Machine Learning for Mask Synthesis and Verification

Haoyu Yang, Yibo Lin and Bei Yu

Abstract The explosion of machine learning and AI techniques have brought great opportunities of data-assisted optimization for VLSI design automation problems. Recent studies have demonstrated promising results dealing with lithography compliance issues. In this chapter, we will introduce successful attempts using machine learning for mask synthesis and verification, including lithograph modeling, hotpsot detection, mask optimization, and layout pattern generation. We hope this chapter can motivate future research on AI-assisted DFM solutions.

1 Introduction

Moore's Law has guided fast and continuous development of VLSI design and manufacturing technologies, which tend to enable the scaling of design feature size to integrate more components into circuit chips. However, the significant gap between circuit feature size and lithography systems has brought great manufacturing challenges.

A classical lithography system consists of mainly five stages that include *source*, *condenser lens*, *mask*, *objective lens*, and *wafer*, as shown in Figure 1. The source stage ejects ultraviolet light beams toward the condenser lens which collects light beams that can go towards the mask stage for further imaging. The remaining light beams that can pass through the mask stage are supposed to leave expected circuit patterns on the wafer stage. As the manufacturing feature size enters single-digit nanometer era, diffraction is inevitable when a light beam enters the mask stage.

Haoyu Yang
nVIDIA Corp., Austin, TX, e-mail: haoyuy@nvidia.com

Yibo Lin
Peking University, China, e-mail: yibolin@pku.edu.cn

Bei Yu
The Chinese University of Hong Kong, Hong Kong SAR, e-mail: byu@cse.cuhk.edu.hk

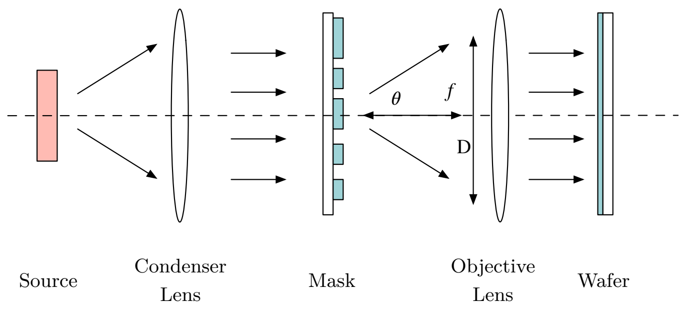


Fig. 1: An example of a classical lithography system.

The objective lens tries to collect diffraction information as much as possible for better transferred image quality. Because of the limited size of the objective lens, higher order diffraction patterns will be discarded when forming the image on the wafer that results in a lower pattern fidelity [1]. Typically, to ensure the mask image can be transferred onto the wafer as accurate as possible, at least the zero and $\pm$1st diffraction order should be captured by the objective lens. Accordingly, the smallest design pitch can be defined as Equation (1),

$$\frac{1}{p} \propto \frac{NA}{\lambda}, \tag{1}$$

where p denotes design pitch, λ is the wavelength of the light source and NA is the numerical aperture of the objective lens which determines how much information can be collected by the objective lens and is given by

$$NA = n \sin\theta_{\max} = \frac{D}{2f}, \tag{2}$$

where n is the index of refraction of the medium, $\theta_{\max}$ is the largest half-angle of the diffraction light that can be collected by the objective lens, D denotes the diameter of physical aperture seen in front of the objective lens and f represents the focal length [2].

Although research has pushed higher NA design of lithography systems, diffraction information loss still causes mismatches between printed patterns on a wafer and the patterns in the design, which is well known as the *lithography proximity effect*. A mainstream solution is called resolution enhancement technique (RET) that includes multiple patterning lithography (MPL) [3, 4, 5, 6], sub-resolution assist feature (SRAF) [7, 8, 9] insertion and optical proximity correction (OPC) [10]. MPL attempts split designs into multiple masks to achieve higher resolution, while SRAF and OPC aim to compensate for the diffraction information loss in the lithography procedure. A lithography system is also subject to process condition variations such as focus and dose, which are likely to deviate from optimal settings. Figure 2 illustrates the different effects of process variations with focus and dose variations resulting in image distortion and contour deviation, respectively. It should be noted that high-quality RETs also make designs robust to process variations.

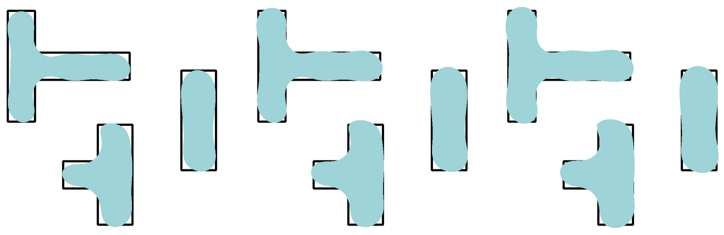


Fig. 2: Lithography contour versus design target under process variations: (a) low dose at focus produces small contours; (b) normal dose at focus produces regular contours; (c) high dose at defocus produces larger contours.

Physics and process limitations have therefore posed various challenges on VLSI design for manufacturing. This chapter focuses on four representative and critical DFM problems that cover lithography modeling (Section 2), layout hotspot detection (Section 3), mask optimization (Section 4), and layout pattern generation (Section 5). We will present recent progress and attempts using emerging machine learning solutions to tackle these challenges.

2 Lithography Modeling

Lithography simulation is critical in modern DFM flows which enable reliable mask optimization and layout verification (Figure 3). Optical modeling and resist modeling are two major steps in the lithography simulation procedure. Optical modeling maps a mask image to light intensity (aerial image) that is projected on a silicon wafer. Resist modeling deals with the interaction between light intensity and resist materials and determines the final shape formed on the silicon wafer.

2.1 Physics in Lithography Modeling

2.1.1 Optical Modeling

Singular value decomposition model is the most popular optical lithography approximation model, which has been widely adopted in mask printability estimation and

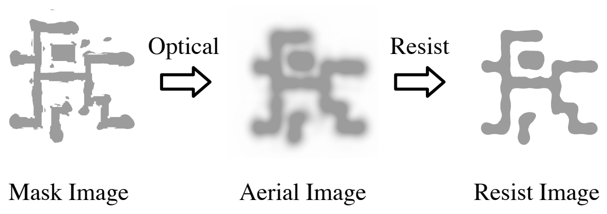


Fig. 3: Lithography simulation with optical and resist model.

optimization [11, 12, 13, 14]. This approximation starts from the Hopkins diffraction model [15] which is given by

$$\boldsymbol{I}(m,n) = \tilde{\boldsymbol{s}}^H \boldsymbol{A} \tilde{\boldsymbol{s}}, \quad m,n = 1,2,...,N, \tag{3}$$

where

$$\tilde{\boldsymbol{s}}_i = \tilde{\boldsymbol{M}}(p,q)\exp[i2\pi(pm+qn)], \quad i = 1,2,...,N^2, \tag{4}$$

where $\tilde{\boldsymbol{M}} = \mathcal{F}(\boldsymbol{M})$ is the mask in Fourier space, $\tilde{\boldsymbol{s}}_i$ is the i^{th} element of $\tilde{\boldsymbol{s}}$ [1], $p = i$ mod N and $q = \lceil \frac{i}{n} \rceil$. $\boldsymbol{A} \in \mathbb{C}^{N^2 \times N^2}$ contains the information of the transmission cross-coefficients that are optical-related parameters. Taking the singular value decomposition of $\boldsymbol{A} = \sum_{k=1}^{N^2} \alpha_k \boldsymbol{v}_k \boldsymbol{v}_k^H$, we have

$$\boldsymbol{I}(m,n) = \sum_{k=1}^{N^2} \alpha_k |\tilde{\boldsymbol{s}}^H \boldsymbol{v}_k|^2, \tag{5}$$

where $\boldsymbol{v}_k \in \mathbb{C}^{N^2}$ is the k^{th} eigenvector of $\boldsymbol{A}$ and α_k is the corresponding eigenvalue. We can therefore define

$$\tilde{\boldsymbol{h}}_k(p,q) = \begin{bmatrix} \boldsymbol{v}_{k,1} & \boldsymbol{v}_{k,N+1} & \cdots & \boldsymbol{v}_{k,N(N-1)+1} \\ \boldsymbol{v}_{k,2} & \boldsymbol{v}_{k,N+1} & \cdots & \boldsymbol{v}_{k,N(N-1)+2} \\ \cdots & \cdots & \cdots & \cdots \\ \boldsymbol{v}_{k,N} & \boldsymbol{v}_{k,2N} & \cdots & \boldsymbol{v}_{k,N^2} \end{bmatrix}. \tag{6}$$

Let $\boldsymbol{h}_k(m,n) = \mathcal{F}^{-1}(\tilde{\boldsymbol{h}}_k(p,q))$ and connect Equations (3) to (5), we have

$$\boldsymbol{I}(m,n) = \sum_{k=1}^{N^2} \alpha_k |\boldsymbol{h}_k(m,n) \otimes \boldsymbol{M}(m,n)|^2, \tag{7}$$

[1] Here $\boldsymbol{s}$ itself is meaningless, and we simply use $\tilde{\boldsymbol{s}}$ to indicate the term is related to frequency domain.

where $\otimes$ denotes the convolution operation and $\boldsymbol{h}_k$'s are usually called the lithography kernels. Given a mask $\boldsymbol{M}(m,n)$ in real domain, we can calculate its aerial image $\boldsymbol{I}(m,n)$ (light intensity projected on resist materials) via Equation (7).

2.1.2 Resist Modeling

Resist models basically aim to perform thresholding on aerial images and obtain the final resist contour. Constant threshold resist (CTR) model is well accepted in literature study for its simplicity [16]. Given an aerial image $\boldsymbol{I}$, the degree of resist chemical reaction $\boldsymbol{D}$ is given by

$$\boldsymbol{D} = \boldsymbol{I}(m,n) \otimes \boldsymbol{G}, \tag{8}$$

where $\boldsymbol{G}$ is a Gaussian kernel to simulate chemical reactions. And the final resist shape $\boldsymbol{Z}$ is defined as

$$\boldsymbol{Z}(m,n) = \begin{cases} 1, & \text{if } \boldsymbol{D}(m,n) > D_{\text{th}}, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

The strong assumption in CTR makes the model less reliable when facing complicated designs. Variable-threshold resist model (VTR) is then proposed to execute more accurate simulation by assigning local patterns with different thresholds. This is determined by

$$D_{\text{th}} = k_1 I_{\max} + k_2 I_{\min} + k_3 s, \tag{10}$$

where $I_{\max}$, $I_{\min}$ and s are max aerial image intensity, min aerial image intensity and the slope of aerial image profile, respectively.

