Layout Hotspot Detection with Feature Tensor Generation and Deep Biased Learning

Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, Bei Yu, and Evangeline F. Y. Young

Abstract—Detecting layout hotspots is a key step in the physical verification flow. Although machine learning solutions show benefits over lithography simulation and pattern matching-based methods, it is still hard to select a proper model for large scale problems and inevitably, performance degradation occurs. To overcome these issues, in this paper we develop a deep learning framework for high performance and large scale hotspot detection. First, we use feature tensor generation to extract representative layout features that fit well with convolutional neural networks while keeping the spatial relationship of the original layout pattern with minimal information loss. Second, we propose a biased learning algorithm to train the convolutional neural network to further improve detection accuracy with small false alarm penalties. In addition, to simplify the training procedure and seek a better trade-off between accuracy and false alarms, we extend the original biased learning to a batch biased learning algorithm. Experimental results show that our framework outperforms previous machine learning-based hotspot detectors in both ICCAD 2012 Contest benchmarks and large scale industrial benchmarks. Source code and trained models are available at https://github.com/phdyang007/dlhsd.

I. Introduction

As transistor feature size enters the nanometer era, manufacturing yield is drastically affected by lithographic process variations caused by the limitations of the conventional 193nm wavelength lithography system. Even with various resolution enhancement techniques (RETs), manufacturing defects are still likely to happen for some sensitive layout patterns, referred to as hotspots, as shown in Fig. 1. Thus hotspot detection during physical verification stage is very important. The conventional hotspot detection flow includes optical proximity correction and lithography simulation on multiple process windows, which has a high accuracy but suffers from runtime overhead issues.

To quickly and correctly recognize hotspots during physical verification, two major methodologies were heavily developed: pattern matching [1]–[3] and machine learning [4]–[9]. In the pattern matching solutions, [1] facilitated the verification flow by integrating the density-based layout encoding, principle components analysis (PCA) and customized city-block distance. Yu et al. [2] developed a design rule-based pattern matching method to recognize hotspots. In [3], an improved tangent space-based distance metric was proposed to perform hotspot pattern analysis and classification. Although pattern matching is a direct and fast method to detect layout characteristics, it has a high error rate for unknown patterns. On the other hand, machine learning techniques are capable of learning hidden relations between layout patterns and their defect characteristics, and can greatly improve detection accuracy. Drmanac et al. [9] proposed a histogram-based layout representation and an unsupervised support vector machine (SVM) model to predict the variability of the lithography process. [7] incorporated hierarchical artificial neural networks (ANN) and SVM models to reduce the false alarm rates. Ding et al. [8] constructed a meta classifier with basic pattern matching and machine learning classifiers to achieve better detection performance. [4] presented a hotspot classification flow with a multi-kernel support vector machine and critical feature extraction. In [5], Adaboost and decision tree are adopted for fast hotspot detection. Very recently, Zhang et al. [6] achieved tremendous performance improvements on the ICCAD Contest 2012 benchmark suite [10] by applying an optimized concentric circle sampling (CCS) feature [11] and an online learning scheme. However, there are several aspects that previous works do not take into account, especially when targeting a very large scale problem size. (1) Scalability: As integrated circuits develop to an ultra large scale, VLSI layout becomes more and more complicated and traditional machine learning techniques do not satisfy the scalability requirements for printability estimation of a large scale layout. That is, it may be hard for machine learning techniques to correctly

Fig. 1: (a) Hotspot and (b) non-hotspot examples.

The preliminary version has been presented at the Design Automation Conference (DAC) in 2017. This work is supported in part by The Research Grants Council of Hong Kong SAR (Project No. CUHK24209017).

H. Yang, Y. Ma, B. Yu and E. F. Y. Young are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, NT, Hong Kong.

J. Su and Y. Zou are with ASML Brion Inc., CA, USA.

model the characteristics of a large amount of layout patterns. (2) Feature Representation: The state-of-the-art layout feature extraction approaches, including density [5] and CCS [11], inevitably suffer from spatial information loss, because extracted feature elements are flattened into 1-D vectors, and ignore potential spatial relations. To overcome the conventional machine learning methodology limitations and issues, in this paper we develop a deep learning-based framework targeting high performance and large scale hotspot detection. Because of the automatic feature learning technique and highly nonlinear neural networks, deep learning is highly successful in image classification tasks [12], [13].

Several attempts were made to detect layout hotspots using deep neural networks. [14]–[18] demonstrated that ordinary convolutional neural networks can offer promising hotspot detection results. To make the deep neural networks more suitable and efficient for layout hotspot detection tasks, we consider the following two aspects. First, because layout hotspots are related to light diffraction, the input clips of a hotspot detector usually have a resolution over 1000×1000, which is much larger than the image size (e.g. $\sim 200 \times 200$) in traditional object recognition tasks. As a result, the corresponding neural networks are not computational and storage efficient. Inspired by the feature map in deep neural networks, we utilize the feature tensor concept, i.e., a multi-dimensional representation of an original layout pattern, that performs compression on input layout images to facilitate learning while keeping the spatial relationship to avoid drastic information loss. Feature tensor compatibility with convolutional neural networks makes our framework more efficient for large scale layout patterns. Second, a good hotspot detector is expected to have a satisfactory trade-off between detection accuracy and false alarms. Note that in the physical verification flow, a missing hotspot region means a possible failure while a false alarm simply induces additional lithographic simulation time. Therefore, hotspot detection accuracy plays a more important role when balancing the trade-off. In our preliminary work [19], we developed a bias learning technique that is easily embedded into traditional neural network training procedures and achieve high detection accuracy with a low false alarm penalty. However, the biased learning requires multiple rounds of end-to-end training and manually determined bias terms, which result in a complex training procedure. To address these concerns, a batch-biased learning algorithm is proposed so that we can train the network with simple back-propagation and single-round mini-batch gradient descent. Because batch-biased learning considers the batch loss of each iteration, we can obtain better trade-offs between accuracy and false alarms. The proposed hotspot detection flow is illustrated in Fig. 2.

The main contributions of this paper are listed as follows.

- A feature tensor extraction method is proposed, where the new feature is compatible with the emerging deep learning structure and can dramatically speed up feed-forward and back-propagation.
- A biased learning technique is embedded into the deep learning framework, that offers significant improvements on hotspot detection accuracy with a minor cost of false alarms.
- We develop a batch-biased learning algorithm to adaptively adjust the biased ground truth with respect to the batch training loss that facilitates the training procedure and provides better trade-offs between accuracy and false alarms.
- Experimental results show that our proposed methods have a great advantage over existing machine learning solutions and achieve a 6.3% accuracy improvements on average for very large scale industrial layouts.

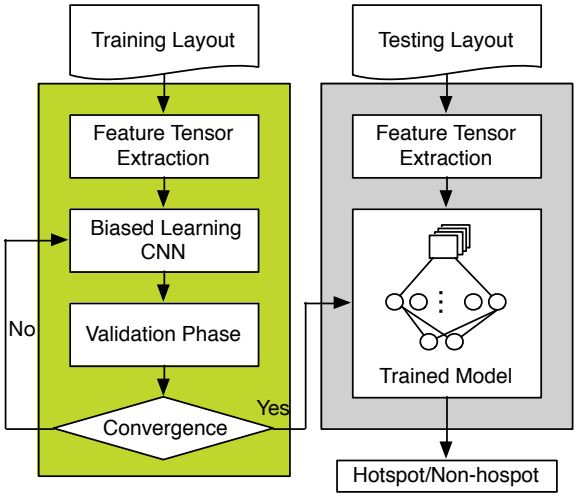


Fig. 2: Proposed hotspot detection flow.

The rest of the paper is organized as follows. Section II introduces basic concepts and problem formulation. Section III and Section IV cover feature tensor generation and biased learning algorithm, respectively. Section V lists the experimental results, followed by the conclusion in Section VI.

II. Preliminaries

In this section, we will introduce some terminologies used in layout hotspot detection. Designed layout patterns are transferred onto silicon wafers through a lithographic process, which involves a lot of variations. Some patterns are sensitive to lithographic process variations and may reduce the manufacturing yield due to potential open or short circuit failures. Layout patterns with a smaller process window and sensitive to process variations are defined as hotspots.

The main objectives of the hotspot detection procedure are identifying as many real hotspots as possible, avoiding incorrect predictions on non-hotspot clips, and reducing runtime. In this paper, we use the following metrics to evaluate performance of a hotspot detector.

Definition 1 (Accuracy [10]). The ratio between the number of correctly predicted hotspot clips and the number of all real hotspot clips.

Definition 2 (False Alarm [10]). The number of non-hotspot clips that are predicted as hotspots by the classifier.

In the actual design flow, detected hotspots (including false positive patterns) are required to perform lithographic simulation, it is reasonable to account for false alarms the overall estimation flow runtime. Therefore, an overall detection and simulation time (ODST) is defined as follows:

Definition 3 (ODST [6]). 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.

With the above definitions, we can formulate the hotspot detection problem as follows:

Problem 1 (Hotspot Detection). Given a set of clips consisting of hotspot and non-hotspot patterns, the objective of hotspot detection is training a classifier that can maximize accuracy and minimize false alarms.

