PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

Imbalance aware lithography hotspot detection: a deep learning approach

Haoyu Yang, Luyang Luo, Jing Su, Chenxi Lin, Bei Yu

Haoyu Yang, Luyang Luo, Jing Su, Chenxi Lin, Bei Yu, "Imbalance aware lithography hotspot detection: a deep learning approach," Proc. SPIE 10148, Design-Process-Technology Co-optimization for Manufacturability XI, 1014807 (28 March 2017); doi: 10.1117/12.2258374

SPIE.

Event: SPIE Advanced Lithography, 2017, San Jose, California, United States

Imbalance Aware Lithography Hotspot Detection: A Deep Learning Approach

Haoyu Yang[1], Luyang Luo[1], Jing Su[2], Chenxi Lin[2] and Bei Yu[1]
[1]The Chinese University of Hong Kong, NT, Hong Kong
[2]ASML Brion Inc., CA 95054, USA

ABSTRACT

With the advancement of VLSI technology nodes, light diffraction caused lithographic hotspots have become a serious problem affecting 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 extreme scaling of transistor feature size and more and more complicated layout patterns, 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. In this paper, we present a deep convolutional neural network (CNN) targeting representative feature learning in lithography hotspot detection. We carefully analyze impact and effectiveness of different CNN hyper-parameters, through which a hotspot-detection-oriented neural network model is established. Because hotspot patterns are always minorities in VLSI mask design, the training data set 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 high false negative results (missing hotspots), which is fatal in hotspot detection problems. To address the imbalance problem, we further apply minority upsampling and random-mirror flipping before training the network. Experimental results show that our proposed neural network model achieves highly 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.

1. INTRODUCTION

As circuit feature size shrinks down to $20nm$, lithographic hotspots caused by light diffraction during manufacturing procedure have become a serious factor that affects manufacture yield. A hotspot is a region of mask layout patterns where failures including open and short circuits are more likely to happen during manufacturing. Therefore, hotspot detection at the post-OPC stage is imperative before transferring designed patterns onto a silicon wafer.

Many studies have been carried out for lithography hotspot detection. The state-of-the-art methods include lithographic simulation,[1,2] pattern matching,[3,4] and recently prevailing machine learning techniques.[5–9] Lithographic simulation is able to imitate fabrication result accurately, but it is computationally expensive. Pattern

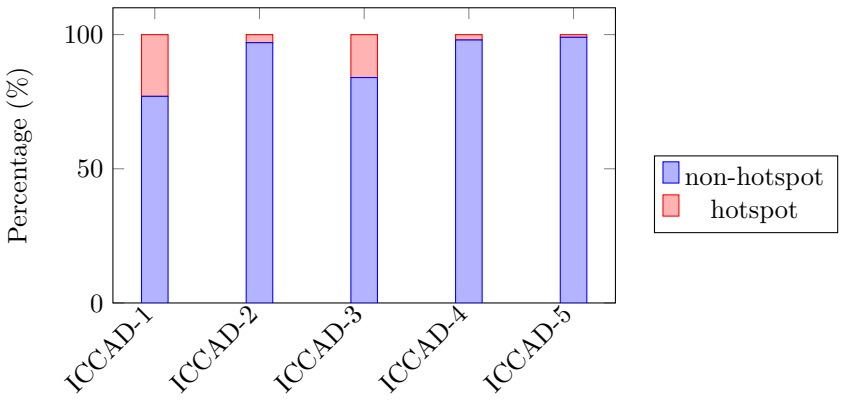

Figure 1. Breakdown of hotspot and non-hotspot pattern percentages for ICCAD 2012 contest benchmark.

Design-Process-Technology Co-optimization for Manufacturability XI, edited by Luigi Capodieci, Jason P. Cain, Proc. of SPIE Vol. 10148, 1014807 · © 2017 SPIE
CCC code: 0277-786X/17/$18 · doi: 10.1117/12.2258374

matching provides speedup in comparison with lithographic simulation, but it is merely applicable to detect already known or similar patterns thus has poor hotspot recognition rate on unknown patterns. Machine learning[†] is an emerging technique that can achieve reasonably good hotspot detection results with fast throughput. In a machine learning flow, before fed into learning engine, raw data should be preprocessed in feature extraction stage to convert complicated layout patterns into low dimensional vectors. The low dimensional vector, also known as feature representation, directly affects hotspot prediction performance. In the case of VLSI layout, the conventional density based feature[4,10] and recently proposed concentric circle area sampling (CCAS) feature[11] capture layout properties and the lithography process, respectively, and therefore have made considerable improvements on hotspot detection accuracy. However, with circuit feature size reduced to several nanometers, layout patterns have become more complicated, hence 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 been 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, in order to take advantages of powerful deep learning models, there are still several aspects that should be taken into consideration: (1) Hyper-parameters 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. However, for the lithography hotspot detection problem, whether a pattern 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 as after lithography enhancement techniques (RETs) the lithographic defects are always the minority. Figure 1 lists the percentages of hotspot and non-hotspot patterns for each test case of ICCAD 2012 benchmark suite.[15] We can see that the amount of non-hotspot patterns are much larger than the amount of hotspot patterns, especially for cases `ICCAD-2`, `ICCAD-4` and `ICCAD-5` that non-hotspot patterns occupy more than 99% of total patterns. Therefore, under conventional training strategies, the neural network may not able to correctly predict hotspot patterns even with ignorable training loss. (3) For hotspot detection application, 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.

