Journal of
Micro/Nanolithography,
MEMS, and MOEMS

Nanolithography.SPIEDigitalLibrary.org

Imbalance aware lithography hotspot detection: a deep learning approach

Haoyu Yang
Luyang Luo
Jing Su
Chenxi Lin
Bei Yu

SPIE.

Haoyu Yang, Luyang Luo, Jing Su, Chenxi Lin, Bei Yu, "Imbalance aware lithography hotspot detection: a deep learning approach," *J. Micro/Nanolith. MEMS MOEMS* **16**(3), 033504 (2017), doi: 10.1117/1.JMM.16.3.033504.

Imbalance aware lithography hotspot detection: a deep learning approach

Haoyu Yang,[a] Luyang Luo,[a] Jing Su,[b] Chenxi Lin,[b] and Bei Yu[a,*]
[a]Chinese University of Hong Kong, Department of Computer Science and Engineering, New Territories, Hong Kong
[b]ASML Brion Inc., San Jose, California, United States

Abstract. With the advancement of very large scale integrated circuits (VLSI) technology nodes, lithographic hotspots become a serious problem that affects manufacture yield. Lithography hotspot detection at the post-OPC stage is imperative to check potential circuit failures when transferring designed patterns onto silicon wafers. Although conventional lithography hotspot detection methods, such as machine learning, have gained satisfactory performance, with the extreme scaling of transistor feature size and layout patterns growing in complexity, conventional methodologies may suffer from performance degradation. For example, manual or *ad hoc* feature extraction in a machine learning framework may lose important information when predicting potential errors in ultra-large-scale integrated circuit masks. We present a deep convolutional neural network (CNN) that targets representative feature learning in lithography hotspot detection. We carefully analyze the impact and effectiveness of different CNN hyperparameters, through which a hotspot-detection-oriented neural network model is established. Because hotspot patterns are always in the minority in VLSI mask design, the training dataset is highly imbalanced. In this situation, a neural network is no longer reliable, because a trained model with high classification accuracy may still suffer from a high number of false negative results (missing hotspots), which is fatal in hotspot detection problems. To address the imbalance problem, we further apply hotspot upsampling and random-mirror flipping before training the network. Experimental results show that our proposed neural network model achieves comparable or better performance on the ICCAD 2012 contest benchmark compared to state-of-the-art hotspot detectors based on deep or representative machine leaning.
 [DOI: 10.1117/1.JMM.16.3.033504]

Keywords: lithography; hotspot detection; deep learning.

Paper 17066P received May 10, 2017; accepted for publication Aug. 1, 2017; published online Aug. 24, 2017.

1 Introduction

As circuit feature size shrinks down to 20 nm, lithographic hotspots have become a serious factor that affects manufacture yield. A hotspot is a region of mask layout patterns where circuit failures are more likely to happen during the manufacturing process because of light diffraction, etch proximate effects, overlay control, and so on. Therefore, hotspot detection at the post-OPC stage is imperative before transferring designed patterns onto a silicon wafer.

Many studies were carried out for lithography hotspot detection. The state-of-the-art methods include lithographic simulation[1,2] assisted with pattern matching[3,4] and current machine learning techniques.[5–9] Lithographic simulation can imitate fabrication results accurately, but it is computationally expensive. Because problematic region area is much smaller than the full chip area, modern physical verification flow usually performs fast classification to extract hotspot candidates for lithography simulation. Pattern matching provides speed improvements in comparison with full chip lithographic simulation, but it only applies to detecting already known or similar patterns, thus, it has a poor hotspot recognition rate on unknown patterns. Machine learning (In this paper, we refer to machine learning as the methods that require manually feature design as opposite to deep learning where features are obtained through training neural network.) is an emerging technique that can achieve reasonably good hotspot detection results with fast throughput. In a machine learning flow, raw data should be preprocessed in the feature extraction stage to convert complicated layout patterns into low-dimensional vectors before being fed into the learning engine. The low-dimensional vector, also known as feature representation, directly affects hotspot prediction performance. For a very large scale integrated circuits (VLSI) layout, the conventional density-based feature[4,10] and the recently proposed concentric circle area sampling (CCAS) feature[11] capture layout properties and the lithography process, respectively, and made considerable improvements on hotspot detection accuracy. However, with circuit feature size reduced to several nanometers, layout patterns are more complicated, and *ad hoc* feature extraction may suffer from important information loss when predicting potential hotspots in ultra-large-scale integrated circuit masks.

Convolutional neural networks (CNN) have proved capable of extracting appropriate image representations and performing accurate classification tasks benefitting from high-efficiency-feature learning and high-nonlinear models.[12–14] However, to take advantage of powerful deep learning models, there are still several aspects that should be considered: (1) hyperparameters are required to be suitable for the nature of the circuit layout. The conventional deep learning model has a pattern shift invariance, due to the nature of convolution and pooling operations. For the lithography hotspot detection problem, whether a pattern

*Address all correspondence to: Bei Yu, E-mail: byu@cse.cuhk.edu.hk

1932-5150/2017/$25.00 © 2017 SPIE

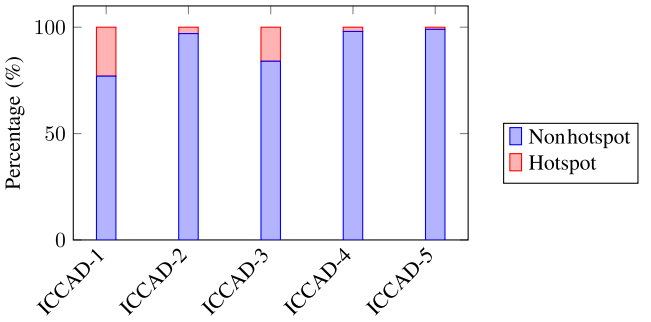

Fig. 1 Breakdown of hotspot and nonhotspot pattern percentages for ICCAD 2012 contest benchmark.

is a hotspot or not is affected by nanometer-level shifts of mask patterns, thus, it is necessary to design compatible convolution and pooling kernel values. (2) Layout datasets are highly imbalanced because after resolution enhancement techniques, the number of lithography hotspots is much less than the number of nonhotspot patterns. Figure 1 shows the percentages of hotspot and nonhotspot patterns for each test case of the ICCAD 2012 benchmark suite.[15] We can see that the number of nonhotspot patterns is much larger than the number of hotspot patterns, especially for cases `ICCAD-2`, `ICCAD-4`, and `ICCAD-5`, where nonhotspot patterns occupy more than 99% of the total patterns. As a result, under conventional training strategies, the neural network may not be able to correctly predict hotspot patterns even with ignorable training loss. (3) For hotspot detection tasks, the mask image size is much larger than those in traditional object recognition tasks, meaning that the neural network should be specifically designed to handle large size inputs.

In this paper, we develop a deep learning-based hotspot detection flow, as shown in Fig. 2. Every original training dataset is divided into two parts: 75% of the samples are preprocessed for deep learning model training, whereas the other 25% of the samples are used for validation. Validation is used to monitor training status and can indicate when to stop training. After obtaining the trained model, instances in the testing dataset are fed into the neural network and their labels will be predicted moving forward. Accuracy and false alarms can then be calculated by comparing prediction results with the corresponding actual results. We carefully analyze impact and effectiveness of different CNN hyperparameters, through which a hotspot-detection-oriented neural network model is established. To address the imbalance problem, we further apply hotspot upsampling and random-mirror flipping of the hotspot patterns before training the network. Finally, we verify our proposed model on the ICCAD 2012 contest benchmark suite.[15] Experimental results show that the proposed model outperforms several state-of-the-art hotspot detectors in most cases while attaining a comparable test runtime.

