Efficient Layout Hotspot Detection via Binarized Residual Neural Network

Yiyang Jiang[1], Fan Yang[1*], Hengliang Zhu[1], Bei Yu[3], Dian Zhou[2] and Xuan Zeng[1*]
[1]State Key Lab of ASIC & System, Microelectronics Department, Fudan University, China
[2]University of Texas at Dallas, USA
[3]Chinese University of Hong Kong, Hong Kong, China

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 can thus be suitable for the hotspot detection problem. In this paper we propose a new deep learning architecture based on binarized neural networks (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 Contest benchmarks show that our architecture outperforms all previous hotspot detectors in detecting accuracy and has an 8x speedup over the best deep learning-based solution.

KEYWORDS

Hotspot Detection, Deep Neural Network, Binarized Neural Network

1 INTRODUCTION

The lithographic printability is one of the most critical issues in nano-scale 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: pattern matching-based and machine learning-based approaches. The pattern matching-based methods characterize the hotspots as explicit patterns and identify the hotspots by matching these patterns. In [1][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, principle 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][9][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], multi-kernel support vector machine and critical feature extraction are adopted for hotspot detection. In [13], unsupervised SVM model and histogram-based layout representationto are applied to predict hotspots. In [14], optimized concentric circle sampling (CCS) feature [15] and online learning scheme are proposed for hotspot detection.

Deep neural network have demonstrated great successes in the image classification, object detection tasks in the community of computer vision. In [16], a convolutional neural network (CNN) architecture is proposed for the hotspot detection. It can achieve 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 convolutional neural network architecture. However, the discrete cosine transforms would 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 might thus be suitable for constructing an efficient hotspot detector. In this paper we propose a new deep learning architecture based on binarized neural networks (BNNs) to speed up the neural networks in hotspot detection. The down-sampled 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 binarized neural networks 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 8x speedup over the best deep learning-based solution.

The rest of this paper is organized as follows. In section 2, the background of hotspot detection is presented. In section 3, we propose the BNN-based hotspot detection method. In section 4, experimental results are shown to demonstrate the efficiency of the proposed method. In section 5, we conclude the paper.

2 BACKGROUND

In this section, we will present the problem formulation of layout hotspot detection and review the background of binarized neural networks.

2.1 Problem Formulation

Lithographic process in chip manufacturing may involve various variations, which can cause potential open or short circuit failures

*Corresponding authors: {yangfan, xzeng}@fudan.edu.cn.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317811

Table 1: Confusion Matrix of Hotspot Detection Problems

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

and result in performance degradation and yield reduction. Layout patterns that are sensitive to process variations are defined as *hotspots*.

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

Table 1 shows the confusion matrix of the hotspot detection problem, where TN denotes True Negative, FN denotes False Negative, FP denotes False Positive and TP denotes True Positive. We have the following definitions.

Definition 2.1. Accuracy: The ratio of correctly predicted hotspots among the set of actual hotspots [17].

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

Definition 2.2. False Alarm: The number of incorrectly predicted non-hotspots [17].

$$False\ Alarm = \#\, FP \tag{2}$$

Definition 2.3. ODST: Abbreviation of Overall Detection and Simulation Time. 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].

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

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

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

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

2.2 Binarized Neural Networks

In recent years, deep convolution neural networks [18] 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 [16]. However deep neural networks often suffer from over-parametrization and enormous redundancy in their models which can result in enormous computational and storage consumption [19]. Parameter quantizing is usually applied to alleviate this problem because high precision filters such as 32-bit floating-point weights are not necessary for deep neural networks. Thus the weights can be quantized to low bit with acceptable accuracy loss. It is demontrated in [20] that a sparse neural network with +1/0/-1 weights can be trained in polynomial time. 32-bit floating-point activations are quantized to 8-bit fixed-point integers in [21]. Neural networks with 3-bit activations and ternary weights are proposed in [22].