2.2 *Machine Learning Solutions for Lithogrpahy Modeling*

Physics shows that rigorous lithography simulations are either computationally expensive or suffer performance drop. Therefore, various machine learning frameworks are proposed to meet both runtime and accuracy requirements. As summarized in Table 1, prior arts can be categorized into three aspects: end-to-end models, optical models, and resist models.

End-to-end models target to complete optical and resist simulation as a whole, i.e., take the input of a mask and output its corresponding resist patterns. End-to-end modes are fast for rough lithography estimation, however, lacks the details of light intensity information. LithoGAN [17] is a very early attempt to use conditional generative adversarial networks (cGAN) for end-to-end modeling. The major component of LithoGAN is a standard cGAN generator, which takes the input of a mask with the target shape located in the center of the mask. cGAN can then gener-

Table 1: Machine Learning Solutions for Lithography Modeling.

Model	Framework	Keywords
End-to-end	[17]	Conditional GAN; Single Contact Simulation; CNN Alignment
	[18]	UNet++; Multi Contact Simulation; Multi-Channel Input; Perceptual Loss
Optical	[19]	Conditional GAN; Thin→Thick Mask Modeling
Resist	[16]	Aerial Image→Resist Threshold; CNN
	[20]	Concentrated Circle Sampling; Multi-layer Perceptron; Resist Height Prediction
	[21]	Aerial Image→Resist Threshold;ResNet;Active Sampling

ate the post-lithography contour of the target shape. The misalignment between the predicted contour and design location will question the reliability of such a framework. Therefore, a CNN forward path is introduced as an assist component to output target shape coordinates and hence fix the alignment issue for the generated contours. Very recently, deep lithography simulator (DLS) [18] is proposed as supporting neural networks for full mask optimization. DLS is still a cGAN-backboned structure. Thanks to its UNet [22] backbone and residual feature layers, DLS is able to perform lithography contour prediction on multiple shapes within a $4\mu m^2$ tile. To tackle the alignment issue, DLS splits the input mask image into three-channel tensors that include SRAF, OPC, and design patterns respectively. Apart from traditional cGAN training objectives, DLS also introduces perceptual loss for high-level feature matching, yielding better generation performance. Instead of directly measuring the differences between the generated contours and ground truth contours, the perceptual loss compares ground truth images and generated images based on high-level representations from pre-trained convolutional layers.

Optical models deal with aerial image generation. A representative work is TEMPO [19], a framework that takes inputs of a mask image and a series of 2D aerial images at different resist heights. The TEMPO will be trained towards generating aerial images under thick mask consumption through an encoder-decoder structure following root mean square error.

Resist models try to estimate the interaction between light beams and resist materials and hence obtain the final resist patterns. Several ML solutions are proposed to achieve fast and accurate resist modeling. The work of [16] starts to investigate standard CNN-based resist threshold prediction framework. It accepts aerial images as input and generates proper thresholds for the corresponding patterns. Another work [20] deals with resist modeling in a more detailed way. Instead of obtaining a resist threshold for the given pattern, it tries to predict the remaining resist height on a given location after the etching process. Given a layout design, the technique of [20] extracts a series of layout features in terms of a position of interest. These features will then be fed into a multi-layer perceptron and output predicted resist height. Although these learning-based frameworks have demonstrated their effectiveness on resist modeling, achieving high accuracy with CNNs requires a large

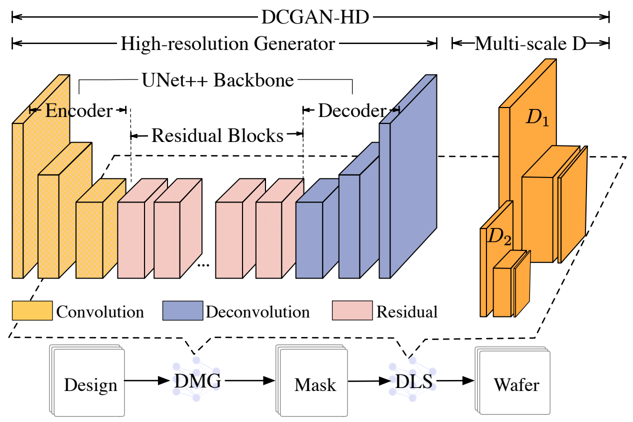


Fig. 4: DCGAN-HD generator [18]. Both deep mask generator (DMG) and deep lithography simulator (DLS) in the paper share the same architecture.

amount of training data, which is often not easy to get at the early stage of process development. Lin *et al.* [21] formulate a data-efficient learning problem leveraging the data from old technology nodes to assist model training at the target node. In this formulation, they construct CNN models taking an aerial image as input and outputting the slicing thresholds at the boundary of the target pattern.

2.3 *Case Studies*

In this section, we will detail two representative works [18] and [21] for machine learning-based lithography modeling.

2.3.1 Deep Lithography Simulator [18]

The key contributions of the deep lithography simulator [18] are the DCGAN-HD generator, multi-scale discriminator, and perceptual loss.

DCGANHD Generator. Left part of Figure 4 shows the architecture of a high-resolution generator, which completes mask-to-wafer mapping from design and SRAF pattern groups. Previous work [12] and [17] adopt traditional UNet [22] for lithography related tasks where input features are down-sampled multiple times. With the decreasing of feature resolution, it is easier for a network to gather high-level features such as context features while low-level information such as the position of each shape becomes harder to collect. However, in lithography physics, low-level information matters more than in the common computer vision tasks. For example, the shape and relative distance of design or SRAF patterns must remain unchanged after the deep mask optimization or deep lithography process. The number and relative distance of via patterns in an input layout have a crucial influence on the result. The features of OPC datasets determine the vital importance of the low-level features. UNet++ [23] is hence proposed for better feature extraction by assembling multiple UNets that have different numbers of downsampling operations. It redesigns the skip pathways to bridge the semantic gap between the encoder and decoder feature maps, contributing to the more accurate low-level feature extraction. The dense skip connections on UNet++ skip pathways improve gradient flow in high-resolution tasks. Although UNet++ has a better performance than UNet, it is not qualified to be the generator of DCGAN-HD. For further improvement, the UNet++ backbone is manipulated with the guidelines suggested in DCGAN [24].

Multi-scale Discriminator. The high-resolution input also imposes a critical challenge to the discriminator design. A simple discriminator that only has three convolutional layers with LeakyReLU and Dropout is presented. Since the patterns in OPC datasets have simple and homogeneous distribution, a deeper discriminator has a higher risk of over-fitting. Therefore, we simplify the discriminator by reducing the depth of the neural network. Meanwhile, a dropout layer is attached after each convolutional layer. We use 3×3 convolution kernels in the generator for parameter-saving purposes and 4×4 kernels in the discriminator to increase receptive fields. However, during training, we find that the simple discriminator fails to distinguish between the real and the synthesized images when more via patterns occur in a window. Because when the number of vias reaches 5 or 6 in a window, the via patterns have a larger impact on each other and the features become more complicated. Inspired by Wang *et al.* in pix2pixHD [25], we design multi-scale discriminators. Different from pix2pixHD [25] which uses three discriminators, our design uses two discriminators that have an identical network structure but operate at different image scales, which are named $D1$, $D2$, as shown in the right part of Figure 4. Specifically, the discriminators $D1$, $D2$ are trained to differentiate real and synthesized images at the two different scales, 1024×1024 and 512×512, respectively, which helps the training of the high-resolution model easier. In our tasks, the multi-scale design also shows its strengths in flexibility. For example, when the training set has only one via in a window, we can use only $D1$ to avoid over-fitting and reduce the training time.

Perceptual Loss. Instead of using per-pixel loss such as L_1 loss or L_2 loss, DSL adopts the perceptual loss which has been proven successful in style transfer [26], image super-resolution and high-resolution image synthesis [25]. A per-pixel loss function is used as a metric for understanding differences between input and output on a pixel level. While the function is valuable for understanding interpolation on a pixel level, the process has drawbacks. For example, as stated in [26], consider two identical images offset from each other by one pixel; despite their perceptual similarity, they would be very different as measured by per-pixel losses. More than that, previous work [24] shows L_2 Loss will cause blur on the output image. Different from per-pixel loss, perceptual loss function in Equation (11) compares ground truth image $\boldsymbol{x}$ with generated image $\hat{\boldsymbol{x}}$ based on high-level representations from pre-trained convolutional neural networks $\boldsymbol{\Phi}$.

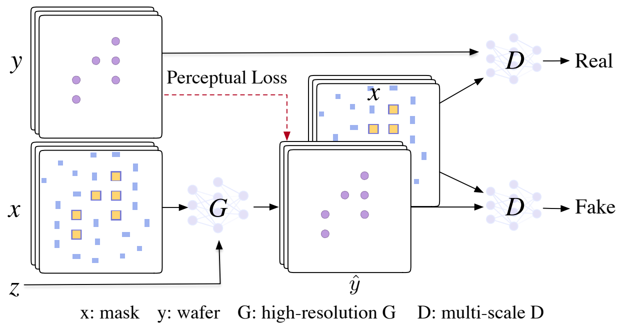


Fig. 5: The training details of DLS, where the input images are mask-wafer pairs [18].

$$\mathcal{L}_{LP}^{G,\Phi}(\boldsymbol{x},\hat{\boldsymbol{x}}) = \mathcal{L}_{L_1}(\Phi(\boldsymbol{x}),\Phi(\hat{\boldsymbol{x}})) = \mathbb{E}_{\boldsymbol{x},\hat{\boldsymbol{x}}}[\|\Phi(\boldsymbol{x})-\Phi(\hat{\boldsymbol{x}})\|_1]. \tag{11}$$

DLS Pipeline. Figure 5 shows the training process of our deep lithography simulator. As a customized design of cGAN, DLS is trained in an alternative scheme using paired mask image $\boldsymbol{x}$ and wafer image $\boldsymbol{y}$ obtained from Mentor Calibre. $\boldsymbol{z}$ indicates randomly initialized images. The objectives of DLS include training the generator G that produces fake wafer images $G(\boldsymbol{x},\boldsymbol{z})$ by learning the feature distribution from $\boldsymbol{x}$–$\boldsymbol{y}$ pairs and training the discriminators D_1, D_2 to identify the paired ($\boldsymbol{x}$, $G(\boldsymbol{x},\boldsymbol{z})$) as fake. This motivates the design of DLS loss function. The first part of the loss function comes from vanilla GAN that allows the generator and the discriminator interacting with each other in an adversarial way:

$$\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[\log D(\boldsymbol{x},\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{x},\boldsymbol{z}}[\log(1-D(\boldsymbol{x},G(\boldsymbol{x},\boldsymbol{z}))]. \tag{12}$$