The rest of this paper is organized as follows: Section 2 studies the effect of hyperparameters and the establishment of a neural network architecture. Section 3 provides a comprehensive study on different learning strategies including imbalance-aware processing and parameters. Section 4 lists the experimental results, followed by a conclusion in Sec. 5.

2 Convolutional Neural Network Architecture

Neural network architecture describes what layers are in the network and how the layers are connected together. Many studies have shown that network architecture can prominently impact model performance.[16–18] In this section, we will discuss the basic layer types of deep learning used in our study and their associated hyperparameters (i.e., parameters or settings that affect the architecture including the kernel size, pooling methods, and activation functions), based on which the effectiveness of our network architecture is further described.

2.1 *Convolutional Neural Network Elements*

2.1.1 *Convolution layer*

Convolution layers are the key structure in the CNN, which are applied for feature extraction. In neural networks, each layer can be regarded as a computational graph that consists of input nodes, edge weights, (In the context of neural networks, it is more common to use the term neuron weights.) and output nodes (or feature maps) that are obtained from the inner product between the input and weights. A characteristic of the convolution layer is that most of the neuron weights are shared, which enables common local feature extraction.[19] There are only a small number of unique weights known as the convolution kernel. Because the input layout image is square, we choose square kernel for better compatibility. Obviously, the kernel size is much smaller than the input size. The operation within each convolution layer becomes the kernel scanning all over the input and within each scanning step, one output node is calculated from the inner product between the kernel and a region of the input, as shown in

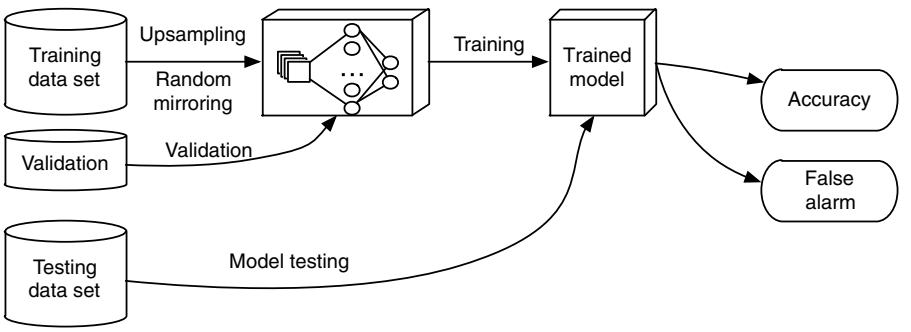

Fig. 2 The proposed deep learning-based hotspot detection flow.

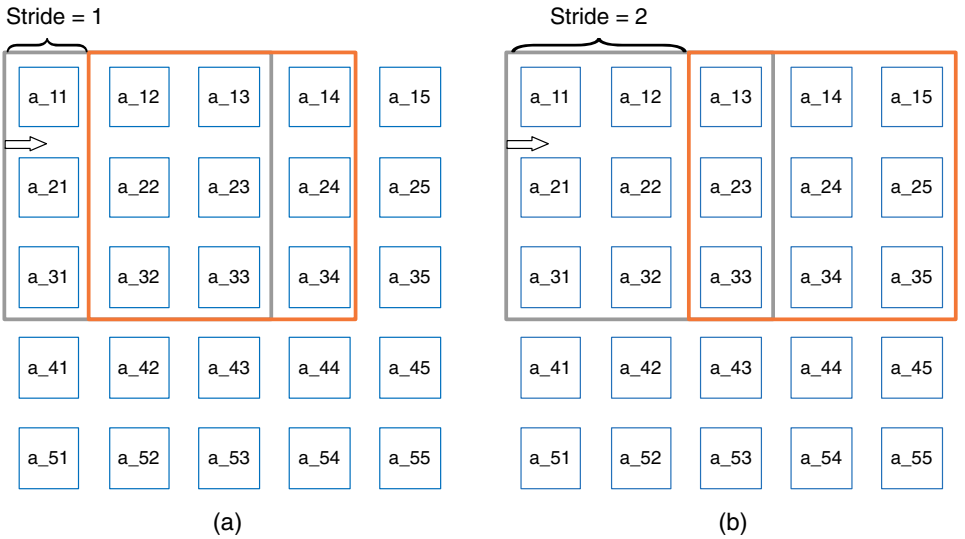

Fig. 3 3×3 convolution example with stride set to (a) 1 and (b) 2, respectively.

$$\mathbf{I} \otimes \mathbf{K}(x, y) = \sum_{i=1}^{c} \sum_{j=1}^{m} \sum_{k=1}^{m} \mathbf{I}(i, x - j, y - k)\mathbf{K}(i, j, k), \quad (1)$$

where $\mathbf{I}(i, j, k)$ is the pixel value of location (j, k) at the i'th feature map output of the previous layer, whereas $\mathbf{K}$ is the convolution kernel. After scanning from the upper-right to the bottom-left corner, a new feature map is generated. Moreover, stacked convolution layers grasp image attributes in different hierarchical levels and generate better feature representation.

A convolution layer is specified by hyperparameters including kernel size m (here, we assume square kernel) and stride s. The stride defines the overlapping between scanning windows, which is set to one in Eq. (1). Note that when efficient dimension reduction is required, stride can be set to any positive integer. Figure 3 shows two scenarios, where stride is set to 1 and 2, respectively.

In traditional computer vision classification tasks, a non-unit stride and a large receptive field (kernel size) are induced for both dimension reduction and feature learning with good performance.[14,20] However, mask layout patterns are more sensitive to small variations than normal objects, as tiny shifts of pattern edges may change a hotspot to a nonhotspot and vice-versa. As far as hotspot detection is concerned, extracting detailed local information in a global scheme is a good choice. Inspired by the work of the ImageNet Challenge 2014,[21] we apply a fixed small receptive field (3×3) to gain sufficient local attributes within each convolutional layer while not extracting image information, pixel by pixel, with a kernel size that is too small. In Fig. 4, we present training curves with different kernel sizes, and we can see that the 3×3 kernel size generates stable and relatively lower validation losses with a limited iteration number. Even when all three kernel sizes have similar performance over time, a kernel size of 3×3 is preferable for the sake of training time. Note that padding is changed along with the kernel sizes in order to maintain a constant output layer size.

Because local regions in the layout are highly correlated, large overlap windows promise to gather sufficient information.

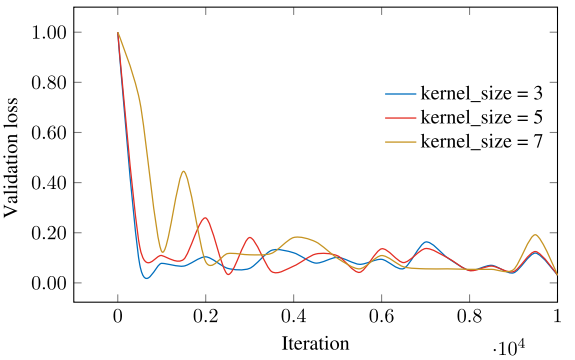

Fig. 4 Comparing different kernel sizes.

Additionally, previous work has shown that a smaller stride ensures that the model is translationally invariant.[21] Therefore, we employ stride $s = 2$ in the first convolution layer to perform dimension reduction and stride $s = 1$ for other convolution layers to acquire enough information.

2.1.2 *Rectified linear unit*

Activation functions have been widely used in multilayer perceptron (neural networks) to perform output regularization.[14,17,22,23] Prevailing candidates are sigmoid and tanh functions that scale the entries of each layer into an interval of (0,1) and $(-1, 1)$, respectively. However, these activation functions suffer from a gradient vanishing problem[24] because of the chained multiplication of numbers within the range (0,1) during back-propagation. In other words, the training of early layers is inefficient in deep neural networks. Nari and Hinton[25] proposed a rectified linear unit (ReLU) for a restricted Boltzmann machine that performs element-wise operations on a feature map as shown in

$$\mathrm{ReLU}(x) = \max\{x, 0\}. \quad (2)$$

The equation indicates that by applying a ReLU, the feature map no longer has an upper boundary, but network sparsity (i.e., the number of nodes with zero response) is

Table 1 Comparison on different activation functions.

