Layout Hotspot Detection with Feature Tensor Generation and Deep Biased Learning

Haoyu Yang[1], Jing Su[2], Yi Zou[2], Bei Yu[1], and Evangeline F. Y. Young[1]

[1]CSE Department, The Chinese University of Hong Kong, NT, Hong Kong
[2]ASML Brion Inc., CA 95054, USA
{hyyang,byu,fyyoung}@cse.cuhk.edu.hk, {jing.su,yi.zou}@asml.com

ABSTRACT

Detecting layout hotspots is one of the key problems in 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 it is inevitable that performance degradation will occur. To overcome these issues, in this paper we develop a deep learning framework for high performance and large scale hotspot detection. First, feature tensor generation is proposed 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. Experimental results show that our framework outperforms previous machine learning-based hotspot detectors in both the ICCAD 2012 Contest benchmarks and large scale industrial benchmarks.

1. INTRODUCTION

As transistor feature size enters the nanometer era, manufacturing yield is drastically affected by lithographic process variations due to 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 (a.k.a, hotspots), thus hotspot detection during physical verification stage is very important. The conventional hotspot detection method is full-chip lithography simulation, which has 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, 2] and machine learning [3–5]. Pattern matching is a direct and fast method to detect layout characteristics, but has a great error rate for unknown patterns. Machine learning techniques, to some extent, are capable of learning hidden relations between layout patterns and their defects characteristics, thus can greatly improve detection accuracy. Very recently, Zhang *et al.* [5] achieve tremendous performance improvement on ICCAD Contest 2012 benchmark suite [6] by applying an optimized concentric circle sampling (CCS) feature [7] and an online learning scheme. However, there are several aspects that previous works have not taken into account, especially when targeting a very large scale problem size. (1) **Scalability**: As integrated circuit has developed 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 model the characteristics of a large amount of layout patterns. (2) **Feature Representation**: The state-of-the-art layout feature extraction approaches, including density [4] and CCS [7], inevitably suffer from spatial information loss, because extracted feature elements are flattened into 1-D vectors, ignoring potential spatial relations.

To overcome the limitations or issues of conventional machine learning methodologies, 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 has shown great success in image classification tasks [8, 9].

The proposed framework is integrated with newly developed techniques that are customized and more suitable for layout hotspot detection. Inspired by the feature map in deep neural networks, we utilize the feature tensor concept, i.e., a multidimensional representation of an original layout pattern, that performs compression to facilitate learning while keeping the spatial relationship to avoid drastic information loss. Feature tensor compatibility with the convolutional neural network makes our framework more efficient when dealing with an ultra large amount of layout patterns. To train a model with high detection accuracy and low false alarm penalty, we further developed a biased learning technique that can be easily embedded into traditional neural network training procedures. 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 feedforward and back-propagation.
- Biased learning technique is proposed and embedded into the deep learning framework, that offers great improvements on hotspot detection accuracy with a minor cost of false alarms.
- Experimental results show that our proposed methods have great advantages over existing machine learning solutions and achieve 5.9% accuracy improvements on average for very large scale industrial layouts.

DAC'17, June 18-22, 2017, Austin, TX, USA
© 2017 ACM. ISBN 978-1-4503-4927-7/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3061639.3062270

The rest of the paper is organized as follows. Section 2 introduces basic concepts and problem formulation. Section 3 and section 4 cover feature tensor generation and biased learning algorithm, respectively. Section 5 lists the experimental results, followed by the conclusion in Section 6.

2. PRELIMINARIES

In this section, we will introduce some terminology 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 [6]). The ratio between the number of correctly predicted hotspot clips and the number of all real hotspot clips.

Definition 2 (False Alarm [6]). 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 for the overall estimation flow runtime. Therefore, an overall detection and simulation time (ODST) is defined as follows:

Definition 3 (ODST [5]). 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 object of hotspot detection is training a classifier that can maximize the Accuracy and minimize the ODST.*

3. FEATURE TENSOR EXTRACTION

Finding a good feature representation is the key procedure in image classification tasks, and so is layout pattern classification. Local density extraction and concentric circle sampling have been widely explored in previous hotspot detection and optical proximity correction (OPC) research [7], which proved to be efficient on hotspot detection tasks because of embedded lithographic prior knowledge. It is notable that layout hotspots are associated with light diffraction, whether a layout suffers from hotspot is not only determined by the pattern itself, but also affected by its surrounding patterns. Therefore, to analyze characteristic of a clip, spatial relations of its local regions have to be kept. However, all of these existing features finally end up flattened into one dimensional vectors which limits hotspot detection accuracy due to a lot of spatial information loss.