Combined with multi-scale discriminators, Equation (12) can be modified as:

$$\begin{aligned}\sum_{k=1,2}\mathcal{L}_{cGAN}\left(G_{DLS},D_{DLS_k}\right) = &\sum_{k=1,2}\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[\log D_{DLS_k}(\boldsymbol{x},\boldsymbol{y})] \\ &+\mathbb{E}_{\boldsymbol{x},\boldsymbol{z}}[\log(1-D_{DLS_k}(\boldsymbol{x},G_{DLS}(\boldsymbol{x},\boldsymbol{z}))],\end{aligned} \tag{13}$$

where D_{DLS_k} is the kth discriminator of DLS. In DLS design, the perceptual loss is added to the objective, we denote $\hat{\boldsymbol{y}}$ as $G(\boldsymbol{x},\boldsymbol{z})$ and loss network Φ is a pre-trained VGG19 on ImageNet. The perceptual loss is given by:

$$\begin{aligned}\mathcal{L}_{LP}^{G_{DLS},\Phi}(\boldsymbol{y},\hat{\boldsymbol{y}}) &= \sum_{j=1\ldots5}\mathcal{L}_{L_1}(\phi_j(\boldsymbol{y}),\phi_j(\hat{\boldsymbol{y}})) \\ &= \sum_{j=1\ldots5}\mathbb{E}_{\boldsymbol{y},\hat{\boldsymbol{y}}}[\|\phi_j(\boldsymbol{y})-\phi_j(\hat{\boldsymbol{y}})\|_1],\end{aligned} \tag{14}$$

Table 2: Results of DLS [18].

Generator	Discriminator	Loss	mIoU (%)	pixAcc (%)
UNet (cGAN)	D (cGAN)	L_1	94.16	97.12
UNet++	D (cGAN)	L_1	93.98	96.74
G (Our)	D (cGAN)	L_1	96.23	97.50
G (Our)	D (Our)	L_1	97.63	98.76
G (Our)	D (Our)	Perceptual	**98.68**	**99.50**

where ϕ_j is the feature representation on j-th layer of the pre-trained VGG19 Φ. By combining Equation (13) and Equation (14):

$$\mathcal{L}_{DLS} = \sum_{k=1,2} \mathcal{L}_{cGAN}(G_{DLS}, D_{DLS_k}) + \lambda_0 \mathcal{L}_{LP}^{G_{DLS},\Phi}(\mathbf{y}, \hat{\mathbf{y}}). \tag{15}$$

Experiments. Since the DLS model is based on the cGAN framework, we set up an ablation experiment to illustrate the advantages of our generator and discriminators. The results shown in Table 2 is the average of 6 groups of the validation set. Firstly, cGAN (used in LithoGAN) provides a baseline mIoU of 94.16% which is far away from practical application. Then, UNet++ is used to replace the UNet generator in cGAN for better performance. However, the original UNet++ is not qualified to be a generator of a cGAN and the mIoU is reduced to 93.98% (as shown in Table 2).

Following DCGAN, we made some amendments in UNet++ and high-resolution generator is adopted in our DLS model. After applying our high-resolution generator, mIoU is improved to 97.63%, which outperforms UNet and UNet++ generators by a large margin when using the same discriminator. The huge gain in mIoU implies that our developed high-resolution generator is a strong candidate for DLS. Next, the newly designed multi-scale discriminators. Results in Table 2 show that mIoU is further boosted to 97.63%.

Lastly, we replace the L_1 loss with the perceptual loss and the mIoU reaches 98.68%. Additionally, DLS can handle multiple vias in a single clip, which overcomes the limitation of LithoGAN [17].

2.3.2 Resist Modeling with Transfer Learning and Active Data Selection [21]

The motivation of learning-based resit modeling is to leverage the simulation speed and accuracy. This comes with dataset efficiency and model design. Lin *et al.* [21] tackles the resist modeling problem with transfer learning and active data selection. The overall flow is shown in Figure 6.

Transfer Learning. Since the lithography configurations evolve from one generation to another with the advancement of technology nodes, there is plenty of historical data available for the old generation. If the lithography configurations have

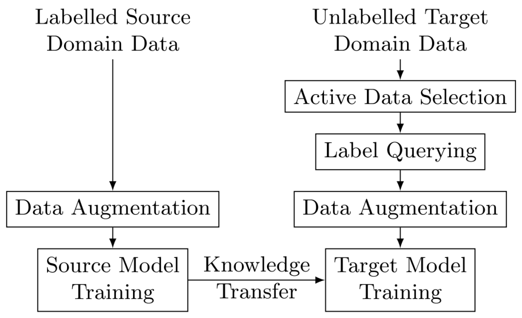


Fig. 6: The overall flow of resist modeling [21].

no fundamental changes, the knowledge learned from the historical data may still be applicable to the new configuration, which can eventually help to reduce the amount of new data required. Transfer learning aims at adapting the knowledge learned from data in source domains to a target domain. The transferred knowledge will benefit the learning in the target domain with faster convergence and better generalization. Suppose the data in the source domain has a distribution P_s and that in the target domain has a distribution P_t. The underlying assumption of transfer learning lies in the common factors that need to be captured for learning the variations of P_s and P_t, so that the knowledge for P_s is also useful for P_t. An intuitive example is that learning to recognize cats and dogs in the source task helps the recognition of ants and wasps in the target task, especially when the source task has a significantly larger dataset than that of the target task. The reason comes from the low-level notions of edges, shapes, etc., shared by many visual categories. In resist modeling, different lithography configurations can be viewed as separate tasks with different distributions.

Typical transfer learning scheme for neural networks fixes the first several layers of the model trained for another domain and finetune the successive layers with data from the target domain. The first several layers usually extract general features, which are considered to be similar between the source and the target domains, while the successive layers are classifiers or regressors that need to be adjusted. Figure 7 shows an example of the transfer learning scheme. We first train a model with source domain data and then use the source domain model as the starting point for the training of the target domain. During the training for the target domain, the first k layers are fixed, while the rest layers are finetuned. For simplicity, we denote this scheme as TF_k, shortened from "Transfer and Fix", where k is the parameter for the number of fixed layers.

Active Data Selection. Although transfer learning is potentially able to improve the accuracy of the target dataset using knowledge from a source dataset, the selection of representative target data samples may further improve the accuracy. Let D be the unlabeled dataset in the target domain and $\boldsymbol{s}$ be the set of selected data samples for label querying, where $|\boldsymbol{s}| \leq k$ and k is the maximum number of data samples for querying. For any $(\boldsymbol{x}_i, y_i) \in D$, $\boldsymbol{x}_i$ is the feature, e.g., aerial image, and

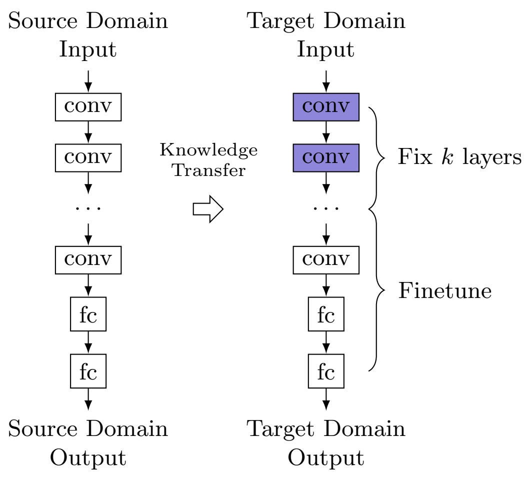


Fig. 7: Transfer learning scheme [21].

y_i is the label, e.g., threshold, where y_i is unknown for D. Consider a loss function $l(\boldsymbol{x}_i, y_i; \boldsymbol{w})$ parameterized over the hypothesis class ($\boldsymbol{w}$), e.g., parameters of a learning algorithm. The objective of active learning is to minimize the average loss of dataset D with a model trained from $\boldsymbol{s}$,

$$\min_{\boldsymbol{s}:|\boldsymbol{s}|\leq k, \boldsymbol{s}\in D} \frac{1}{n}\sum_{i=1}^{n} l\left(\boldsymbol{x}_i, y_i; \boldsymbol{w}_{\boldsymbol{s}}\right), \tag{16}$$

where $n = |D|$, and $\boldsymbol{w}_s$ represents the parameters of a model trained from $\boldsymbol{s}$.

Experiments. Table 3 presents the accuracy metrics, i.e., relative threshold RMS error (ε_r^{th}) and CD RMS error (ε^{CD}), for learning N7$_b$ from various source domain datasets. Since we consider the data efficiency of different learning schemes, we focus on the small training dataset for N7$_b$, from 1% to 20%. Situations such as no source domain data ($\emptyset$), only source domain data from N10 (N10$^{50\%}$), only source domain data from N7$_a$ (N7$_a^{50\%}$), and combined source domain datasets, are examined. The fidelity between relative threshold RMS error and CD RMS error is very consistent, so they share almost the same trends. Transfer learning with any source domain dataset enables an average improvement of 22% to 38% from that without knowledge transfer. In small training datasets of N7$_b$, ResNet also achieves around 10% better performance on average than CNN in the transfer learning scheme. At 1% of N7$_b$, combined source domain datasets have better performance compared with that of N10$^{50\%}$ only, but the benefits vanish with the increase of the N7$_b$ dataset.