Activation function	Expression	Validation loss
ReLU	$\max\{x, 0\}$	**0.16**
Sigmoid	$\frac{1}{1+\exp(-x)}$	87.0
TanH	$\frac{\exp(2x)-1}{\exp(2x)+1}$	0.32
BNLL	$\log[1+\exp(x)]$	87.0
WOAF	NULL	87.0

Bold value stands for the best results among all the values in the columns or rows.

augmented and the model is still nonlinear. In particular, overfitting can be reduced with a sparse feature map. These properties are necessary to train a deep neural network model efficiently and reliably. Because of these reasons, each convolution layer is followed by a ReLU in convolutional neural network design.

To evaluate the effectiveness of a ReLU layer, we replace the ReLU with several classical activation functions, including sigmoid, tanh, and binomial normal log likelihood (BNLL) during training. The experimental results in Table 1 show that the deep neural network model with ReLU layers reports the best performance (0.16 of validation loss). We can also notice that the networks with sigmoid, BNLL, or without active functions (WOAF) suffer from extremely large validation loss.

2.1.3 *Pooling layer*

In CNN design, following one or more convolution and ReLU layers, the pooling layer extracts the local region statistical attributes in the feature map. Typical attributes include the maximum or average value within a predefined region (kernel) that corresponds to max pooling and average pooling as shown in Fig. 5. Similar to a convolution layer, a pooling kernel also scans over the feature map and generates more compact feature representations.

Pooling layers make the feature map invariant to minor translations of the original image and a large pooling kernel enhances this property. When we perform hotspot detection, however, we do not benefit much from translation-invariance since pattern locations do affect the classification result. In this situation, the main purpose of pooling layers in this work is to reduce the feature map dimensions.[26] To decide the best pooling method, we compare the performance of models with maximum pooling, average pooling, and random choosing a value from the scan window (stochastic pooling). As shown in Table 2, max and average pooling do not show obvious result differences and both pooling methods outperform the stochastic approach. For best results in this work, we use max pooling in our network.

Table 2 Comparison on different pooling methods.

Pooling method	Kernel	Test accuracy (%)
Max	2×2	**96.25**
Ave	2×2	**96.25**
Stochastic	2×2	90.00

2.1.4 *Fully connected layer*

Following the convolution hierarchy, a feature map will become smaller and deeper, and finally reach the unit size. The layer generating unit size feature is called fully connected (FC) layer. The FC layers form the last several layers of the convolutional neural network and can be regarded as a special case of a convolution layer with a kernel size equal to the feature map size of the previous layer. If the kernel and feature map are square, we have the following parameter relationship as in Eq. (1)

$$\text{size of}(\mathbf{I}) = \text{size of}(\mathbf{K}). \tag{3}$$

Note that vertexes reflect the probability of an object being predicted as each class. The main difference between a flattened deep learning feature vector and machine learning features (e.g., CCAS feature and density features) is that each node in the FC layer contains the information from a global view while user-designed features extracted through sampling may lose spatial information.

The FC layers in the deep neural network also play a role to prevent overfitting. Srivastava et al.[27] discovered that when there is a random percentage drop of vertexes and connected neurons in an FC layer during training, deep neural network performance significantly improves. To evaluate the effect of dropout, we trained the network with variant dropout ratios from 0 to 0.8. As shown in Fig. 6, the validation curve indicates that the model performance is best when the dropout ratio is between 0.4 and 0.7, therefore, we set the dropout ratio to 0.5 in our FC layer.

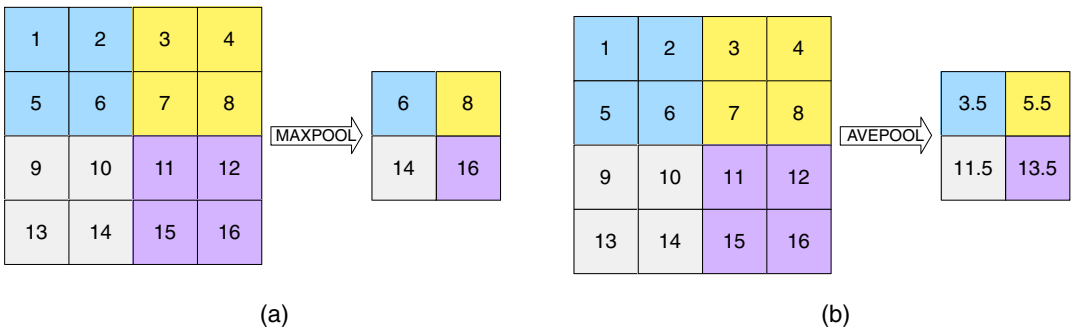

Fig. 5 Examples of (a) max pooling and (b) average pooling.

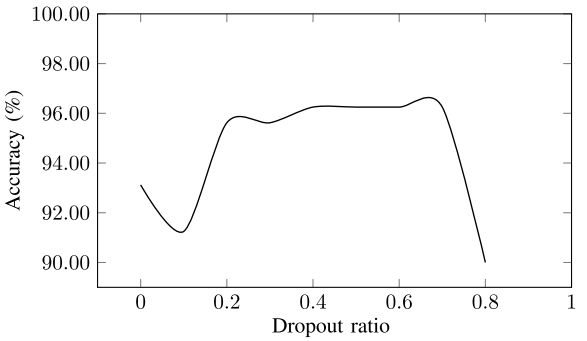

Fig. 6 Dropout ratio effect.

2.2 *Architecture Summary*

According to the above analyses on deep neural network elements and layout clip size, we can apply the architecture as shown in Fig. 7. The architecture differs from Ref. 28 on the first two layers. Because the layout clip of 1024×1024 nm^2 is much larger than the image size of a conventional computer vision benchmark dataset such as ImageNet,[29] in Ref. 28, we set a 2×2 convolution and 2×2 pooling with a stride of 2 to reduce the feature map to benefit both storage and training. However, the CNN is applied on post-OPC masks, small edge displacements will be filtered by the nonoverlapped pooling. For that reason, we replace the first two layers with two 3×3 convolution layers with stride 2 that reduce the input dimension while attaining the edge displacement information. Following the first two convolution layers are four convolution stages, each of which contains three 3×3 unit stride convolutions and one stride-2 2×2 pooling.

The feature map depth is doubled at the first four convolution stages to obtain deeper representation. After layer-by-layer abstraction, deep feature representation is obtained and flattened by the first FC layer. However, the flattened feature vector still has a high dimension. To generate the final output, we add two additional FC layers to reduce the output vertex number to two, then output hotspot/nonhotspot probability for each input target. Note that each convolution layer is followed by one ReLU. More configuration information is listed in Table 3.

Table 3 Neural network configuration.