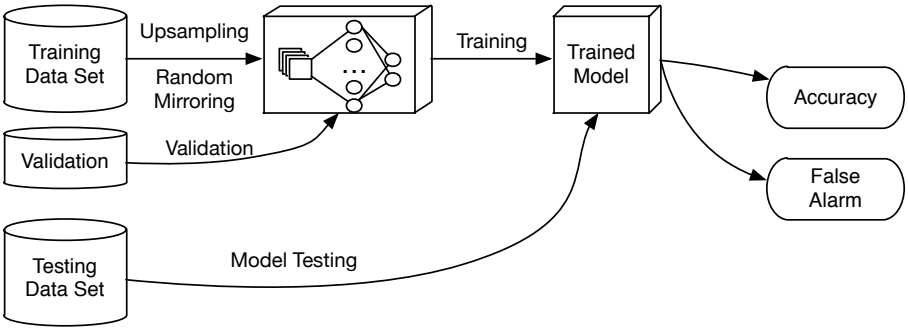

Figure 2. The proposed deep learning based hotspot detection flow.

In this paper, we develop a deep learning-based hotspot detection flow, as illustrated in Figure 2. Every original training dataset is divided into two parts: 75% of the samples are preprocessed for deep learning model training, while 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 thereafter. Accuracy and false alarms can then be calculated by comparing prediction results with corresponding actual results. We carefully analyze impact and effectiveness of different CNN hyper-parameters, through which a hotspot-detection-oriented neural network model is established. To address the imbalance problem, we further apply minority upsampling and random-mirror flipping of the hotspot patterns before training the network. Finally, we verify our proposed model on

[†]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.

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 running time.

The rest of the paper is organized as follows: Section 2 studies the effect of hyper parameters 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 and discussion in Section 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 hyper-parameters, based on which the effectiveness of our network architecture is further described.

2.1 Convolutional Neural Network Elements

2.1.1 Convolution Layer

The operation within each convolution layer is as follows,

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

where $\mathbf{I}(i, j, k)$ is the pixel value of location (j, k) at i^{th} feature map output of previous layer, while $\mathbf{K}$ is the convolution kernel. We can see from Equation (1) that the pixel value of the output feature map is obtained from the inner product between the convolution kernel and the input feature map. After scanning from the upper-right to the bottom-left corner, a new feature map is generated. Usually a single convolution kernel is shared during the generation of a feature map as proposed in [19], and this weight sharing scheme enables common local feature extraction within a feature map. Moreover, stacked convolution layers grasp image attributes in different hierarchical levels and generate better feature representation.

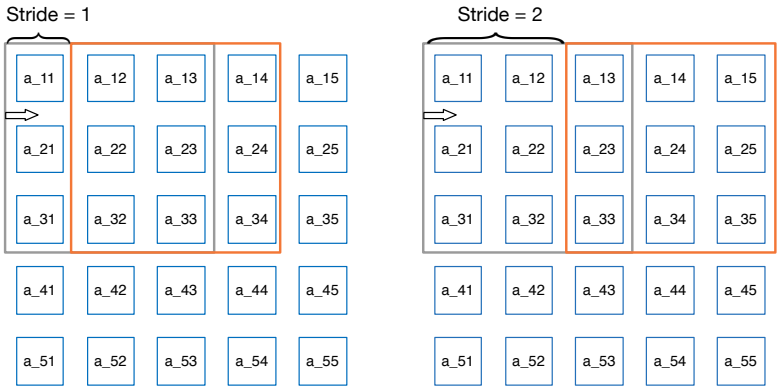

Figure 3. 3×3 convolution example with stride set to 1 (left) and 2 (right), respectively.

A convolution layer is specified by hyper-parameters 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 Equation (1). Note that in the case when efficient dimension reduction is required, stride can be set to any positive integer. Figure 3 illustrates 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 non-hotspot 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 Table 1, we compare model performances with different kernel sizes, and we can see that the proposed 3×3 kernel size outperforms other kernel sizes. Note that here padding is also changed along with the kernel sizes in order to keep the output size of the corresponding layer unchanged.

Table 1. Comparison on Different Kernel Size.

Kernel Size	Padding	Test Accuracy
3×3	1	**96.25%**
5×5	2	93.75%
7×7	3	87.50%

Because local regions in the layout are highly correlated, large overlap windows promise to gather sufficient information. In addition, previous work has shown that a smaller stride ensures the model to be translational 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 multi-layer perceptron to perform output regularization. Prevailing candidates are sigmoid and tanh functions that scale the entries of each layer into an interval of $(-1, 1)$ and $(0, 1)$, respectively. However, these activation functions suffer from a gradient vanishing problem,[22] due to the chained multiplication of numbers within the range $(-1, 1)$ during backpropogation. In other words, the training of early layers is inefficient in deep neural networks. Nari et al. proposed a rectified linear unit (ReLU) for a restricted Boltzmann Machine,[23] which performs element-wise operations on a feature map as shown in Equation (2):

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

The equation indicates that by applying ReLU, the feature map no longer has an upper boundary, but network sparsity is augmented and 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 an ReLU layer, we replace ReLU with several classical activation functions, including sigmoid, tanh and binomial normal log likelihood (BNLL) during training. The experimental results in Table 2 show that the deep neural network model with ReLU layer 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 extreme large validation loss.