Table 3: Relative Threshold RMS Error and CD RMS Error for N7_b with Different Source Domain Datasets

Source Datasets		$\emptyset$		N$10^{50\%}$				N$7_a^{50\%}$				N$10^{50\%}$ + N$7_a^{5\%}$		N$10^{50\%}$ + N$7_a^{10\%}$	
Neural Networks		CNN		CNN TF$_0$		ResNet TF$_0$		CNN TF$_0$		ResNet TF$_0$		ResNet TF$_0$		ResNet TF$_0$	
		ε_r^{th} (10^{-2})	ε^{CD}	ε_r^{th} (10^{-2})	ε^{CD}	ε_r^{th} (10^{-2})	ε^{CD}	ε_r^{th} (10^{-2})	ε^{CD}	ε_r^{th} (10^{-2})	ε^{CD}	ε_r^{th} (10^{-2})	ε^{CD}	ε_r^{th} (10^{-2})	ε^{CD}
N7_b	1%	4.31	4.67	2.23	2.36	1.88	2.00	1.55	1.65	1.36	1.45	1.70	1.83	1.63	1.75
	5%	2.57	2.74	1.79	1.94	1.57	1.71	1.60	1.73	1.40	1.52	1.66	1.81	1.63	1.77
	10%	1.95	2.10	1.73	1.87	1.52	1.65	1.54	1.67	1.39	1.50	1.54	1.66	1.56	1.69
	15%	1.83	1.98	1.60	1.76	1.42	1.56	1.49	1.62	1.32	1.44	1.43	1.56	1.45	1.58
	20%	1.67	1.81	1.56	1.70	1.36	1.48	1.47	1.59	1.31	1.42	1.38	1.51	1.41	1.54
ratio		1.00	1.00	0.78	0.79	0.68	0.69	0.69	0.70	0.62	0.62	0.69	0.69	0.69	0.70

3 Layout Hotspot Detection

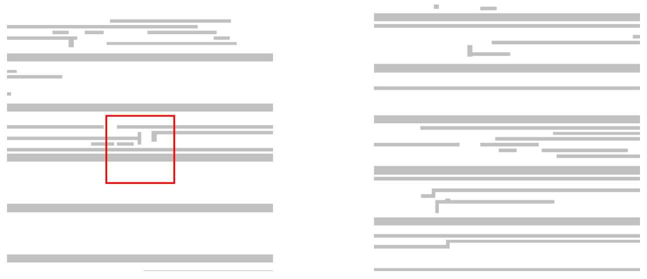

(a) Hotspot Pattern (b) Non-Hotspot Pattern

Fig. 8: Layout design examples that contain hotspots and hotspot-free. (Layout source [27])

Design-process weak points also known as hotspots cause systematic yield loss in semiconductor manufacturing. One of the main goals of DFM is to detect such hotspots. Traditionally, searching for hotspots heavily relies on lithography and manufacturing simulation which are accurate yet time-consuming. Since a layout design can be treated as an image, popular machine learning techniques in computer vision can be naturally applied for manufacturability hotspot detection. In this section, we will introduce how dedicated classification and object detection techniques enable fast and accurate hotspot detection frameworks.

Table 4: Machine Learning Solutions for Hotspot Detection.

Model	Framework	Keywords
Traditional ML	[28]	Layout Density; Decision Tree; AdaBoost
	[29]	Concentrated Circle Sampling; Naive Bayes; SmoothBoost
	[30]	Critical Feature Extraction; Multi-kernel SVM
Deep Learning	[31]	Feature Tensor Extraction; CNN; Batch Biased Learning
	[32]	ResNet; Binary Neural Networks
	[33]	Multi-task Learning; Transformer
	[34]	Metric Learning; Transformer
	[35]	Object Detection; Mask R-CNN; Hotspot Non-Maximum Suppression
	[36]	Multi-task Learning; CNN

3.1 *Machine Learning for Layout Hotspot Detection*

Research of machine learning-based hotspot detectors dates back to decades ago before the exploding of deep learning. This section will introduce the recent progress of hotspot detection research from traditional machine learning to emerging deep learning techniques.

Analyzing the layout to reduce lithography hotspots in the early stage is a necessary step in semiconductor manufacturing. Common traditional machine learning techniques for hotspot detection include SVM [30, 37], decision tree [28] and Bayes method [29]. [28, 29] are also two representation contributions using ensemble learning. Feature engineering plays an important role in legacy machine learning solutions, which convert original designs into complete and discriminating layout representations. Effective layout features includes layout density [28], Concentrated Circle Sampling [29], Critical Dimension [30] and Tangent Space [38]. Layout feature engineering and learning model together contribute to the research of traditional machine learning on hotspot detection.

Exploding of deep learning techniques has made feature engineering trivial, where, mostly, layout images can be easily converted to discriminative feature space via trained convolutional neural networks. [39, 40] are the two earliest works using CNNs for lithography hotspot detection, where standard VGG-like CNN structures are employed for automatic feature learning and label prediction.

Following the pioneering CNN solution on hotspot detection. [31] presented a feature tensor extraction method to perform representative layout features and got remarkable speed up with the deep neural network. A batch-biased learning training algorithm is also proposed to achieve a better trade-off between hotspot detection accuracy and false-positive penalty. More advanced layout feature-friendly neural network architectures are investigated to enhance hotspot detection performance. [32], for the first time, brings the binary neural network to hotspot detection tasks,

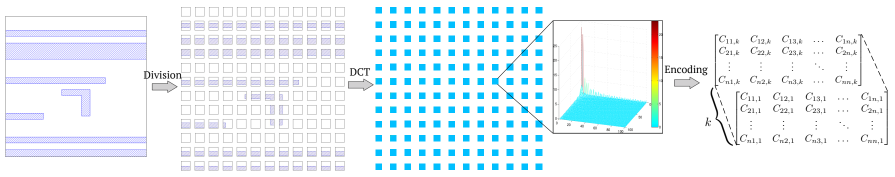


Fig. 9: Feature Tensor generation example ($n = 12$). The original clip (1200×1200 nm^2) is divided into 12×12 blocks and each block is converted to a 100×100 image representing a 100×100 nm^2 sub region of the original clip. Feature tensor is then obtained by encoding on first k DCT coefficients of each block.

which achieves state-of-the-art results with much smaller neural models and faster inference time. [34] starts investigating the popular transformer structure that incorporates spatial and channel attention in CNN layers. Such methods have been demonstrated efficient in feature learning. [36] considers the availability of data in hotspot detection tasks. When there is a limited number of labeled data, the proposed multi-task learning in [36] learns regular classification cross-entropy loss and an unsupervised contrastive loss to handle the drawbacks of pseudo-label training flow.

Conventional classification-based solutions have potential challenges due to the increased design complexity. To tackle this problem, Chen *et al.* proposed a region-based hotspot detection framework to detect multiple hotspots on large scales [35]. The framework takes a full/large-scale layout design as the input and performs classification and regression at the same time to localize the hotspot regions. They report a $45\times$ speedup compared to the classification-based deep learning model. A transformer encoder-based single-stage detection network in [33] can capture the long-range dependencies between polygons and makes the region-based detector more robust.

3.2 Case Studies

In this section, we will introduce two representative hotspot detection solutions based on classification and object detection.

3.2.1 Detecting Lithography Hotspots with Feature Tensor Generation and Batch-biased Learning [31]

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

Spectral analysis of mask patterns for wafer clustering was recently explored in literature and achieved good clustering performance. Inspired by these research, we express the sub-region as a finite combination of different frequency components. The high sparsity of the discrete cosine transform (DCT) makes it preferable over other frequency representations in terms of spectral feature extraction, and it is consistent with the expected properties of the feature tensor. To sum up, the process of feature tensor generation contains the following steps.

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

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

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

where $B = \frac{N}{n}$ is sub-region size, (x,y) and (m,n) are original layout image and frequency domain indexes respectively. Particularly, the upper-left side of DCT coefficients in each block corresponds to low-frequency components, that contain high-density information, as depicted in Fig. 9.

Step 3: Flatten $\mathbf{D}_{i,j}$'s into vectors in Zig-Zag form [41] with the larger index being higher frequency coefficients as follows.

$$\mathbf{C}^*_{i,j} = [\mathbf{D}_{i,j}(0,0), \mathbf{D}_{i,j}(0,1), \mathbf{D}_{i,j}(1,0), ..., \mathbf{D}_{i,j}(B,B)]^\intercal. \tag{17}$$

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

$$\mathbf{C}_{i,j} = \mathbf{C}^*_{i,j}[:k], \tag{18}$$

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

$$\mathbf{F} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} & \mathbf{C}_{13} & \ldots & \mathbf{C}_{1n} \\ \mathbf{C}_{21} & \mathbf{C}_{22} & \mathbf{C}_{23} & \ldots & \mathbf{C}_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_{n1} & \mathbf{C}_{n2} & \mathbf{C}_{n3} & \ldots & \mathbf{C}_{nn} \end{bmatrix}, \tag{19}$$

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

The nature of discrete cosine transform ensures that high-frequency coefficients are near zero. As shown in Fig. 9, large responses only present at the entries with smaller indexes, i.e. low-frequency regions. Therefore, most information is kept even when a large number of elements in $\mathbf{C}^*_{i,j}$ are dropped. The feature tensor also has the following advantages when applied in neural networks: (1) Highly compatible with the data packet transference in convolutional neural networks and (2) forward propagation time is significantly reduced when compared with using an original layout image as input, because the scale of the neural network is reduced with the smaller input size.

Batch-biased Learning. Intuitively, a too confident classifier is not necessary to give a good prediction performance and on the contrary, may induce more training pressure or even overfitting problems. Therefore, we propose a new learning algorithm to dynamically adjust the learning taget for different instances. We define a bias function as follows:

$$\varepsilon(l) = \begin{cases} \frac{1}{1+\exp(\beta l)}, & \text{if } l \leq 0.3, \\ 0, & \text{if } l > 0.3, \end{cases} \tag{20}$$

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

The training procedure is summarized as **Algorithm** 1, where β is a hyper-parameter defined in Equation (20). We initialize the neural network (line 1) and update the weight until meeting the convergence condition (lines 2–16). Within each iteration, we first sample the same amount of hotspot and non-hotspot instances to make sure the training procedure is balanced (lines 3–4); we then calculate the average loss of non-hotspot instances to obtain the bias level and the biased ground truth (lines 5–6); the gradients of the hotspot and non-hotspot instances are calculated separately (lines 8–9); the rest of the steps are the normal weight update through back-propagation and learning rate decay (lines 11–15).

Experiments. We compare the hotspot detection results with four state-of-the-art hotspot detectors in TABLE 5. Here "Accu (%)" denotes the hotspot prediction accuracy and "FA#" represents the number of false positives. "SPIE'15 [28]" is a traditional machine learning-based hotspot detector that applies the density-based layout features and the AdaBoost–DecisionTree model. "ICCAD'16 [29]" takes the lithographic properties into account during feature extraction and adopts the more robust Smooth Boosting algorithm. "SPIE'17 [42]" is another deep learning solution for hotspot detection that takes the original layout image as input and contains more than 20 layers. "SOCC'17 [43]" employs deep neural networks that replace all pooling layers with strided convolution layers and contain the same number of layers as SPIE'17.

Overall, the framework performs better than traditional machine learning techniques (SPIE'15 [28] and ICCAD'16 [29]) with at least a 2% advantage for the detection accuracy (98.88% of BBL v.s. 96.89% of ICCAD'16) on target layouts. Traditional machine learning models are effective for the benchmarks with regular

Algorithm 1 Batch Biased-learning

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

Table 5: Performance comparison with state-of-the-art hotspot detectors on target layouts.