Layer	Kernel size	Stride	Padding	Output vertexes
Conv1-1	$3 \times 3 \times 4$	2	0	$512 \times 512 \times 4$
Conv1-2	$3 \times 3 \times 4$	2	0	$256 \times 256 \times 4$
Conv2-1	$3 \times 3 \times 8$	1	1	$256 \times 256 \times 8$
Conv2-2	$3 \times 3 \times 8$	1	1	$256 \times 256 \times 8$
Conv2-3	$3 \times 3 \times 8$	1	1	$256 \times 256 \times 8$
Pool2	2×2	2	0	$128 \times 128 \times 8$
Conv3-1	$3 \times 3 \times 16$	1	1	$128 \times 128 \times 16$
Conv3-2	$3 \times 3 \times 16$	1	1	$128 \times 128 \times 16$
Conv3-3	$3 \times 3 \times 16$	1	1	$128 \times 128 \times 16$
Pool3	2×2	2	0	$64 \times 64 \times 16$
Conv4-1	$3 \times 3 \times 32$	1	1	$64 \times 64 \times 32$
Conv4-2	$3 \times 3 \times 32$	1	1	$64 \times 64 \times 32$
Conv4-3	$3 \times 3 \times 32$	1	1	$64 \times 64 \times 32$
Pool4	2×2	2	0	$32 \times 32 \times 32$
Conv5-1	$3 \times 3 \times 32$	1	1	$32 \times 32 \times 32$
Conv5-2	$3 \times 3 \times 32$	1	1	$32 \times 32 \times 32$
Conv5-3	$3 \times 3 \times 32$	1	1	$32 \times 32 \times 32$
Pool5	2×2	2	0	$16 \times 16 \times 32$
FC1	—	—	—	2048
FC2	—	—	—	512
FC3	—	—	—	2

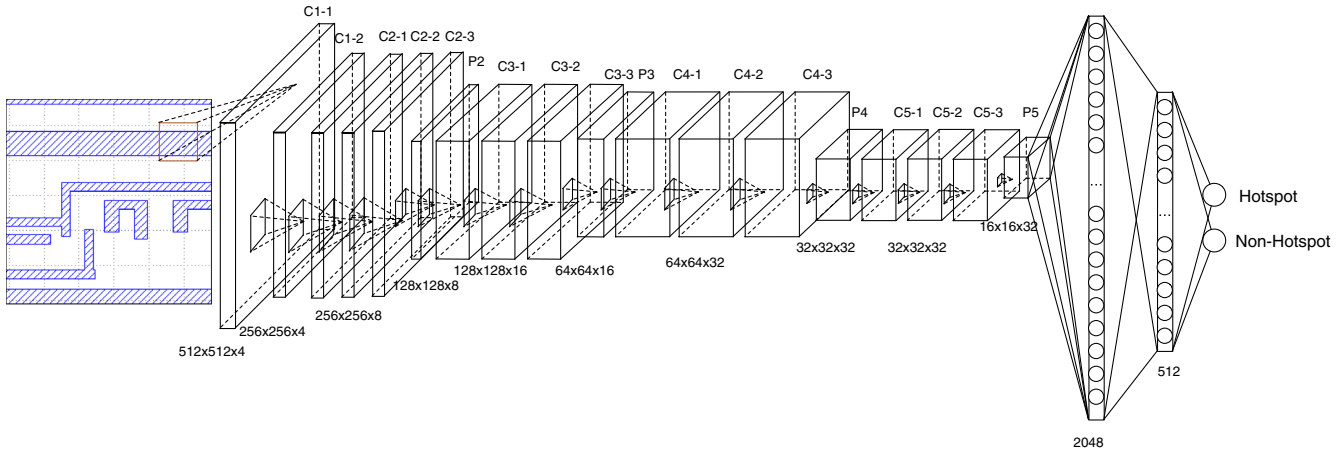

Fig. 7 Architecture overview.

3 Imbalance Aware Learning

The previous sections describe the effects of different network configurations. Preliminary results show that our designed CNN has the potential to perform well on hotspot detection problems. Because hotspot patterns are always in the minority in VLSI mask design, the training dataset is highly imbalanced. In this situation, a neural network is no longer reliable because a trained model with high classification accuracy may still suffer from a high number of false negative results (missing hotspots), which is fatal in hotspot detection problems. The rest of this section will focus on the imbalanced layout dataset and basic learning strategies.

3.1 *Random-Mirror Flipping and Upsampling*

Existing methods to handle imbalanced data include multilabel learning,[30] majority downsampling,[31] pseudoinstance generation,[32] and so on, which are general solutions aiming to make the dataset more balanced. Because of the nature of mask layout and CNN, these approaches are not directly applicable. For instance, Zhang et al.[30] assigned different labels to the majority to balance the number of instances in each category. However, this may cause insufficient training samples of individual classes, as a large training dataset is needed to efficiently train deep neural networks. Similarly, majority downsampling cannot apply to the deep neural network-based method either. Recently, Shin and Lee[33] performed layout pattern shifting to artificially generate hotspot patterns, but the approach might be invalid because a shift larger than 10 nm is enough to change the layout pattern attribute. It should be noted that a straightforward way to handle imbalanced mask patterns is naïve upsampling, i.e., duplicating hotspot samples. Here, we use α to denote the upsampling factor and intuitively

$$\alpha = \frac{\#\text{ of nonhotspot}}{\#\text{ of hotspot}}. \tag{4}$$

As it is normal to find only one hotspot instance within more than 100 samples, directly duplicating them may raise the following problems: (1) in mini-batch gradient descent (MGD),[26] if one batch contains too many identical instances, a large gradient will be generated in one direction that will lead the training procedure away from the optimal solution. (2) Even with duplicated instances, hotspot pattern types are still limited. Therefore, the trained model will suffer from overfitting and have low detection accuracy.

Assume that the source of the lithography system is up–down and left–right symmetric, we first propose augmenting the training dataset with mirror-flipped and 180-deg-rotated version of the original layout clips to enhance the effectiveness of hotspot upsampling. In this case, each hotspot instance has equal probabilities of taking one of four orientations (see Fig. 8). Because MGD randomly picks training instances for some mini-batch size, we fix the batch size to 8 to ensure diversity of each mini-batch. Overfitting can also be reduced through random mirroring. We study the impact of different upsampling factors on a highly imbalanced dataset (i.e., hotspot 95 and nonhotspot 4452). The experimental results indicate that validation performance does not show further improvement when the upsampling factor increases beyond a certain value (~20) (see Fig. 9).

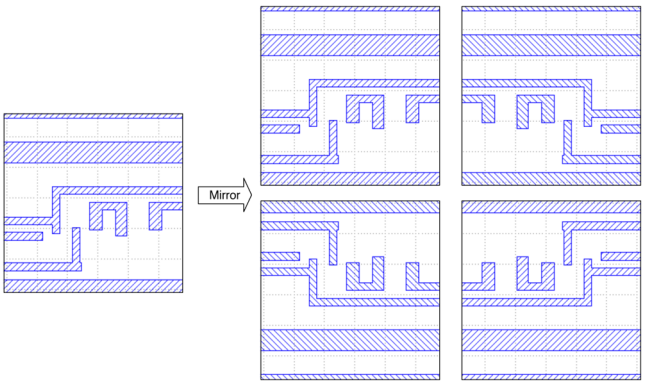

Fig. 8 Random-mirror flipping with X, Y, and XY.

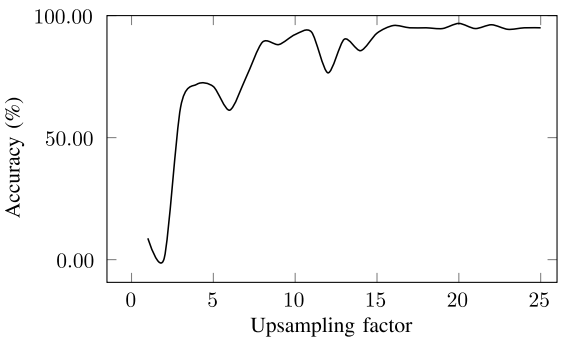

Fig. 9 Upsampling effects.

3.2 *Training Neural Networks*

We use the prevailing MGD method to train the neural network. As described in Sec. 2, there are lots of hyperparameters associated with learning that can affect learning speed and determine the final model performance.[16] This makes tuning hyperparameters a very important part of neural network design.