III. Feature Tensor Extraction

Finding a good feature representation is a key procedure in image classification tasks, and so is layout pattern classification. Local density extraction and concentric circle sampling were widely explored in previous hotspot detection and optical proximity correction (OPC) research [11], and were proved to be efficient on hotspot detection tasks because of the embedded lithographic prior knowledge. It is notable that layout hotspots are associated with light diffraction, therefore whether a layout pattern contains hotspots is not only determined by the pattern itself, but is also affected by the surrounding patterns. Therefore, to analyze clip characteristics, we mush be aware of the spatial relations of its local regions. However, all of these existing features finally are flattened into one dimensional vectors that limit hotspot detection accuracy due to a large amount of spatial information loss.

To address the above issue, we propose a feature tensor extraction method that provides a lower scale representation of the original clips while keeping the spatial information of the clips. After feature tensor extraction, each layout image $\mathbf{I}$ is converted into a hyper-image (image with a customized number of channels) $\mathbf{F}$ with the following properties: (1) size of each channel is much smaller than $\mathbf{I}$ and (2) an approximation of $\mathbf{I}$ can be recovered from $\mathbf{F}$.

Spectral analysis of mask patterns for wafer clustering was recently explored in literature [20], [21] and achieved good clustering performance. Inspired by that work, we express the sub-region as a finite combination of different frequency components. High sparsity of the discrete cosine transform (DCT) makes it preferable over other frequency representations in terms of spectral feature extraction, and it is consistent with the expected properties of the feature tensor.

To sum up, the process of feature tensor generation contains the following steps.

Step 1: Divide each layout clip into $n \times n$ sub-regions, then obtain feature representations of all sub-regions for multi-level perceptions of layout clips.

Step 2: Convert each sub-region of the layout clip $\mathbf{I}_{i,j}$ $(i, j = 0, 1, \ldots, n-1)$ into a frequency domain:

$$\mathbf{D}_{i,j}(m,n) = \sum_{x=0}^{B}\sum_{y=0}^{B} \mathbf{I}_{i,j}(x,y) \cos[\frac{\pi}{B}(x+\frac{1}{2})m] \cos[\frac{\pi}{B}(y+\frac{1}{2})n],$$

where $B = \frac{N}{n}$ is sub-region size, (x, y) and (m, n) are original layout image and frequency domain indexes respectively. Particularly, the left-upper side of DCT coefficients in each block correspond to low frequency components, that contain high density information, as depicted in Fig. 3.

Step 3: Flatten $\mathbf{D}_{i,j}$s into vectors in Zig-Zag form [22] with the larger index being higher frequency coefficients as follows.

$$\mathbf{C}^*_{i,j} = [\mathbf{D}_{i,j}(0,0), \mathbf{D}_{i,j}(0,1), \mathbf{D}_{i,j}(1,0), ..., \mathbf{D}_{i,j}(B,B)]^\intercal. \quad (1)$$

Step 4: Pick the first $k \ll B \times B$ elements of each $\mathbf{C}^*_{i,j}$,

$$\mathbf{C}_{i,j} = \mathbf{C}^*_{i,j}[:k], \quad (2)$$

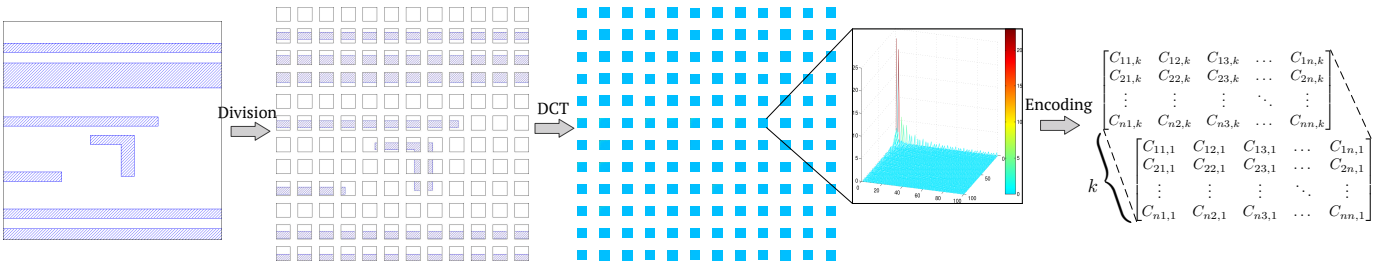


Fig. 3: Feature Tensor generation example ($n = 12$). The original clip (1200×1200 nm^2) is divided into 12×12 blocks and each block is converted to a 100×100 image representing a 100×100 nm^2 sub region of the original clip. Feature tensor is then obtained by encoding on first k DCT coefficients of each block.

and combine $\mathbf{C}_{i,j}, i, j \in \{0, 1, ..., n-1\}$ with their spatial relationships unchanged. Finally, the feature tensor is given as follows:

$$\mathbf{F} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} & \mathbf{C}_{13} & \dots & \mathbf{C}_{1n} \\ \mathbf{C}_{21} & \mathbf{C}_{22} & \mathbf{C}_{23} & \dots & \mathbf{C}_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_{n1} & \mathbf{C}_{n2} & \mathbf{C}_{n3} & \dots & \mathbf{C}_{nn} \end{bmatrix}, \tag{3}$$

where $\mathbf{F} \in \mathbb{R}^{n \times n \times k}$. By reversing above procedure, an original clip can be recovered from an extracted feature tensor.

The nature of discrete cosine transform ensures that high frequency coefficients are near zero. As shown in Fig. 3, large responses only present at the entries with smaller indexes, i.e. low frequency regions. Therefore, most information is kept even when a large amount of elements in $\mathbf{C}^*_{i,j}$ are dropped.

The feature tensor also has the following advantages when applied in neural networks: (1) Highly compatible with the data packet transference in convolutional neural networks and (2) forward propagation time is significantly reduced when compared with using an original layout image as input, because the scale of the neural network is reduced with the smaller input size.

IV. Training The Neural Networks

This section discusses the the convolutional neural network details. First, we introduce the basis and the architecture of convolutional neural networks. Then, we present a customized training procedure that looks for better trade-offs between accuracy and false alarms. Finally, we list additional training and testing strategies.

A. Convolutional Neural Network Architecture

To address the weak scalability of traditional machine learning techniques, we introduce the convolutional neural network (CNN) as preferred classifier. CNN is built with several convolution stages and fully connected layers, where convolution stages perform feature abstraction and fully connected (FC) layers generate the probability of testing instances drawn from each category (Fig. 4).

In this paper, our convolutional neural network has two convolution stages followed by two fully connected layers, and each convolution stage consists of two convolution layers, a ReLU layer and a max-pooling layer. In each convolution, a set of kernels perform convolution on a tensor $\mathbf{F}$ as follows:

$$\mathbf{F} \otimes \mathbf{K}(j,k) = \sum_{i=1}^{c} \sum_{m_0=1}^{m} \sum_{n_0=1}^{m} \mathbf{F}(i, j-m_0, k-n_0)\mathbf{K}(i, m_0, n_0), \tag{4}$$

where $\mathbf{F} \in \mathbb{R}^{c \times n \times n}$, and kernel $\mathbf{K} \in \mathbb{R}^{c \times m \times m}$. In this work, the convolution kernel size is set to 3×3 and the numbers of output feature maps in two convolution stages are 16 and 32 respectively. ReLU is an element-wise operation that follows each convolution layer as a replacement of the traditional sigmoid activation function. As shown in Equation (5), ReLU ensures that the network is nonlinear and sparse.

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0. \end{cases} \tag{5}$$

The max-pooling layer performs 2×2 down-sampling on the output of the previous layer and is applied as the output layer of each convolution stage. Following the two convolution stages are two FC layers with output node numbers of 250 and 2, respectively. A 50% dropout is applied on the first FC layer during training to alleviate overfitting. The second FC layer is the output layer of the entire neural network, where two output nodes generate the predicted probabilities of an input instance being hotspot and non-hotspot. Detailed configurations are shown in TABLE I.

TABLE I: Neural Network Configuration.

Layer	Kernel Size	Stride	Output Node #
conv1-1	3	1	$12 \times 12 \times 16$
conv1-2	3	1	$12 \times 12 \times 16$
maxpooling1	2	2	$6 \times 6 \times 16$
conv2-1	3	1	$6 \times 6 \times 32$
conv2-2	3	1	$6 \times 6 \times 32$
maxpooling2	2	2	$3 \times 3 \times 32$
fc1	-	-	250
fc2	-	-	2

B. Mini-batch Gradient Descent

Determining the gradient of each neuron and the parameter updating strategy in the neural network are two key mechanisms in the training procedure. Back-propagation [23] is widely applied to calculate gradients when training large neural networks. Each training instance $\mathbf{F}$ has a corresponding gradient set $\mathcal{G} = \{\mathbf{G}_1, \mathbf{G}_2, ..., \mathbf{G}_v\}$, where each element is a gradient matrix associated with a specific layer and v is the total layer number. All the neural network parameters are then updated with the obtained $\mathcal{G}$.