Benchmarks	SPIE'15[28]		ICCAD'16[29]		SOCC'17[43]		SPIE'17[42]		BBL+AUG	
	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#
`ICCAD`	84.20	2919	97.70	4497	96.90	**1960**	97.70	2703	**98.40**	3535
`Industry0`	93.63	**30**	96.07	1148	97.77	100	97.55	85	**99.36**	387
Average	88.92	1475	96.89	2823	97.34	**1030**	97.63	1394	**98.88**	1961
Ratio	0.899	0.752	0.980	1.439	0.984	**0.525**	0.987	0.711	**1.000**	1.000

patterns (`ICCAD` and `Industry0`) with the highest detection accuracy of 97.7% on the `ICCAD` and 96.07% on the `Industry0` achieved by [29].

3.2.2 Faster Region-based Hotspot Detection [35]

The proposed region-based hotspot detection (R-HSD) neural network, as illustrated in Figure 10, is composed of three steps: (1) feature extraction, (2) clip proposal network, and (3) refinement. In this section, we will discuss each step in detail. At first glance, R-HSD problem is similar to the object detection problem, which is a hot topic in the computer vision domain recently. In object detection problems, objects with different shapes, types, and patterns are the target to be detected. However, as we will discuss, there is a gap between hotspot detection and object detection, e.g. the hotspot pattern features are quite different from the objects in real scenes,

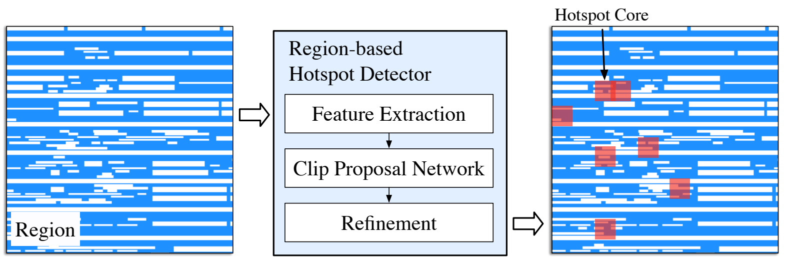


Fig. 10: The proposed region-based hotspot detection flow.

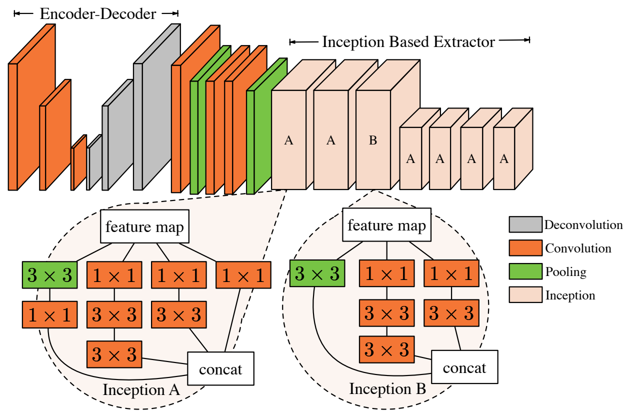


Fig. 11: Tensor structure of feature extractor.

thus typical strategies and frameworks utilized in object detection cannot be applied here directly.

Inception-based Feature Extraction. According to the recent progress of deep learning-based research in the computer vision region, a deeper neural network can give a much more robust feature expression and get higher accuracy compared to a shallow structure as it increases the model complexity. However, it also brings sacrifice on the speed at both the inference stage and training stage. Another point we need to be concerned about is that the feature expression of the layout pattern is monotonous, while the feature space of layout patterns is still in low dimension after we transform it by the Encoder-Decoder structure. According to these issues, we propose an Inception-based structure. The following three points are the main rules of our design:

- Increase the number of filters in width at each stage. For each stage, multiple filters do the convolution operation with different convolution kernel sizes and then concatenate them in channel direction as feature fusion.
- Prune the depth of the output channel for each stage.
- Downsample the feature map size in height and width direction.

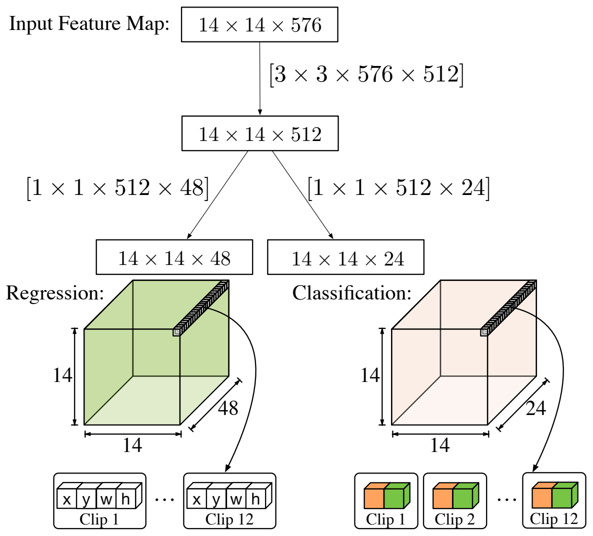

Input Feature Map:
$14 \times 14 \times 576$
$[3 \times 3 \times 576 \times 512]$
$14 \times 14 \times 512$
$[1 \times 1 \times 512 \times 48]$
$[1 \times 1 \times 512 \times 24]$
$14 \times 14 \times 48$
$14 \times 14 \times 24$
Regression:
Classification:
14
48
24
x y w h
Clip 1
Clip 2
Clip 12

Fig. 12: Kernel Work flow of Clip Proposal Network.

With the above rules, the inception structure [44] can take a good balance between accuracy and time. The blobs shown in Figure 11 are what we apply in our framework. We construct module A with the operation stride one and four branches. The aim of module A is to extract multiple features without downsampling the feature map. The operation stride of each layer in module B is two. This setting makes the output feature map half of the input and further reduces the operations . We only use one Module B here, because the feature map size should not be too small, while the low dimension of feature expression in final layers may bring negative effects to the final result.

The 1×1 convolution kernel with low channel numbers brings the dimension reduction which controls the number of the parameters and operations. The multiple branches bring more abundant feature expressions, which give the network ability to do the kernel selection with no operation penalty.

Clip Proposal Networks. Given the extracted features, a clip proposal network is developed here to detect potential hotspot clips. For both feature maps and convolutional filters, the tensor structures of the clip proposal network are illustrated in Figure 12. Per preliminary experiments, clips with a single aspect ratio and scale (e.g. square equal to the ground truth) may lead to poor performance. Therefore, for each pixel in the feature map, a group of 12 clips with different aspect ratios is generated. The network is split into two branches: one is for classification and the other is for regression. In the classification branch, for each clip, a probability of hotspot and a probability of non-hotspot are calculated through the softmax function. In the regression branch, the location and the shape of each clip are determined by a vector $[x, y, w, h]$.

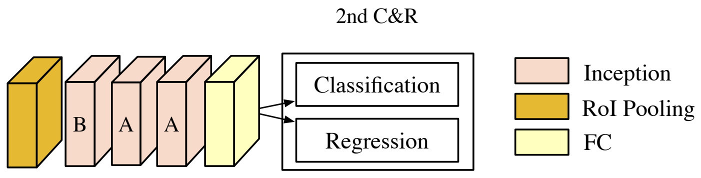


Fig. 13: Tensor structure of Refinement.

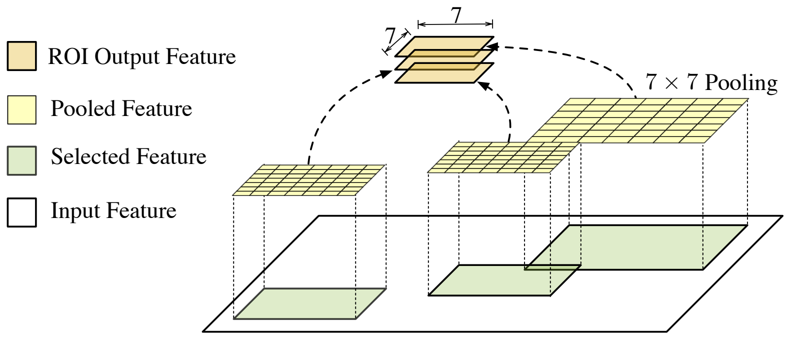


Fig. 14: Visualized 7×7 RoI pooling.

Refinement Network. After the prediction of the first classification and regression in the clip proposal network stage, we get a rough prediction on hotspot localization and filtered region which is classified as non-hotspot. While the greedy method of clip filtering cannot guarantee all the reserved clips are classified correctly, the false alarm may be too high. To bring a robust detection with a lower false alarm, we further construct refinement stage in the whole neural network, which includes a Region of Interests (RoI) pooling layer, three inception modules, as well as another classification and regression. The structure of Refinement is shown in Figure 13.

The coordinates of each clip are the actual location from the original input image. We scale down the coordinates to conform with the spatial extent of the last feature map before the refinement. In traditional image processing, the most common ways to resize the image are cropping and warping. The crop method cuts the pattern boundary to fix the target size which leads to information loss. The warping will reshape the size which changes the shape of origin features. Here we apply Region of Interests (RoI) pooling to transform the selected feature region $h \times w$ to a fixed spatial size of $H \times W$ (H and W are the hyperparameters, and we use 7×7 in this work). For each pooled feature region $\lfloor h/H \times w/W \rfloor$, the max-pooling is applied independently. The RoI pooling transforms clips with different sizes into a fixed size which reserves the whole feature information and makes further hotspot classification and regression feasible. Figure 14 gives an example of RoI pooling operations.

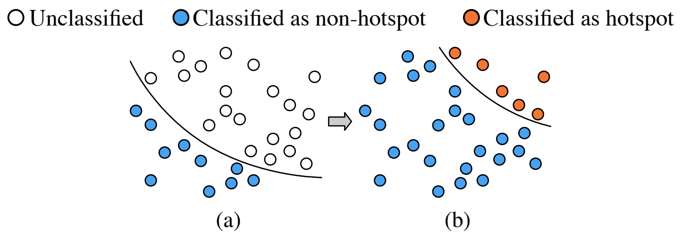


Fig. 15: (a) The first hotspot classification in clip proposal network; (b) The labelled hotspots are fed into the 2nd hotspot classification in refinement stage to reduce false alarm.

Besides classification and regression in clip proposal network, here additional classification and regression are designed to finetune the clip location and give a more reliable classification result. At this stage, most non-hotspot clips have been removed, thus the two stages of hotspot classification can efficiently reduce false alarms. Figure 15 illustrates the flow of the two-stage hotspot classification.

Loss Function. We design a multi-task loss function called Classification and Regression (C&R) to calibrate our model. As shown in Figure 12 and Figure 13, C&R is applied both in clip proposal network stage and refinement stage. The input tensors of the 1st C&R are boxes in Figure 12. W, H, and C are width, height, and channel respectively. The probability score of the hotspot, non-hotspot and prediction of clip coordinates are grouped in the channel direction. Here x and y are the coordinates of the hotspot, which means the center of the clip area. w, h are the width and height of the clip. In the 2nd C&R, the tensor flow of the classification and regression are the same as [45] using fully-connected layers.