3.2.1 *Learning rate*

In MGD, the learning rate γ defines how fast the neuron weights are updated. When the gradient on one node $\frac{\partial l}{\partial w_i}$ is obtained, connected neuron weight is updated according to the following strategy:

$$w_i = w_i - \gamma \frac{\partial l}{\partial w_i}. \tag{5}$$

In general, the classifier will learn faster with a larger γ, but it may never reach an optimal solution. Conversely, we cannot benefit much with a smaller γ either, as under this condition, the learning procedure is time consuming and can be easily trapped at the local minimum and saddle point. Therefore, it is reasonable to apply an adaptive learning rate that starts from some initial value and decays after a fixed iteration interval. This scheme ensures a stable and quick training process. We study the effect of γ by choosing the initial values of 0.1, 0.01, 0.001, 0.0001, and decaying them by a factor of 10 at every 500 iterations. The corresponding learning curves are presented in Fig. 10. We can see that the network is trained more efficiently with

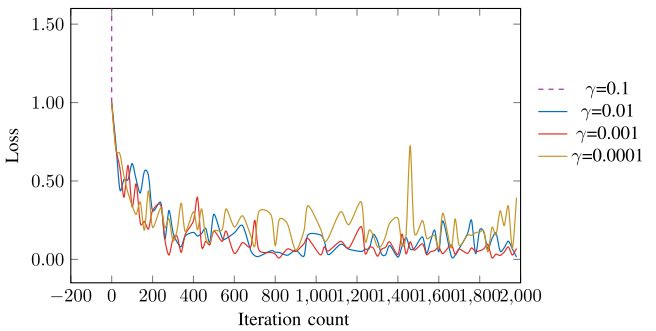

Fig. 10 Effect of initial learning rate.

$\gamma = 0.001$. It also shows that nonsuitable initial learning rate might cause an unstable network. In particular, when $\gamma = 0.1$, the network cannot learn any useful information from the training dataset and the training loss diverges.

3.2.2 *Momentum*

Polyak[34] analyzed the physical meaning of gradient descent and proposed the momentum method, which can speed up convergence in iterative learning. The key idea is to modify the weight update scheme according to

$$v = \mu v - \gamma \frac{\partial l}{\partial w_i}, \tag{6}$$

$$w_i = w_i + v, \tag{7}$$

where v is the weight update speed initialized as 0 and μ is the momentum factor. Equations (6) and (7) indicate that in gradient descent optimization, the update speed is directly associated with the loss gradient. The momentum μ here slightly reduces the update speed and produces better training convergence.[35] Momentum by default is set to 0.9; however, the efficiency varies for different applications. We test normal momentum values of 0.5, 0.9, 0.95, 0.99 (see Table 4), as suggested in CS231n.[36] The results show that the momentum of 0.99 has the lowest validation loss.

3.2.3 *Weight decay*

A common problem in training large neural networks is that when a training dataset is not informative, network overfitting is more likely to happen. Instead of adding a weight penalty on the loss function, constraints can be applied on the gradient descent procedure by introducing a weight decay term $-\gamma w_i$ when learning neuron weights.[37] Then, Eqs. (6) and (7) become

Table 4 Momentum configuration.

μ	Learning rate	Validation loss
0.5	0.001	0.21
0.9	0.001	0.22
0.95	0.001	0.21
0.99	0.001	**0.16**

Table 5 Effect of weight decay.

λ	Learning rate	Momentum	Validation loss
10^{-3}	0.001	0.99	0.95
10^{-4}	0.001	0.99	1.19
10^{-5}	0.001	0.99	0.37
10^{-6}	0.001	0.99	**0.2**

$$v = \mu v - \gamma \frac{\partial l}{\partial w_i} - \gamma \lambda w_i, \tag{8}$$

$$w_i = w_i + v, \tag{9}$$

where λ is the decay factor and is usually around 10^{-4} to 10^{-6}.[22] A virtue of Eq. (8) is that when neuron weights are small, the term $\gamma \lambda w_i$ is ignorable and neuron weights can get penalties from the decay factor when they are large. Therefore, neuron weights can be kept from increasing infinitely. To show the effect of different weight decay factors, we train the neural network with the solver configuration listed in Table 5, and the results show that the model is learned more efficiently with $\lambda = 10^{-6}$.

3.2.4 *Weight initialization*

The weight initialization procedure determines the initial values assigned to each neuron before the gradient descent update starts. Because weight initialization defines the optimization starting point, an improper initialization may cause bad performance or even failed training, and it requires careful determination.

Many approaches were studied in the literature, and in most applications, random Gaussian enjoys the best performance. However, the results are highly affected by standard deviation. To make the training procedure more efficient, initial weight distribution should ensure the variance remains invariant when passing through each layer. Otherwise, the neuron response and the gradient will suffer from unbounded growth or may vanish. To address this problem, Glorot and Bengio[38] proposed an initialization method by taking the variance of each layer into consideration, and experiment results have shown that it is more efficient than general Gaussian initialization. Consider the layer represented as follows:

$$y = \sum_{i=1}^{N} x_i w_i, \tag{10}$$

where x_i is the i'th input and w_i is the corresponding neuron weight. The variance relationship can be written as

$$\hat{V}(y) = \sum_{i=1}^{N} \hat{V}(x_i)\hat{V}(w_i). \tag{11}$$

We assume all the variables are identically distributed, then

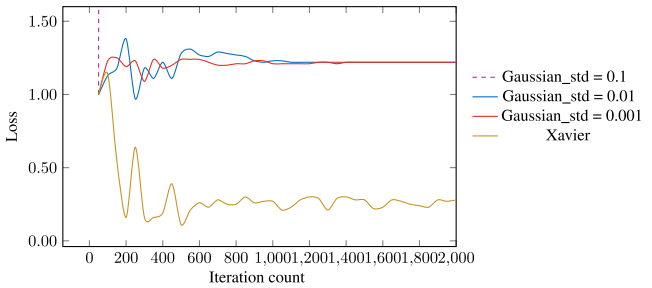

Fig. 11 Importance of weight initialization.

$$\hat{V}(y) = N\hat{V}(x_i)\hat{V}(w_i). \tag{12}$$

Equation (12) indicates that input and output have the same variance if and only if

$$N\hat{V}(w_i) = 1, \tag{13}$$

$$\hat{V}(w_i) = \frac{1}{N}, \tag{14}$$

which is the rule of Xavier weight initialization. Figure 11 shows the validation loss during training and illustrates that Xavier outperforms ordinary Gaussian initialization. It is also notable that improper weight initialization might cause a training failure (dashed curve).

4 Experimental Results

We have discussed how our neural network architecture is designed and some techniques adopted for applying hotspot detection. In this section, we will focus on the experimental results. We first introduce evaluation metrics and the ICCAD 2012 contest benchmark[15] information. Then, we exemplify feature learning by visualizing intermediate neuron responses. Next, we compare our framework with other deep learning solutions in the hotspot detection literature, and finally, we compare the result with two machine learning-based hotspot detectors that have achieved satisfactory performance. All experiments are conducted using caffe[39] on a platform with an Intel Xeon processor and a GTX Titan graphic card.

4.1 *Evaluation Metrics and Benchmark Information*

As described in Ref. 15, a good hotspot detector should be able to recognize hotspot patterns as much as possible and have a low false alarm rate. Therefore, the following evaluation metrics are adopted:

Definition 1 (accuracy). The ratio between the number of correctly detected hotspot clips and the number of all hotspot clips.[15]

Definition 2 (false alarm). The number of nonhotspot clips that are reported as hotspots by the detector.For the actual design flow, lithographic simulation should be performed on all detected hotspot clips including false alarms. As suggested in Ref. 9, a unified runtime evaluation metric called overall detection and simulation time (ODST) is defined.[15]

Table 6 ICCAD 2012 contest benchmark.

