, *Senior Member, IEEE*, and Xuan Zeng, *Senior Member, IEEE*

***Abstract*—Layout hotspot detection is of great importance in the physical verification flow. Deep neural network models have been applied to hotspot detection and achieved great successes. The layouts can be viewed as binary images. The binarized neural network (BNN) can thus be suitable for the hotspot detection problem. In this article, we propose a new deep learning architecture based on BNNs to speed up the neural networks in hotspot detection. A new binarized residual neural network is carefully designed for hotspot detection. Experimental results on ICCAD 2012 and 2019 benchmarks show that our architecture outperforms previous hotspot detectors in detecting accuracy and has an 8× speedup over the best deep learning-based solution. Since the BNN-based model is quite computationally efficient, a good tradeoff can be achieved between the efficiency and performance of the hotspot detector by applying ensemble learning approaches. Experimental results show that the ensemble models achieve better hotspot detection performance than the original with acceptable speed loss.**

***Index Terms*—Binarized neural network (BNN), deep neural network, hotspot detection.**

I. Introduction

THE LITHOGRAPHIC printability is one of the most critical issues in nanoscale integrated circuits. Although various resolution enhancement techniques have been proposed to improve the printability in the past years, there still exist sensitive layout patterns which would lead to manufacture defects. These lithographic hotspots should be detected and fixed at early design stages.

Two classes of hotspot detection approaches have been proposed recently: 1) pattern matching-based approach and 2) machine-learning-based approach. The pattern matching-based methods characterize the hotspots as explicit patterns and identify the hotspots by matching these patterns. In [1] and [2], the hotspots are encoded by strings and modified transitive closure graphs. In [3], a graph is used to represent the layout. The hotspots are encoded as the critical "faces" of the graph. In [4], density-based layout encoding and principal components analysis (PCA) are integrated to detect the hotspot. In [5], a tangent space-based distance metric is proposed to classify the hotspot patterns. Generally, pattern matching-based approaches are relatively fast, but impossible to detect the unseen patterns.

To address this problem, machine-learning-based approaches have been proposed recently. In the machine-learning-based approaches, implicit models are built by learning from the existing training data. It is possible to detect the unseen hotspots through the generalization capacities of the machine learning models. However, the false alarm issues should be carefully treated in the machine learning approaches [6], [7]. In [8]–[10], the neural network and support vector machine (SVM) are proposed for hotspot detection. In [11], Adaboost and decision tree are adopted for fast hotspot detection. In [12], multikernel SVM and critical feature extraction are adopted for hotspot detection. In [13], an unsupervised SVM model and histogram-based layout representation are applied to predict hotspots. In [14], optimized concentric circle sampling (CCS) feature [15] and online learning scheme are proposed for hotspot detection. In [16], an algorithm for pattern matching which dissects patterns into rectangles based on polygon edges is proposed. In [17], a methodology for machine-learning-based hotspot detection that uses lithography information to build SVM during its learning process is proposed.

Deep neural networks have demonstrated great successes in the image classification, object detection tasks in the community of computer vision. Some breakthroughs are achieved in hotspot detection problem as well [18]–[22]. Yang *et al.* [18] proposed a deep learning model that takes the original layout image as input and contains more than 20 layers to detect the hotspots. Yang *et al.* [19] proposed a deep neural network that replaces all pooling layers with stride convolution layers. In [22], a convolutional neural network (CNN) architecture is proposed for the hotspot detection. It can achieve a nice balance between the accuracy and the suppression of false alarms. The features of the hotspots are represented as the truncated coefficients of the discrete cosine transforms of the patterns. And floating-point arithmetic is employed in the CNN architecture. However, the discrete cosine transforms would

Manuscript received September 30, 2019; revised May 1, 2020; accepted July 16, 2020. Date of publication August 11, 2020; date of current version June 18, 2021. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFA0709602; in part by the National Natural Science Foundation of China under Grant 61822402, Grant 61774045, Grant 61929102, and Grant 62011530132; and in part by the Research Grants Council of Hong Kong SAR under Project CUHK24209017. The preliminary version has been presented at the ACM/IEEE Design Automation Conference (DAC) in 2019. This article was recommended by Associate Editor S. Held. *(Corresponding authors: Fan Yang; Xuan Zeng.)*

Yiyang Jiang, Fan Yang, and Xuan Zeng are with the State Key Laboratory of ASIC & System, Microelectronics Department, Fudan University, Shanghai 200433, China (e-mail: yangfan@fudan.edu.cn; xzeng@fudan.edu.cn).

Bei Yu is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

Dian Zhou is with the Department of Electrical Engineering, University of Texas at Dallas, Richardson, TX 75080 USA.

Digital Object Identifier 10.1109/TCAD.2020.3015918

miss the spatial information of the patterns and the floating-point arithmetic-based neural network would be computation intensive.

The layouts can be viewed as binary images. The binarized neural network (BNN) might thus be suitable for constructing an efficient hotspot detector. In our preliminary work [23], we propose a new deep learning architecture based on BNNs to speed up the neural networks in hotspot detection. The downsampled images of the patterns are taken as the inputs directly, and the spatial information of the patterns can be fully exploited in our approach. A new binarized residual neural network is carefully designed for hotspot detection. Compared with the floating-point arithmetic-based neural network, the BNNs are computationally efficient. Experimental results on ICCAD 2012 Contest benchmarks show that our architecture outperforms all previous hotspot detectors in detecting accuracy and has an 8× speedup over the best deep learning-based solution. To further improve the performance of the well-performed BNN-based architecture, we adopt the ensemble learning approaches which combine the BNN-based model and its two shallower virant models. The experimental results show that the ensemble model achieves better hotspot detection performance compared with the original BNN-base model with acceptable efficiency loss.

The remainder of this article is organized as follows. In Section II, the background of hotspot detection is presented. In Section III, we propose the BNN-based hotspot detection method. In Section IV, we show the details of the ensemble learning models. In Section V, experimental results are shown to demonstrate the efficiency of the proposed method. In Section VI, we conclude this article.

II. Background

In this section, we will present the problem formulation of layout hotspot detection and then review the background of BNNs and ensemble learning algorithms.

A. Problem Formulation

The lithographic process in chip manufacturing may involve various variations, which can cause potential open or short circuit failures and result in performance degradation and yield reduction. Layout patterns that are sensitive to process variations are defined as *hotspots*.

In the hotspot detection process, the most important issue is to correctly detect as many hotspots as possible. Identifying an instance as a hotspot which is nonhotspot should also be avoided. The following metrics are used to evaluate the performance of the hotspot detector.

Table I shows the confusion matrix of the hotspot detection problem, where *TN* denotes the true negative, *FN* denotes the false negative, *FP* denotes the false positive, and *TP* denotes the true positive. We have the following definitions.

Definition 1 (Accuracy): The ratio of correctly predicted hotspots among the set of actual hotspots [24]

$$\text{Accuracy} = \frac{\#\ \text{TP}}{\#\ \text{TP} + \#\ \text{FN}}. \tag{1}$$

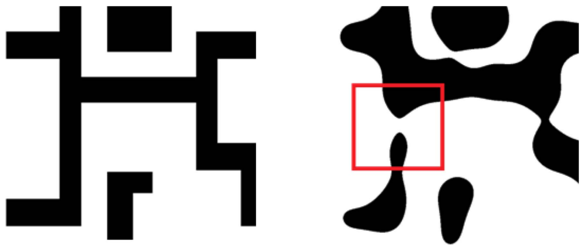

Fig. 1. Hotspot in the layout.

TABLE I
Confusion Matrix of Hotspot Detection Problems

Prediction	Real	
	Non-hotspot	Hotspot
Non-Hotspot	$\# TN$	$\# FN$
Hotspot	$\# FP$	$\# TP$

Definition 2 (False Alarm): The number of incorrectly predicted nonhotspots [24]

$$\text{False Alarm} = \#\ \text{FP}. \tag{2}$$

Definition 3 [Overall Detection and Simulation Time (ODST)]: The sum of the lithography simulation time for layout patterns detected as hotspots (including real hotspots and false alarms) and the learning model evaluation time [14]

$$\begin{aligned}\text{ODST} = {} & (\#\ \text{FP} + \#\ \text{TP})t_{ls} \\ & + (\#\ \text{TN} + \#\ \text{FN} + \#\ \text{FP} + \#\ \text{TP})t_{ev}\end{aligned} \tag{3}$$

where t_{ls} is the lithography simulation time per instance and t_{ev} is the model evaluation time per instance.

Observe that through utilizing multicore parallelism, the simulation time for each layout core is around 10 s, thus in ODST calculation, we set t_{ls} to 10 s.

With the above definitions, the hotspot detection problem is formulated as follows.

Problem 1 (Hotspot Detection): Given a dataset that contains hotspot and nonhotspot instances, train a classifier that can maximize the *accuracy* and minimize the *false alarm*.