In the task of region-based hotspot detection, h_i is the predicted probability of clip i being a hotspot, h_i' is the groundtruth of clip i. $\boldsymbol{l}_i = \{l_x, l_y, l_w, l_h\} \in \mathbb{R}^4$ and $\boldsymbol{l}_i' = \{l_x', l_y', l_w', l_h'\} \in \mathbb{R}^4$ are assigned as coordinates of clips with index i representing the encoded coordinates of the prediction and groundtruth respectively. The encoded coordinates can be expressed as:

$$\begin{aligned} l_x &= (x - x_g)/w_g, & l_y &= (y - y_g)/h_g, \\ l_x' &= (x' - x_g)/w_g, & l_y' &= (y' - y_g)/w_g, \\ l_w &= \log(w/w_g), & l_h &= \log(h/h_g), \\ l_w' &= \log(w'/w_g), & l_h' &= \log(h'/h_g), \end{aligned} \tag{21}$$

Variables x, x_g and x' are for the prediction of clip, g-clip, and the ground-truth clip respectively (same as the y, w, and h).

The classification and regression loss function for clips can be expressed as:

$$
\begin{aligned}
L_{C\&R}(h_i, \boldsymbol{l}_i) = & \alpha_{loc} \sum_i h_i^{'} l_{loc}(\boldsymbol{l}_i, \boldsymbol{l}_i^{\prime}) + \sum_i l_{hotspot}(h_i, h_i^{'}) \\
& + \frac{1}{2}\beta(\|\boldsymbol{T}_{loc}\|_2^2 + \|\boldsymbol{T}_{hotspot}\|_2^2),
\end{aligned} \tag{22}
$$

where β is a hyperparameter which controls the regularization strength. α_{loc} is the hyperparameter which controls the balance between two tasks. $\boldsymbol{T}_{loc}$ and $\boldsymbol{T}_{hotspot}$ are the weights of the neural network. For elements $l_i[j]$ and $l_i^{\prime}[j]$ ($j \in [1,4]$) in $\boldsymbol{l}_i, \boldsymbol{l}_i^{\prime}$ respectively, l_{loc} can be expressed as

$$
l_{loc}(l_i[j], l_i^{\prime}[j]) = \begin{cases} \frac{1}{2}(l_i[j] - l_i^{\prime}[j])^2, & \text{if } |l_i[j] - l_i^{\prime}[j]| < 1, \\ |l_i[j] - l_i^{\prime}[j]| - 0.5, & \text{otherwise,} \end{cases} \tag{23}
$$

which is a robust L_1 loss used to avoid the exploding gradients problem at training stage. $l_{hotspot}$ is the cross-entropy loss which is calculated as:

$$
l_{hotspot}(h_i, h_i^{'}) = -(h_i \log h_i^{'} + h_i^{'} \log h_i). \tag{24}
$$

Experiments. We list the detailed result comparison in Table 6. Column “Bench” lists three benchmarks used in our experiments. Columns “Acc”, “FA”, “Time” denotes hotspot detection accuracy, false alarm count and detection runtime, respectively. Column “TCAD’18” lists the result of a deep learning-based hotspot detector proposed in [31] that adopts frequency-domain feature extraction and biased learning strategy. We also implement two baseline frameworks that employ Faster R-CNN [46] and SSD [47], respectively, which are two classic techniques matching our region-based hotspot detection objectives well. The corresponding results are listed in columns “Faster R-CNN [46]” and “SSD [47]”. The results show that our framework gets better hotspot detection accuracy on average with 6.14% improvement with $\sim$ 200 less false alarm penalty compared to [31]. Especially, our framework behaves much better on `Case2` with 93.02% detection accuracy compared to 77.78%, 1.8%, and 71.9% for [31], Faster R-CNN and SSD, respectively. The advantage of the proposed two-stage classification and regression flow can also be seen here that [31] achieves similar hotspot detection accuracy compared to our framework but has extremely large false alarms that will introduce additional. It should be noted that the detection runtime is much faster than [31] thanks to the region-based detection scheme. We can also observe that although Faster R-CNN and SSD are originally designed for large region object detection, they perform poorly on hotspot detection tasks which reflects the effectiveness and efficiency of our customized framework.

Table 6: Comparison with State-of-the-art

Bench	TCAD'18 [31]			Faster R-CNN [46]			SSD [47]			Ours		
	Acc (%)	FA	Time (s)	Acc (%)	FA	Time (s)	Acc (%)	FA	Time (s)	Acc (%)	FA	Time (s)
Case2	77.78	48	60.0	1.8	3	1.0	71.9	519	1.0	93.02	17	2.0
Case3	91.20	263	265.0	57.1	74	11.0	57.4	1730	3.0	94.5	34	10.0
Case4	100.00	511	428.0	6.9	69	8.0	77.8	275	2.0	100.00	201	6.0
Average	89.66	274.0	251.0	21.9	48.7	6.67	69.0	841.3	2.0	95.8	84	6.0
Ratio	1.00	1.00	1.00	0.24	0.18	0.03	0.87	3.07	**0.01**	**1.07**	**0.31**	0.02

4 Mask Optimization

RETs try to minimize the error when transferring a design onto silicon wafers. Mainstream solutions vary from lithography source configuration [48, 49], mask pattern optimization [50, 11, 8, 51] to multiple patterning lithography [52, 5]. Among the above, mask optimization is one of the most critical stages in sign-off flows. It tweaks the features or contexts of design layout patterns to circumvent side effects from the lithography proximity effect, which requires frequent interaction with lithography simulation engines, resulting in significant computation overhead. Also, mask optimization recipes need to be carefully crafted for good result convergence.

4.1 Machine Learning for Mask Optimization

Mask Optimization involves constrained optimization and mask printability querying. This requires an extensive understanding of lithography physics. The exploding of machine learning and deep neural networks bring opportunities to learn lithography behavior either implicitly or explicitly. We will introduce recent research from the perspective of SRAF and OPC (as in Table 7).

SRAF Insertion. Sub-resolution assist feature (SRAF) insertion is one representative strategy among numerous RET techniques. Without printing SRAF patterns themselves, the small SRAF patterns can transfer light to the positions of target patterns, and therefore SRAFs are able to improve the robustness of the target patterns under different lithographic variations. [8] is one of the earliest works using machine learning for SRAF insertion. Given a layout location, the machine learning model is trained to identify whether the location should include an SRAF or not. Concentrated circle area sampling is investigated for feature extraction and applied on legacy machine learning models. [7] improves [8] from the perspective of feature engineering. A supervised dictionary learning is proposed to further optimize the concentrated circle area sampling features. However, legacy machine learning approaches still suffer runtime and accuracy issues. Therefore, GAN-SRAF [51] for the first time brings conditional generative adversarial networks for SRAF insertion.

Table 7: Machine Learning Solutions for Mask Optimization.

Task	Framework	Kewords
SRAF	[8]	Concentrated Area Sampling; Decision Tree; Logistic Regression
	[7]	Concentrated Area Sampling; Dictionary Learning; SVM
	GAN-SRAF [51]	Conditional GAN; Heat Map Encoding
OPC	[53]	Concentrated Area Sampling; EPE Prediction; XGBoost
	[54]	Concentrated Area Sampling; Bayes Model; Markov Chain Monte Carlo
	GAN-OPC [55]	Conditional GAN; ILT-guided Training; UNet
	DAMO [18]	Conditional GAN; UNet++; Perceptual Loss
	DevelSet [14]	Conditional GAN; LevelSet; Signed Distance Field

A novel heat map encoding is proposed to address the problem that neural networks are not talented to generate sharp edges.

OPC. Early attempts using machine learning for OPC regard the entire optimization process as a black box. [54] builds up a Bayes model to directly predict the edge correction level in model-based OPC that can reduce the overall OPC iterations by a significant amount. Since model-based OPC limits the solution space for mask optimization, [55, 14] tend to generate initial solutions at the format of coverage image and levelset field respectively. These initial solutions can then be fed into the legacy inverse lithography technique (ILT) engine for faster and better optimization performance.

Machine learning-based OPC solutions all deal with the overhead of lithography simulation. Instead of reducing iterations and lithography simulator query count, [53] works on per-iteration lithography query time. It builds a machine learning-based lithography estimator that predicts EPE at certain OPC control points and guides edge segment movement. DAMO [18] is also a framework developed for fast OPC iterations using a deep lithography simulator (DLS). The DLS is able to backpropagate lithography error gradients back to another neural network for mask generation.

4.2 Case Studies

We will discuss details of one legacy machine learning solution for SRAF insertion [7] and the pioneer work using GAN for mask optimization [55].

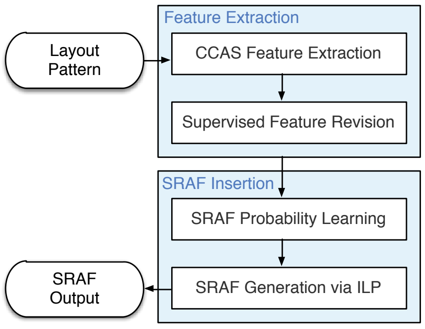


Fig. 16: The proposed SRAF insertion flow.

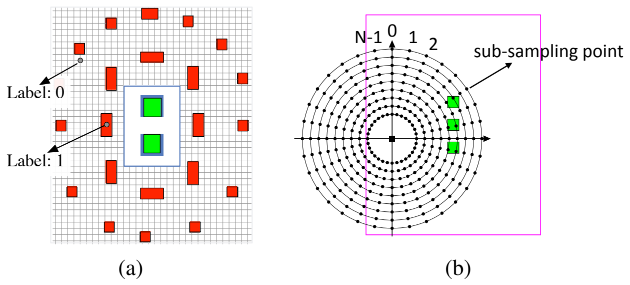


Fig. 17: (a) SRAF label; (b) CCAS feature extraction method in machine learning model-based SRAF generation.

4.2.1 SRAF Insertion with Supervised Dictionary Learning [7]

The overall flow of SRAF insertion is shown in Figure 16, which consists of two stages: feature extraction and SRAF insertion. In the feature extraction stage, after feature extraction via concentric circle area sampling (CCAS), we propose supervised feature revision, namely, mapping features into a discriminative low-dimension space. Through dictionary training, our dictionary consists of atoms which are representatives of original features. The original features are sparsely encoded over a well-trained dictionary and described as combinations of atoms. Due to space transformation, the new features (i.e. sparse codes) are more abstract and discriminative with little important information loss for classification. Therefore, proposed supervised feature revision is expected to avoid over-fitting of a machine learning model. In the second stage, based on the predictions inferred by the learning model, SRAF insertion can be treated as a mathematical optimization problem accompanied by taking design rules into consideration.