Bench	Training HS#	Training NHS#	Testing HS#	Testing NHS#
ICCAD-1	99	340	226	3869
ICCAD-2	174	5285	498	41,298
ICCAD-3	909	4643	1808	46,333
ICCAD-4	95	4452	177	31,890
ICCAD-5	26	2716	41	19,327

Definition 3 (ODST). The sum of all lithographic simulation time for clips predicted as hotspots and the elapsed deep learning model evaluation time.[9]

Note that an industrial lithography simulator[40] adopted in this paper takes 10 s to perform lithography simulation on each clip, therefore, ODST can be calculated using

$$\text{ODST} = \text{test time} + 10\ \text{s} \times \#\,\text{of false alarm}. \tag{15}$$

The evaluation benchmark contains five test cases: "ICCAD-1" to "ICCAD-5." The benchmark details are listed in Table 6. The "training HS#" and "training NHS#" columns denote the number of hotspot and nonhotspot patterns in training sets. The "testing HS#" and "testing NHS#" columns are for the number of hotspots and nonhotspots in testing sets. We can see that in the training set, the number of hotspots and the number of nonhotspots are highly imbalanced, which induces pressure on normal neural network training procedures.

4.2 *Layer Visualization*

Deep neural networks enhance classification tasks by learning representative features efficiently. In our designed network architecture, there are five convolution stages that extract different levels of feature representations. Figure 12 shows the neuron response for one input example. Subfigures "origin," "pool1," "pool2," "pool3," "pool4," and "pool5" correspond to the original clip and the neuron response of the first, second, third, fourth, and fifth pooling layers, respectively. Tiles within each subfigure are features extracted by specific convolution kernels. The visualization of neuron responses illustrates that different convolution kernels focus on different image properties and learned feature maps are associated with each other as well as the original input.

4.3 *Receiver Operating Characteristic*

Hotspot detection is a binary classification problem where only positive or negative is reported by the classifier. There are four cases of prediction results: (1) true positive (TP): hotspot instances that are predicted as hotspots; (2) false positive (FP): nonhotspot instances that are predicted as hotspots; (3) true negative (TN): nonhotspot instances that are predicted as nonhotspot; (4) false negative (FN): hotspot instances that are predicted as nonhotspots.

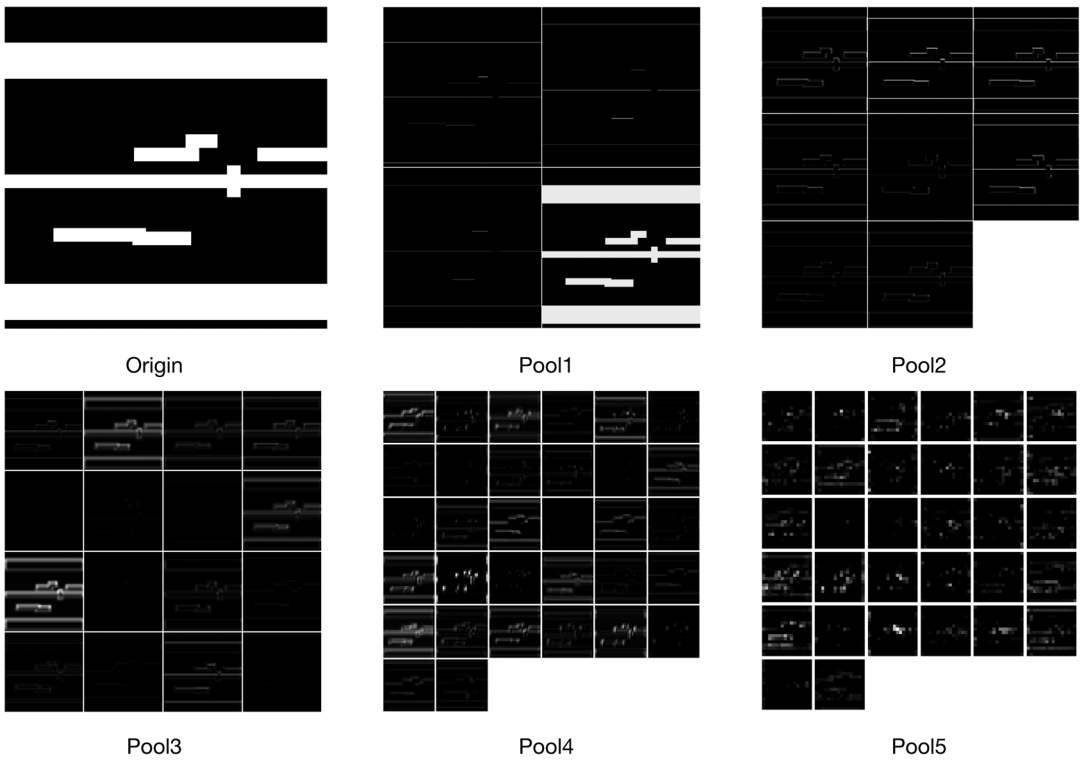

Fig. 12 Neuron response of pooling layers in five convolution stages.

True positive rate (TPR) and false positive rate (FPR) are defined as follows:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{16}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}, \tag{17}$$

which correspond to accuracy and false alarm defined in our work, respectively. We use a receiver operating characteristic (ROC) curve to depict the trade-off between TPR and FPR. In this experiment, we train the CNN model with combined 28-nm benchmarks (`ICCAD2` to `ICCAD5`) and obtain the corresponding ROC curve as in Fig. 13.

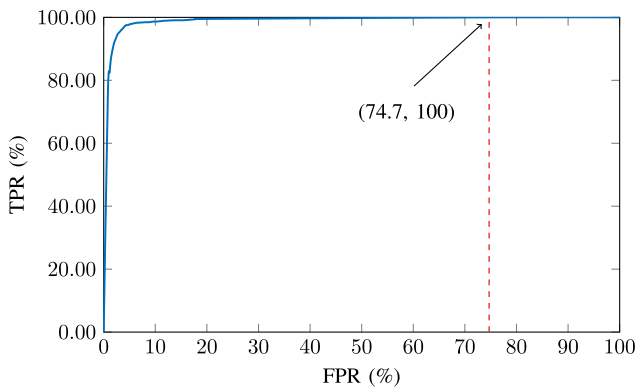

Fig. 13 The ROC curve, where the *x*-axis corresponds to false alarm rate, whereas the *y*-axis is hotspot detection accuracy. Note that TPR reaches 100% at the cost of FPR of 74.6% (red dashed line).

4.4 *Downscaling on Input Layout Images*

By default configuration, layout patterns are converted into images with 1-nm resolution, which results in a large clip image size (1024×1024). To improve runtime, it is worth examining the impact of layout resolution on hotspot detection accuracy.

In this experiment, we conduct density-based downscaling (In this paper, we refer to downscaling as changing the resolution of a layout instead of proportionally reducing the design pitch) on the layout images (implemented by doing average pooling). Figure 14 shows the performance of models trained with 2×2, 4×4, 8×8, and 16×16 downscaled layout images, where each pixel in the scaled image has an average value of the corresponding square region of the original image. Although smaller input size ensures faster feed-forward time, the runtime improvements are limited, possibly because of other bottlenecks. Also, detection accuracy drops due to the information loss of density-based downscaling. To pursue a higher hotspot detection accuracy, we still apply the original net (Fig. 7) for the following experiments.