B. Binarized Neural Networks

In recent years, deep convolution neural networks [25] have led to a series of breakthroughs in various aspects of computer vision, including image classification, object detection, and semantic segmentation. Recently, they are also adopted in hotspot detection problems [22]. However deep neural networks often suffer from overparametrization and enormous redundancy in their models which can result in enormous computational and storage consumption [26]. Parameter quantizing is usually applied to alleviate this problem because high precision filters such as 32-b floating-point weights are not necessary for deep neural networks. Thus, the weights can be quantized to a low bit with acceptable accuracy loss. It is demonstrated in [27] that a sparse neural network with

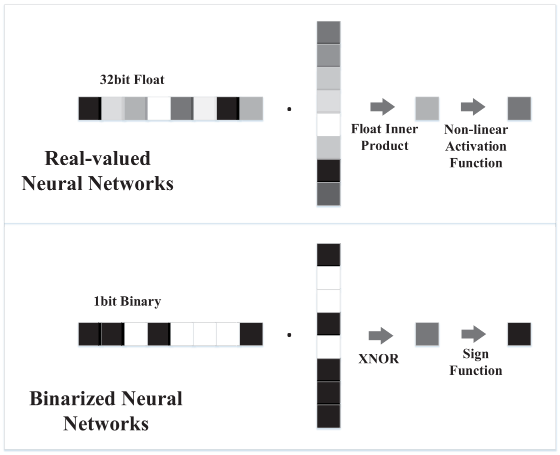
32bit Float
Float Inner Product
Non-linear Activation Function
Real-valued Neural Networks
1bit Binary
XNOR
Sign Function
Binarized Neural Networks

Fig. 2. Difference between real-valued neural networks and BNNs.

+1/0/−1 weights can be trained in polynomial time. 32-b floating-point activations are quantized to 8-b fixed-point integers in [28]. Neural networks with 3-b activations and ternary weights are proposed in [29].

BNN is a special type of parameter quantizing method because the weights are extremely quantized to 1 b. Fig. 2 shows the difference between real-valued neural networks and BNNs. Due to the precision loss of parameters, BNNs were believed to face serious performance degradation [30]. However, expectation backpropagation (EPB) is proposed in [31] to train a high-performance BNN. In BinaryNet [32], [33], real-valued weights are used for binarization and they are updated ignoring the binarization in the backpropagation process. A BNN is obtained by retraining a trained neural network with binary weights and binary inputs in [34]. Rastegari *et al.* [35] adopted a new way of binarizing parameters and activations and achieved huge advance in large datasets such as ImageNet Large Scale Visual Recognition Competition [36] (ILSVRC) 2012.

BNNs are inherently suitable for hardware implementation because binarization replaces floating-point operations with binary operations which can be very efficiently operated in logic circuits, such as FPGA and ASIC. It also reduces the storage and memory bandwidth requirements which is suitable for low-power embedded implementation. An FPGA-based BNN accelerator synthesized from C++ to Verilog is implemented in [37]. An architecture based on the two-stage arithmetic unit (TSAU) is proposed in [38] to implement the low-bit CNN on FPGA. A BNN accelerator on the Xeon+FPGA platform is implemented in [39].

C. Ensemble Learning

Ensemble learning is a type of method that combines multiple base learners to make a decision as shown in Fig. 3. The main premise of ensemble learning is that the errors of an individual model might be compensated by other models so that it may achieve a better performance than the single models. Gomes *et al.* [40] and Kulkarni and Sinha [41] listed some comprehensive surveys on ensemble learning.

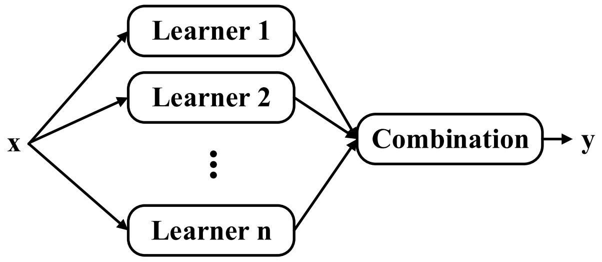

Fig. 3. Common ensemble architecture.

There are two main types of ensemble methods. One is sequential ensemble methods that are able to convert weak learners to strong learners. The base learners are generated sequentially, with AdaBoost [42] as a representative. The other is parallel ensemble methods which exploit the independence between the base learners so as to reduce the error by combining independent base learners. The base learners are generated in parallel, with Bagging [43] and Random Forest [44] as representatives.

III. Proposed BNN-Based Hotspot Detector

Different from common RGB images and grayscale images, layout patterns are inherently binarized. Thus, the BNN might be suitable for classifying the hotspots and nonhotspots. A BNN-based architecture is carefully designed to detect layout hotspot efficiently considering the binarization property of the layout patterns. We will present the details of the proposed BNN-based hotspot detector in this section.

A. Convolutional Neural Networks and Gradient Descent

CNNs are widely adopted in deep learning models. A CNN architecture usually consists of several convolution layers and fully connected (FC) layers. An L-layer CNN architecture can be defined as $<\mathcal{W}, \mathcal{T}, \otimes, f>$. $\mathcal{W}$ is the set of weights of the network. $\mathcal{W}_{l,k}$ is the kth convolution filter of the lth layer. $\mathcal{W}_{l,k} \in \mathbb{R}^{c_{\text{in}} \times w_k \times h_k}$, where $(c_{\text{in}}, w_k, h_k)$ denote the number of input channels, the width, and height of the convolution kernel, respectively. $\mathcal{T}$ is the set of input tensors of the network, where $\mathcal{T}_l$ is the input tensor of the lth layer and the output tensor of the $(l-1)$th layer as well. $\mathcal{T}_l \in \mathbb{R}^{c_{\text{in}} \times w_{\text{in}} \times h_{\text{in}}}$, where $(c_{\text{in}}, w_{\text{in}}, h_{\text{in}})$ represents the channel, width, and height of the input tensor. $\otimes$ represents the convolution computation which is defined in

$$\mathcal{T}_l \otimes \mathcal{W}_{l,k}\big[j,k\big] = \sum_{c=1}^{c_k} \sum_{w=1}^{w_k} \sum_{h=1}^{h_k} \mathcal{W}_{l,k}[c,w,h]\mathcal{T}_l\big[c, j-w, k-h\big]. \tag{4}$$

f represents the activation function so that $\mathcal{T}_{l+1,k} = f(\mathcal{T}_l \otimes \mathcal{W}_{l,k})$. Following the tradition, we use the rectified linear unit (ReLU) activation function which is defined in (5) instead of the old fashioned sigmoid activation function defined in (6)

$$\text{ReLU}(x) = \begin{cases} x & x > 0 \\ 0 & \text{else} \end{cases} \tag{5}$$

$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}. \tag{6}$$

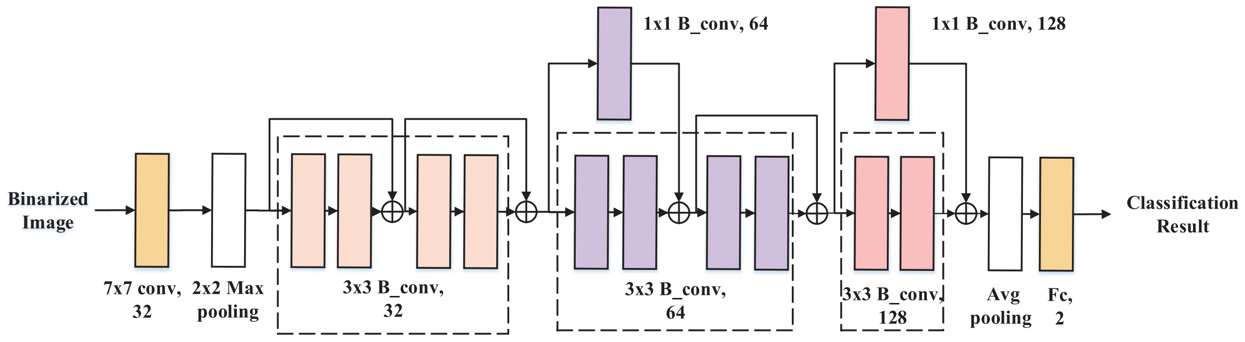

Fig. 4. Redesigned binarized network based on residual network.

The 2×2 max pooling performs 2×2 downsampling that outputs the max value of the local 2×2 feature. It reveals the overfitting problem by providing an abstracted form of the representation and reduces the computational cost by reducing the number of parameters. Global average pooling (GAP) is widely used to reveal the overfitting by reducing the total number of parameters in the model in the last few years. Compared with max pooling, the GAP performs a more extreme type of dimensionality reduction. For a tensor with the shape of $c \times w \times h$, GAP reduces the tensor to have dimensions $c \times 1 \times 1$ in size where each $w \times h$ feature map is averaged to a single number.

Gradient descent is an optimization algorithm used for finding the weights or coefficients of machine learning algorithms, including artificial neural networks, logistic regression, etc. The model makes predictions on training data and the error on the predictions is used to update the weight so as to reduce the error. All parameters are updated according to their gradients. Backpropagation [45] is widely applied to calculate gradients when training neural networks.