To address the above issue, we propose a feature tensor extraction method to provide lower scale representation of 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 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 [10, 11] and achieves 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 moreover, 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 feature representations of all sub-regions are obtained 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, which contain high density information, as depicted in Figure 1.

Step 3: Flatten $\mathbf{D}_{i,j}$s into vectors in Zig-Zag form [12] with 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 first $k \ll B \times B$ elements of each $\mathbf{C}_{i,j}^{*}$,

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

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

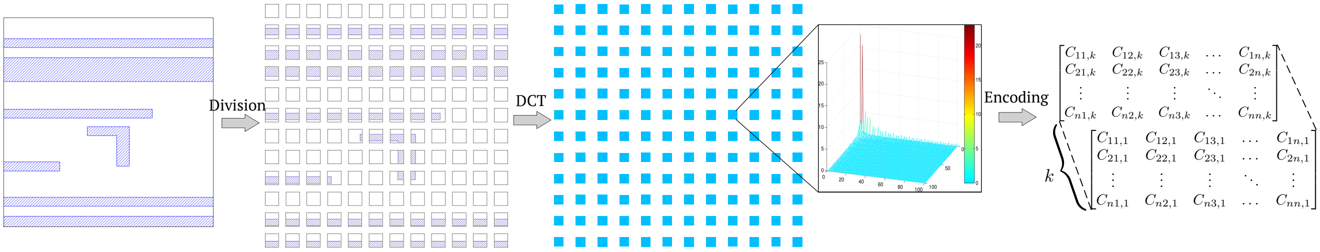
Division
DCT
Encoding
$C_{11,k}$ $C_{12,k}$ $C_{13,k}$ … $C_{1n,k}$
$C_{21,k}$ $C_{22,k}$ $C_{23,k}$ … $C_{2n,k}$
$C_{n1,k}$ $C_{n2,k}$ $C_{n3,k}$ … $C_{nn,k}$
$C_{11,1}$ $C_{12,1}$ $C_{13,1}$ … $C_{1n,1}$
$C_{21,1}$ $C_{22,1}$ $C_{23,1}$ … $C_{2n,1}$
$C_{n1,1}$ $C_{n2,1}$ $C_{n3,1}$ … $C_{nn,1}$
k

Figure 1: 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 100×100 image representing 100×100 nm^2 sub region of the original clip. Feature tensor is then obtained by encoding on DCT coefficients of each block.

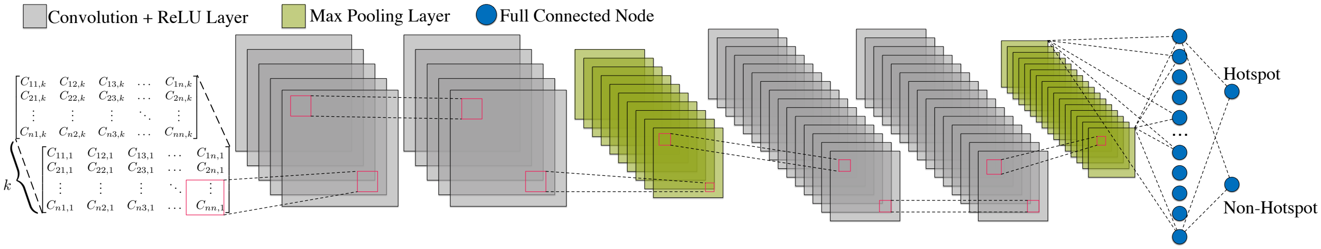

Figure 2: The proposed convolutional neural network structure.

follows:

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

where $\mathbf{F} \in \mathbb{R}^{n \times n \times k}$. Particularly, through 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 Figure 1, large responses only present at the entries with smaller indexes, i.e. low frequency regions. Therefore, most information are kept even when large amounts 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 in comparison with using original layout image as input, because the scale of the neural network is reduced with smaller input size.

4. BIASED LEARNING ALGORITHM

4.1 Convolutional Neural Network Architecture

To address the weak scalability of traditional machine learning techniques, we introduce 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 layers generate the probability of testing instances drawn from each category (Figure 2).

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:

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

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.

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

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 two convolution stages are two fully connected (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 1.

Table 1: 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

4.2 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 [13] 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 [14] 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) [15] 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 (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 training process.