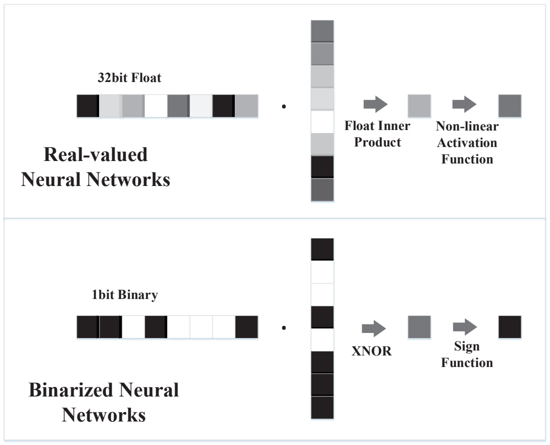
32bit Float
Float Inner Product
Non-linear Activation Function
Real-valued Neural Networks
1bit Binary
XNOR
Sign Function
Binarized Neural Networks

Figure 1: Difference between Real-valued Neural Networks and Binarized Neural Networks

Binarized neural network (BNN) is a special type of parameter quantizing method because the weights are extremely quantized to 1 bit. Figure 1 shows the difference between Real-valued Neural Networks and Binarized Neural Networks. Due to precision loss of parameters, BNNs were believed to face serious performance degradation [23]. However, expectation back propagation (EPB) is proposed in [24] to train a high-performance BNN. In BinaryNet [25, 26], real-valued weights are used for binarization and they are updated ignoring the binarization in the back propagation process. A BNN is obtained by re-training a trained neural network with binary weights and binary inputs in [27]. [28] adopts a new way of binarizing parameters and activations and achieves huge advance in large datasets such as ILSVRC [29] (ImageNet Large Scale Visual Recognition Competition) 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 [30]. An architecture based on the two-stage arithmetic unit (TSAU) is proposed in [31] to implement the low-bit CNN on FPGA. A BNN accelerator on the Xeon+FPGA platform is implemented in [32].

3 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 non-hotspots. 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.

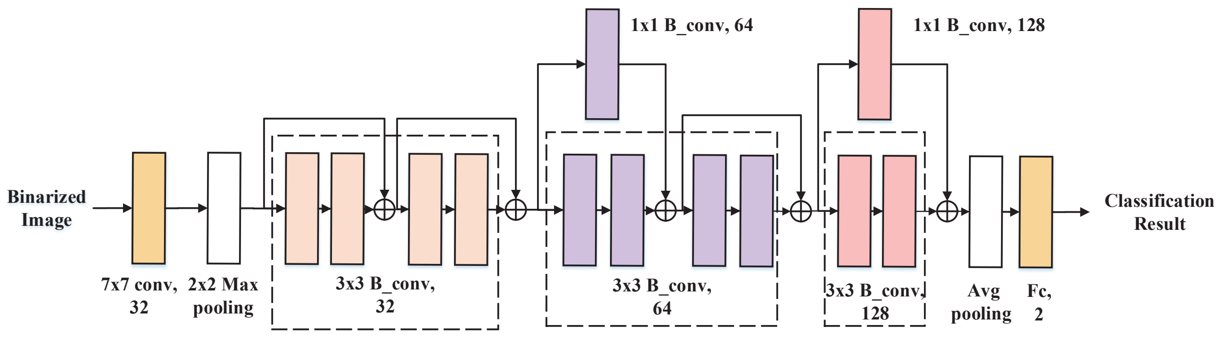

Figure 2: Redesigned Binarized Network based on Residual Network

3.1 Network Architecture

With networks going deeper, some problems begin to influence the convergence of the network like gradient vanishing/exploding [33] and accuracy saturation. ResNets [34] 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 over-fitting 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 Figure 2. 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: Batch Normalization [35], Binarizing and Binary Convolution. Figure 3 shows the structure of a convolution block. Batch Normalization layer normalizes the input tensor by its mean and variance. Binarizing layer binarizes the input tensors as the input of the next binary convolution layer. Following the practice in [28], the Batch Normalization layer is placed before the binarizing layer to further reduce the information loss due to binarization.

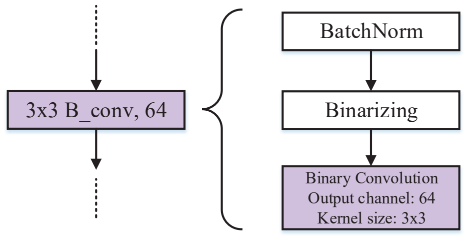

Figure 3: Typical BNN block structure

3.2 Binarization Approach

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 k-th convolution filter of the l-th layer. $\mathcal{W}_{l,k} \in \mathbb{R}^{c_{in} \times w_k \times h_k}$, where (c_{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 l-th layer and the output tensor of the $(l-1)$-th layer as well. $\mathcal{T}_l \in \mathbb{R}^{c_{in} \times w_{in} \times h_{in}}$, where (c_{in}, w_{in}, h_{in}) represents the channel, width and height of the input tensor. f and $\otimes$ represents activation function and convolution computation so that $\mathcal{T}_{l+1,k} = f(\mathcal{T}_l \otimes \mathcal{W}_{l,k})$.