Gradient descent can vary in terms of the number of instances used to update the parameters. The three main kinds of gradient descent are batch, stochastic, and mini-batch.

In the batch gradient descent (BGD), the error of each example in the training set is calculated and the model gets updated after all training instances are evaluated. The BGD updates the model at the end of each training epoch. The BGD is computationally efficient and generates a more stable error gradient which can result in a more stable convergence. However, the stable gradient might lead to premature convergence of the model to a less minima. The BGD usually requires the entire training dataset in the memory which is not suitable for large datasets.

In the stochastic gradient descent (SGD), the error is calculated and the model is updated for each training instance in the training set. The high model update frequency can result in faster training speed on some problems and avoid local minima. However, the noisy gradient can make the model hard to converge. Also updating the model too frequently is more computationally expensive than the BGD.

The mini-BGD (MGD) is a compromise approach. The training set is split into small batches to calculate error and update the model parameters. MGD tries to find a balance between the efficiency of BGD and the robustness of SGD and is suitable for large datasets. MGD is widely used in the field of deep learning. The MGD requires an additional hyperparameter "batch size" which should be carefully tuned.

B. Network Architecture

With networks going deeper, some problems begin to influence the convergence of the network like gradient vanishing/exploding [46] and accuracy saturation. ResNets [47] bypass information between layers via identity connections called "shortcut connections" to relieve the effect of the accuracy saturation problem.

Based on the philosophy of ResNet, we design our binarized architecture to fit the hotspot detection problem. Considering the size of the training set and the computational complexity, a too deep network architecture is not appropriate. The network is preliminarily set to be with fewer than 20 layers.

Our baseline network architecture is the ResNet-18 model. The convolution layers of the original model are replaced by binary convolution layers whose input/output tensors and weights are binarized. We use two 3×3 binary convolution layers as the basic building block. To further reduce the time complexity and address the overfitting problem, the number of layers is reduced and the number of filters of each layer is readjusted. We generally follow the rule that the deeper a layer is, the more filters it contains and keep as few filters as possible for all layers. Finally, we derive a 12-layer network which achieves high speed and satisfies the accuracy requirement at the same time.

Our network architecture is shown in Fig. 4. The 1×1 convolution blocks in the shortcut connections appear where the input tensor and output tensor of a residual block do not have the same shapes. The input tensor is convolved with 1×1 kernel to acquire the same tensor shape as the output tensor so that they can be summed at the end of a residual block.

In the network architecture, each convolution block consists of three cascaded layers: 1) batch normalization [48]; 2) binarizing; and 3) binary convolution. Fig. 5 shows the structure of a convolution block. The batch normalization layer normalizes the input tensor by its mean and variance. The binarizing layer binarizes the input tensors as the input of the next binary convolution layer. Following the practice in [35], the batch normalization layer is placed before the binarizing layer to further reduce the information loss due to binarization.

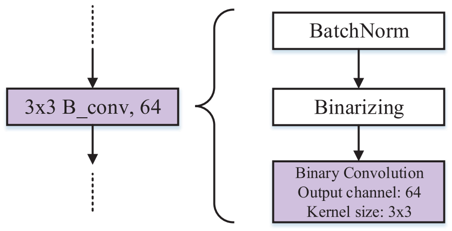

Fig. 5. Typical BNN block structure.

C. Binarization Approach

After binarization, the parameters and input tensors of the layers become binary. The corresponding L-layer BNN can be defined as $<\mathcal{B}, \mathcal{A}_B, \mathcal{I}, \mathcal{A}_T, \circledast, f>$. The binary filter $W_B \in \mathcal{B}$ and the scaling factor $\alpha_B \in \mathcal{A}_B$ are used to estimate the original full-precision filter $W \in \mathcal{W}$. The binary input tensor $T_B \in \mathcal{I}$ and the scaling factor $\alpha_T \in \mathcal{A}_T$ are used to estimate the original input tensor $T_{\text{in}} \in \mathcal{T}$. Here, $W_B \in \{+1, -1\}^{c_{\text{in}} \times w_k \times h_k}$ and $T_B \in \{+1, -1\}^{c_{\text{in}} \times w_{\text{in}} \times h_{\text{in}}}$. $\alpha_T \in \mathbb{R}^{c_{\text{in}} \times w_{\text{in}} \times h_{\text{in}}}$, $\alpha_B \in \mathbb{R}^+$. $\circledast$ represents the binarized convolution operation, which is much faster than the full-precision convolution.

The convolution operation consists of shift kernel operations and dot product operations. The weight filter slides over the input tensor and the output is the inner product of the kernel vector and the vector of the corresponding block in the input tensor. To minimize the binarization loss of the convolution operation, the gap between inner products of full-precision input tensors and weight filters and those of binarized input tensors and weight filters needs to be minimized.

Let $\mathbf{W}$ be the kernel which is an n-element vector and $\mathbf{X}$ be the vector of the corresponding block in the input tensor, $n = w_k \times h_k$. Let $\mathbf{W}_B$ and $\mathbf{X}_B$ be the binarized kernel and input vector and α_W and α_X be the corresponding scaling factors so that $\mathbf{W} \odot \mathbf{X} \approx \alpha_W \mathbf{W}_B \odot \alpha_X \mathbf{X}_B$. Here, $\mathbf{W}, \mathbf{X} \in \mathbb{R}^n$, $\mathbf{W}_B, \mathbf{X}_B \in \{-1, +1\}^n$, and $\alpha_W, \alpha_X \in \mathbb{R}^+$.

The binarizing method we adopt is similar to XNOR-Net's [35] except that different scaling factors for input vector in different input channels are adopted which can estimate the input tensor more accurately.

The binarization loss of inner product operation L_i is defined in

$$L_i(\mathbf{W}_B, \mathbf{X}_B, \alpha_W, \alpha_X) = \|\mathbf{W} \odot \mathbf{X} - \alpha_W \mathbf{W}_B \odot \alpha_X \mathbf{X}_B\|^2. \quad (7)$$

Minimizing binarization loss can be rewritten to the optimization problem in

$$\mathbf{W}_B^*, \mathbf{X}_B^*, \alpha_W^*, \alpha_X^* = \underset{\mathbf{W}_B, \mathbf{X}_B, \alpha_W, \alpha_X}{\arg\min} \; L_i(\mathbf{W}_B, \mathbf{X}_B, \alpha_W, \alpha_X). \quad (8)$$

To simplify the problem, we define $\mathbf{C} = \mathbf{W} \odot \mathbf{X}$, $\mathbf{C}_B = \mathbf{W}_B \odot \mathbf{X}_B$ and $\alpha = \alpha_W \alpha_X$, where $\mathbf{C} \in \mathbb{R}^n$, $\mathbf{C}_B \in \{-1, +1\}^n$ and $\alpha \in \mathbb{R}^+$. Equation (8) can be rewritten as

$$\mathbf{C}_B^*, \alpha^* = \underset{\mathbf{C}_B, \alpha}{\arg\min} \|\mathbf{C} - \alpha \mathbf{C}_B\|^2. \quad (9)$$

Solving the optimization problem, we have

$$\begin{aligned} \mathbf{C}_B^* &= \text{sign}(\mathbf{C}) \\ \alpha^* &= \frac{1}{n}\|\mathbf{C}\|_{l1}. \end{aligned} \quad (10)$$

Since $\mathbf{W}_B^*$ and $\mathbf{X}_B^*$ are independent, we decompose $\mathbf{C}_B^*$ and α^* to get $\mathbf{W}_B^*, \mathbf{X}_B^*, \alpha_W^*, \alpha_X^*$

$$\begin{aligned} \mathbf{W}_B^* &= \text{sign}(\mathbf{W}), \quad \mathbf{X}_B^* = \text{sign}(\mathbf{X}) \\ \alpha_W^* &= \frac{\|\mathbf{W}\|_{l1}}{n}, \quad \alpha_X^* = \frac{\|\mathbf{X}\|_{l1}}{n}. \end{aligned} \quad (11)$$

According to the calculations above, the estimated weight $\widetilde{\mathbf{W}}$ and the estimated corresponding input vector $\widetilde{\mathbf{X}}$ are

$$\begin{aligned} \widetilde{\mathbf{W}} &= \frac{1}{n}\text{sign}(\mathbf{W})\|\mathbf{W}\|_{l1} \\ \widetilde{\mathbf{X}} &= \frac{1}{n}\text{sign}(\mathbf{X})\|\mathbf{X}\|_{l1}. \end{aligned} \quad (12)$$

D. Training Binarized Networks

Like normal CNNs, we train our network with the MGD [49] approach which can utilize computation resources more efficiently than SGD. A group of instances are randomly picked from the training set to perform forward and backward process in each training iteration.