Supervised Feature Revision. With considering concentric propagation of diffracted light from mask patterns, the recently proposed CCAS [56] layout feature is used in

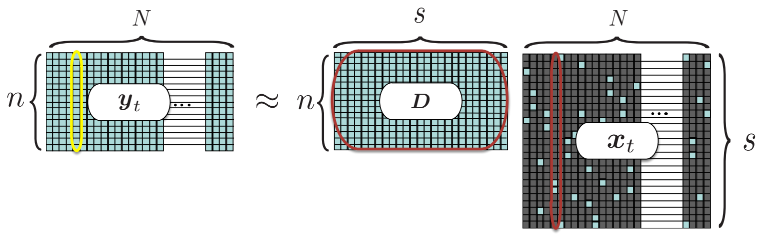


Fig. 18: The overview of a dictionary learning model.

the SRAF generation domain. In SRAF insertion, the raw training data set is made up of layout clips, which include a set of target patterns and model-based SRAFs. Each layout clip is put on a 2-D grid plane with a specific grid size so that real training samples can be extracted via the CCAS method at each grid. For every sample, according to the model-based SRAFs, the corresponding label is either “1” or “0”. As Figure 17a illustrates, “1” means inserting an SRAF at this grid and “0” vice versa. Figure 17b shows the feature extraction method in SRAF generation. However, since adjacent circles contain similar information, the CCAS feature has much redundancy. In fact, the redundancy will hinder the fitting of a machine learning model.

With the CCAS feature as input, the dictionary learning model is expected to output the discriminative feature of low-dimension. In the topic of data representation [57], a self-adaptive dictionary learning model can sparsely and accurately represent data as linear combinations of atoms (i.e., columns) from a dictionary matrix. This model reveals the intrinsic characteristics of raw data. In recent arts, sparse decomposition and dictionary construction are coupled in a self-adaptive dictionary learning framework. As a result, the framework can be modeled as a constrained optimization problem. The joint objective function of a self-adaptive dictionary model for feature revision problem is proposed as Equation (25):

$$\min_{\boldsymbol{x},\boldsymbol{D}} \frac{1}{N}\sum_{t=1}^{N}\{\frac{1}{2}\left\|\boldsymbol{y}_t-\boldsymbol{D}\boldsymbol{x}_t\right\|_2^2+\lambda\left\|\boldsymbol{x}_t\right\|_p\}, \tag{25}$$

where $\boldsymbol{y}_t \in \mathbb{R}^n$ is an input CCAS feature vector, and $\boldsymbol{D}=\left\{\boldsymbol{d}_j\right\}_{j=1}^{s}, \boldsymbol{d}_j \in \mathbb{R}^n$ refers to the dictionary made up of atoms to encode input features. $\boldsymbol{x}_t \in \mathbb{R}^s$ indicates sparse codes (i.e. sparse decomposition coefficients) with p the type of norm. Meanwhile, N is the total number of training data vectors in memory. The above equation, illustrated in Figure 18, consists of a series of reconstruction error, $\|\boldsymbol{y}_t-\boldsymbol{D}\boldsymbol{x}_t\|_2^2$, and the regularization term $\|\boldsymbol{x}_t\|_p$. In Figure 18, every grid represents a numerical value, and dark grid of $\boldsymbol{x}_t$ indicates zero. It can be seen that the motivation of dictionary learning is to sparsely encode input CCAS features over a well-trained dictionary.

However, from Equation (25), it is easy to discover that the main optimization goal is minimizing the reconstruction error in a mean squared sense, which may not be compatible with the goal of classification. Therefore, we try to explore the supervised information, and then propose our joint objective function as Equation (26). An assumption has been made in advance that each atom is associated with a particular label, which is true as each atom is selected to represent a subset of the training CCAS features ideally from one class (i.e. occupied with an SRAF or not).

$$\min_{\boldsymbol{x},\boldsymbol{D},\boldsymbol{A}} \frac{1}{N}\sum_{t=1}^{N}\{\frac{1}{2}\left\|\left(\boldsymbol{y}_t^\top, \sqrt{\alpha}\boldsymbol{q}_t^\top\right)^\top - \begin{pmatrix}\boldsymbol{D}\\ \sqrt{\alpha}\boldsymbol{A}\end{pmatrix}\boldsymbol{x}_t\right\|_2^2 + \lambda\|\boldsymbol{x}_t\|_p\}. \tag{26}$$

In Equation (26), α is a hyper-parameter balancing the contribution of each part to reconstruction error. $\boldsymbol{q}_t \in \mathbb{R}^s$ is defined as discriminative sparse code of t-th input feature vector. Hence, $\boldsymbol{A} \in \mathbb{R}^{s\times s}$ transforms original sparse code $\boldsymbol{x}_t$ into discriminative sparse code. In $\boldsymbol{q}_t$, the non-zero elements indicate that the corresponding atoms share the same label with t-th input. Given dictionary $\boldsymbol{D}$, it is obvious that $\boldsymbol{q}_t$ has some fixed types.

To illustrate physical meaning of Equation (26) clearly, we can also rewrite it via splitting the reconstruction term into two terms within l_2-norm as (27):

$$\min_{\boldsymbol{x},\boldsymbol{D},\boldsymbol{A}} \frac{1}{N}\sum_{t=1}^{N}\{\frac{1}{2}\|\boldsymbol{y}_t - \boldsymbol{D}\boldsymbol{x}_t\|_2^2 + \frac{\alpha}{2}\|\boldsymbol{q}_t - \boldsymbol{A}\boldsymbol{x}_t\|_2^2 + \lambda\|\boldsymbol{x}_t\|_p\}. \tag{27}$$

The first term $\|\boldsymbol{y}_t - \boldsymbol{D}\boldsymbol{x}_t\|_2^2$ is still the reconstruction error term. The second term $\|\boldsymbol{q}_t - \boldsymbol{A}\boldsymbol{x}_t\|_2^2$ represents discriminative error, which imposes a constraint on the approximation of $\boldsymbol{q}_t$. As a result, the input CCAS features from the same class share quite similar representations.

Since the latent supervised information has been used, the label information can also be directly employed. After adding the prediction error term into initial objective function Equation (26), we propose our final joint objective function as Equation (28):

$$\min_{\boldsymbol{x},\boldsymbol{D},\boldsymbol{A},\boldsymbol{W}} \frac{1}{N}\sum_{t=1}^{N}\{\frac{1}{2}\left\|\left(\boldsymbol{y}_t^\top, \sqrt{\alpha}\boldsymbol{q}_t^\top, \sqrt{\beta}h_t\right)^\top - \begin{pmatrix}\boldsymbol{D}\\ \sqrt{\alpha}\boldsymbol{A}\\ \sqrt{\beta}\boldsymbol{W}\end{pmatrix}\boldsymbol{x}_t\right\|_2^2 + \lambda\|\boldsymbol{x}_t\|_p\}, \tag{28}$$

where $h_t \in \mathbb{R}$ is the label with $\boldsymbol{W} \in \mathbb{R}^{1\times s}$ the related weight vector, and therefore $\|h_t - \boldsymbol{W}\boldsymbol{x}_t\|_2^2$ refers to the classification error. α and β are hyperparameters that control the contribution of each term to reconstruction error and balance the trade-off. So this formulation can produce a good representation of the original CCAS feature.

Online Algorithm. According to our proposed formulation (i.e. Equation (28)), the joint optimization of both dictionary and sparse codes is non-convex, but the sub-problem with one variable fixed is convex. Hence, Equation (28) can be divided into two convex sub-problems. Note that, in a taste of linear algebra, our new input with label information, i.e. $\left(\boldsymbol{y}_t^\top, \sqrt{\alpha}\boldsymbol{q}_t^\top, \sqrt{\beta}h_t\right)^\top$ in Equation (28), can be still regarded as the original $\boldsymbol{y}_t$ in Equation (25). So is the new merged dictionary consisting of $\boldsymbol{D}$, $\boldsymbol{A}$ and $\boldsymbol{W}$. For simplicity of description and derivation, in following analysis, we will use $\boldsymbol{y}_t$ referring to $\left(\boldsymbol{y}_t^\top, \sqrt{\alpha}\boldsymbol{q}_t^\top, \sqrt{\beta}h_t\right)^\top$ and $\boldsymbol{D}$ standing for merged dictionary with $\boldsymbol{x}$ as the sparse codes.

Two-stage sparse coding and dictionary constructing alternatively perform in iterations. Thus, in the t-th iteration, the algorithm firstly draws the input sample $\boldsymbol{y}_t$ or a mini-batch over the current dictionary $\boldsymbol{D}_{t-1}$ and obtains the corresponding sparse codes $\boldsymbol{x}_t$. Then it uses two updated auxiliary matrices, $\boldsymbol{B}_t$ and $\boldsymbol{C}_t$ to help computing $\boldsymbol{D}_t$.

The objective function for sparse coding is shown in (29):

$$\boldsymbol{x}_t \overset{\Delta}{=} \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{y}_t - \boldsymbol{D}_{t-1}\boldsymbol{x}\|_2^2 + \lambda \|\boldsymbol{x}\|_1. \tag{29}$$

If the regularizer adopts l_0-norm, solving Equation (29) is NP-hard. Therefore, we utilize l_1-norm as a convex replacement of l_0-norm. In fact, Equation (29) is the classic Lasso problem [58], which can be solved by any Lasso solver.