Stochastic gradient descent (SGD), where each training data instance is randomly presented to the machine learning model, has proved more efficient to train large data sets [24] than conventional batch learning, where a complete training set is presented to the model for each iteration. However, as a data set scales to the ultra large level, e.g. millions of instances, SGD has difficulty to efficiently utilize computation resources. Therefore, it takes a long time for the model to cover every instance in the training set. A compromise approach called mini-batch gradient descent (MGD) [25] can be applied where a group of instances are randomly picked to perform gradient descent. Additionally, MGD is naturally compatible with the online method allowing it to facilitate convergence and avoid large storage requirements for training ultra large instances.

However, for large nonlinear neural networks, back-propagation and MGD do not have rigorous convergence criteria. A fraction, empirically 25%, of training instances

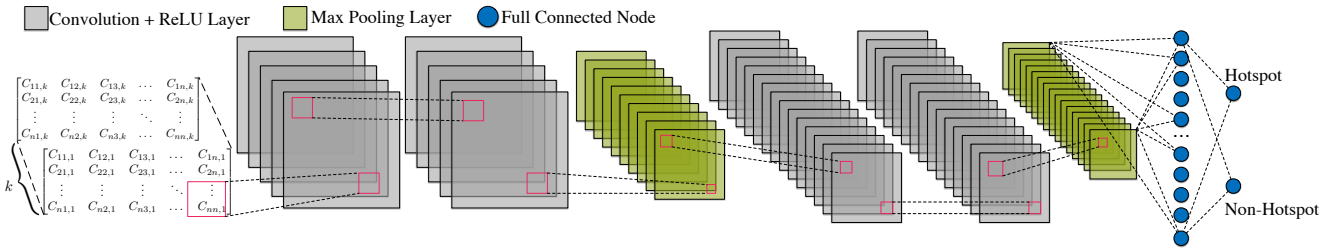


Fig. 4: The proposed convolutional neural network structure.

(validation set) is separated out and is never shown to the network for weight updating. We then test the trained model on the validation set every few iterations. When the test performance on the validation set does not show much variation or starts getting worse, the training procedure is considered to be converged. To make sure MGD reaches a more granular solution, we reduce the learning rate along with the training process.

Algorithm 1 Mini-batch Gradient Descent (MGD)

1: function MGD($\mathbf{W}$, λ, α, k, $\mathbf{y}_h^*$, $\mathbf{y}_n^*$)
2: Initialize parameters $j \leftarrow 0$, $\mathbf{W} > \mathbf{0}$;
3: while not stop condition do
4: $j \leftarrow j + 1$;
5: Sample m training instances $\{\mathbf{F}_1, \mathbf{F}_2, ..., \mathbf{F}_m\}$;
6: for $i \leftarrow 1, 2, ..., m$ do
7: $\mathcal{G}_i \leftarrow \text{backprop}(\mathbf{F}_i)$;
8: end for
9: Calculate gradient $\bar{\mathcal{G}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathcal{G}_i$;
10: Update weight $\mathbf{W} \leftarrow \mathbf{W} - \lambda\bar{\mathcal{G}}$;
11: if $j \mod k = 0$ then
12: $\lambda \leftarrow \alpha\lambda$, $j \leftarrow 0$;
13: end if
14: end while
15: return Trained model f;
16: end function

The details of MGD with learning rate decay are shown in Algorithm 1, where $\mathbf{W}$ is the neuron weights, λ is the learning rate, $\alpha \in (0, 1)$ is the decay factor, k is the decay step, $\mathbf{y}_h^*$ is the hotspot ground truth and $\mathbf{y}_n^*$ is the non-hotspot ground truth. MGD can be regarded as a function that returns the model with the best performance on the validation set. Indicator j will count up through iterations (line 4), and in each iteration, m training instances $\{\mathbf{F}_1, \mathbf{F}_2, ..., \mathbf{F}_m\}$ are randomly sampled from the training set (line 5). Gradients of these training instances ($\mathcal{G}_i$) are calculated using back-propagation (lines 6–8). Then neuron weights $\mathbf{W}$ are updated by subtracting the average gradient of sampled instances $\lambda\bar{\mathcal{G}}$ scaled by learning rate γ (line 14). When j is an integer multiple of k, λ is reduced to $\alpha\lambda$, i.e. the learning rate decays every k iterations (lines 11–13). At the end of MGD, a trained model that has satisfactory performance on validation set will be returned (line 15).

Although SGD has shown an advantage in emerging machine learning techniques, it cannot fully utilize GPU resources. MGD, on the other hand, is more compatible with parallel computing and can speed up training procedures. To evaluate the efficiency of MGD, we train the neural network on the ICCAD benchmark using MGD and SGD separately with the configuration in TABLE III. The training procedure is shown in Fig. 5, where the X-axis is the elapsed time (s) in the training procedure and the Y-axis is the cross-entropy loss on the validation set. The curve shows that MGD behaves much more stably than SGD, which indicates the neural network with the MGD learning strategy is more efficient and effective than conventional SGD.

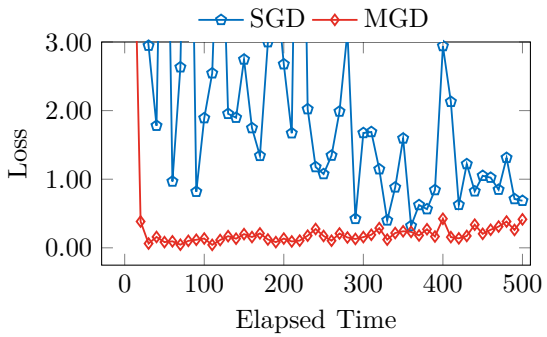


Fig. 5: Stochastic gradient descent (SGD) v.s. Mini-batch gradient descent (MGD).

C. Learning Towards Biased Target

Softmax cross entropy can provide speedup for back-propagation while attaining comparable network performance with the mean square error [25]. In a n-category classification task, the instance that belongs to class c has a ground truth $\mathbf{y}^* \in \mathbb{R}^n$ where $\mathbf{y}^*$ has the property that $\mathbf{y}^*(c) = 1$ and $\sum_{i=1}^{n} \mathbf{y}^*(i) = 1$. Each entry of $\mathbf{y}^*$ is regarded as the probability of the instance drawn from each category. The predicted label vector $\mathbf{y}$ by classifier is defined similarly.

In the task of hotspot detection, $\mathbf{y}^* = \mathbf{y}_n^* = [1, 0]$ and $\mathbf{y}^* = \mathbf{y}_h^* = [0, 1]$ are assigned as the ground truths for non-hotspot and hotspot. To generate loss with respect to ground truth, score $\mathbf{x} = [x_h, x_n]$ predicted by the neural network is scaled to a $(0, 1)$ interval by the softmax function shown in Equation (6).

$$\mathbf{y}(0) = \frac{\exp x_h}{\exp x_h + \exp x_n}, \quad \mathbf{y}(1) = \frac{\exp x_n}{\exp x_h + \exp x_n}, \quad (6)$$

and then, cross-entropy loss is calculated as follows,

$$l(\mathbf{y}, \mathbf{y}^*) = -(\mathbf{y}^*(0) \log \mathbf{y}(0) + \mathbf{y}^*(1) \log \mathbf{y}(1)). \quad (7)$$

In case of the situation we need to calculate $\log 0$, we define,

$$\lim_{x \to 0} \quad x \log x = 0. \quad (8)$$

Because each entry of softmax label $\mathbf{y}_i$ is the probability of given instance $\mathbf{F}_i$ being non-hotspot $\mathcal{N}$ and hotspot $\mathcal{H}$, we have,

$$\mathbf{F} \in \begin{cases} \mathcal{N}, & \text{if } \mathbf{y}(0) > 0.5, \\ \mathcal{H}, & \text{if } \mathbf{y}(1) > 0.5. \end{cases} \quad (9)$$

$$\mathbf{y}(0) + \mathbf{y}(1) = 1. \quad (10)$$

To improve the hotspot detection accuracy, a straightforward approach is shifting the decision boundary, as shown in Equation (11).

$$\mathbf{F} \in \begin{cases} \mathcal{N}, & \text{if } \mathbf{y}(0) > 0.5 + \lambda, \\ \mathcal{H}, & \text{if } \mathbf{y}(1) > 0.5 - \lambda, \end{cases} \quad (11)$$

where $\lambda > 0$ is the shifting level. However, this method can cause a large increase in false alarms.

The conventional training procedure applies ground truth label $\mathbf{y}_n^* = [1, 0]$ for non-hotspot instances and $\mathbf{y}_h^* = [0, 1]$ for hotspot instances. For non hotspot instances, the classifier is trained towards $\mathbf{y}_n^*$. If the training procedure meets the stop criteria, then for most non-hotspot clips, f will predict them to have a high probability, close to 1, to be non-hotspots. However, as can be seen in Equation (9), the instance would be predicted as a non-hotspot as long as the predicted probability is greater than 0.5. Thus, to some extent, the classifier is too confident as expected.