During the training process, the main objective is to update the real-valued kernel $\mathbf{W}$. Backpropagation [45] is widely applied to calculate gradients when training neural networks. Modern deep learning libraries can easily calculate the gradients of the kernels of a normal CNN with backpropagation. The key difference between CNN and BNN architecture is the sign function sign(r). The derivative of the sign function is zero almost everywhere which can interdict the propagation of the gradients. To compute the gradient for sign function, we adopt the straight-through estimator introduced in [33] which considers the saturation effect

$$\frac{\partial \text{sign}(x)}{\partial x} = \mathbf{1}_{\|x\|<1} \quad (13)$$

where $\mathbf{1}_{\|x\|<1}$ is the indicator function which is defined as follows:

$$\mathbf{1}_{\|x\|<1} = \begin{cases} 1, & \|x\| < 1 \\ 0, & \text{else.} \end{cases} \quad (14)$$

Adopting the binarization approach, in the forward process, the estimated convolution kernel $\widetilde{\mathbf{W}}$ is

$$\widetilde{\mathbf{W}} = \alpha_W^* \mathbf{W}_B^* \quad (15)$$

whose gradient can be calculated via standard backward propagation according to (13).

The gradients of the real-valued weights are calculated in (16) via the chain rule

$$\begin{aligned} \frac{\partial l}{\partial \mathbf{W}} &= \frac{\partial l}{\partial \widetilde{\mathbf{W}}} \frac{\partial \widetilde{\mathbf{W}}}{\partial \mathbf{W}} \\ &= \frac{\partial l}{\partial \widetilde{\mathbf{W}}} \frac{\partial \left(\frac{1}{n}\|\mathbf{W}\|_{l1}\text{sign}(\mathbf{W})\right)}{\partial \mathbf{W}} \\ &= \frac{\partial l}{\partial \widetilde{\mathbf{W}}} \left(\frac{1}{n} + \alpha_W^* \mathbf{1}_{\|W\|<1}\right) \end{aligned} \quad (16)$$

where $(\partial l/\partial \mathbf{W})$ and $(\partial l/\partial \widetilde{\mathbf{W}})$ denote the gradients of the loss function l with respect to the full-precision weight $\mathbf{W}$ and estimated weight $\widetilde{\mathbf{W}}$.

Algorithm 1 Training a BNN

```
Input: $(\mathcal{T}_0, Y)$: a minibatch of input tensors and labels;
 1: $l(Y, Y_{\text{out}})$: loss function;
 2: $\mathcal{W}^t$: current real-valued weight;
 3: $L$: number of layers;
 4: $n$: kernel size of layers;
 5: $\eta^t$: current learning rate;
Output: $\mathcal{W}^{t+1}$: updated real-valued weight; $\eta^{t+1}$: updated
    learning rate.
 6: 1. Forward Process:
 7: for $k = 1$ to $L$ do
 8:     $\mathcal{B}_k^t = \textbf{BinarizeWeight}(\mathcal{W}_k^t)$
 9:     $\mathcal{T}_{k+1} = \textbf{BinarizeInput}(\textbf{BatchNorm}(\mathcal{T}_k)) \circledast \mathcal{B}_k^t$
10: end for
11: $Y_{\text{out}} = \mathcal{T}_{L+1}$
12: 2. Backward Process:
13: for $k = L$ to $1$ do
14:     $\frac{\partial l}{\partial \mathcal{T}_k} = \textbf{BinaryBackward}(\frac{\partial l}{\partial \mathcal{T}_{k+1}}, \mathcal{T}_k)$
15:     $\frac{\partial l}{\partial \mathcal{B}_k^t} = \textbf{BinaryBackward}(\frac{\partial l}{\partial \mathcal{T}_{k+1}}, \mathcal{B}_k^t)$
16:     $\frac{\partial l}{\partial \mathcal{W}_k^t} = \frac{1}{n_l}(1 + \|\mathcal{W}_k^t\|_{l1} \mathbf{1}_{\|\mathcal{W}_k^t\|<1}) \frac{\partial l}{\partial \mathcal{B}_k^t}$
17: end for
18: 3. Update Parameters:
19: $\mathcal{W}^{t+1}, \eta^{t+1} = \textbf{Update}(\mathcal{W}^t, \frac{\partial l}{\partial \mathcal{W}^t}, \eta^t)$
20: return $\mathcal{W}^{t+1}, \eta^{t+1}$
```

The procedure for training a BNN is shown in Algorithm 1. The called procedures are listed as follows.

1) *BatchNorm():* Batch normalization function.
2) *BinarizeWeight():* Weight binarization function.
3) *BinarizeInput():* Input tensor binarization function.
4) *BinaryBackward():* Binarized backward function.
5) *Update():* Optimizer for updating weights and learning rates.

In Algorithm 1, we binarize the convolution kernel and input tensor and compute the output from first to the Lth layer first. Next, we calculate the gradients of the binarized weights ($\partial l/\partial \mathcal{B}^t$) using the standard backpropagation algorithm. Then, we calculate the gradients of the real-valued weights ($\partial l/\partial \mathcal{W}^t$) according to (16). Finally, we update the parameters and learning rate with an appropriate optimizer, e.g., NAdam [50] in this article.

E. Implementation Details

We present the implementation details in this section.

1) Redundant Computations in Overlapping Areas: During the binary convolution, each time the binary convolution kernel $\mathbf{W}_B \in \{-1, +1\}R^{c_{\text{in}} \times w_k \times h_k}$ shifts over the input tensor $T_{\text{in}} \in \mathbb{R}^{c_{\text{in}} \times w_{\text{in}} \times h_{\text{in}}}$, the scaling factor α_X for the corresponding input vector $\mathbf{X} \in \mathbb{R}^{c_{\text{in}} \times w_k \times h_k}$ needs to be recomputed. Because the stride of convolution is usually lower than the kernel size, there are overlaps between these input vectors, which leads to a large number of redundant computations. To address this problem, we first compute $|T_{\text{in}}|$ which is the scaling factor of the input tensor for every single pixel. For kernels whose shape is not 1×1, these scaling factors need to be averaged

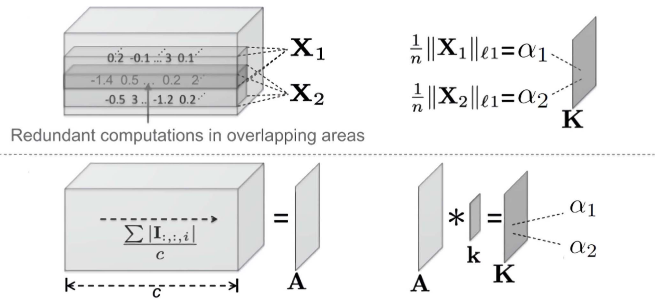
0.2 -0.1 ... 3 0.1
-1.4 0.5 ... 0.2 2
-0.5 3 ... -1.2 0.2
$\mathbf{X}_1$
$\mathbf{X}_2$
Redundant computations in overlapping areas
$\frac{1}{n}\|\mathbf{X}_1\|_{\ell 1} = \alpha_1$
$\frac{1}{n}\|\mathbf{X}_2\|_{\ell 1} = \alpha_2$
K
$\frac{\sum |\mathbf{I}_{:,:,i}|}{c}$
c
=
A
A
*
k
=
K
α_1
α_2

Fig. 6. Remove redundant computations during scaling factor calculations.

by the shape of the kernel. Next, we construct a matrix $\mathbf{K}$ with shape of $[w_k, h_k]$ whose every element is $(1/w_k h_k)$. Matrix $\mathbf{K}$ is used to average $|T_{\text{in}}|$ locally by convolving $|T_{\text{in}}|$ with $\mathbf{K}$ for each channel. The scaling factor for input tensor is α_T

$$\alpha_T(c) = |T_{\text{in}}(c, :, :)| \otimes \mathbf{K} \tag{17}$$

where $\otimes$ represents the full-precision convolution. The final output of the binary convolution layer is expressed as

$$T_{\text{out}} = \alpha_B(\text{sign}(T_{\text{in}}) \circledast \text{sign}(W_B)) \odot \alpha_T \tag{18}$$

where α_B is the scaling factor for the kernel and $\circledast$ represents the binary convolution which is much faster than the full-precision convolution. The procedure is illustrated in Fig. 6.

2) Biased Learning Algorithm: The loss function we use is softmax cross-entropy which can provide speedup for backpropagation and is widely adopted in deep learning models. In the hotspot detection task, each instance x_i that belongs to class c has a ground-truth label y_i^*, that is

$$y_i^*[k] = \begin{cases} 1 & k = c \\ 0 & k \neq c \end{cases} \tag{19}$$

where $k \in \{0, 1\}$ in the hotspot detection problem. The predicted vector and ground-truth label y_i^* are regard as the probability distribution of the classes. The hotspot instance has a label of $y_h^* = [0, 1]$ and the nonhotspot instance has a label of $y_n^* = [1, 0]$. To generate the loss, the output vector $[x_n, x_h]$ is normalized with the softmax function

$$x'_n = \frac{e^{x_n}}{e^{x_n} + e^{x_h}}$$
$$x'_h = \frac{e^{x_h}}{e^{x_n} + e^{x_h}}. \tag{20}$$

The cross entropy loss is defined in

$$L = -\big(\log(x'_n) y_i^*[0] + \log(x'_h) y_i^*[1]\big). \tag{21}$$

Note that the dataset is quite biased which contains much more nonhotspot instances than the hotspot instances. To further improve the detecting accuracy of our model, the biased learning in [22] is adopted after the model is trained with Algorithm 1. The trained model is fine-tuned with nonhotspot's label changed to $y_n^* = [1 - \epsilon, \epsilon]$ and hotspot's label keeps the same. ϵ is the bias term. In our experiment, we set $\epsilon = 0.2$. The bias learning method improves the detecting accuracy but also increases the false alarms at the same time.