4.5 *Result Comparison with Existing Deep Learning Flow*

To the best of our knowledge, there were two attempts that applied deep learning on hotspot detection.[33,41] Because no detailed results are reported in Ref. 41, we only compare our framework with Ref. 33. Shin and Lee[33] proposed a four-level convolutional neural network to perform hotspot classification. Table 7 lists the network configuration, which differs from our work in three aspects: (1) a 5×5 kernel is employed in each convolution layer while we prefer a smaller kernel size because the layout is sensitive to

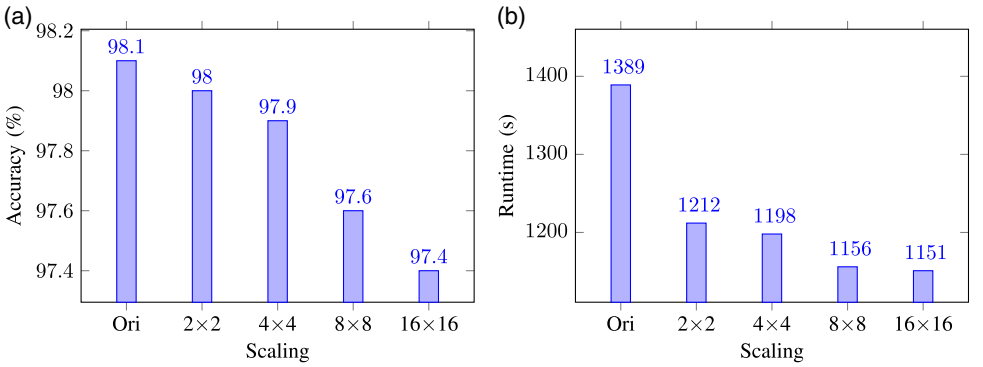

Fig. 14 The effect of scaling on layout images, where the *x*-axis corresponds to scale down factors, whereas *y*-axes are (a) hotspot detection accuracy and (b) test runtime.

variation. (2) The network scale is much smaller than ours (10 layers versus 21 layers). (3) In preprocessing, the training and testing datasets in Ref. 33 are sampled from layout clips with 10-nm precision (10 nm for each pixel) and training hotspot patterns are randomly shifted to avoid imbalance problems that may cause unreliable training instances because there is enough nanometer level variation to modify a layout clip attribute.

Experimental results are listed in Table 8. Columns "FA#," "CPU (s)," "ODST (s)," and "accu. (%)," correspond to the number of false alarms, the running time of the prediction flow, ODST as defined above, and the hotspot detection accuracy, respectively. The rows "ICCAD-1" to "ICCAD-5" are the results of five test cases, where the row "average" lists the average value of four metrics, and the row "ratio" offers normalized comparison by setting our experimental result to 1.0. Note that for different test cases, the instance numbers are different, for accuracy, we adopted a weighted average.

The comparison shows that detection accuracy of our framework is better than Ref. 33 for each test case and has a 2.3% advantage of average detection accuracy. As far as the false alarm is concerned, Ref. 33 takes 104% more ODST than ours. Results also indicate that our CNN architecture is effective and efficient. Because we introduce additional parameters for the network, the average detection accuracy drops 0.2% compared to our previous architecture in Ref. 28 caused by overfitting effect.

Table 7 Neural network configuration of Ref. 33.

Layer	Kernel size	Output vertexes
Conv1	$5\times5\times25$	—
Pool1	2×2	52×25
Conv2	$5\times5\times40$	—
Pool2	2×2	24×40
Conv3	$5\times5\times60$	—
Pool3	2×2	10×60
Conv4	$5\times5\times80$	—
Pool4	2×2	3×80
FC1	—	100
FC2	—	2

4.6 *Results Comparison with Machine Learning Hotspot Detectors*

Many machine learning technologies were explored for layout hotspot detection and achieved better performance than the traditional pattern matching approach. Here, we compare our experiment with two representative machine learning-based hotspot detectors[8,9] that adopt support vector machine and smooth boosting, respectively.

As shown in Table 9, columns and rows are similarly defined as in Table 8. For detection accuracy, our framework achieves the best results on cases ICCAD-2 (98.7%) and ICCAD-3 (98.0%), meanwhile the framework gains similar results on rest cases with only a 3.2% difference on ICCAD-4. On average, the proposed deep learning approach outperforms[8] on accuracy (consistency with total number of correctly detected hotspots) by 5.3% and achieves the same detection accuracy as Ref. 9. Our framework also has approximately a 50,000-s and 3600-s ODST advantage, respectively.

For some test cases, deep learning does not perform better than machine learning. The main reason is that the size of training dataset is much smaller than required to efficiently train a deep neural network and machine learning is, therefore, more suitable. However, with advanced VLSI technology nodes, layout patterns will be more and more complicated, and in real applications, deep learning has the potential to perform better, thanks to its robustness and effectiveness.

4.7 *Performance Evaluation on Post-OPC Layouts*

Hotspot detection tasks for post-OPC mask layouts are more challenging than the datasets above. We conduct the experiment on three industrial post-OPC datasets and compare the results with the original architecture, as shown in Table 10. With the refined architecture, the CNN model improves the average hotspot detection accuracy by 2.6% and reduces ODST by 15.9%.

Table 8 Performance comparisons with Ref. 33.

	JM3'16[33]				SPIE'17[28]				Ours			
Bench	FA#	CPU (s)	ODST (s)	Accu. (%)	FA#	CPU (s)	ODST (s)	Accu. (%)	FA#	CPU (s)	ODST (s)	Accu. (%)
ICCAD-1	**386**	**15**	**3875**	95.1	147	51	1521	99.6	1037	50	10,420	**100**
ICCAD-2	1790	**208**	18,108	98.8	561	390	6000	**99.8**	**83**	501	**1331**	98.7
ICCAD-3	7077	**322**	71,092	97.5	2660	434	27,034	97.8	**3108**	546	**31,626**	**98.0**
ICCAD-4	892	**129**	9049	93.8	1785	333	18,183	**96.4**	**296**	346	**3306**	94.5
ICCAD-5	**172**	**82**	**1802**	92.7	242	232	2652	**95.1**	394	264	4204	**95.1**
Average	2063	**151**	20,781	96.7	1079	288	11,078	**98.2**	**984**	341	**10,177**	98.0
Ratio	—	—	2.04	0.99	—	—	1.09	1.01	—	—	1.0	1.0

Table 9 Performance comparisons with state-of-the-art machine learning.

	TCAD'15[8]				ICCAD'16[9]				Ours			
Bench	FA#	CPU (s)	ODST (s)	Accu. (%)	FA#	CPU (s)	ODST (s)	Accu. (%)	FA#	CPU (s)	ODST (s)	Accu. (%)
ICCAD-1	1493	38	14,968	94.7	**788**	**10**	**7890**	**100**	1037	50	10,420	**100**
ICCAD-2	11,834	234	118,574	98.2	544	**103**	5543	99.4	**83**	501	**1331**	**98.7**
ICCAD-3	13,850	778	139,278	91.9	**2052**	**110**	**20,630**	97.5	3108	546	31,626	**98.0**
ICCAD-4	3664	356	36,996	85.9	3341	**69**	33,478	**97.7**	**296**	346	**3306**	94.5
ICCAD-5	1205	**20**	12,070	92.9	**94**	41	**980**	**95.1**	394	264	4204	**95.1**
Average	6409	285	64,377	92.9	1363	67	13,702	**98.0**	**984**	341	**10,177**	**98.0**
Ratio	—	—	6.33	0.948	—	—	1.35	**1.0**	—	—	**1.0**	**1.0**

Table 10 Performance evaluation on post-OPC layouts.

	SPIE'17[28]				Ours			
Benchmarks	FA#	CPU (s)	ODST (s)	Accu. (%)	FA#	CPU (s)	ODST (s)	Accu. (%)
Industry1	519	271	5461	97.7	**328**	297	**3577**	**98.4**
Industry2	760	362	7962	89.6	**676**	443	**7203**	**90.7**
Industry3	1966	416	20,076	77.1	**1686**	502	**17,362**	**83.4**
Average	1082	350	11,166	88.2	**897**	414	**9381**	**90.8**
Ratio	—	—	1.19	0.971	—	—	**1.0**	**1.0**

5 Conclusion

In this paper, we explore the feasibility of deep learning as an alternative approach for lithography hotspot detection in the submicron era. We study the effectiveness of the associated hyperparameters to make the architecture and the learning procedures match well with the layout nature. In particular, upsampling and random-mirror flipping are applied to address the side effects caused by imbalanced datasets.