Intuitively, a too confident classifier is not necessary to give a good prediction performance and on the contrary, may induce more training pressure or even overfitting problem. We exemplify the case using a linear classifier. As illustrated in Fig. 6, a more confident classifier results in an optimized loss, but cannot guarantee higher classification accuracy. Therefore, an assumption can be made that the hotspot detection accuracy can be further improved by sacrificing the training loss of non-hotspot samples. Meanwhile, the induced false alarm penalties are expected to be lower than directly shifting the decision boundary.

Assumption 1. Given a pre-trained convolutional neural network model with ground truth $\mathbf{y}_n^* = [1, 0]$ and $\mathbf{y}_h^* = [0, 1]$ and hotspot detection accuracy a on a given test set. Fine-tune the network with $\mathbf{y}_n^\epsilon = [1 - \epsilon, \epsilon]$, $\epsilon \in [0, 0.5)$, we can obtain the hotspot detection accuracy a' and false alarm fs' of the new model. Shifting the decision boundary of the original model to reach the detection accuracy a', the corresponding false alarm is fs''. We have $a' \geq a$ and $fs'' \geq fs'$.

Because it is hard to have a solid proof of the above assumption due to the uncertainty of the deep neural networks, we conduct a sketch explanation of $a' \geq a$ by analyzing the training actions of the fully connected layer.

Proof. Consider a trained classifier f with $O_{l-1}(\mathbf{F}_i)$ as the output of the second last layer, and the neurons have weight $\mathbf{W}_l$. Then the output can be expressed as follows:

$$\mathbf{x}_i = \mathbf{W}_l^\intercal O_{l-1}(\mathbf{F}_i), \quad (12)$$

where $\mathbf{W}_l$ is learned towards the target $\mathbf{y}_n^* = [1, 0]$. We will show in Equations (29)–(32) that training gradients of non-hotspot instances with biased labels are smaller than those without bias. Considering the fine-tune process with $\mathbf{y}_n^\epsilon = [1 - \epsilon, \epsilon]$, training instances with predicted probability less than $1 - \epsilon$ are only supposed to generate minor gradient to update the network, since they can not even make prominent difference with the target $\mathbf{y}_n$ when the stopping criteria is met. For those non-hotspot instances that have predicted probability in $(1 - \epsilon, 1)$ (confident instances), neuron weights are updated along the gradient generated by confident instances.

Also, gradient vanishing theory [26] indicates a later layer in the neural network learns faster, therefore within a limited number of iterations, updates of the front layer can be ignored. We assume the layers before O_{l-1} are fixed for some iterations. Let the updated weight for the output layer be $\mathbf{W}_l'$, and the current output for confident instance $\mathbf{F}_c$ is,

$$\mathbf{x}_c' = \mathbf{W}_l'^\intercal O_{l-1}(\mathbf{F}_c). \quad (13)$$

Before adjusting the ground truth, we have

$$\mathbf{x}_c = \mathbf{W}_l^\intercal O_{l-1}(\mathbf{F}_c). \quad (14)$$

Note that $\mathbf{x} \in \mathbb{R}^2$, therefore $\mathbf{W}_l$ has two columns $\mathbf{W}_{l,1}$ and $\mathbf{W}_{l,2}$. Similarly, $\mathbf{W}_l' = [\mathbf{W}_{l,1}', \mathbf{W}_{l,2}']$. We define $\mathbf{w}$ and $\mathbf{w}'$ as follows:

$$\mathbf{w} = \mathbf{W}_{l,1} - \mathbf{W}_{l,2}, \quad \mathbf{w}' = \mathbf{W}_{l,1}' - \mathbf{W}_{l,2}'. \quad (15)$$

Here $\mathbf{w}'$ is updated from $\mathbf{w}$ through gradient descent:

$$\begin{aligned} \mathbf{w}' &= \mathbf{w} - \alpha \nabla_{\mathbf{w}} (\mathbf{x}_c(0) - \mathbf{x}_c(1))(\nabla_{\mathbf{x}_c(0)} L_c - \nabla_{\mathbf{x}_c(1)} L_c) \\ &= \mathbf{w} - \alpha O_{l-1}(\mathbf{F}_c)(\nabla_{\mathbf{x}_c(0)} L_c - \nabla_{\mathbf{x}_c(1)} L_c), \end{aligned} \quad (16)$$

where L_c is the cross-entropy loss and $\alpha > 0$ is the learning rate. Besides,

$$L_c = -\mathbf{y}_n^\epsilon(0) \log \mathbf{y}_c(0) - \mathbf{y}_n^\epsilon(1) \log \mathbf{y}_c(1), \quad (17)$$

$$\mathbf{y}_c(0) = \frac{\exp \mathbf{x}_c(0)}{\exp \mathbf{x}_c(0) + \exp \mathbf{x}_c(1)}, \quad (18)$$

$$\mathbf{y}_c(1) = \frac{\exp \mathbf{x}_c(1)}{\exp \mathbf{x}_c(0) + \exp \mathbf{x}_c(1)}. \quad (19)$$

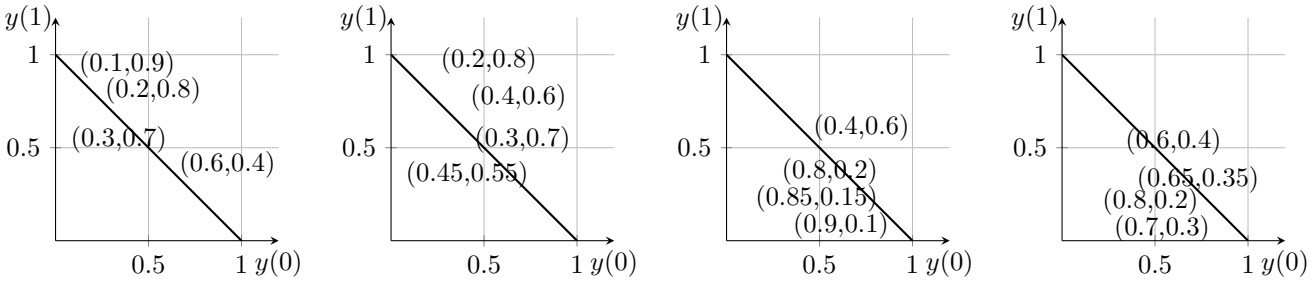
Substitute Equation (18) and Equation (19) into Equation (17),

$$\begin{aligned} L_c = &-\mathbf{y}_n^\epsilon(0)\mathbf{x}_c(0) - \mathbf{y}_n^\epsilon(1)\mathbf{x}_c(1) \\ &+ \log(\exp \mathbf{x}_c(0) + \exp \mathbf{x}_c(1)), \end{aligned} \quad (20)$$

$$\nabla_{\mathbf{x}_c(0)} L_c = -\mathbf{y}_n^\epsilon(0) + \mathbf{y}_c(0), \quad (21)$$

$$\nabla_{\mathbf{x}_c(1)} L_c = -\mathbf{y}_n^\epsilon(1) + \mathbf{y}_c(1). \quad (22)$$

$$\nabla_{\mathbf{x}_c(0)} L_c - \nabla_{\mathbf{x}_c(1)} L_c > 0. \quad (23)$$



(a) acc=0.75, loss=0.69 (b) acc=1, loss=0.73 (c) fa=1, loss=0.61 (d) fa=0, loss=0.66

Fig. 6: Training loss versus accuracy. The loss in (a) and (b) are calculated with respect to the hotspot ground truth. The loss in (c) and (d) are calculated with respect to the non-hotspot ground truth. An optimal loss does not guarantee a better discriminant performance: (a) illustrates a smaller loss than (b) but suffers a lower hotspot prediction accuracy (75% v.s. 100%); (c) illustrates a smaller loss than (b) but suffers a higher false alarm (1 v.s. 0).

For hotspot instances:

$$\begin{aligned}&\mathbf{w}'^{\intercal} O_{l-1}(\mathbf{F}_h)\\ =&\mathbf{w}^{\intercal} O_{l-1}(\mathbf{F}_h) - \alpha O_{l-1}^{\intercal}(\mathbf{F}_c)(\nabla_{\mathbf{x}_c(0)} L_c - \nabla_{\mathbf{x}_c(1)} L_c) O_{l-1}(\mathbf{F}_h).\end{aligned} \tag{24}$$

Because O_{l-1} is ReLU output, we have $O_{l-1}(\mathbf{F}_c) > \mathbf{0}$ and $O_{l-1}(\mathbf{F}_h) > \mathbf{0}$. Therefore,

$$\mathbf{w}^{\intercal} O_{l-1}(\mathbf{F}_h) > \mathbf{w}'^{\intercal} O_{l-1}(\mathbf{F}_h), \tag{25}$$

which indicates that

$$\mathbf{x}_h(0) - \mathbf{x}_h(1) > \mathbf{x}'_h(0) - \mathbf{x}'_h(1), \tag{26}$$

$$\Rightarrow \quad \frac{\exp \mathbf{x}_h(1)}{\exp \mathbf{x}_h(0) + \exp \mathbf{x}_h(1)} < \frac{\exp \mathbf{x}'_h(1)}{\exp \mathbf{x}'_h(0) + \exp \mathbf{x}'_h(1)}. \tag{27}$$