IV. Models Ensemble

Due to the efficiency of the BNN model, the ensemble learning approaches are acceptable for further improving the performance with some efficiency loss.

Considering a binary classification problem, $y \in \{-1, +1\}$ and the ground-truth function is f, if the error rate of classifier h_i is ϵ, we have

$$P(h_i(x) \neq f(x)) = \epsilon. \tag{22}$$

If we integrate T classifiers by simply voting, we construct a ensemble classifier $H(x)$, that is

$$H(x) = \text{sign}\left(\sum_{i=1}^{T} h_i(x)\right). \tag{23}$$

If the error rates of classifier h_i are independent, the error rate of the ensemble classifier $H(x)$ is

$$\begin{aligned} P(H(x) \neq f(x)) &= \sum_{k=0}^{T/2} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \\ &\leq exp\left(-\frac{1}{2} T(1-2\epsilon)^2\right). \end{aligned} \tag{24}$$

Theoretically, the error rate will go down to near zero with more and more classifiers integrated. The variance of the integrated classifiers is important for improving the performance of the ensemble model. Thus, we propose two shallower network structures with ten layers and eight layers marked as BNN-10 and BNN-8 to improve the diversity without increasing too much computation complexity. The details of BNN-10 and BNN-8 are shown in Figs. 7 and 8.

Generally, given a dataset of n examples $D = \{(x_i, y_i)\}$ $|D| = n$, the ensemble learning model H uses an aggregation function G that aggregates T inducers, $\{h_1, h_2, \ldots, h_T\}$ to predict a single output

$$y = H(x) = G(h_1, h_2, \ldots, h_T)(x) \tag{25}$$

where $y \in \mathbb{R}$ for regression problems and $y \in Z$ for classification problems. To fit this hotspot detection problem, the following ensemble policies are adopted for the ensemble model. In this article, $h_i(x) = [h_i^0(x), h_i^1(x)]$, where $h_i^k(x)$ is the output for the kth class of the classifier h_i. Because the hotspot detection problem is a binary classification problem, $k \in \{0, 1\}$. In the voting-based policies $h_i^k(x) \in \{0, 1\}$, the first-level classifiers output the predicted class indices. In the averaging-based and stacking policies $h_i(x) \in \mathbb{R}^+$, the first-level classifiers output the softmax probability distributions of classes.

A. Voting-Based Policies

Majority Voting: In this policy, each classifier h_i has the same weight. The class which gets the most votes is selected

$$H(x) = \begin{cases} [0, 1] & \sum_{i=1}^{T} h_i^0(x) < \sum_{i=1}^{T} h_i^1(x) \\ [1, 0] & \text{else.} \end{cases} \tag{26}$$

Weighted Voting: In this policy, each classifier h_i has weight w_i based on its error rate ϵ_i on the test set. The sum of weights w_i is 1. The definition of w_i is

$$w_i = \frac{\frac{1}{\epsilon_i}}{\sum_{k=1}^{T} \frac{1}{\epsilon_k}}. \tag{27}$$

The class which gets the most weighted votes is selected as well

$$H(x) = \begin{cases} [0, 1] & \sum_{i=1}^{T} w_i h_i^0(x) < \sum_{i=1}^{T} w_i h_i^1(x) \\ [1, 0] & \text{else.} \end{cases} \tag{28}$$

B. Averaging-Based Policies

Simple Averaging: In this policy, the output of the single classifier h_i is the predicted probability distribution, the ensemble model simply averages these probability distributions as the final output

$$H(x) = \frac{1}{T} \sum_{i=1}^{T} h_i(x). \tag{29}$$

Weighted Averaging: Different from the simple averaging, each classifier h_i gets weight w_i based on its error rate ϵ_i in the weighted averaging policy. The definition of w_i is shown in (27). The ensemble model H averages the outputs of the first-level classifiers h_i according to their weights as the final output

$$H(x) = \sum_{i=1}^{T} w_i h_i(x). \tag{30}$$

C. Stacking

Different from the above policies, in stacking procedure, a model is trained to combine the individual first-level classifiers. The combiner is called meta-learner. The basic idea of the stacking algorithm is to train the first-level classifiers with the original training dataset and then generate a new dataset for training the meta-learner, where the outputs of the first-level classifiers are treated as input features while the original labels are still regarded as labels of the new training dataset. The general procedure of stacking is illustrated in Algorithm 2.

A new dataset is built from the outputs of the first-level classifiers while training the meta-learner. If this dataset is generated from exactly the same dataset used to train the classifiers, that may cause overfitting. So following the practice of the previous paper, the cross-validation procedure is adopted to exclude the data used for generating the new dataset from the training set used for training the first-level classifiers. In the experiment, we adopt the k-fold cross-validation procedure. The whole dataset D is split into k equal parts $D_1, \ldots, D_k$. D_{-j} is defined as $D_{-j} = D \backslash D_j$. The first-level classifier h_t is trained on D_{-j} as h_t^{-j}. For an instance x_i in D_j, the output of h_t^{-j} is $z_{it} = h_t^{-j}(x_i)$ and the new dataset generated from the instance x_i is $z_i = (z_{i1}, \ldots, z_{iT}), y_i$. The new dataset for training the meta-learner is

$$D' = \{(z_i, y_i)\}_{i=1}^{m}. \tag{31}$$

The meta-learner h' learns a function from z_i to y_i. Then, the first-level classifiers are regenerated by training on the whole training dataset.

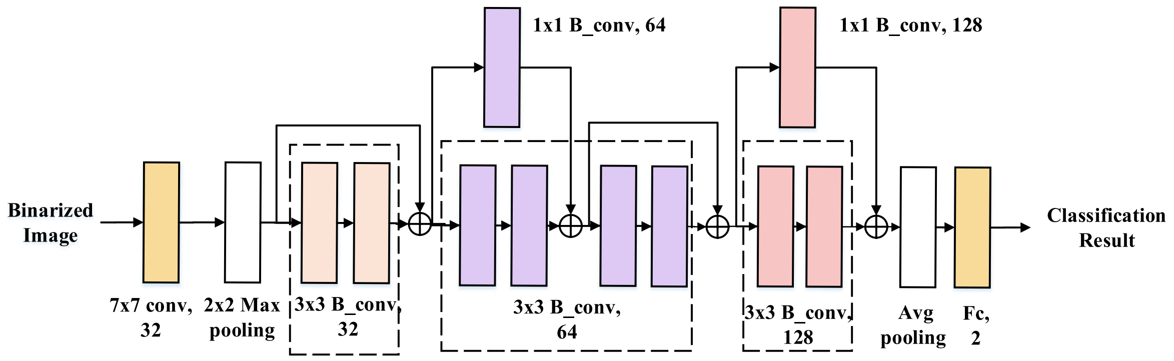

Fig. 7. Network structure of BNN-10.

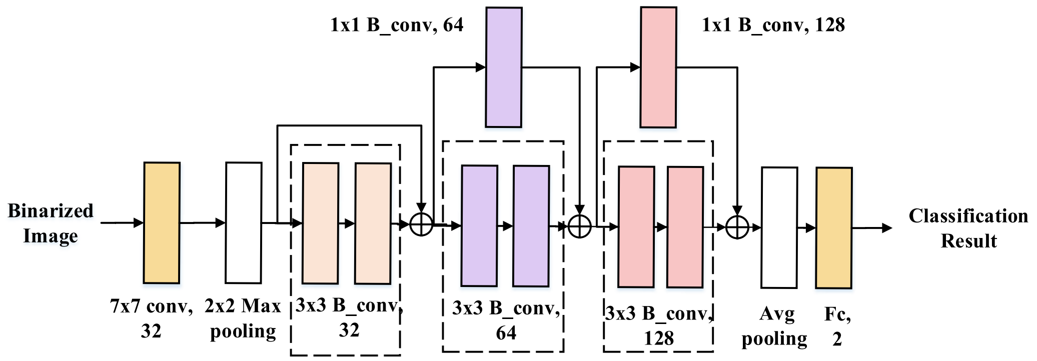

Fig. 8. Network structure of BNN-8.