Table 2. 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

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 max or average value within a predefined region (kernel) that correspond to max pooling and average pooling as illustrated in Figure 4. 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.[24] To decide the best pooling method, we compare the performance of models with max pooling, average pooling, and random chosen value from the scan window (stochastic pooling). As shown in Table 3, 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 3. 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, feature map will become smaller and deeper, and finally reach unit size. The layer generating unit size feature is called fully connected (FC) layer. The FC layers form the last several layers of convolutional neural network, and can be regarded as a special case of convolution layer with kernel size equal to the feature map size of previous layer. If the kernel and feature map are square, we have the following relationship of the parameters as in Equation (1),

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

Note that vertexes in last FC layer are associated with target of classifier and each vertex reflects 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 FC layer contains the information from a global view, while user-designed features are 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. 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 improvements.[25] To evaluate the effect of dropout, we trained the network with variant dropout ratios from 0 to 0.8. As shown in Figure 5, the validation curve indicates that the model performance is best when dropout ratio is between 0.4 and 0.7, therefore we set the dropout ratio to 0.5 in our FC layer.

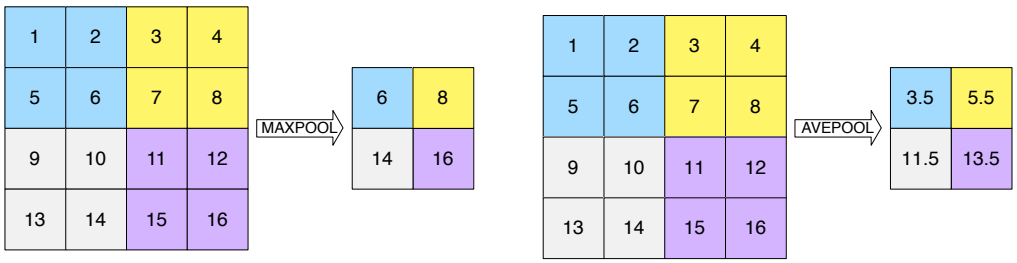

Figure 4. Examples of max pooling (left) and average pooling (right).

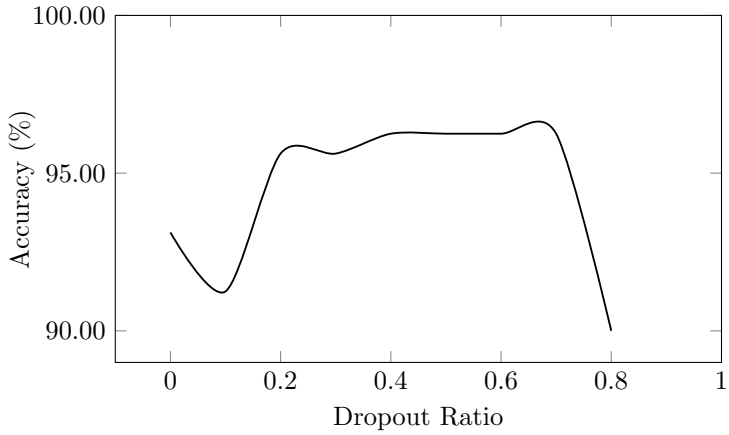

Figure 5. Dropout Ratio Effect.

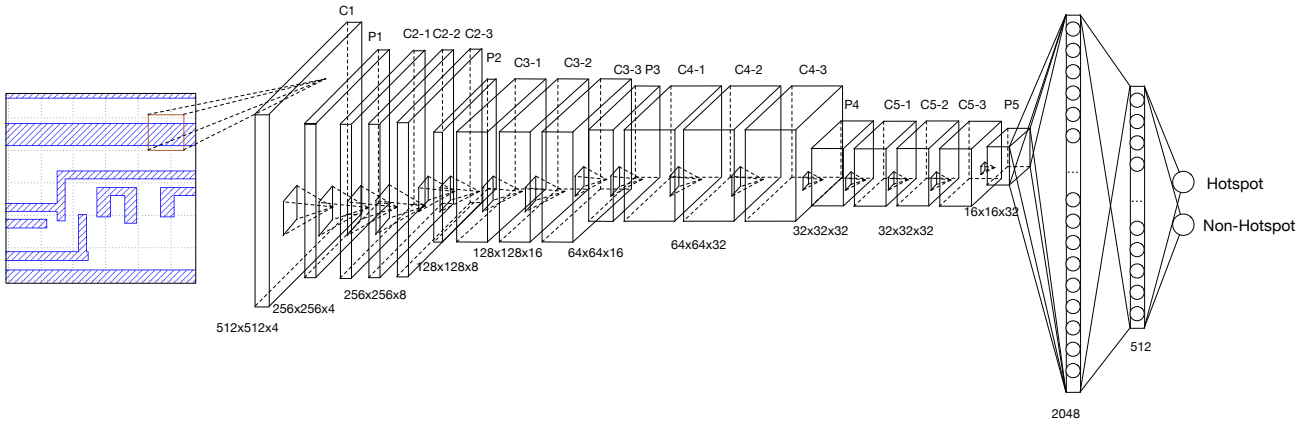

Figure 6. Architecture Overview.