Therefore, the predicted probability of hotspot instances being real hotspots is expected to be greater, and the classifier is more confident about those wrongly detected patterns that have predict probability around 0.5. In other words, $a' \geq a$. □

From a physical point of view, the biased term ϵ can be regarded as a force "dragging" the decision boundary closer to non-hotspot instances. However, in the space defined by the pre-trained model, all the instances are located in different locations and have different distances to the decision boundary. For any non-hotspot instance, the loss with and without the biased term ϵ is given by Equations (28) and (29) respectively.

$$l = -\log \mathbf{y}(0), \tag{28}$$

$$\begin{aligned} l_b &= -(1-\epsilon)\log \mathbf{y}(0) - \epsilon \log \mathbf{y}(1) \\ &= -(1-\epsilon)\log \mathbf{y}(0) - \epsilon \log(1 - \mathbf{y}(0)). \end{aligned} \tag{29}$$

The training speed of the neural network is determined by the gradient of the loss with respect to neuron weights. Because the relationships between the prediction score $\mathbf{y}$ and neuron weights are the same regardless of the loss function, the training speeds of the two cases are determined by the following equations:

$$\frac{\partial l}{\partial \mathbf{y}(0)} = -\frac{1}{\mathbf{y}(0)}, \tag{30}$$

$$\frac{\partial l_b}{\partial \mathbf{y}(0)} = \frac{\epsilon + \mathbf{y}(0) - 1}{\mathbf{y}(0)(1 - \mathbf{y}(0))}. \tag{31}$$

Observe that when $\mathbf{y}(0) \leq 0.5$, i.e. the network makes an incorrect prediction,

$$\left|\frac{\partial l}{\partial \mathbf{y}(0)}\right| > \left|\frac{\partial l_b}{\partial \mathbf{y}(0)}\right|, \tag{32}$$

which indicates if the bias is directly applied at the random initialized networks, the network with bias updates slower than the network without bias. Therefore, we apply multiple rounds fine-tuning on the pre-trained model instead of directly training the network with bias.

The bias term ϵ cannot be increased without limitations, because at some point, most of the non-hotspot patterns will cross the middle line, where the probability is 0.5, causing a significant increase of false alarms. Because the approach improves the performance of hotspot detection at the cost of confidence on non-hotspots, we call it biased learning. As the uncertainty exists for large CNN, a validation procedure is applied to decide when to stop biased learning. To sum up, biased learning is iteratively carrying out normal MGD with changed non-hotspot ground truths, as shown in Algorithm 2.

Algorithm 2 Biased-learning

Require: ϵ, $\delta\epsilon$, t, $\mathbf{W}$, λ, α, k, $\mathbf{y}_h^*$, $\mathbf{y}_n^*$;
1: $i \leftarrow 0$, $\epsilon \leftarrow 0$, $\mathbf{y}_h^* \leftarrow [0, 1]$;
2: while $i < t$ do
3: $\quad \mathbf{y}_n^* \leftarrow [1 - \epsilon, \epsilon]$;
4: $\quad f_\epsilon \leftarrow \text{MGD}(\mathbf{W}, \lambda, \alpha, k, \mathbf{y}_n^*, \mathbf{y}_h^*)$; ▷ Algorithm 1
5: $\quad i \leftarrow i + 1$, $\epsilon \leftarrow \epsilon + \delta\epsilon$;
6: end while

Here ϵ is the bias, $\delta\epsilon$ represents the bias step, and t is the maximum iteration of biased learning. In biased learning, the hotspot ground truth is fixed at $[0, 1]$ while the non-hotspot truth is $[1 - \epsilon, \epsilon]$. Initially, the normal MGD is

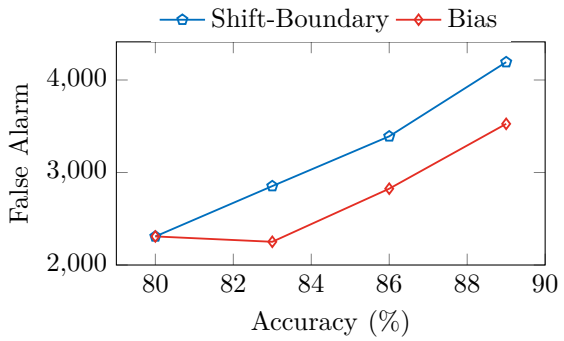


Fig. 7: Bias learning shows a smaller false alarm penalty to obtain the same hotspot detection accuracy.

applied with $\epsilon = 0$ (line 1). After getting the converged model, fine-tune it with ϵ updated by $\epsilon = \epsilon + \delta\epsilon$. Repeat the procedure until the framework reaches the maximum bias adjusting time t (lines 2–6).

Assumption 1 shows that the biased learning algorithm can improve hotspot detection accuracy by taking advantage of the ReLU property. Because biased learning is applied through training, the false alarm penalty on the improvement of hotspot accuracy is expected to be limited. Here we evaluate the biased learning algorithm by training the neural network with $\epsilon = 0$ to obtain an initial model and fine-tuning with $\epsilon = 0.1, 0.2, 0.3$. Then we perform boundary shifting on the initial model to achieve the same test accuracy with three fine-tuned models. As shown in Fig. 7, biased learning has 600 less false alarm penalties for the same improvement of hotspot detection accuracy equivalent to saving $6000s$ of ODST consumption, which demonstrates the validity of Assumption 1.

D. Batch Biased Learning

In the biased learning algorithm, we perform fine-tuning on the pre-trained models with a fixed and biased ground truth until meeting the stop condition. From the deduction above, we also notice that it might not be suitable to apply the same biased ground truth on all the non-hotspot instances. Therefore, to dynamically adjust the bias for different instances, we define a bias function as follows:

$$\epsilon(l) = \begin{cases} \frac{1}{1+\exp(\beta l)}, & \text{if } l \leq 0.3, \\ 0, & \text{if } l > 0.3, \end{cases} \tag{33}$$

where l is the training loss of the current instance or batch in terms of the unbiased ground truth and β is a manually determined hyper-parameter that controls how much the bias is affected by the loss. Because the training loss of the instance at the decision boundary is $-\log 0.5 \approx 0.3$, we set the bias function to take effect when $l \leq 0.3$. With the bias function, we can train the neural network in a single-round MGD and obtain a better model performance. Because $\epsilon(l)$ is fixed within each training step, no additional computing effort is required for back-propagation, as indicated by Equation (31).

The training procedure is summarized as Algorithm 3, where β is a hyper-parameter defined in Equation (33). Similar to MGD, we initialize the neural network (line 1) and update the weight until meeting the convergence condition (lines 2–16). Within each iteration, we first sample the same amount of hotspot and non-hotspot instances to make sure the training procedure is balanced (lines 3–4); we then calculate the average loss of non-hotspot instances to obtain the bias level and the biased ground truth (lines 5–6); the gradients of the hotspot and non-hotspot instances are calculated separately (lines 8–9); the rest of the steps are the normal weight update through back-propagation and learning rate decay (lines 11–15).

Algorithm 3 Batch Biased-learning

Require: $\mathbf{W}$, λ, α, k, $\mathbf{y}_h^*$, $\mathbf{y}_n^*$, β;
1: Initialize parameters, $\mathbf{y}_h^* \leftarrow [0, 1]$;
2: while not stop condition do
3: Sample m non-hotspot instances $\{\mathbf{N}_1, \mathbf{N}_2, ..., \mathbf{N}_m\}$;
4: Sample m hotspot instances $\{\mathbf{H}_1, \mathbf{H}_2, ..., \mathbf{H}_m\}$;
5: Calculate average loss of non-hotspot samples l_n with ground truth $[1, 0]$;
6: $\mathbf{y}_n^* \leftarrow [1 - \epsilon(l_n), \epsilon(l_n)]$;
7: for $i \leftarrow 1, 2, ..., m$ do
8: $\mathcal{G}_{h,i} \leftarrow \text{backprop}(\mathbf{H}_i)$;
9: $\mathcal{G}_{n,i} \leftarrow \text{backprop}(\mathbf{N}_i)$;
10: end for
11: Calculate gradient $\bar{\mathcal{G}} \leftarrow \frac{1}{2m} \sum_{i=1}^{m} (\mathcal{G}_{h,i} + \mathcal{G}_{n,i})$;
12: Update weight $\mathbf{W} \leftarrow \mathbf{W} - \lambda \bar{\mathcal{G}}$;
13: if $j \mod k = 0$ then
14: $\lambda \leftarrow \alpha\lambda$, $j \leftarrow 0$;
15: end if
16: end while

E. Data Augmentation and Ensemble Testing

Many studies have shown that data preprocessing provides a greater generalization ability of the deep learning model [12], [16], [27]. Observe that (1) the orientation of a clip does not affect its property under most illumination settings and (2) usually there are more non-hotspot clips than hotspot clips, we include several data augmentation techniques in the training stage accordingly.