The details of MGD with learning rate decay are shown in

Algorithm 1 Mini-batch Gradient Descent (MGD)

```
1: function MGD($\mathbf{W}$, $\lambda$, $\alpha$, $k$, $\mathbf{y}_h^*$, $\mathbf{y}_n^*$)
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 $i = 1, 2, ..., m$ do
7:             $\mathcal{G}_i \leftarrow$ 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:        Update weight $\mathbf{W} \leftarrow \mathbf{W} - \lambda\bar{\mathcal{G}}$;
15:    end while
16:    return Trained model $f$;
17: end function
```

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. The 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 training set (line 5). Gradients of these training instances ($\mathcal{G}_i$) are calculated for 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. learning rate decays every k iterations (lines 11–13). At the end of MGD, a trained model that has satisfying performance on validation set will be returned (line 16).

4.3 Learning Towards Biased Target

Softmax cross entropy is able to provide speedup for back-propagation while attain comparable network performance with mean square error [15]. In n-category classification task, the instance belongs to class c has ground truth $\mathbf{y}^* \in \mathbb{R}^n$ where $\mathbf{y}^*$ has the property $\mathbf{y}^*(c) = 1$ 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. 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 ground truth of 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 $(0, 1)$ interval by softmax function shown in Equation (6).

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

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

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

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

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

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,

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

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

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

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

where $\lambda > 0$ is shifting level. However, this method takes effect at large cost of false alarm.

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 $\mathbf{y}_n^*$. Suppose training procedure meets stop criteria, then for most non-hotspot clips, f will predict them to have high probability, close to 1, to be non-hotspot. However, as can be seen in Equation (9), the instance would be predicted as 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 to the network. Therefore, an assumption can be made that when a training procedure meets the convergence criteria, the hotspot detection accuracy can be further improved by reducing the confident level of the non-hotspot data set. We have the following theorem.

Theorem 1: *Given a trained convolutional neural network 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 $\mathbf{y}_n^{\epsilon} = [1 - \epsilon, \epsilon]$, $\epsilon \in [0, 0.5)$, and obtain the hotspot detection accuracy a' of the new model. We have $a' \geq a$.*

The proof of Theorem 1 is in Appendix. 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 alarm. 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 truth, as shown in Algorithm 2.