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 Figure 6. The layout clip of $1024 \times 1024nm^2$ is much larger than the image size of a conventional computer vision benchmark dataset such as ImageNet.[26] Therefore, we set a 2×2 convolution and 2×2 pooling with stride of 2 to reduce the feature map which benefits both storage and training. Following the first pooling layer 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 is still of high dimension. To generate final output, we add two additional FC layers to reduce output vertex number to two, then output hotspot/non-hotspot probability for each input target. Note that each convolution layer is followed by one Rectified Linear Unit. More configuration information is listed in Table 4.

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 minorities in VLSI mask design, the training data set 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 high 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.

Table 4. Neural Network Configuration.

Layer	Kernel Size	Stride	Padding	Output Vertexes
Conv1-1	$2 \times 2 \times 4$	2	0	$512 \times 512 \times 4$
Pool1	2×2	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

3.1 Random Mirror Flipping and Upsampling

Existing methods to handle imbalanced data include multi-label learning,[27] majority downsampling,[28] pseudo instance generation,[29] and so on, that 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. assigns different labels to the majority to balance the number of instances in each category.[27] However, this may cause insufficient training sample of individual classes, as 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 et al. performed layout pattern shifting to artificially generate hotspot patterns,[30] but the approach might be invalid because a shift larger that $10nm$ 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 minority samples. Here we use α to denote the duplicating factor and intuitively,

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

As it is normal to find only one minority instance within more than 100 samples, directly duplicating minorities may raise following problems. (1) In mini-batch gradient descent (MGD),[24] if one batch contains too many identical instances, a large gradient will be generated in one direction, which 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.

To enhance the effectiveness of minority upsampling, we first propose augmenting the training dataset with mirror-flipped and 180-degree-rotated version of the original layout clips, because the orientation of the clip does not change its attribute. In this case, each hotspot instance has an equal probabilities of taking one of four orientations (see Figure 7). Because MGD randomly picks training instances for some mini-batch size, we fix 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., minority 95 and majority 4452). The experimental results indicate that validation performance does not show further improvement when the upsampling factor increases beyond a certain value (approximately 20) (see Figure 8).

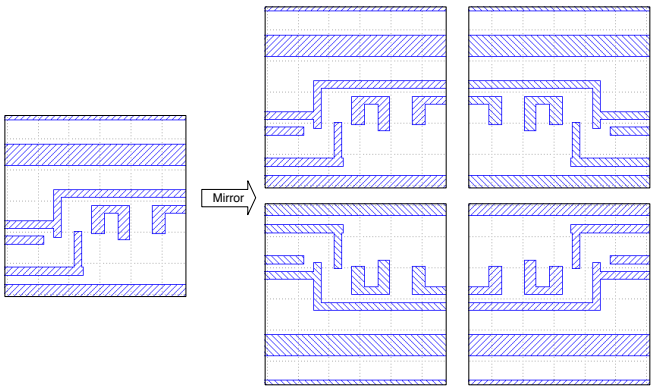

Figure 7. Random Mirror Flipping with X, Y, and XY.

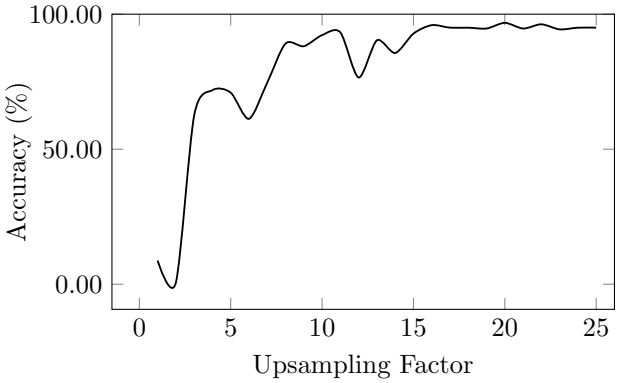

Figure 8. Upsampling Effects.

3.2 Training Neural Networks

We use the prevailing mini-batch gradient descent method to train the neural network. As described in Section 2, there are lots of hyper parameters associated with learning that can affect learning speed, and determine the final model performance.[16] This makes tuning hyper-parameters 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 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. On the other hand, 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 Figure 9. We can see that the network is trained more efficiently with $\gamma = 0.001$. 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 et al. analyzed the physical meaning of gradient descent and proposed the *momentum* method, which can speed up convergence in iterative learning.[31] The key idea is to modify the weight update scheme according to the following equations:

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

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

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 update speed and produces better training convergence.[32] Momentum by default is set 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 5), as suggested in CS231n.[33] The results show that the momentum of 0.99 enjoys the lowest validation loss.