- We randomly perform flipping (top-bottom transformation) and mirroring (left-right transformation) on each feature tensor along its last axis. It is easy to derive that the convolution operation is not flipping invariant, therefore random flipping and mirroring will introduce diversity to the dataset and increase the trained model's generalization ability (how well the model fits the testing data [28], [29]).
- We force the number of hotspot and non-hotspot instances to be equal in each mini-batch. As shown in the experiment of [16], a highly imbalanced dataset causes performance degradation. Sampling

TABLE II: Benchmark Statistics

Benchmarks	Training Set		Testing Set		Size/Clip
	HS#	NHS#	HS#	NHS#	(μm^2)
ICCAD	1204	17096	2524	13503	3.6×3.6
Industry0	3629	80299	942	20412	1.2×1.2
Industry1	34281	15635	17157	7801	1.2×1.2
Industry2	15197	48758	7520	24457	1.2×1.2
Industry3	24776	49315	12228	24817	1.2×1.2

equal amount of instances from different categories is expected to benefit both the training progress and the model performance.

To make better use of the flipping variance property, an ensemble method is applied in the testing phase.

- We do prediction on four directions of each clip and take the average of the prediction scores as the final prediction result.

Although ensemble testing will induce additional testing runtime, it offers better model performance.

V. Experimental Results

A. Experimental Setup

We implement our deep biased learning framework in Python with the TensorFlow library [30], and test it on a platform with a Xeon E5 processor and Nvidia Graphic card. To fully evaluate the proposed framework, we employ four test cases. Because the individual test cases in the ICCAD 2012 contest [10] are not large to verify the scalability of our framework, we merge all the $28nm$ patterns into a unified test case ICCAD.

Additionally, we adopt four more complicated industry test cases: Industry0 – Industry3. The details for all test cases are listed in the TABLE II.

Columns "Train HS#" and "Train NHS#" list the total number of hotspots and the total number of non-hotspots in the training set. Columns "Test NHS#" and "Test HS#" list the total number of hotspots and total number of non-hotspots in the testing set. Images in the testing set of ICCAD have a resolution of 3600×3600 which is larger than the images in industry benchmarks (1200×1200). Therefore, during the testing phase, each clip in ICCAD Testing Set is divided into nine 1200×1200 blocks before feeding into the testing flow. Mask images in the four benchmarks are from different OPC stages, and have different complexities, as shown in Fig. 8. Note that classic hotspot detection problems aim to find and revise problematic designs at an early stage of the whole layout verification flow which corresponds to cases ICCAD benchmarks and Industry0 that contains before-OPC patterns and are labeled based on the results of the entire layout verification flow, including OPC and lithography simulation. Industry1-Industry3 are benchmark sets that contain layouts from intermediate OPC results which are labeled hotspot or non-hotspot based on the lithography simulation results of current OPC step. The motivation of introducing intermediate OPCed layouts is to show some potential of embedding efficient hotspot detectors into OPC engines and facilitate the procedure.

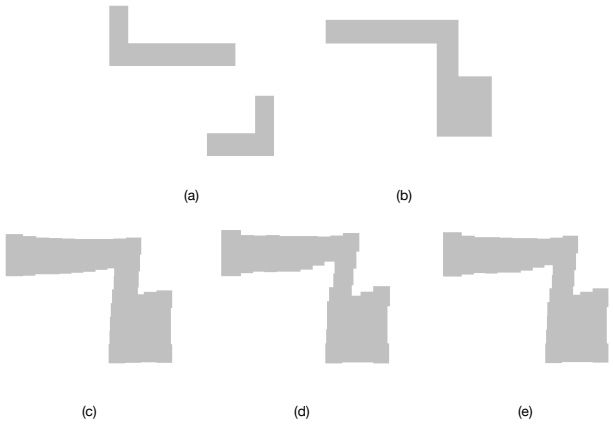


Fig. 8: Benchmark examples. (a) and (b) are before-OPC patterns from ICCAD and Industry0, respectively. (b)–(d) correspond to Industry1-Industry3, which are from intermediate OPC results.. The pattern becomes more complicated after OPC and is more challenging for machine learning based hotspot detectors.

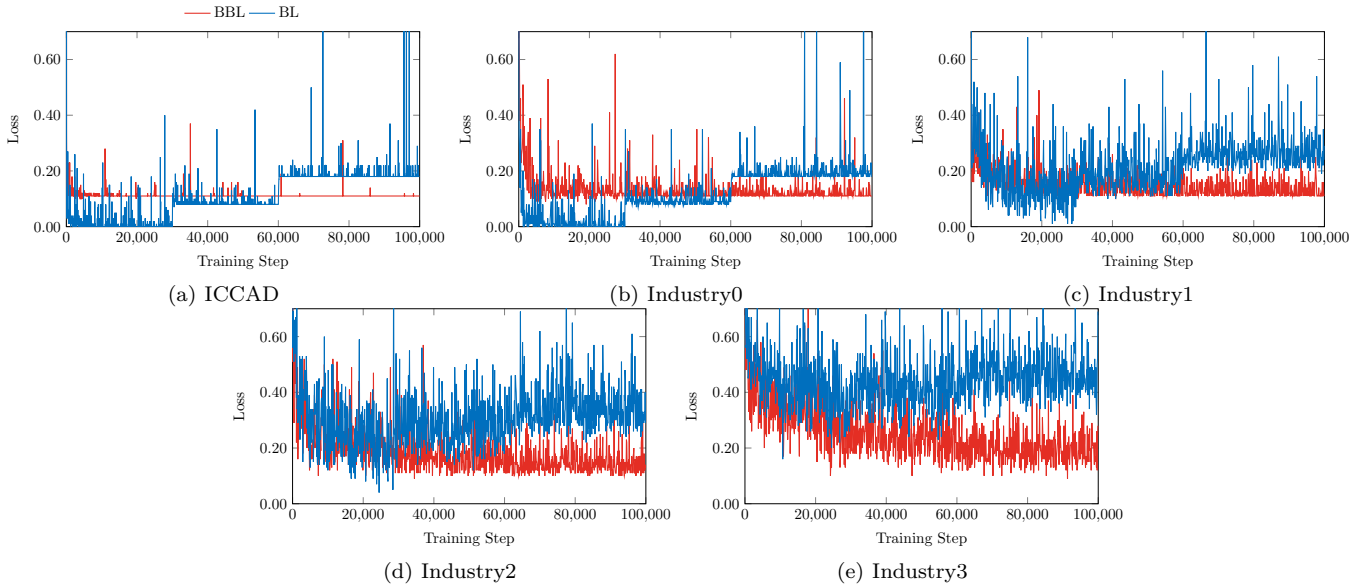
TABLE III: Training Configurations

Configurations	Value
Optimizer	Adam [31]
Initial Learning Rate (λ)	0.001
Learning Rate Decay (α)	0.75
LR Decay Step (k)	10,000
Bias Function Coefficient (β)	6 and 43
Batch Size	32
Feature Tensor Channel	32

B. Model Training

We train five individual models for each benchmark set following Algorithm 3. TABLE III lists the details of the training configurations. "Adam" is an improved optimizer to conduct a gradient descent proposed in [31] that is proved to converge faster. Parameters α and k denote that the learning rate (λ) drops to $\alpha\lambda$ every k steps. β is the coefficient that appears in Equation (33) and controls how much the bias is affected by the loss of a non-hotspot batch, therefore, it also controls the trade-offs between accuracy and false alarms. Here we use $\beta = 6$ for the datasets (ICCAD, Industry0 and Industry1) with more regular patterns and $\beta = 43$ for the datasets (Industry2 and Industry3) with complicated patterns. Because it takes more effort to fit the complicated patterns with the neural networks, we pick a larger β to avoid additional perturbation on the neural networks at early training stages (Fig. 9). Feature Tensor Channel represents the number of remaining elements after dropping high frequency components in the feature tensor extraction procedure.

We visualize the training loss of Batch Biased Learning (BBL) and Biased Learning (BL) of five benchmarks in



(a) ICCAD (b) Industry0 (c) Industry1

(d) Industry2 (e) Industry3

Fig. 9: Visualize training of the batch biased learning (red) and the biased learning (blue). On each benchmarks, batch biased learning exhibits lower loss than the biased learning at convergence.

Fig. 9, where (a), (b), (c), (d) and (e) correspond to ICCAD, Industry0, Industry1, Industry2 and Industry3, respectively. The "Loss" is the average cross-entropy loss given by Equation (7) with respect to the unbiased ground truth. For the ICCAD and Industry0 dataset, the batch biased learning converges quickly within 20,000 steps, while the biased learning requires manual adjustment of the bias and finally converges at a higher loss level. (see the blue curve in Fig. 9(a) and Fig. 9(b)). Also, for the benchmarks Industry1-Industry3, the batch biased learning (red curve) has better convergent result than the biased learning (blue).

C. Model Testing

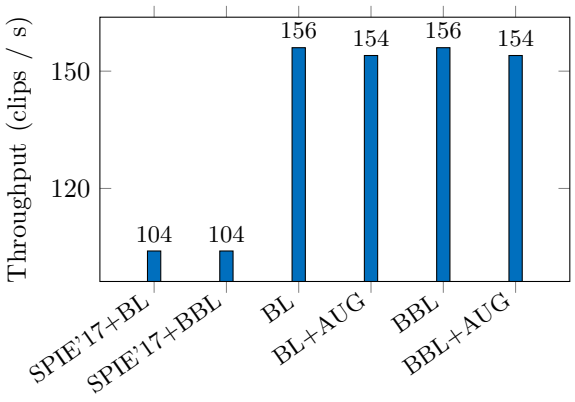


Fig. 10: Throughput comparison of different neural network models.