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 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 $T_{in} \in \mathcal{T}$. Here, $W_B \in \{+1,-1\}^{c_{in} \times w_k \times h_k}$ and $T_B \in \{+1,-1\}^{c_{in} \times w_{in} \times h_{in}}$. $\alpha_T \in \mathbb{R}^{c_{in} \times w_{in} \times h_{in}}, \alpha_B \in \mathbb{R}^+$. $\circledast$ 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 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, 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 [28] 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 Equation (4).

$$L_i(\mathbf{W}_B, \mathbf{X}_B, \alpha_W, \alpha_X) = \|\mathbf{W} \odot \mathbf{X} - \alpha_W \mathbf{W}_B \odot \alpha_X \mathbf{X}_B\|^2 \tag{4}$$

Minimizing binarization loss can be rewritten to the optimization problem in Equation (5).

$$\mathbf{W}_B^*, \mathbf{X}_B^*, \alpha_W^*, \alpha_X^* = \underset{\mathbf{W}_B, \mathbf{X}_B, \alpha_W, \alpha_X}{\arg\min} \quad L_i(\mathbf{W}_B, \mathbf{X}_B, \alpha_W, \alpha_X) \tag{5}$$

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 (5) can be rewritten as:

$$\mathbf{C}_B^*, \alpha^* = \underset{\mathbf{C}_B, \alpha}{\arg\min} \|\mathbf{C} - \alpha \mathbf{C}_B\|^2 \tag{6}$$

Solving the optimization problem, we have:

$$\begin{aligned} \mathbf{C}_B^* &= sign(\mathbf{C}) \\ \alpha^* &= \frac{1}{n} \|\mathbf{C}\|_{l1} \end{aligned} \tag{7}$$

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

$$\begin{aligned} \mathbf{W}_B^* &= sign(\mathbf{W}), \quad & \mathbf{X}_B^* &= sign(\mathbf{X}) \\ \alpha_W^* &= \frac{\|\mathbf{W}\|_{l1}}{n}, \quad & \alpha_X^* &= \frac{\|\mathbf{X}\|_{l1}}{n} \end{aligned} \tag{8}$$

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

$$\begin{aligned} \widetilde{\mathbf{W}} &= \frac{1}{n} sign(\mathbf{W}) \|\mathbf{W}\|_{l1} \\ \widetilde{\mathbf{X}} &= \frac{1}{n} sign(\mathbf{X}) \|\mathbf{X}\|_{l1} \end{aligned} \tag{9}$$

3.3 Training Binarized Networks

Like normal CNNs, we train our network with Mini-batch Gradient Descent (MGD) [36] approach which can utilize computation resources more efficiently than Stochastic Gradient Descent (SGD). A group of instances are randomly picked from the training set to perform forward and backward process in each training iteration.

During training process, the main object is to update the real-valued kernel $\mathbf{W}$. Back-propagation [37] 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 back-propagation. The key difference between CNN and BNN architecture is the sign function $sign(r)$. The derivative of 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 [26] which considers the saturation effect

$$\frac{\partial sign(x)}{\partial x} = \mathbf{1}_{\|x\|<1} \tag{10}$$

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

$$\mathbf{1}_{\|x\|<1} = \begin{cases} 1 & \|x\| < 1 \\ 0 & \text{else} \end{cases} \tag{11}$$

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

$$\widetilde{\mathbf{W}} = \alpha_W^* \mathbf{W}_B^* \tag{12}$$

whose gradient can be calculated via standard backward propagation according to Equation (10).