Table 5. 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**

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.[34] Then Equations (6) and (7) become:

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

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

where λ is decay factor and is usually around $10^{-4} - 10^{-6}$.[35] A virtue of Equation (8) is that the term $\gamma \lambda w_i$ is ignorable when neuron weights are small, and neuron weights can get penalty from decay factor when they are

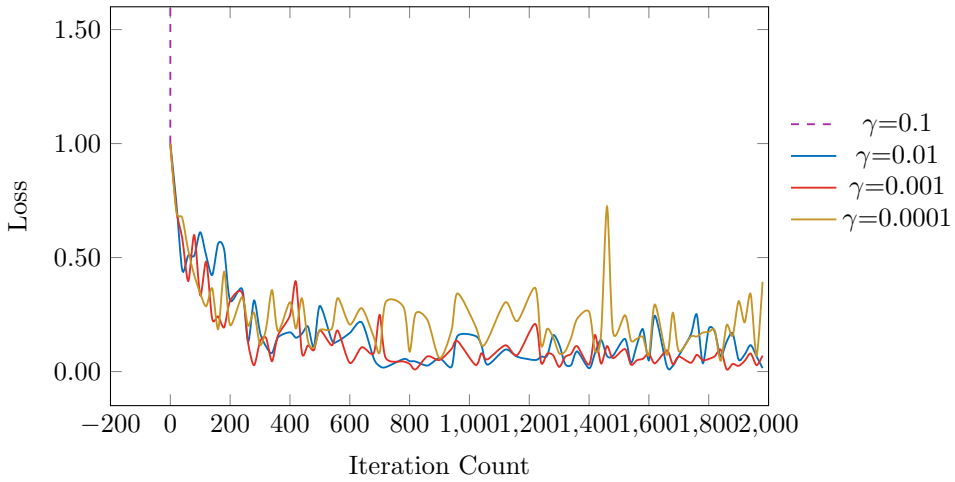

Figure 9. Effect of Initial Learning Rate.

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 6 and the results show that the model is learned more efficiently with $\lambda = 10^{-6}$.

Table 6. 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**

3.2.4 Weight Initialization

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

Many approaches have been studied in literature, and in most applications, random Gaussian enjoys the best performance. However, the results are highly affected by standard deviation. In order 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, Xavier et al. proposed an initialization method by taking the variance of each layer into consideration,[36] 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

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

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 10 reports the validation loss during training and illustrates that Xavier outperforms ordinary Gaussian initialization. It is also notable that improper weight initialization might cause 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 literature for hotspot detection and finally, we compare the result with two machine learning based hotspot detectors that have achieved satisfactory performance. All experiments are conducted using `caffe`[37] on a platform with an Intel Xeon Processor and a GTX Titan graphic card.

4.1 Evaluation Metrics and Benchmark Info

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

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

Definition 2 (False Alarm[15]) *The number of non-hotspot clips that are reported as hotspots by detector.*

As far as real design flow is concerned, lithographic simulation should be performed on all detected hotspot clips including false alarms. As suggested in [9], an unified runtime evaluation metric called overall detection and simulation time (ODST) is defined.

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

Note that an industrial lithography simulator[38] adopted in this paper takes 10s to perform lithography simulation on each clip, therefore, ODST can be calculated using the following equation,

$$\text{ODST} = \text{Test Time} + 10\text{s} \times \#\text{ of False Alarm}. \tag{15}$$

The evaluation benchmark contains five test cases: "ICCAD-1"-"ICCAD-5". The benchmark details are listed in Table 7. The "Training HS#" and "Training NHS#" columns denote the number of hotspot and non-hotspot patterns in training sets. The "Testing HS#" and "Testing NHS#" columns are for the number of hotspots and non-hotspots in testing sets. We can see that in the training set, the number of hotspots and the number of non-hotspots are highly imbalanced, which induces pressure on normal neural network training procedures.

Table 7. ICCAD 2012 Contest Benchmark.

Bench	Training HS#	Training NHS#	Testing HS#	Testing NHS#
ICCAD-1	99	340	226	3869
ICCAD-2	174	5285	498	41298
ICCAD-3	909	4643	1808	46333
ICCAD-4	95	4452	177	31890
ICCAD-5	26	2716	41	19327