To evaluate the effectiveness of the batch biased learning, we first compare the test results on four datasets (statistics are listed in TABLE II) with our preliminary results in [19], as shown in TABLE IV and TABLE V. "Accu (%)" and "FA #" denote the hotspot detection accuracy (Definition 1) and the false alarm number (Definition 2). "FA (%)" is the percentage representation of false alarms that are coherent with "FA #". "SPIE'17 [16]+BL" corresponds to the results obtained by inserting biased learning (Algorithm 2) in SPIE'17 [16] neural networks model; "SPIE'17 [16]+BBL" lists the results obtained by embedding batch biased learning (Algorithm 3) in SPIE'17 [16] neural networks model; "BL [19]" corresponds to the original biased learning algorithm (Algorithm 2) that is applied in our preliminary work; Column "BL [19] + AUG" contains the results obtained from the original biased learning and the data augmentation mentioned in section IV-E; "BBL" lists the testing results of the batch biased learning as in Algorithm 3; "BBL + AUG" also includes the data augmentation in the batch biased learning procedure. Note that original SPIE'17 [16] also employs data augmentation on raw layout images.

In this paper, we propose a batch biased learning algorithm that can train the neural network in one round MGD and seek a better trade-off than the biased learning algorithm. TABLE IV lists the testing results of target layouts ICCAD and Industry0, which show that BBL surpasses BL on both average detection accuracy and false alarm. With the aid of data augmentation, detection accuracy is further improved from 97.8% to 98.8% with ignorable false alarm penalty. For intermediate OPCed layouts Industry1-3 in TABLE V, BBL also significantly reduces average false alarm from 2347 to 1981 when achieving almost the same detection accuracy. We can also notice that for OPCed layouts with data augmentation, BBL does not exhibit as efficiently as on target layouts. One reason falls on the complexity of OPC patterns

TABLE IV: Performance comparison between the biased learning and the batch biased learning on target layouts.

Benchmarks	SPIE'17 [16]+BL		SPIE'17 [16]+BBL		BL [19]		BL [19]+AUG		BBL		BBL+AUG	
	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#
ICCAD	97.60	2054	98.04	3237	98.20	3413	98.40	3878	98.00	2745	98.40	3535
Industry0	98.73	316	99.47	469	97.35	142	99.26	246	97.66	136	99.36	387
Average	98.17	1185	98.76	1853	97.78	1778	98.83	2062	97.83	1441	98.88	1961
Ratio	0.993	0.604	0.999	0.945	0.989	0.906	0.999	1.052	0.989	0.735	1.000	1.000

TABLE V: Performance comparison between the biased learning and the batch biased learning on OPCed layouts.

Benchmarks	SPIE'17 [16]+BL		SPIE'17 [16]+BBL		BL [19]		BL [19]+AUG		BBL		BBL+AUG	
	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#
Industry1	98.61	311	98.30	362	98.90	680	99.00	558	98.20	402	98.20	409
Industry2	94.62	1156	95.49	1550	93.60	2165	94.80	1483	94.70	1702	95.50	1691
Industry3	89.95	2541	93.72	4361	91.30	4196	91.30	3917	90.00	3840	91.40	3888
Average	94.39	1336	95.84	2091	94.60	2347	95.03	1986	94.30	1981	95.03	1996
Ratio	0.993	0.669	1.008	1.048	0.995	1.176	1.000	0.995	0.992	0.993	1.000	1.000

TABLE VI: Performance comparison with state-of-the-art hotspot detectors on target layouts.

Benchmarks	SPIE'15 [5]		ICCAD'16 [6]		SOCC'17 [18]		SPIE'17 [16]		BBL+AUG	
	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#
ICCAD	84.20	2919	97.70	4497	96.90	1960	97.70	2703	98.40	3535
Industry0	93.63	30	96.07	1148	97.77	100	97.55	85	99.36	387
Average	88.92	1475	96.89	2823	97.34	1030	97.63	1394	98.88	1961
Ratio	0.899	0.752	0.980	1.439	0.984	0.525	0.987	0.711	1.000	1.000

TABLE VII: Performance comparison with state-of-the-art hotspot detectors on OPCed layouts.

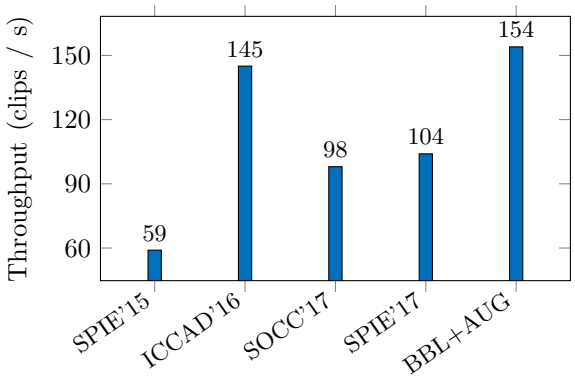
Benchmarks	SPIE'15 [5]		ICCAD'16 [6]		SOCC'17 [18]		SPIE'17 [16]		BBL+AUG	
	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#
Industry1	93.20	2204	89.90	1136	98.50	318	97.74	519	98.20	409
Industry2	44.80	1320	88.40	7402	91.1	710	89.62	760	95.50	1691
Industry3	44.00	3144	82.30	8609	84.01	1628	77.14	1966	91.40	3888
Average	60.67	2223	86.87	5716	91.20	885	88.17	1082	95.03	1996
Ratio	0.638	1.114	0.914	2.864	0.960	0.444	0.928	0.542	1.000	1.000

which induce more challenge on neural network training, when unbiased labels dominate the training procedure compared to BL which forces non-hotspot labels to be biased.

We also embed the proposed biased learning and batch biased learning algorithms into large neural networks designed in SPIE'17 [16] which takes raw layout images as input. Compared with original SPIE'17 [16] (as shown in TABLE VI and TABLE VII), BL and BBL can significantly improve the hotspot detection performance. For target layouts, average detection accuracy increased from 97.63% to 98.17% and 98.76% with BL and BBL, respectively. BL also reduces the false alarm count from 1394 to 1185. For OPCed layouts, BL and BBL dramatically increase the detection accuracy from 88.17% to 94.39% and 95.84% respectively. In particular, BBL can further improve the hotspot detection accuracy when inserted in large SPIE'17 nets at relatively large false alarm penalty, because biased labels dominate during the training procedure of BBL for neural networks with larger capacity. Fig. 10 illustrates the testing speed of different neural network models that correspond to TABLE IV and TABLE V. From the experimental results, we can clearly see that the proposed methods can achieve similar hotspot detection accuracy compared to larger neural network models but with much less computing costs (104 clips/s of deep networks in SPIE'17 v.s. 156 clips/s in our proposed network architecture). Although data augmentation and ensemble testing slightly induces computing costs, we can still observe great advantage of the proposed models over SPIE'17 nets.

We then compare the hotspot detection results with four state-of-the-art hotspot detectors in TABLE VI and TABLE VII. "SPIE'15 [5]" is a traditional machine learning-based hotspot detector that applies the density-based layout features and the AdaBoost [32]–DecisionTree model. "ICCAD'16 [6]" takes the lithographic properties into account during feature extraction and adopts the more robust Smooth Boosting [33] algorithm. "SPIE'17 [16]" is another deep learning solution for hotspot detection that takes the original layout image as input and contains more than 20 layers. "SOCC'17 [18]" employs a deep neural networks that replace all pooling layers with strided convolution layers and contains the same number of layers as SPIE'17.

Overall, our framework performs better than traditional



- SPIE'15 & ICCAD'16: tested on 3.3GHz Quad-Core Intel processor.
- SOCC'17, SPIE'17 & BBL+AUG: tested on NVIDIA GPU.

Fig. 11: Throughput comparison with state-of-the-art hotspot detectors.

machine learning techniques (SPIE'15 [5] and ICCAD'16 [6]) with at least a 2% advantage for the detection accuracy (98.88% of BBL v.s. 96.89% of ICCAD'16) on target layouts. Traditional machine learning models are effective for the benchmarks with regular patterns (ICCAD and Industry0) with the highest detection accuracy of 97.7% on the ICCAD and 96.07% on the Industry0 achieved by [6]. However, manually designed features, including the density-based features and the CCAS, have difficulties to grasp the attributes of the post-OPC mask layouts. For OPCed layouts, [5] suffers a large performance degradation with only approximately 45% detection accuracy on the most complicated case Industry3. Although prior knowledge and a more robust Smooth Boosting [33] algorithm are applied in [6], the hotspot detection accuracy inevitably drops around 8%. Deep learning solutions [18], [16] and BBL+AUG exhibit better performances with hotspot detection accuracy of 91.20%, 88.17% and 95.03% respectively. It is notable that the architecture of our framework contains significantly fewer layers than the framework in SPIE'17 and SOCC'17. Although the shallow architecture results in acceptable more false alarms than deep models, our framework can still offer significant higher accuracy on ICCAD, Industry0, Industry2 and Industry3.