Algorithm 2 Stacking Procedure

```
Input:
1: $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$: Training set;
2: $\zeta_1, \ldots, \zeta_T$: first-level classifiers learning algorithms;
3: $\zeta$: Ensemble model learning algorithm.
4: for $t = 1$ to $T$ do
5:     $h_i = \zeta_i(D)$
6: end for
7: $D' = \varnothing$
8: for $i = 1$ to $n$ do
9:     for $t = 1$ to $T$ do
10:        $z_{it} = h_t(x_i)$
11:    end for
12:    $D' = D' \cup ((z_{i1}, \ldots, z_{iT}), y_i)$
13: end for
14: $h' = \zeta(D')$
Output:
15: $H(x) = h'(h_1(x), \ldots, h_T(x))$: Ensemble classifier.
```

Because we only have three models, the T-fold cross-validation procedure can lose too many instances for training the first-level classifiers although these they will be trained on the whole training set after the whole cross-validation procedure. As a result, we adopt tenfold cross-validation in the training process.

Following the practice in [51] and [52], class probability distributions are used instead of crisp class labels as features for the new dataset to show the confidence of the individual first-level classifiers. Multiresponse linear regression (MLR) [53] is adopted for the meta-learner. MLR is a variant of the least-square linear regression algorithm. For conventional linear regression model M_{lr}, to predict the probability of the kth class p_k in a K-class classification problem with input instance as x_i

$$\begin{aligned} p_k &= M_{lr}([z_{i1}, \ldots, z_{iT}]) \\ \min \quad & \|p_k - y_i[k]\|^2 \end{aligned} \tag{32}$$

where z_{it} is the output of the t-th classifier which has K features. The model is trained by minimizing the square error between p_k and $y_i[k]$, where $y_i[k]$ is the kth element of y_i. The class with highest p_k is selected as the predicted class.

The MLR model is a little different from LR. Only the probability of the kth class predicted by first-level classifiers is used to construct the model when predict the probability of the kth class

$$\begin{aligned} p_k &= M_{mlr}([z_{i1}[k], \ldots, z_{iT}[k]]) \\ \min \quad & \|p_k - y_i[k]\|^2 \end{aligned} \tag{33}$$

where $z_{it}[k]$ is the kth element of the output of the t-th first-level classifier. With MLR, better performance is usually achieved than adopting LR.

V. Experimental Results

A. Experimental Setup

All our training and testing code is built on MXNet [54] referring to the implementation in [55]. We train and test our

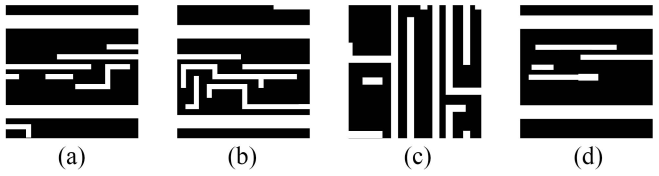

Fig. 9. Dataset examples. (a) and (b) Hotspots. (c) and (d) Nonhotspots.

TABLE II
BENCHMARK STATISTICS

Benchmark	#Train HS	#Train NHS	#Test HS	#Test NHS
ICCAD 12	1204	17096	2524	13503
ICCAD 19-1	467	17758	1001	14621
ICCAD 19-2	467	17758	64310	65523

TABLE III
TRAINING HYPERPARAMETER SETTINGS

Hyperparameter	Value
Feature Size(l_s)	128
Batch Size	128
Bias Term (ϵ)	0.2
Data Augumentation Method	Random flipping
Initial Learning Rate (λ)	0.15
Learning Rate Decay (α)	0.5
Learning Rate Decay Epoches (K)	40
optimizer	NAdam [50]

model on a machine with a 4-core Intel CPU and a Nvidia GTX 1060 GPU.

Following the practice in [22], we merge all the patterns of the ICCAD 2012 contest to verify the scalability of our model. Fig. 9 shows some samples of the dataset. Due to the fact that the ICCAD 2012 benchmark lacks the diversity of patterns, we further experiment on the more challenging ICCAD 2019 benchmark. The statistics of the datasets is listed in Table II. For training data, columns "# Training HS" and "# Training NHS" show the numbers of hotspots and nonhotspots in the training set, respectively. For testing data, columns "# Testing HS" and "# Testing NHS" give the numbers of hotspots and nonhotspots in the testing set. The ICCAD 19-1 dataset and ICCAD 19-2 dataset share the same training set.

B. Training Details

We present the training details in this section. Table III shows the hyperparameter settings.

1) Data Preprocessing: Different from normal image classification problems, the hotspots might be anywhere in the input layout clips so the widely used randomly cropping augmentation method is not adopted in this article. Note that the input layout clips are all square. The input binary images are simply downsampled to $[l_s, l_s]$. After careful tuning, l_s is finally set as 128 which achieves a nice balance between accuracy and speed. Random horizontal and vertical flipping are performed for training which can improve the diversity of the dataset and increase the trained model's generalization ability due to the flipping variance of the convolution operation. Compared with preprocessing methods adopted in previous works, such as DCT-based feature tensor extraction [22] and maximal circle mutual information (MCMI) scheme [14], our preprocessing method keeps the most spatial information of the original patterns.

2) Training Hyperparameters: The real-valued kernels are initialized with Xavier initializer [46]. We do not use dropout [56] following the practice in ResNet paper [47].

The optimizer adopted for training the model is Nesterov-accelerated adaptive moment estimation (NAdam) optimizer [50], which combines adaptive moment estimation (Adam) optimizer [57] and Nesterov-accelerated gradient (NAG) optimizer [58]. We train our model using mini-batches of size 128 for 150 epoches and pick the model with the best performance on the validation set. We set the initial learning rate as 0.15. The learning rate decay scheme is to exponentially decay each 40 epoch which is used in [59]. The bias term is set to 0.2 as mentioned in Section III-E.

C. Experimental Results on ICCAD 2012 Benchmark

To evaluate the performance of our ensemble learning model, we compare the results of the ensemble model on the ICCAD 2012 benchmark with our preliminary results in [23] as shown in Table IV. Four main metrics are listed in the table. "False Alarm (%)" denotes the ratio of false alarms (Definition 2) to all nonhotspots. "Runtime (s)" denotes the evaluating time of the model. "ODST (s)" denotes the overall detection simulation time (Definition 3). "Accuracy (%)" denotes the hotspot detection accuracy. All deep learning models are accelerated with a middle-end graphic card Nvidia GTX1060. The others are tested on CPU only. Following [60], we set the lithography simulation time per instance t_{ls} in (3) as 10 s to calculate the ODST.

"DAC19 [23]" lists the results of the model in our preliminary work [23]; "BNN-10" and "BNN-8" correspond to the results obtained by the two shallower models with ten layers and eight layers, respectively; "E-MV" and "E-WV" show the results of the ensemble models which take majority voting policy and weighted voting policy in Section IV-A; "E-SA" and "E-WA" correspond to the ensemble models with simple averaging policy and weighted averaging policy in Section IV-B; and "E-Stacking" lists the result of the ensemble model with stacking algorithm in Algorithm 2.

In this article, we adopt ensemble learning approaches to further improve the performance of the binary neural network models. As listed in Table IV, the two shallower networks get lower accuracy and more false alarms than the original model due to the network structure complexity loss but with more computation and storage efficiency. The next five ensemble models (E-MV, E-WV, E-SA, E-WA, and E-Stacking) combine the original model and the two shallower models BNN-10 and BNN-8 to achieve a better performance. The ensemble models almost outperform the original BNN model in accuracy (99.3%, 99.2%, 99.3%, 99.3%, and 99.4% versus 99.2% of [23]). The "E-SA" and "E-Stacking" also outperform the original model in false alarm rate (20.5% of "E-SA," 19.7% of "E-Stacking" versus 20.6% of [23]). The "E-MV," "E-WV," and "E-WA" achieve comparable performance in false alarms as well. The runtime of the ensemble models is almost

TABLE IV
PERFORMANCE COMPARISONS BETWEEN INDIVIDUAL MODELS AND THE ENSEMBLE MODELS ON ICCAD 2012 BENCHMARK

Method	False Alarm (%)	Runtime (s)	ODST (s)	Accuracy (%)	Training Time (s/epoch)
DAC'19 [23]	20.6	60	52970	99.2	65
BNN-10	22.4	51	55141	98.3	62
BNN-8	25.1	**43**	55653	98.0	**52**
E-MV	20.8	136	53216	99.3	-
E-WV	22.7	133	55713	99.2	-
E-SA	20.5	133	52883	99.3	-
E-WA	21.5	135	54185	99.3	-
E-Stacking	**19.7**	136	**51776**	**99.4**	-

TABLE V
PERFORMANCE COMPARISONS WITH PREVIOUS HOTSPOT DETECTORS ON ICCAD 2012 BENCHMARK

Dataset	Method	False Alarm (%)	Runtime (s)	ODST (s)	Accuracy (%)
ICCAD 2012	SPIE'15 [11]	21.6	2672	53112	84.2
	ICCAD'16 [14]	33.3	1052	70628	97.7
	TCAD'18 [22]	26.2	482	60672	98.4
	DAC'19 [23]	20.6	**60**	52970	99.2
	E-Stacking	**19.7**	136	**50776**	**99.4**

TABLE VI
PERFORMANCE COMPARISONS WITH STATE-OF-THE-ART HOTSPOT DETECTORS ON ICCAD 2019 BENCHMARK

Dataset	Method	False Alarm (%)	Runtime (s)	ODST (s)	Accuracy (%)
ICCAD 2019-1	TCAD'18 [22]	2.6	470	11820	76.0
	DAC'19 [23]	3.5	**59**	13219	**80.9**
	E-Stacking	**2.5**	130	**11720**	**80.9**
ICCAD 2019-2	TCAD'18 [22]	87.8	3807	1147327	88.4
	DAC'19 [23]	84.1	**498**	1071878	89.7
	E-Stacking	**83.9**	1090	**1058820**	**89.8**

the same which is about 2.25 times of the original model. But considering the efficiency of the original model, the runtime increase is acceptable. The best-performed "E-Stacking" model also outperforms the original model in ODST (51 776 s of "E-Stacking" versus 52 970 s of [23]) which enormously remedies the runtime increase.