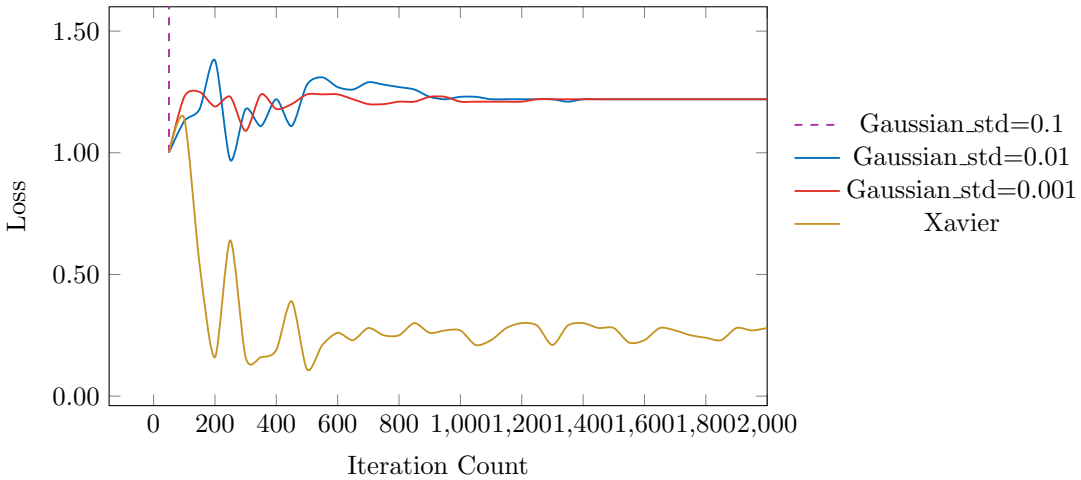

Figure 10. Importance of weight initialization.

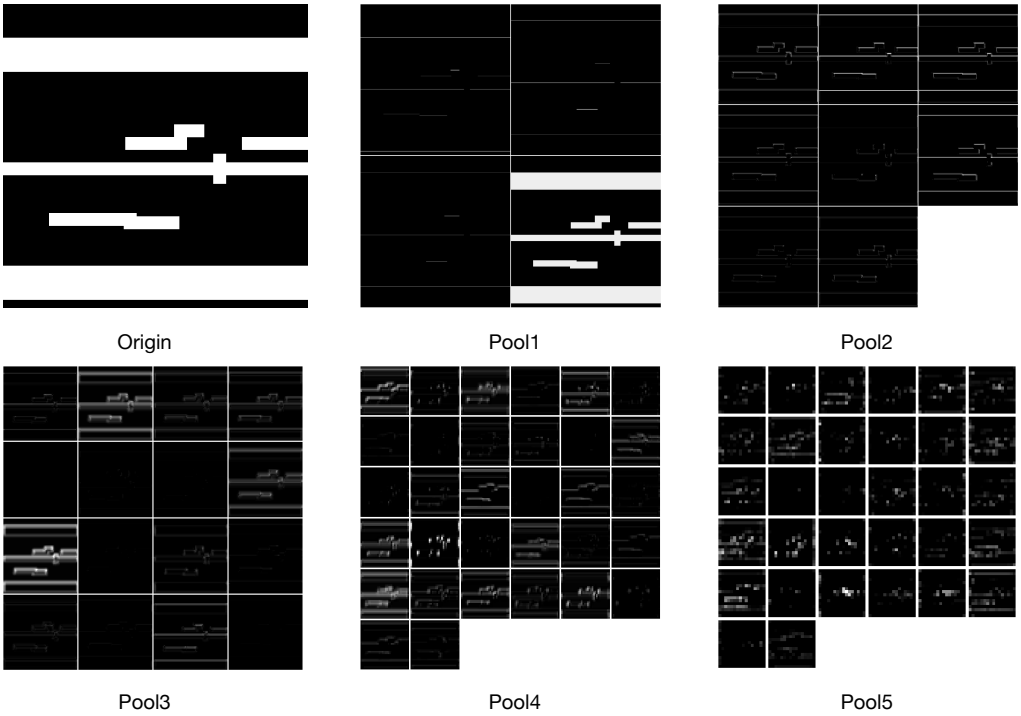

Figure 11. Neuron Response of Pooling Layers in Five Convolution Stages.

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 11 shows the neuron response for one input example. Subfigures "Origin", "Pool1", "Pool2", "Pool3", "Pool4" and "Pool5" correspond to original clip, and the neuron response of the 1^{st}, 2^{nd}, 3^{rd}, 4^{th} and 5^{th} 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 Result Comparison with Existing Deep Learning Flow

To the best of our knowledge, there were two attempts that applied deep learning on hotspot detection.[30,39] Because no detailed results are reported in [39], we only compare our framework with [30]. In [30], Shin et al. proposed a four-level convolutional neural network to perform hotspot classification. Table 8 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 variation; (2) The network scale is much smaller than ours (10 layers versus 21 layers); (3) In preprocessing, the training and testing datasets in [30] are sampled from layout clips with $10nm$ precision ($10nm$ for each pixel) and training hotspot patterns are randomly shifted to avoid imbalance problem that might cause unreliable training instances because there is enough nanometer level variation to modify a layout clip attribute.

Experimental results are listed in Table 9. 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`"-"`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.

Table 8. Neural Network Configuration of [30]

Layer	Kernel Size	Output Vertexes
Conv1	$5 \times 5 \times 25$	–
Pool1	2×2	52×25
Conv2	$5 \times 5 \times 40$	–
Pool2	2×2	24×40
Conv3	$5 \times 5 \times 60$	–
Pool3	2×2	10×60
Conv4	$5 \times 5 \times 80$	–
Pool4	2×2	3×80
FC1	–	100
FC2	–	2

The comparison shows that detection accuracy of our framework is better than [30] for each test case and has a 2.5% advantage of average detection accuracy. As far as the false alarm is concerned, [30] takes 88% more overall detection and simulation time than ours. Results also indicate that our CNN architecture enjoys effectiveness and efficiency.