Although neural networks are not as computational efficient as traditional machine learning methods, with the aid of parallel computing units (e.g. GPU), we are able to achieve comparable and acceptable processing speed. Fig. 11 presents the detecting speed of different hotspot detectors adopted in this work, where SPIE'15 and ICCAD'16 are tested on CPU only and neural network implementations are tested on graphic cards. Runtime reports show that although neural network models are not computational friendly due to the complicated convolutional operations, we are still able to complete the task with comparable and acceptable time, which also show the potential of enhancing the layout verification flow with dedicated computing units instead of CPU only.

VI. Conclusion

To address the existing problems of machine learning-based printability estimation techniques, we first propose a high-dimensional feature (feature tensor) extraction method that can reduce the size of training instances while keeping the spatial information. The feature tensor is also compatible with powerful convolutional neural networks. Additionally, to improve hotspot detection accuracy, we first develop a biased learning algorithm, which takes advantage of the ReLU function in CNN to prominently increase accuracy while reducing false alarm penalties. We further propose a batch biased learning algorithm to automatically adjust the training ground truth according to the current batch loss, that can offer better trade-offs between accuracy and false alarms. The experimental results show that the batch biased learning algorithm is more efficient during training and our framework outperforms the other hotspot solutions on complicated benchmarks. Source code and trained models are available at https://github.com/phdyang007/dlhsd. As the technology node keeps shrinking down, we hope this paper can be a demonstration that deep learning has the potential to provide satisfactory solutions for advanced design for manufacturability (DFM) research.

References

[1] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), vol. 33, no. 11, pp. 1671–1680, 2014.

[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 ACM/IEEE Design Automation Conference (DAC), 2012, pp. 1167–1172.

[3] F. Yang, S. Sinha, C. C. Chiang, X. Zeng, and D. Zhou, "Improved tangent space based distance metric for lithographic hotspot classification," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), vol. 36, no. 9, pp. 1545–1556, 2017.

[4] Y.-T. Yu, G.-H. Lin, I. H.-R. Jiang, and C. Chiang, "Machine-learning-based hotspot detection using topological classification and critical feature extraction," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), vol. 34, no. 3, pp. 460–470, 2015.

[5] 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 Proceedings of SPIE, vol. 9427, 2015.

[6] H. Zhang, B. Yu, and E. F. Y. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016, pp. 47:1–47:8.

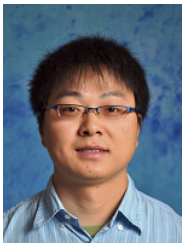
[7] D. Ding, A. J. Torres, F. G. Pikus, and D. Z. Pan, "High performance lithographic hotspot detection using hierarchically refined machine learning," in IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), 2011, pp. 775–780.

[8] D. Ding, B. Yu, J. Ghosh, and D. Z. Pan, "EPIC: Efficient prediction of IC manufacturing hotspots with a unified meta-classification formulation," in IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC), 2012, pp. 263–270.

[9] D. G. Drmanac, F. Liu, and L.-C. Wang, "Predicting variability in nanoscale lithography processes," in ACM/IEEE Design Automation Conference (DAC), 2009, pp. 545–550.

[10] A. J. Torres, "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite," in IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2012, pp. 349–350.
[11] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical bayes model," in Proceedings of SPIE, vol. 9426, 2015.
[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Conference on Neural Information Processing Systems (NIPS), 2012, pp. 1097–1105.
[13] T. Xiao, H. Li, W. Ouyang, and X. Wang, "Learning deep feature representations with domain guided dropout for person re-identification," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 1249–1258.
[14] T. Matsunawa, S. Nojima, and T. Kotani, "Automatic layout feature extraction for lithography hotspot detection based on deep neural network," in SPIE Advanced Lithography, vol. 9781, 2016.
[15] M. Shin and J.-H. Lee, "Accurate lithography hotspot detection using deep convolutional neural networks," Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3), vol. 15, no. 4, p. 043507, 2016.
[16] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography hotspot detection: A deep learning approach," in SPIE Advanced Lithography, vol. 10148, 2017.
[17] ——, "Imbalance aware lithography hotspot detection: a deep learning approach," Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3), vol. 16, no. 3, p. 033504, 2017.
[18] H. Yang, Y. Lin, B. Yu, and E. F. Young, "Lithography hotspot detection: From shallow to deep learning," in IEEE International System-on-Chip Conference (SOCC), 2017, pp. 233–238.
[19] H. Yang, J. Su, Y. Zou, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," in ACM/IEEE Design Automation Conference (DAC), 2017, pp. 62:1–62:6.
[20] W. Zhang, X. Li, S. Saxena, A. Strojwas, and R. Rutenbar, "Automatic clustering of wafer spatial signatures," in ACM/IEEE Design Automation Conference (DAC), 2013, pp. 71:1–71:6.
[21] S. Shim, W. Chung, and Y. Shin, "Synthesis of lithography test patterns through topology-oriented pattern extraction and classification," in Proceedings of SPIE, vol. 9053, 2014.
[22] G. K. Wallace, "The JPEG still picture compression standard," IEEE Transactions on Consumer Electronics (TCE), vol. 38, no. 1, pp. xviii–xxxiv, 1992.
[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, no. 6088, pp. 533–536, 1986.
[24] W. A. Gardner, "Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique," Signal Processing, vol. 6, no. 2, pp. 113–133, 1984.
[25] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016, http://www.deeplearningbook.org.
[26] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 6, no. 02, pp. 107–116, 1998.
[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," International Conference on Learning Representations (ICLR), 2015.
[28] C. Zhang, Q. Liao, A. Rakhlin, K. Sridharan, B. Miranda, N. Golowich, and T. Poggio, "Theory of deep learning III: Generalization properties of SGD," Center for Brains, Minds and Machines (CBMM), Tech. Rep., 2017.
[29] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in International Conference on Learning Representations (ICLR), 2017.
[30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean et al., "TensorFlow: A system for large-scale machine learning," in USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2016, pp. 265–283.
[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
[32] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in European conference on computational learning theory. Springer, 1995, pp. 23–37.
[33] R. A. Servedio, "Smooth boosting and learning with malicious noise," Journal of Machine Learning Research, vol. 4, no. Sep, pp. 633–648, 2003.

Haoyu Yang received his B.E. degree from Qiushi Honors College, Tianjin University in 2015. He is currently pursuing his Ph.D. Degree at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include VLSI design for manufacturability and machine learning.



Jing Su graduated from University of Science and Technology of China in 2008 with his B.S. degree in Physics. He later received his Ph.D. degree in theoretical and computational AMO physics at University of Colorado at Boulder in 2014. Currently he is a senior design engineer in the Advanced Technology Development group at ASML-Brion. His present research interest covers a wide range of topics in lithography, including advanced mask optimization, data mining, and machine learning.

Yi Zou graduated from Zhejiang University in 1998 with his B.S. degree in Control Science and Engineering. He received his MPhil degree in Industrial Engineering at Hong Kong University of Science and Technology in 2000, and his Ph.D. degree in Microchip Simulation at Boston University in 2004. Currently Yi Zou is Director of Engineering at ASML. His research interests include, but are not limited to: optical proximity correction (OPC) and lithography scanner modeling and control via advanced data analytics.

Yuzhe Ma received his B.E. degree from the Department of Microelectronics, Sun Yat-sen University in 2016. He is currently pursuing his Ph.D. degree at the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research interests include VLSI design for manufacturing, physical design and machine learning on chips.

Bei Yu (S'11–M'14) received the Ph.D. degree from The University of Texas at Austin in 2014. He is currently an Assistant Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. He has served in the editorial boards of Integration, the VLSI Journal and IET Cyber-Physical Systems: Theory & Applications. He received five Best Paper Awards from Integration, the VLSI Journal in 2018, International Symposium on Physical Design 2017, SPIE Advanced Lithography Conference 2016, International Conference on Computer Aided Design 2013. and Asia and South Pacific Design Automation Conference 2012, and four ICCAD/ISPD contest awards.

Evangeline F.Y. Young received her B.Sc. degree in Computer Science from The Chinese University of Hong Kong (CUHK). She received her Ph.D. degree from The University of Texas at Austin in 1999. She is currently a professor in the Department of Computer Science and Engineering in CUHK. Her research interests include Optimization, algorithms and VLSI CAD. She works actively on floorplanning, placement, routing, DFM and EDA on Physical Design in general. Dr. Young has served on the organization committees of ISPD, ARC and FPT and on the program committees of conferences including DAC, ICCAD, ISPD, ASP-DAC, SLIP, DATE and GLSVLSI. She also served on the editorial boards of IEEE TCAD, ACM TODAES and Integration, the VLSI Journal. Besides, her research group has won several championships and prizes in renown EDA contests, including the 2016, 2015, 2013 and 2012 CAD Contests at ICCAD, DAC 2012 and ISPD 2011 Routability-driven Placement Contests and ISPD 2010 High Performance Clock Network Synthesis Contest.