The gradients of the real-valued weights is calculated in Equation (13) via chain rule.

$$\begin{aligned} \frac{\partial l}{\partial \mathbf{W}} &= \frac{\partial l}{\partial \widetilde{\mathbf{W}}} \frac{\partial \widetilde{\mathbf{W}}}{\partial \mathbf{W}} \\ &= \frac{\partial l}{\partial \widetilde{\mathbf{W}}} \frac{\partial (\frac{1}{n} \|\mathbf{W}\|_{l1} sign(\mathbf{W}))}{\partial \mathbf{W}} \\ &= \frac{\partial l}{\partial \widetilde{\mathbf{W}}} (\frac{1}{n} + \alpha_W^* \mathbf{1}_{\|W\|<1}) \end{aligned} \tag{13}$$

where $\frac{\partial l}{\partial \mathbf{W}}$ and $\frac{\partial l}{\partial \widetilde{\mathbf{W}}}$ 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;
$l(Y, Y_{out})$: loss function;
$\mathcal{W}^t$: current real-valued weight;
$L$: number of layers;
$n$: kernel size of layers;
$\eta^t$: current learning rate;
Output: $\mathcal{W}^{t+1}$: updated real-valued weight; $\eta^{t+1}$: updated learning rate.
1. Forward Process:
for $k = 1$ to $L$ do
    $\mathcal{B}_k^t = \mathbf{BinarizeWeight}(\mathcal{W}_k^t)$
    $\mathcal{T}_{k+1} = \mathbf{BinarizeInput}(\mathbf{BatchNorm}(\mathcal{T}_k)) \circledast \mathcal{B}_k^t$
end for
$Y_{out} = \mathcal{T}_{L+1}$
2. Backward Process:
for $k = L$ to 1 do
    $\frac{\partial l}{\partial \mathcal{T}_k} = \mathbf{BinaryBackward}(\frac{\partial l}{\partial \mathcal{T}_{k+1}}, \mathcal{T}_k)$
    $\frac{\partial l}{\partial \mathcal{B}_k^t} = \mathbf{BinaryBackward}(\frac{\partial l}{\partial \mathcal{T}_{k+1}}, \mathcal{B}_k^t)$
    $\frac{\partial l}{\partial \mathcal{W}_k^t} = \frac{1}{n_l}(1 + \|\mathcal{W}_k^t\|_{l1} \mathbf{1}_{\|\mathcal{W}_k^t\|<1}) \frac{\partial l}{\partial \mathcal{B}_k^t}$
end for
3. Update Parameters:
$\mathcal{W}^{t+1}, \eta^{t+1} = \mathbf{Update}(\mathcal{W}^t, \frac{\partial l}{\partial \mathcal{W}^t}, \eta^t)$
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.

- **BatchNorm()**: Batch Normalization function;
- **BinarizeWeight()**: weight binarization function;
- **BinarizeInput()**: input tensor binarization function;
- **BinaryBackward()**: binarized backward function;
- **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 L-th layer firstly. Next, we calculate the gradients of the binarized weights $\frac{\partial l}{\partial \mathcal{B}^t}$ using standard back propagation algorithm. Then, we calculate the gradients of the real-valued weights $\frac{\partial l}{\partial \mathcal{W}^t}$ according to Equation

(13). Lastly we update the parameters and learning rate with an appropriate optimizer, *e.g.*, NAdam [38] in this paper.

3.4 Implementation Details

We present the implementation details in this subsection.

3.4.1 Data Preprocessing. Different from normal image classification problems, the hotspots might be anywhere in the input layout clips so the widely used randomly cropping augumentation method is not adopted in this paper. Note that the input layout clips are all square. The input binary images are simply down-sampled 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. Compared with preprocessing methods adopted in previous works such as DCT-based feature tensor extraction [16] and Maximal Circle Mutual Information (MCMI) scheme [14], our preprocessing method keeps the most spatial information of the original patterns.

3.4.2 Training Hyperparameters. The real-valued kernels are initialized with Xavier initializer [33]. We do not use dropout [39] following the practice in ResNet paper [34].

The optimizer adopted for training the model is NAdam (Nesterov-accelerated Adaptive Moment Estimation) optimizer [38], which combines Adam (Adaptive Moment Estimation) optimizer [40] and NAG (Nesterov Accelerated Gradient) optimizer [41]. We train our model using mini-batches of size 128. We set the initial learning rate as 0.15. The learning rate decay scheme is to exponentially decay each time the validation loss plateaus after an epoch which is used in [42].