Table 9. Performance Comparisons with [30]

Bench	JM3'16 [30]				Ours			
	FA#	CPU (s)	ODST (s)	Accu (%)	FA#	CPU (s)	ODST (s)	Accu (%)
ICCAD-1	386	**15**	3875	95.1%	**147**	51	**1521**	**99.6%**
ICCAD-2	1790	**208**	18108	98.8%	**561**	390	**6000**	**99.8%**
ICCAD-3	7077	**322**	71092	97.5%	**2660**	434	**27034**	**97.8%**
ICCAD-4	**892**	**129**	**9049**	93.8%	1785	333	18183	**96.4%**
ICCAD-5	**172**	**82**	**1802**	92.7%	242	232	2652	**95.1%**
Average	2063	**151**	20781	96.7%	**1079**	288	**11078**	**98.2%**
Ratio	–	–	1.88	0.98	–	–	1.0	1.0

4.4 Result Comparison with Machine Learning Hotspot Detectors

Many machine learning technologies were explored for layout hotspot detection and achieved better performance than traditional pattern matching approach. Here we compare our experiment with two representative machine learning based hotspot detectors [8] and [9], which adopt Support Vector Machine and Smooth Boosting, respectively.

As shown in Table 10, columns and rows are similarly defined as Table 9. For detection accuracy, our framework achieves the best result on cases ICCAD-2 (99.8%) and ICCAD-3 (97.8%), meanwhile gains similar results on rest cases with only 0.2% difference on ICCAD-1 and 1.3% difference on ICCAD-4. On average, the proposed deep learning approach outperforms [8] and [9] on accuracy (consistency with total number of correctly detected hotspot) by 5.3% and 0.2%, and has around 50,000s and 2,700s ODST advantages, 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 node, 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.

Table 10. Performance Comparisons with State-of-the-art Machine Learning

Bench	TCAD'15 [8]				ICCAD'16 [9]				Ours			
	FA#	CPU (s)	ODST (s)	Accu (%)	FA#	CPU (s)	ODST (s)	Accu (%)	FA#	CPU (s)	ODST (s)	Accu (%)
ICCAD-1	1493	38	14968	94.7%	788	**10**	7890	**100%**	**147**	51	**1521**	99.6%
ICCAD-2	11834	234	118574	98.2%	**544**	**103**	**5543**	99.4%	561	390	6000	**99.8%**
ICCAD-3	13850	778	139278	91.9%	**2052**	**110**	**20630**	97.5%	2660	434	27034	**97.8%**
ICCAD-4	3664	356	36996	85.9%	3341	**69**	33478	**97.7%**	**1785**	333	**18183**	96.4%
ICCAD-5	1205	**20**	12070	92.9%	**94**	41	**980**	**95.1%**	242	232	2652	**95.1%**
Average	6409	285	64377	92.9%	1363	67	13702	98.0%	**1079**	288	**11078**	**98.2%**
Ratio	–	–	5.81	0.946	–	–	1.24	0.997	–	–	**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. Effectiveness of the associated hyper parameters are studied 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 dataset. 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 offering better solutions to some emerging design for manufacturability (DFM) problems as circuit layout advances to extreme scale.

Acknowledgment

This work is supported in part by The Chinese University of Hong Kong (CUHK) Direct Grant for Research. The authors would like to thank Evangeline F. Y. Young from CUHK and Lauren Katzive from ASML for helpful comments.

REFERENCES

[1] Kim, J. and Fan, M., "Hotspot detection on Post-OPC layout using full chip simulation based verification tool: A case study with aerial image simulation," in [*Proceedings of SPIE*], **5256** (2003).

[2] Roseboom, E., Rossman, M., Chang, F.-C., and Hurat, P., "Automated full-chip hotspot detection and removal flow for interconnect layers of cell-based designs," in [*Proceedings of SPIE*], **6521** (2007).

[3] Yu, Y.-T., Chan, Y.-C., Sinha, S., Jiang, I. H.-R., and Chiang, C., "Accurate process-hotspot detection using critical design rule extraction," in [*ACM/IEEE Design Automation Conference (DAC)*], 1167–1172 (2012).

[4] Wen, W.-Y., Li, J.-C., Lin, S.-Y., Chen, J.-Y., and Chang, S.-C., "A fuzzy-matching model with grid reduction for lithography hotspot detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* **33**(11), 1671–1680 (2014).

[5] Ding, D., Torres, J. A., and Pan, D. Z., "High performance lithography hotspot detection with successively refined pattern identifications and machine learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* **30**(11), 1621–1634 (2011).

[6] Gao, J.-R., Yu, B., and Pan, D. Z., "Accurate lithography hotspot detection based on PCA-SVM classifier with hierarchical data clustering," in [*Proceedings of SPIE*], **9053** (2014).