Algorithm 2 Biased-learning

```
Input: $\epsilon$, $\delta\epsilon$, $t$, $\mathbf{W}$, $\lambda$, $\alpha$, $k$, $\mathbf{y}_h^*$, $\mathbf{y}_n^*$;
1: $i \leftarrow 0$, $\epsilon \leftarrow 0$, $\mathbf{y}_h^* \leftarrow [0, 1]$;
2: while $i < t$ do
3:     $\mathbf{y}_n^* \leftarrow [1 - \epsilon, \epsilon]$;
4:     $f_\epsilon \leftarrow$ MGD($\mathbf{W}$, $\lambda$, $\alpha$, $k$, $\mathbf{y}_n^*$, $\mathbf{y}_h^*$);    ▷ Algorithm 1
5:     $i \leftarrow i + 1$, $\epsilon \leftarrow \epsilon + \delta\epsilon$;
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 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).

Theorem 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. Details of the effect of biased learning are presented in following section.

5. EXPERIMENTAL RESULTS

We implement our deep biased learning framework in Python with the TensorFlow library [16], and test it on a platform with a Xeon E5 processor and Nvidia K620 Graphic card. To fully evaluate the proposed framework, we employ four test cases. Because the individual test cases in the ICCAD 2012 contest [6] 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 three more complicated industry test cases: `Industry1` – `Industry3`. The details for all test cases are listed in the left side of Table 2. 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.

In the first experiment, the neural network is trained following the biased learning algorithm with $\lambda = 1e-4, \alpha = 0.5, k = 10,000, \epsilon = 0, \delta\epsilon = 0.1$ and $t = 4$. We compare the test results with two state-of-the-art machine learning-based hotspot detectors, as shown in Table 2. Columns "FA#", "CPU(s)", "ODST(s)" and "Accu" denote false alarm, model testing time, overall detection simulation time (ODST) and hotspot detection accuracy respectively. An industry simulator described in [17] shows that the simulation time of each instance is approximately $10s$, therefore, we induce a $10s$ penalty on each detected hotspot instance for the ODST final calculation. The experimental results show that within all test cases, our framework achieves the best hotspot detection accuracy on average of 95.5% compared to 89.6% of [5] and 66.6% of [4]. The table also shows that when the benchmark is larger, traditional machine learning techniques suffer great performance decay. ODST is a unified metric on testing and lithographic simulation running time and it takes all the predicted hotspots into consideration. Although [5] exhibits a better ODST, it suffers from a lower hotspot detection accuracy.

Note that in Table 2 only the testing runtime is reported. As far as the proposed framework is concerned, it might take hours in the training procedure compared with approximately 30 minutes in other machine learning works. However, the training procedure can be done once and for all. The trained model can be effectively updated with newly incoming instances, due to the online update capability of MGD. Besides, the fine tuning procedure only has time complexity of $\mathcal{O}(m)$ for a fixed network configuration, where m is the total number of new instances. Therefore, the initial training time consumption can be ignorable when the large performance improvement is considered.

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 provide speed up on training procedures. To evaluate the efficiency of MGD, in the second experiment, we train the neural network on the `ICCAD` benchmark with MGD and SGD separately. SGD has learning a rate of 1e-4, while MGD has an initial learning rate of 1e-3 [1]. The training procedure is shown in Figure 3, where the X-axis is the elapsed time (s) in training procedure and the Y-axis is the test accuracy on the validation set. The curve shows that when MGD reaches 90% validation accuracy, SGD is still around 70%, which indicates the neural network with the MGD learning strategy is more efficient and effective than conventional SGD.

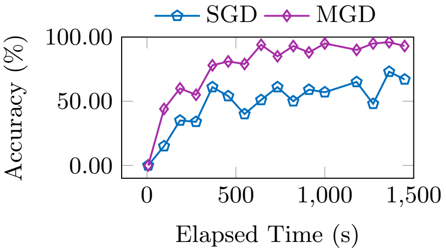

Figure 3: Stochastic gradient descent(SGD) v.s. Mini-batch gradient descent (MGD).

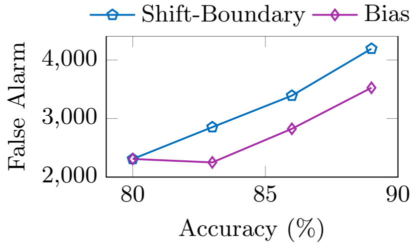

Figure 4: Bias learning shows less false alarm penalty to obtain the same hotspot detection accuracy.

In the last experiment, we evaluate the biased learning algorithm. The above neural network is trained with $\epsilon = 0$ to obtain an initial model, and is fine-tuned with $\epsilon = 0.1, 0.2, 0.3$ on `Industry3`. Then we perform boundary shifting on initial model to achieve the same test accuracy with three fine-tuned models. As shown in Figure 4, biased learning has 600 less false alarm penalties for same improvement of hotspot detection accuracy which is equivalent to saving $6000s$ of ODST consumption.

6. 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 spatial information. The feature tensor is also compatible with powerful convolutional neural networks. Additionally, to improve hotspot detection accuracy, we develope a biased learning algorithm, that takes advantage of the ReLU function in CNN to prominently increase accuracy while reducing false alarm penalties. The experimental results show that our framework outperforms the existing machine learning techniques. 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 DFM research.