3.4.3 Other Details. During the binary convolution, each time the binary convolution kernal $\mathbf{W}_B \in \{-1, +1\}R^{c_{in} \times w_k \times h_k}$ shifts over the input tensor $T_{in} \in \mathbb{R}^{c_{in} \times w_{in} \times h_{in}}$, the scaling factor α_X for the corresponding input vector $\mathbf{X} \in \mathbb{R}^{c_{in} \times w_k \times h_k}$ 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 $|T_{in}|$ which is the scaling factor of the input tensor for every single pixle. For kernels whose shape is not 1×1, these scaling factors needs to be averaged by the shape of the kernel. Next, we construct a matrix $\mathbf{K}$ with shape of $[w_k, h_k]$ whose every element is $\frac{1}{w_k h_k}$. Matrix $\mathbf{K}$ is used to average $|T_{in}|$ locally by convolving $|T_{in}|$ with $\mathbf{K}$ for each channel. The scaling factor for input tensor is α_T:

$$\alpha_T(c) = |T_{in}(c, :, :)| \otimes \mathbf{K} \tag{14}$$

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

$$T_{out} = \alpha_B(sign(T_{in}) \circledast sign(W_B)) \odot \alpha_T \tag{15}$$

where α_B is the scaling factor for the kernel and $\circledast$ represents the binary convolution which is much faster than full-precision convolution.

We use softmax cross entropy as our loss function. The hospot instance has a label of $y_h^* = [0, 1]$ and the non-hotspot instance has a label of $y_n^* = [1, 0]$. Note that the dataset is quite biased which contains much more non-hospot instances than the hospot instances. To further improve the detecting accuracy of our model, the biased learning in [16] is adopted after the model is trained with Algorithm 1. The trained model is finetuned with non-hotspot'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.

Table 2: ICCAD-2012 benchmark statistics

Benchmark	#Train HS	#Train NHS	#Test HS	#Test NHS
ICCAD	1204	17096	2524	13503

Table 3: Performance comparisons with state-of-the-art hotspot detectors

Method	FA#	Runtime (s)	ODST (s)	Accu (%)
SPIE'15 [11]	2919	2672	53112	84.2
ICCAD'16 [14]	4497	1052	70628	97.7
DAC'17 [16]	3413	482	59402	98.2
Ours	**2787**	**60**	**52970**	**99.2**

4 EXPERIMENTAL RESULTS

All our training and testing code is built on MXNet [43] referring to the implementation in [44]. We train and test our model on a machine with a 4-core Intel CPU and a Nvidia GTX 1060 GPU.

Following the practice in [16], we merge all the patterns of the ICCAD 2012 contest to verify the scalability of our model. The statistics of the dataset is listed in Table 2. For training data, columns "# Training HS" and "# Training NHS" show the numbers of hotspots and non-hotspots in training set respectively. For testing data, columns "# Testing HS" and "# Testing NHS" give the numbers of hotspots and non-hotspots in testing set.

The experimental results of our model and three state-of-the-art works [11, 14, 16] are shown in table 3. In the experiment, four metrics, i.e., false alarm (column "FA#"), evaluating time (column "Runtime"), overall detection simulation time (column "ODST") and detecting accuracy (column "Accu") are used to evaluate the performance of the hotspot detectors. Our model and [16]'s model is accelerated with a middle-end graphic card Nvidia GTX1060. Following [45], we set the lithography simulation time per instance t_{ls} in Equation (3) as 10 seconds to calculate the ODST.

The experimental results show that our model achieves the best performance in the ICCAD-2012 testcase. The detecting accuracy of our model is 99.2% compared to 84.2% of [11], 97.7% of [14] and 98.2% of [16]. We also get the fewest 2787 false alarms compared to 2919 of [11], 4497 of [14] and 3413 of [16]. The evaluating time of our model is 60s which is 8 times faster than [16]'s deep learning-based model running on the same platform. The ODST of our model is the lowest among the four listed methods as well.

5 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 paper, considering the binary characteristic of the lithography layout, we propose a BNN-based architecture to address this problem. The down-sampled 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 Contest benchmarks show that our architecture outperforms previous hotspot detectors and achieves an 8x speedup over the best deep learning-based solution. Note that BNNs are more compatible with digital circuits than traditional CNNs. Thus the more efficient hardware-accelerated hotspot detectors are expected.

ACKNOWLEDGEMENTS

This research is supported partly by National Key Research and Development Program of China (2016YFB0201304), partly by National Natural Science Foundation of China (NSFC) research projects 61822402, 61774045, 61574046 and 61574044, partly by the Research Grants Council of Hong Kong SAR (Project No. CUHK24209017).