Some of the ensemble models get more false alarms than the original model. We think the reason might be that the models trained with biased learning approaches do not predict confidently enough for nonhotpots. There is a tradeoff between the accuracy and false alarm achieved by decreasing the confidence of nonhotspot predictions. Therefore, some inappropriate weight distributions can possibly lead to more false alarms. Also results in [61]–[63] show that weighted averaging is not clearly superior to simple averaging because of overfitting problems.

We then compare our model's hotspot detection results with four previous hotspot detectors in Table V. "SPIE'15 [11]" adopts the density-based layout features and the AdaBoost-DecisionTree model. "ICCAD'16 [14]" applies optimized CCS feature and online learning scheme for hotspot detection. "TCAD'18 [22]" improves the hotspot detection performance with DCT-based feature extraction and deep biased learning algorithm. "DAC'19 [23]" enormously improves the efficiency of the hotspot detector with a residual BNN architecture. "E-Stacking" is our best-performed ensemble model which adopts the stacking algorithm in Algorithm 2 to combine the model of [23] and its two shallower variants.

The experimental results show that our model achieves the best performance in the ICCAD-2012 testcase. Overall our "E-Stacking" ensemble model outperforms the previous hotspot detectors in both accuracies (99.4% of E-Stacking versus 84.2% of SPIE'15; 97.7% of ICCAD'16; 98.4% of TCAD'18; and 99.2% of DAC'19) and false alarm rate (19.7% of E-Stacking versus 21.6% of SPIE'15; 33.3% of ICCAD'16; 26.2% of TCAD'18; and 20.6% of DAC'19). Due to the fewest false alarms our ensemble model achieves, "E-Stacking" gets the lowest ODST among all hotspot detectors as well. Although the ensemble "E-Stacking" model loses some computation efficiency compared with the original BNN model in DAC'19 due to the models ensemble (136 s of "E-Stacking" versus 60 s of DAC'19), the runtime of "E-Stacking" is still acceptable because it is still more efficient than all other hotpot detectors. That should be owed to the efficiency of the BNN architecture.

D. Experimental Results on ICCAD 2019 Benchmark

We further experiment our method on the more challenging ICCAD 2019 dataset [64]. Table VI shows the results of the state-of-the-art methods [22], [23] and the best-performed "E-Stacking" model on the benchmark.

In the ICCAD 2019-1 dataset, the "E-Stacking" ensemble model outperforms the other two state-of-the-art methods in both accuracy (80.9% of E-Stacking versus 76.0% of TCAD'18 and 80.9% of DAC'19) and false alarm rate (2.5%

of E-Stacking versus 2.6% of TCAD'18 and 3.5% of DAC'19). In the ICCAD 2019-2 dataset, the "E-Stacking" ensemble model gets the best performance as well in both accuracy (89.8% of E-Stacking versus 88.4% of TCAD'18 and 89.9% of DAC'19) and false alarm rate (83.9% of E-Stacking versus 87.8% of TCAD'18 and 84.1% of DAC'19). We also notice that in the ICCAD 2019 benchmark paper [64] the VTS'18 [65] method gets a quite low false alarm rate by generating synthetic hotspot patterns and adding them to the training set so as to enhance the training set. Since the generated patterns for training is not released yet, we do not compare it with the other methods which are trained on the released training set temporarily. The "E-Stacking" model loses some efficiency compared with the single model (130 s of E-Stacking versus 59 s of DAC'19 and 1090 s of E-Stacking versus 498 s of DAC'19) which is similar to the results in the previous experiments. Due to the fewest false alarms, the "E-Stacking" model still gets the lowest ODST among the three methods in both ICCAD 2019-1 and ICCAD 2019-2 datasets.

VI. Conclusion

Deep neural network models have been applied to hotspot detection and achieved great successes. However, the deep neural network models can also lead to enormous computational and storage consumption. In this article, considering the binary characteristic of the lithography layout, we propose a BNN-based architecture to address this problem. The downsampled images of the patterns are taken as the inputs directly, and the spatial information of the patterns can be captured in our approach. The experimental results on ICCAD 2012 and 2019 benchmarks show that the BNN-based architecture outperforms previous hotspot detectors and achieves an 8× speedup over the best deep learning-based solution. We further adopt ensemble learning approaches which combine the BNN-based model and its two shallower virant models to further improve the performance of the BNN model. The experimental results show that the ensemble model achieves better hotspot detection performance compared with the original BNN-base model with acceptable efficiency loss. Note that BNNs are more compatible with digital circuits than traditional CNNs. Thus, the more efficient hardware-accelerated hotspot detectors are expected.

References

[1] H. Yao, S. Sinha, J. Xu, C. Chiang, Y. Cai, and X. Hong, "Efficient range pattern matching algorithm for process-Hotspot detection," in *Proc. ICCAD*, 2006, pp. 2–15.

[2] Y.-T. Yu, Y.-C. Chan, S. Sinha, I. H.-R. Jiang, and C. Chiang, "Accurate process-Hotspot detection using critical design rule extraction," in *Proc. DAC*, 2012, pp. 1167–1172.

[3] A. B. Kahng, C.-H. Park, and X. Xu, "Fast dual graph based Hotspot detection," in *Proc. SPIE*, 2006, Art. no. 63490H.

[4] W.-Y. Wen, J.-C. Li, S.-Y. Lin, J.-Y. Chen, and S.-C. Chang, "A fuzzy-matching model with grid reduction for lithography Hotspot detection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 11, pp. 1671–1680, Nov. 2014.

[5] F. Yang, S. Sinha, C. C. Chiang, X. Zeng, and D. Zhou, "Improved tangent space-based distance metric for lithographic Hotspot classification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1545–1556, Sep. 2017.

[6] J.-Y. Wuu, F. G. Pikus, and M. Marek-Sadowska, "Efficient approach to early detection of lithographic Hotspots using machine learning systems and pattern matching," in *Proc. SPIE*, 2012, Art. no. 79740U.

[7] D. D. *et al.*, "Efficient prediction of IC manufacturing Hotspots with a unified meta-classification formulation," in *Proc. ASP-DAC*, 2012, pp. 263–270.

[8] D. Ding and D. Z. Pan, "Machine learning based lithographic Hotspot detection with critical feature extraction and classifications," in *Proc. Int. Conf. Integr. Circuit Design Technol.*, 2009, pp. 219–222.

[9] D. Ding, A. J. Torres, F. G. Pikus, and D. Z. Pan, "High performance lithographic Hotspot detection using hierarchically refined machine learning," in *Proc. ASPDAC*, 2011, pp. 775–780.

[10] Y. T. Y. et al, "Machine-learning-based Hotspot detection using topological classification and critical feature extraction," in *Proc. DAC*, 2013, pp. 95–115.

[11] T. Matsunawa, J.-R. Gao, B. Yu, and D. Z. Pan, "A new lithography Hotspot detection framework based on adaboost classifier and simplified feature extraction," in *Proc. Design Process Technol. Cooptim. Manufacturability IX*, vol. 9427, 2015, Art. no. 94270S.

[12] D. Ding, B. Yu, J. Ghosh, and D. Z. Pan, "Epic: Efficient prediction of ic manufacturing Hotspots with a unified meta-classification formulation," in *Proc. IEEE 17th Asia–South Pac. Design Autom. Conf. (ASP-DAC)*, 2012, pp. 263–270.

[13] D. G. Drmanac, F. Liu, and L.-C. Wang, "Predicting variability in nanoscale lithography processes," in *Proc. ACM 46th Annu. Design Autom. Conf.*, 2009, pp. 545–550.

[14] H. Zhang, B. Yu, and E. F. Young, "Enabling online learning in lithography Hotspot detection with information-theoretic feature optimization," in *Proc. ACM 35th Int. Conf. Comput. Aided Design*, 2016, p. 47.

[15] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical bayes model," in *Proc. Opt. Microlithography XXVIII*, vol. 9426, 2015, Art. no. 94260X.

[16] J. W. Park, R. Todd, and X. Song, "Geometric pattern match using edge driven dissected rectangles and vector space," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 12, pp. 2046–2055, Feb. 2016.