The experimental results show that the designed network architecture is more robust and performs better than existing deep learning architectures and representative machine leaning approaches. This study also demonstrates that deep neural networks have potential to offer better solutions to some emerging design for manufacturability problems as circuit layouts advance to extreme scale.

Acknowledgments

This work was supported in part by the Research Grants Council of Hong Kong SAR (Project No. CUHK24209017). The authors would like to thank Evangeline F. Y. Young from CUHK, Yi Zou and Lauren Katzive from ASML for helpful comments.

References

1. J. Kim and M. Fan, "Hotspot detection on post-OPC layout using full chip simulation based verification tool: a case study with aerial image simulation," *Proc. SPIE* **5256**, 919 (2003).
2. E. Roseboom et al., "Automated full-chip hotspot detection and removal flow for interconnect layers of cell-based designs," *Proc. SPIE* **6521**, 65210C (2007).
3. Y.-T. Yu et al., "Accurate process-hotspot detection using critical design rule extraction," in *ACM/IEEE Design Automation Conf. (DAC)*, pp. 1167–1172 (2012).
4. W.-Y. Wen et al., "A fuzzy-matching model with grid reduction for lithography hotspot detection," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **33**(11), 1671–1680 (2014).
5. D. Ding, J. A. Torres, and D. Z. Pan, "High performance lithography hotspot detection with successively refined pattern identifications and machine learning," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **30**(11), 1621–1634 (2011).
6. J.-R. Gao, B. Yu, and D. Z. Pan, "Accurate lithography hotspot detection based on PCA-SVM classifier with hierarchical data clustering," *Proc. SPIE* **9053**, 90530E (2014).
7. B. Yu et al., "Accurate lithography hotspot detection based on principal component analysis-support vector machine classifier with hierarchical data clustering," *J. Micro/Nanolithogr. MEMS MOEMS* **14**(1), 011003 (2015).
8. Y.-T. Yu et al., "Machine-learning-based hotspot detection using topological classification and critical feature extraction," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **34**(3), 460–470 (2015).
9. H. Zhang, B. Yu, and E. F. Y. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 1–8 (2016).
10. T. Matsunawa et al., "A new lithography hotspot detection framework based on AdaBoost classifier and simplified feature extraction," *Proc. SPIE* **9427**, 94270S (2015).
11. T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical bayes model," *J. Micro/Nanolith. MEMS MOEMS* **15**, 021009 (2015).
12. G. E. Hinton, "What kind of graphical model is the brain?" in *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 1765–1775 (2005).
13. G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.* **18**(7), 1527–1554 (2006).
14. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Conf. on Neural Information Processing Systems (NIPS)*, pp. 1097–1105 (2012).
15. A. J. Torres, "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 349–350 (2012).
16. Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*, G. B. Orr and K.-R. Müller, Eds., pp. 437–478, Springer, Heidelberg, Germany (2012).
17. L. Deng, "Three classes of deep learning architectures and their applications: a tutorial survey," in *APSIPA Transactions on Signal and Information Processing* (2012).
18. Y. Sun, X. Wang, and X. Tang, "Hybrid deep learning for face verification," in *IEEE Int. Conf. on Computer Vision (ICCV)*, pp. 1489–1496 (2013).
19. G. B. Huang, H. Lee, and E. Learned-Miller, "Learning hierarchical representations for face verification with convolutional deep belief networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 2518–2525 (2012).
20. M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conf. on Computer Vision (ECCV)*, pp. 818–833 (2014).
21. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint (2014).
22. L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, G. B. Orr and K.-R. Müller, Eds., pp. 421–436, Springer, Heidelberg, Germany (2012).
23. T. Xiao et al., "Learning deep feature representations with domain guided dropout for person re-identification," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 1249–1258 (2016).
24. S. Hochreiter et al., "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Neural Networks*, J. F. Kolen and S. C. Kremer, Eds., IEEE Press, Hoboken, New Jersey (2001).
25. V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Int. Conf. on Machine Learning (ICML)*, pp. 807–814 (2010).
26. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, Massachusetts (2016).
27. N. Srivastava et al., "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.* **15**(1), 1929–1958 (2014).
28. H. Yang et al., "Imbalance aware lithography hotspot detection: a deep learning approach," *Proc. SPIE* **10148**, 1014807 (2017).
29. O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vision* **115**(3), 211–252 (2015).
30. M.-L. Zhang, Y.-K. Li, and X.-Y. Liu, "Towards class-imbalance aware multi-label learning," in *Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 4041–4047 (2015).
31. W. W. Y. Ng et al., "Diversified sensitivity-based undersampling for imbalance classification problems," *IEEE Trans. Cybern.* **45**(11), 2402–2412 (2015).
32. H. He et al., "ADASYN: adaptive synthetic sampling approach for imbalanced learning," in *Int. Joint Conf. on Neural Networks (IJCNN)*, pp. 1322–1328 (2008).
33. M. Shin and J.-H. Lee, "Accurate lithography hotspot detection using deep convolutional neural networks," *J. Micro/Nanolithogr. MEMS MOEMS* **15**(4), 043507 (2016).
34. B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Comput. Math. Math. Phys.* **4**(5), 1–17 (1964).
35. I. Sutskever et al., "On the importance of initialization and momentum in deep learning," in *Int. Conf. on Machine Learning (ICML)*, pp. 1139–1147 (2013).
36. A. Karpathy, "Stanford University CS231n: convolutional neural networks for visual recognition," http://cs231n.github.io/neural-networks-3/ (07 March 2017).
37. J. Moody et al., "A simple weight decay can improve generalization," in *Conf. on Neural Information Processing Systems (NIPS)*, pp. 950–957 (1995).
38. X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, pp. 249–256 (2010).
39. Y. Jia et al., "Caffe: convolutional architecture for fast feature embedding," in *ACM Int. Multimedia Conf.*, pp. 675–678 (2014).
40. S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 271–274 (2013).
41. T. Matsunawa, S. Nojima, and T. Kotani, "Automatic layout feature extraction for lithography hotspot detection based on deep neural network," *Proc. SPIE* **9781**, 97810H (2016).

Haoyu Yang received his BEng degree from Tianjin University, Tianjin, China, in 2015. He is now a PhD student in the Department of Computer Science and Engineering, Chinese University of Hong Kong. His research interests include design for manufacturability and deep learning.

Luyang Luo is an undergraduate student in the Department of Computer Science and Engineering, Chinese University of Hong Kong. He has joined the CUHK summer research program under the supervision of Prof. Bei Yu. He is currently interested and involved in deep-learning related study.

Jing Su received his BS degree in physics from the University of Science and Technology of China in 2008 and his PhD in theoretical and computational AMO physics from the 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 multiple patterning, advanced mask optimization, and machine learning.

Chenxi Lin received his BS degree in electrical engineering from Peking University, China, in 2008, and his PhD from the University of Southern California in 2013. His PhD research focused on the optical characterization and optimal design of nanostructured semiconductor thin-film materials for photovoltaic applications. He is currently a senior design engineer in the Advanced Technologies Department at ASML Brion Technologies, with a strong interest in data science and its applications in optical lithography.

Bei Yu is currently an assistant professor in the Department of Computer Science and Engineering, Chinese University of Hong Kong. He has served in the editorial boards of *Integration, the VLSI Journal*, and *IET Cyber-Physical Systems: Theory and Applications*. His current research interests include combinatorial algorithm and machine learning with applications in VLSI computer aided design and cyber-physical systems.