7. 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 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 *Proc. DAC*, 2012, pp. 1167–1172.

[3] 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 TCAD*, vol. 34, no. 3, pp. 460–470, 2015.

[1] Because the average gradient on a random picked group of samples is smaller than the gradient of single instance, we pick the larger learning rate for MGD to ensure a fair comparison.

Table 2: Performance Comparisons with two state-of-the-art hotspot detectors

Bench	Train	Train	Test	Test	SPIE'15 [4]				ICCAD'16 [5]				Ours			
	HS#	NHS#	HS#	NHS#	FA#	CPU (s)	ODST (s)	Accu (%)	FA#	CPU (s)	ODST (s)	Accu (%)	FA#	CPU (s)	ODST (s)	Accu (%)
ICCAD	1204	17096	2524	13503	**2919**	2415	**52857**	84.2%	4497	**989**	70618	97.7%	3413	1232	60147	**98.2%**
Industry1	34281	15635	17157	7801	2204	557	182500	93.2%	1136	**179**	**165780**	89.9%	**680**	266	176748	**98.9%**
Industry2	15197	48758	7520	24457	**1320**	752	**47641**	44.8%	7402	**233**	140729	88.4%	2165	341	92298	**93.6%**
Industry3	24776	49315	12228	24817	**3144**	879	**86122**	44.0%	8609	**279**	187005	82.3%	4196	404	154005	**91.3%**
Average	-	-	-	-	**2397**	1150	45109	66.6%	5411	**420**	**35359**	89.6%	2613	561	44577	**95.5%**
Ratio	-	-	-	-	-	-	1.01	0.70	-	-	**0.79**	0.94	-	-	1.0	**1.0**

[4] T. Matsunawa, J.-R. Gao, B. Yu, and D. Z. Pan, "A new lithography hotspot detection framework based on AdaBoost classifier and simplified feature extraction," in *Proc. SPIE*, vol. 9427, 2015.

[5] H. Zhang, B. Yu, and E. F. Y. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *Proc. ICCAD*, 2016, pp. 47:1–47:8.

[6] A. J. Torres, "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite," in *Proc. ICCAD*, 2012, pp. 349–350.

[7] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical bayes model," in *Proc. SPIE*, vol. 9426, 2015.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1097–1105.

[9] T. Xiao, H. Li, W. Ouyang, and X. Wang, "Learning deep feature representations with domain guided dropout for person re-identification," in *Proc. CVPR*, 2016, pp. 1249–1258.

[10] W. Zhang, X. Li, S. Saxena, A. Strojwas, and R. Rutenbar, "Automatic clustering of wafer spatial signatures," in *Proc. DAC*, 2013, pp. 71:1–71:6.

[11] S. Shim, W. Chung, and Y. Shin, "Synthesis of lithography test patterns through topology-oriented pattern extraction and classification," in *Proc. SPIE*, vol. 9053, 2014.

[12] G. K. Wallace, "The JPEG still picture compression standard," *IEEE TCE*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[13] 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.

[14] 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.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, 2016, pp. 265–283.

[17] S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in *Proc. ICCAD*, 2013, pp. 271–274.

[18] 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.

APPENDIX

A. PROOF OF THEOREM 1

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:

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

where $\mathbf{W}_l$ is learned towards the target $\mathbf{y}_n^* = [1, 0]$. 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 training instances that have predicted probability in $(1-\epsilon, 1)$ (confident instances), neuron weights will be updated along the gradient generated by confident instances.

Also, gradient vanishing theory [18] indicates a later layer in the neural network learns faster, therefore within 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,

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

Before adjusting the ground truth, we have

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

Since $\mathbf{x}_c$ is obtained from the classifier towards $\mathbf{y}_n^* = [1, 0]$ and $\mathbf{x}'_c$ is obtained from the trained model with the ground truth $\mathbf{y}_n^{\epsilon}$,

$$\frac{\exp \mathbf{x}_c(0)}{\exp \mathbf{x}_c(0) + \exp \mathbf{x}_c(1)} > \frac{\exp \mathbf{x}'{}_c(0)}{\exp \mathbf{x}'{}_c(0) + \exp \mathbf{x}'{}_c(1)} > 1-\epsilon. \tag{15}$$

That is,

$$\mathbf{x}_c(0) - \mathbf{x}_c(1) > \mathbf{x}'_c(0) - \mathbf{x}'_c(1). \tag{16}$$

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:

$$\mathbf{w} = \mathbf{W}_{l,1} - \mathbf{W}_{l,2}, \quad \mathbf{w}' = \mathbf{W}'_{l,1} - \mathbf{W}'_{l,2}. \tag{17}$$

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

$$\mathbf{w}' = \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathbf{w}^{\intercal} O_{l-1}(\mathbf{F}_c) = \mathbf{w} - \alpha O_{l-1}(\mathbf{F}_c), \tag{18}$$

where $\alpha > 0$ is the learning rate. For hotspot instances:

$$\mathbf{w}'^{\intercal} O_{l-1}(\mathbf{F}_h) = \mathbf{w}^{\intercal} O_{l-1}(\mathbf{F}_h) - \alpha O_{l-1}^{\intercal}(\mathbf{F}_c) O_{l-1}(\mathbf{F}_h). \tag{19}$$

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,

$$\mathbf{w}'^{\intercal} O_{l-1}(\mathbf{F}_h) < \mathbf{w}^{\intercal} O_{l-1}(\mathbf{F}_h), \tag{20}$$

which indicates that

$$\mathbf{x}_h(0) - \mathbf{x}_h(1) > \mathbf{x}'_h(0) - \mathbf{x}'_h(1), \tag{21}$$

$$\Rightarrow \frac{\exp \mathbf{x}_h(1)}{\exp \mathbf{x}_h(0) + \exp \mathbf{x}_h(1)} < \frac{\exp \mathbf{x}'_h(1)}{\exp \mathbf{x}'_h(0) + \exp \mathbf{x}'_h(1)}. \tag{22}$$

Therefore, the predicted probability of hotspot instances being hotspot 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, $a' \geq a$.