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 *ICCAD*, 2006.
[2] Y. T. Yu et al. Accurate process-hotspot detection using critical design rule extraction. In *DAC*, pages 1167–1172, 2012.
[3] A. B. Kahng et al. Fast dual graph based hotspot detection. In *SPIE*, 2006.
[4] Wan-Yu Wen, Jin-Cheng Li, Sheng-Yuan Lin, Jing-Yi Chen, and Shih-Chieh Chang. A fuzzy-matching model with grid reduction for lithography hotspot detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(11):1671–1680, 2014.
[5] Fan Yang, Subarna Sinha, Charles C Chiang, Xuan Zeng, and Dian Zhou. Improved tangent space-based distance metric for lithographic hotspot classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(9):1545–1556, 2017.
[6] J.-Y. Wuu et al. Efficient approach to early detection of lithographic hotspots using machine learning systems and pattern matching. In *SPIE*, 2012.
[7] D. Ding et al. Efficient prediction of ic manufacturing hotspots with a unified meta-classification formulation. In *ASP-DAC*, pages 263–270, 2012.
[8] Duo Ding and David Z.Pan. Machine learning based lithographic hotspot detection with critical feature extraction and classifications. In *International conference on integrated circuit design technology*, 2009.
[9] Duo Ding, Andres J. Torres, Fedor G. Pikus, and David Z. Pan. High performance lithographic hotspot detection using hierarchically refined machine learning. In *ASPDAC*, 2011.
[10] Y. T. Yu et al. Machine-learning-based hotspot detection using topological classification and critical feature extraction. In *DAC*, 2013.
[11] Tetsuaki Matsunawa, Jhih-Rong Gao, Bei Yu, and David Z Pan. A new lithography hotspot detection framework based on adaboost classifier and simplified feature extraction. In *Design-Process-Technology Co-optimization for Manufacturability IX*, volume 9427, page 94270S. International Society for Optics and Photonics, 2015.
[12] Duo Ding, Bei Yu, Joydeep Ghosh, and David Z Pan. Epic: Efficient prediction of ic manufacturing hotspots with a unified meta-classification formulation. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 263–270. IEEE, 2012.
[13] Dragoljub Gagi Drmanac, Frank Liu, and Li-C Wang. Predicting variability in nanoscale lithography processes. In *Proceedings of the 46th Annual Design Automation Conference*, pages 545–550. ACM, 2009.
[14] Hang Zhang, Bei Yu, and Evangeline FY Young. Enabling online learning in lithography hotspot detection with information-theoretic feature optimization. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 47. ACM, 2016.
[15] Tetsuaki Matsunawa, Bei Yu, and David Z Pan. Optical proximity correction with hierarchical bayes model. In *Optical Microlithography XXVIII*, volume 9426, page 94260X. International Society for Optics and Photonics, 2015.
[16] Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, Bei Yu, and Evangeline FY Young. Layout hotspot detection with feature tensor generation and deep biased learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
[17] J Andres Torres. Iccad-2012 cad contest in fuzzy pattern matching for physical verification and benchmark suite. In *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, pages 349–350. IEEE, 2012.
[18] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
[19] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.
[20] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *International Conference on Machine Learning*, pages 584–592, 2014.
[21] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, volume 1, page 4. Citeseer, 2011.
[22] Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pages 1–6. IEEE, 2014.
[23] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.
[24] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pages 963–971, 2014.
[25] M Courbariaux and Y Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or- 1. arxiv: 1602.02830, 2016, 2017.
[26] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.
[27] Minje Kim and Paris Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.
[28] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.
[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
[30] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 15–24. ACM, 2017.
[31] Li Jiao, Cheng Luo, Wei Cao, Xuegong Zhou, and Lingli Wang. Accelerating low bit-width convolutional neural networks with embedded fpga. In *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, pages 1–4. IEEE, 2017.
[32] Duncan JM Moss, Eriko Nurvitadhi, Jaewoong Sim, Asit Mishra, Debbie Marr, Suchit Subhaschandra, and Philip HW Leong. High performance binary neural networks on the xeon+ fpga platform. In *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*, pages 1–4. IEEE, 2017.
[33] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.
[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.
[37] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.
[38] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.
[39] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
[40] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
[41] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269, pages 543–547, 1983.
[42] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
[43] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.
[44] Haojin Yang, Martin Fritzsche, Christian Bartz, and Christoph Meinel. Bmxnet: An open-source binary neural network implementation based on mxnet. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 1209–1212. ACM, 2017.
[45] Shayak Banerjee, Zhuo Li, and Sani R Nassif. Iccad-2013 cad contest in mask optimization and benchmark suite. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pages 271–274. IEEE, 2013.