[7] Yu, B., Gao, J.-R., Ding, D., Zeng, X., and Pan, D. Z., "Accurate lithography hotspot detection based on principal component analysis-support vector machine classifier with hierarchical data clustering," *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)* **14**(1), 011003 (2015).

[8] Yu, Y.-T., Lin, G.-H., Jiang, I. H.-R., and Chiang, C., "Machine-learning-based hotspot detection using topological classification and critical feature extraction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* **34**(3), 460–470 (2015).

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

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

[11] Matsunawa, T., Yu, B., and Pan, D. Z., "Optical proximity correction with hierarchical bayes model," in [*Proceedings of SPIE*], **9426** (2015).

[12] Hinton, G. E., "What kind of graphical model is the brain?," in [*International Joint Conference on Artificial Intelligence (IJCAI)*], 1765–1775 (2005).

[13] Hinton, G. E., Osindero, S., and Teh, Y. W., "A fast learning algorithm for deep belief nets," *Neural computation* **18**(7), 1527–1554 (2006).

[14] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "Imagenet classification with deep convolutional neural networks," in [*Conference on Neural Information Processing Systems (NIPS)*], 1097–1105 (2012).

[15] Torres, A. J., "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite," in [*IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*], 349–350 (2012).

[16] Bengio, Y., "Practical recommendations for gradient-based training of deep architectures," in [*Neural Networks: Tricks of the Trade*], Orr, G. B. and Müller, K.-R., eds., 437–478, Springer (2012).

[17] Deng, L., "Three classes of deep learning architectures and their applications: a tutorial survey," *APSIPA transactions on signal and information processing* (2012).

[18] Sun, Y., Wang, X., and Tang, X., "Hybrid deep learning for face verification," in [*IEEE International Conference on Computer Vision (ICCV)*], 1489–1496 (2013).

[19] Huang, G. B., Lee, H., and Learned-Miller, E., "Learning hierarchical representations for face verification with convolutional deep belief networks," in [*IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*], 2518–2525 (2012).

[20] Zeiler, M. D. and Fergus, R., "Visualizing and understanding convolutional networks," in [*European Conference on Computer Vision (ECCV)*], 818–833 (2014).

[21] Simonyan, K. and Zisserman, A., "Very deep convolutional networks for large-scale image recognition," *arXiv preprint* (2014).

[22] Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J., "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in [*A Field Guide to Dynamical Recurrent Neural Networks*], Kolen, J. F. and Kremer, S. C., eds., IEEE Press (2001).

[23] Nair, V. and Hinton, G. E., "Rectified linear units improve restricted boltzmann machines," in [*International Conference on Machine Learning (ICML)*], 807–814 (2010).

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

[25] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research* **15**(1), 1929–1958 (2014).

[26] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L., "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision* **115**(3), 211–252 (2015).

[27] Zhang, M.-L., Li, Y.-K., and Liu, X.-Y., "Towards class-imbalance aware multi-label learning," in [*International Joint Conference on Artificial Intelligence (IJCAI)*], 4041–4047 (2015).

[28] Ng, W. W. Y., Hu, J., Yeung, D. S., Yin, S., and Roli, F., "Diversified sensitivity-based undersampling for imbalance classification problems," *IEEE Transactions on Cybernetics (TCYB)* **45**(11), 2402–2412 (2015).

[29] He, H., Bai, Y., Garcia, E. A., and Li, S., "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in [*International Joint Conference on Neural Networks (IJCNN)*], 1322–1328 (2008).

[30] Shin, M. and Lee, J.-H., "Accurate lithography hotspot detection using deep convolutional neural networks," *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)* **15**(4), 043507 (2016).
[31] Polyak, B. T., "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics* **4**(5), 1–17 (1964).
[32] Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E., "On the importance of initialization and momentum in deep learning.," *International Conference on Machine Learning (ICML)* **28**, 1139–1147 (2013).
[33] Karpathy, A., "Stanford university cs231n: Convolutional neural networks for visual recognition." http://cs231n.github.io/neural-networks-3/.
[34] Moody, J., Hanson, S., Krogh, A., and Hertz, J. A., "A simple weight decay can improve generalization," *Conference on Neural Information Processing Systems (NIPS)* **4**, 950–957 (1995).
[35] Bottou, L., "Stochastic gradient descent tricks," in [*Neural networks: Tricks of the Trade*], Orr, G. B. and Müller, K.-R., eds., 421–436, Springer (2012).
[36] Glorot, X. and Bengio, Y., "Understanding the difficulty of training deep feedforward neural networks," in [*International Conference on Artificial Intelligence and Statistics (AISTATS)*], **9**, 249–256 (2010).
[37] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T., "Caffe: Convolutional architecture for fast feature embedding," in [*ACM International Multimedia Conference (MM)*], 675–678 (2014).
[38] Banerjee, S., Li, Z., and Nassif, S. R., "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in [*IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*], 271–274 (2013).
[39] Matsunawa, T., Nojima, S., and Kotani, T., "Automatic layout feature extraction for lithography hotspot detection based on deep neural network," in [*SPIE Advanced Lithography*], **9781** (2016).