[17] J. W. Park, A. Torres, and X. Song, "Litho-aware machine learning for Hotspot detection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 7, pp. 1510–1514, Jul. 2018.

[18] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography Hotspot detection: A deep learning approach," *J. Micro/Nanolithography MEMS MOEMS*, vol. 16, no. 3, 2017, Art. no. 033504.

[19] H. Yang, Y. Lin, B. Yu, and E. F. Young, "Lithography Hotspot detection: From shallow to deep learning," in *Proc. 30th IEEE Int. System-on-Chip Conf. (SOCC)*, 2017, pp. 233–238.

[20] T. Matsunawa, S. Nojima, and T. Kotani, "Automatic layout feature extraction for lithography Hotspot detection based on deep neural network," in *Proc. Design Process Technol. Cooptim. Manufacturability X*, vol. 9781, 2016, Art. no. 97810H.

[21] M. Shin and J.-H. Lee, "Accurate lithography Hotspot detection using deep convolutional neural networks," *J. Micro/Nanolithography MEMS MOEMS*, vol. 15, no. 4, 2016, Art. no. 043507.

[22] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Young, "Layout Hotspot detection with feature tensor generation and deep biased learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 6, pp. 1175–1187, Jun. 2019.

[23] Y. Jiang, F. Yang, H. Zhu, B. Yu, D. Zhou, and X. Zeng, "Efficient layout Hotspot detection via binarized residual neural network," in *Proc. DAC*, 2019, pp. 1–147.

[24] J. A. Torres, "ICCAD-2012 cad contest in fuzzy pattern matching for physical verification and benchmark suite," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2012, pp. 349–350.

[25] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[26] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2148–2156.

[27] S. Arora, A. Bhaskara, R. Ge, and T. Ma, "Provable bounds for learning some deep representations," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 584–592.

[28] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Proc. Deep Learn. Unsupervised Feature Learn. NIPS Workshop*, vol. 1, 2011, p. 4.

[29] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, 2014, pp. 1–6.

[30] M. Courbariaux, Y. Bengio, and J.-P. David. (2014). *Training Deep Neural Networks With Low Precision Multiplications*. [Online]. Available: https://arxiv.org/abs/1412.7024

[31] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 963–971.

[32] M. Courbariaux and Y. Bengio. (2017). *BinaryNet: Training Deep Neural Networks With Weights and Activations Constrained to + 1 or* −1. [Online]. Available: https://arxiv.org/abs/1602.02830.

[33] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[34] M. Kim and P. Smaragdis. (2016). *Bitwise Neural Networks*. [Online]. Available: https://arxiv.org/abs/1601.06071

[35] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNoR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[36] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[37] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2017, pp. 15–24.

[38] L. Jiao, C. Luo, W. Cao, X. Zhou, and L. Wang, "Accelerating low bit-width convolutional neural networks with embedded FPGA," in *Proc. 27th Int. Conf. IEEE Field Program. Logic Appl. (FPL)*, 2017, pp. 1–4.

[39] D. J. Moss *et al.*, "High performance binary neural networks on the xeon+ FPGA platform," in *27th Int. Conf. IEEE Field Program. Logic Appl. (FPL)*, 2017, pp. 1–4.

[40] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *ACM Comput. Surveys*, vol. 50, no. 2, p. 23, 2017.

[41] V. Y. Kulkarni and D. P. K. Sinha, "Pruning of random forest classifiers: A survey and future," in *Proc. Int. Conf. Data Sci. Eng. (ICDSE)*, 2012, pp. 2051–2845.

[42] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[43] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[44] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, p. 533, 1986.

[46] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Stat.*, 2010, pp. 249–256.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[48] S. Ioffe and C. Szegedy. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. [Online]. Available: https://arxiv.org/abs/1502.03167

[49] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[50] T. Dozat, "Incorporating nesterov momentum into adam," in *Proc. ICLR Workshop*, 2016, pp. 1–6.

[51] D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, 1992.

[52] K. M. Ting and I. H. Witten, "Issues in stacked generalization," *J. Artif. Intell. Res.*, vol. 10, no. 1, pp. 271–289, 1999.

[53] L. Breiman, "Stacked regressions," *Mach. Learn.*, vol. 24, no. 1, pp. 49–64, 1996.

[54] T. Chen *et al.* (2015). *MxNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems*. [Online]. Available: https://arxiv.org/abs/1512.01274 .

[55] H. Yang, M. Fritzsche, C. Bartz, and C. Meinel, "BMXNet: An open-source binary neural network implementation based on MxNet," in *Proc. ACM Multimedia Conf.*, 2017, pp. 1209–1212.

[56] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. (2012). *Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors*. [Online]. Available: https://arxiv.org/abs/1207.0580

[57] D. P. Kingma and J. Ba. (2014). *Adam: A Method for Stochastic Optimization*. [Online]. Available: https://arxiv.org/abs/1412.6980

[58] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2)," in *Proc. Doklady AN USSR*, vol. 269, 1983, pp. 543–547.

[59] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.

[60] S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 cad contest in mask optimization and benchmark suite," in *IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2013, pp. 271–274.

[61] L. Xu, A. Krzyzak, and C. Y. Suen, "Methods of combining multiple classifiers and their applications to handwriting recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 3, pp. 418–435, May/Jun. 1992.

[62] T. K. Ho, J. J. Hull, and S. N. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 1, pp. 66–75, Jan. 1994.

[63] J. Kittler, M. Hater, and R. P. Duin, "Combining classifiers," in *Proc. IEEE 13th Int. Conf. Pattern Recognit.*, vol. 2, 1996, pp. 897–901.

[64] G. R. Reddy, K. Madkour, and Y. Makris, "Machine learning-based Hotspot detection: Fallacies, pitfalls and marching orders," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2019, pp. 1–8.

[65] G. R. Reddy, C. Xanthopoulos, and Y. Makris, "Enhanced Hotspot detection through synthetic pattern generation and design of experiments," in *Proc. IEEE 36th VLSI Test Symp. (VTS)*, 2018, pp. 1–6.

Yiyang Jiang received the B.E. degree from the Department of Microelectronics, Fudan University, Shanghai, China, in 2016, where he is currently pursuing the Ph.D. degree with the State Key Laboratory of Application Specific Integrated Circuits and System.

His research interests include VLSI design for manufacturability and machine learning.

Fan Yang (Member, IEEE) received the B.S. degree from Xi'an Jiaotong University, Xi'an, China, in 2003, and the Ph.D. degree from Fudan University, Shanghai, China, in 2008.

From 2008 to 2011, he was an Assistant Professor with Fudan University, where he is currently an Associate Professor with the Microelectronics Department. His research interests include model order reduction, circuit simulation, high-level synthesis, yield analysis, and design for manufacturability.

Bei Yu (Member, IEEE) received the Ph.D. degree from the University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

Dr. Yu received six Best Paper Awards from the International Conference on Tools with Artificial Intelligence in 2019, the Integration, the VLSI Journal in 2018, the International Symposium on Physical Design in 2017, the SPIE Advanced Lithography Conference in 2016, the International Conference on Computer Aided Design in 2013, and the Asia and South Pacific Design Automation Conference in 2012, and six ICCAD/ISPD Contest Awards. He has served as the TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He is an Editor of IEEE TCCPS Newsletter.

Dian Zhou (Senior Member, IEEE) received the B.S. degree in physics and the M.S. degree in electrical engineering from Fudan University, Shanghai, China, in 1982 and 1985, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1990.

He joined the University of North Carolina at Charlotte, Charlotte, NC, USA, as an Assistant Professor in 1990, where he became an Associate Professor in 1995. He joined the University of Texas at Dallas, Richardson, TX, USA, as a Full Professor in 1999. His research interests include high-speed VLSI systems, CAD tools, mixed-signal ICs, and algorithms.

Xuan Zeng (Senior Member, IEEE) received the B.S. and Ph.D. degrees in electrical engineering from Fudan University, Shanghai, China, in 1991 and 1997, respectively.

She is a Full Professor with the Department of Microelectronics, Fudan University, where she was the Director of the State Key Laboratory of ASIC & System from 2008 to 2012. She was a Visiting Professor with the Department of Electrical Engineering, Texas A&M University, College Station, TX, USA, and the Department of Microelectronics, Technische Universiteit Delft, Delft, The Netherlands, in 2002 and 2003, respectively. Her current research interests include analog circuit modeling and synthesis, design for manufacturability, high-speed interconnect analysis and optimization, and circuit simulation.

Prof. Zeng received the Best Paper Award from the 8th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference in 2017, the Changjiang Distinguished Professor with the Ministry of Education Department of China in 2014, the Chinese National Science Funds for Distinguished Young Scientists in 2011, the First-Class of Natural Science Prize of Shanghai in 2012, the 10th For Women in Science Award in China in 2013, and the Shanghai Municipal Natural Science Peony Award in 2014. He is an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS, the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and the *ACM Transactions on Design Automation on Electronics and Systems*.