Two auxiliary matrices $\boldsymbol{B}_t \in \mathbb{R}^{(n+s+1)\times s}$ and $\boldsymbol{C}_t \in \mathbb{R}^{s\times s}$ are defined respectively in (30) and (31):

$$\boldsymbol{B}_t \leftarrow \frac{t-1}{t}\boldsymbol{B}_{t-1} + \frac{1}{t}\boldsymbol{y}_t\boldsymbol{x}_t^\top, \tag{30}$$

$$\boldsymbol{C}_t \leftarrow \frac{t-1}{t}\boldsymbol{C}_{t-1} + \frac{1}{t}\boldsymbol{x}_t\boldsymbol{x}_t^\top. \tag{31}$$

The objective function for dictionary construction is:

$$\boldsymbol{D}_t \overset{\Delta}{=} \arg\min_{\boldsymbol{D}} \frac{1}{t}\sum_{i=1}^{t}\{\frac{1}{2}\|\boldsymbol{y}_i - \boldsymbol{D}\boldsymbol{x}_i\|_2^2 + \lambda\|\boldsymbol{x}_i\|_1\}. \tag{32}$$

Algorithm 2 Supervised Online Dictionary Learning (SODL)

```
Require: Input merged features $\boldsymbol{Y} \leftarrow \{\boldsymbol{y}_t\}_{t=1}^N, \boldsymbol{y}_t \in \mathbb{R}^{(n+s+1)}$ (including original CCAS features,
    discriminative sparse code $\boldsymbol{Q} \leftarrow \{\boldsymbol{q}_t\}_{t=1}^N, \boldsymbol{q}_t \in \mathbb{R}^s$ and label information $\boldsymbol{H} \leftarrow \{h_t\}_{t=1}^N, h_t \in \mathbb{R}$).
Ensure: New features $\boldsymbol{X} \leftarrow \{\boldsymbol{x}_t\}_{t=1}^N, \boldsymbol{x}_t \in \mathbb{R}^s$, dictionary $\boldsymbol{D} \leftarrow \left\{\boldsymbol{d}_j\right\}_{j=1}^s, \boldsymbol{d}_j \in \mathbb{R}^{(n+s+1)}$.
1: Initialization: Initial merged dictionary $\boldsymbol{D}_0, \boldsymbol{d}_j \in \mathbb{R}^{(n+s+1)}$ (including initial transforma-
   tion matrix $\boldsymbol{A}_0 \in \mathbb{R}^{s\times s}$ and initial label weight matrix $\boldsymbol{W}_0 \in \mathbb{R}^{1\times s}$), $\boldsymbol{C}_0 \in \mathbb{R}^{s\times s} \leftarrow \boldsymbol{0}$, $\boldsymbol{B}_0 \in$
   $\mathbb{R}^{(n+s+1)\times s} \leftarrow \boldsymbol{0}$;
2: for $t \leftarrow 1$ to $N$ do
3:     Sparse coding $\boldsymbol{y}_t$ and obtaining $\boldsymbol{x}_t$;          ▷ Equation (29)
4:     Update auxiliary variable $\boldsymbol{B}_t$;                      ▷ Equation (30)
5:     Update auxiliary variable $\boldsymbol{C}_t$;                      ▷ Equation (31)
6:     Update dictionary $\boldsymbol{D}_t$;                              ▷ Algorithm 3;
7: end for
```

Algorithm 2 summarizes the algorithm details of the proposed supervised online dictionary learning (SODL) algorithm. We use coordinate descent algorithm as the solving scheme to Equation (29) (line 3). To accelerate the convergence speed, Equation (32) involves the computations of past signals $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t$ and the sparse codes $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$. One way to efficiently update dictionary is to introduce some sufficient statistics, i.e. $\boldsymbol{B}_t \in \mathbb{R}^{(n+s+1)\times s}$ (line 4) and $\boldsymbol{C}_t \in \mathbb{R}^{s\times s}$ (line 5), into Equation (32) without directly storing the past input data sample $\boldsymbol{y}_i$ and corresponding sparse codes $\boldsymbol{x}_i$ for $i \leq t$. These two auxiliary variables play important roles in updating atoms, which summarize the past information from sparse coefficients and input data. We further exploit block coordinate method with warm start [59] to solve Equation (32) (line 6). As a result, through some gradient calculations, we bridge the gap between Equation (32) and sequentially updating atoms based on Equations (33) and (34).

$$\boldsymbol{u}_j \leftarrow \frac{1}{\boldsymbol{C}[j,j]}(\boldsymbol{b}_j - \boldsymbol{D}\boldsymbol{c}_j) + \boldsymbol{d}_j. \tag{33}$$

$$\boldsymbol{d}_j \leftarrow \frac{1}{\max\left(\left\|\boldsymbol{u}_j\right\|_2, 1\right)}\boldsymbol{u}_j. \tag{34}$$

For each atom $\boldsymbol{d}_j$, the updating rule is illustrated in Algorithm 3. In Equation (33), $\boldsymbol{D}_{t-1}$ is selected as the warm start of $\boldsymbol{D}$. $\boldsymbol{b}_j$ indicates the j-th column of $\boldsymbol{B}_t$, while $\boldsymbol{c}_j$ is the j-th column of $\boldsymbol{C}_t$. $\boldsymbol{C}[j,j]$ denotes the j-th element on diagonal of $\boldsymbol{C}_t$. Equation (34) is an l_2-norm constraint on atoms to prevent atoms from becoming arbitrarily large (which may lead to arbitrarily small sparse codes).

Algorithm 3 Rules for Updating Atoms

```
Require: $\boldsymbol{D}_{t-1} \leftarrow \{\boldsymbol{d}_j\}_{j=1}^{s}, \boldsymbol{d}_j \in \mathbb{R}^{(n+s+1)}$,
         $\boldsymbol{B}_t \leftarrow \{\boldsymbol{b}_j\}_{j=1}^{s}, \boldsymbol{b}_j \in \mathbb{R}^{(n+s+1)}$,
         $\boldsymbol{C}_t \leftarrow \{\boldsymbol{c}_j\}_{j=1}^{s}, \boldsymbol{c}_j \in \mathbb{R}^{s}$.
Ensure: dictionary $\boldsymbol{D}_t \leftarrow \{\boldsymbol{d}_j\}_{j=1}^{s}, \boldsymbol{d}_j \in \mathbb{R}^{(n+s+1)}$.
1: for $j \leftarrow 1$ to $s$ do
2:     Update the $j$-th atom $\boldsymbol{d}_j$;          ▷ Equations (33) and (34)
3: end for
```

SRAF Insertion via ILP. Through SODL model and classifier, the probabilities of each 2-D grid can be obtained. Based on design rules for the machine learning model, the label for a grid with a probability less than the threshold is "0". It means that the grid will be ignored when doing SRAF insertion. However, in [8], the scheme to insert SRAFs is a little naive and greedy. Actually, combined with some relaxed SRAF design rules such as maximum length and width, minimum spacing, the SRAF insertion can be modeled as an integer linear programming (ILP) problem. With the ILP model to formulate SRAF insertion, we will obtain a global view for SRAF generation.

In the objective of the ILP approach, we only consider valid grids whose probabilities are larger than the threshold. The probability of each grid is denoted as $p(i, j)$, where i and j indicate the index of a grid. For simplicity, we merge the current small grids into new bigger grids, as shown in Figure 19a. Then we define

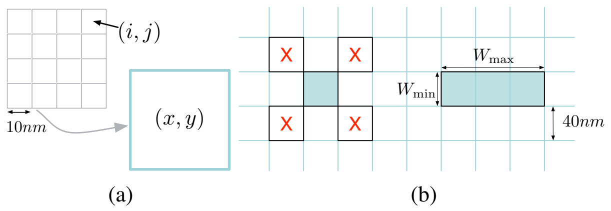


Fig. 19: (a) SRAF grid model construction; (b) SRAF insertion design rule under the grid model.

$c(x,y)$ as the value of each merged grid, where x,y denote the index of merged grid. The rule to compute $c(x,y)$ is as follows.

$$c(x,y) = \begin{cases} \sum_{(i,j)\in(x,y)} p(i,j), & \text{if } \exists\ p(i,j) \geq \text{threshold}, \\ -1, & \text{if all } p(i,j) < \text{threshold}. \end{cases} \tag{35}$$

The motivation behind this approach is twofold. One is to speed up the ILP. Because we can pre-determine some decision variables whose values are negative. The other is to keep the consistency of machine learning prediction.

In ILP for SRAF insertion, our real target is to maximize the total probability of valid grids with feasible SRAF insertion. Accordingly, it manifests to put up with the objective function, which is to maximize the total value of merged grids. The ILP formulation is shown in Formula (36).

$$\max_{a(x,y)} \quad \sum_{x,y} c(x,y)\cdot a(x,y) \tag{36a}$$

$$\text{s.t.} \quad a(x,y) + a(x-1,y-1) \leq 1, \qquad \forall(x,y), \tag{36b}$$

$$a(x,y) + a(x-1,y+1) \leq 1, \qquad \forall(x,y), \tag{36c}$$

$$a(x,y) + a(x+1,y-1) \leq 1, \qquad \forall(x,y), \tag{36d}$$

$$a(x,y) + a(x+1,y+1) \leq 1, \qquad \forall(x,y), \tag{36e}$$

$$a(x,y) + a(x,y+1) + a(x,y+2) + a(x,y+3) \leq 3, \qquad \forall(x,y), \tag{36f}$$

$$a(x,y) + a(x+1,y) + a(x+2,y) + a(x+3,y) \leq 3, \qquad \forall(x,y), \tag{36g}$$

$$a(x,y) \in \{0,1\}, \qquad \forall(x,y). \tag{36h}$$

Here $a(x,y)$ refers to the insertion situation at the merged grid (x,y). According to the rectangular shape of an SRAF and the spacing rule, the situation of two adjacent SRAFs on the diagonal is forbidden by Constraints (36b) to (36e); e.g. Constraint (36b) requires the $a(x,y)$ and the left upper neighbor $a(x-1,y-1)$ cannot be 1 at the same time, otherwise design rule violations can be generated.

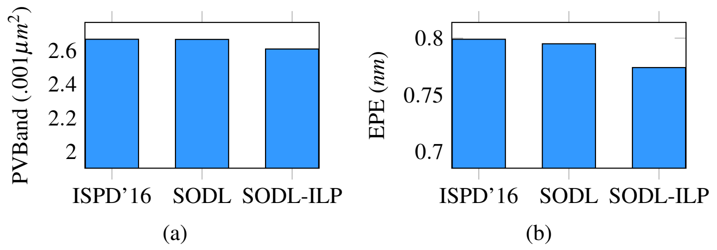


Fig. 20: Result comparison with previous work.

Constraints (36f) to (36g) restrict the maximum length of SRAFs. Figure 19b actively illustrates these linear constraints coming from design rules.

Experiments. Figure 20 compares the results with a state-of-the-art machine learning based SRAF insertion tool "ISPD'16" [8]. It can be seen from the figure that the SODL algorithm outperforms [8] in terms of EPE and PVBand by 3%. This indicates the predicted SRAFs by the SODL model match the reference results better than [8]. In other words, the proposed SODL based feature revision can efficiently improve machine learning model generality.

4.2.2 GAN-OPC [55]

GAN-OPC [55] is the earliest work bringing GAN to OPC problems, which aims to train a generative network to provide an initial mask for ILT optimization instantly. The major contributions include customized conditional GAN design and efficient training flow.

Conditional GAN. As shown in Figure 21, the GAN-OPC framework is backboned with conditional GAN structure, which consists of an encode-decoder generator design and a regular CNN-based discriminator. The objective is therefore given by,

$$\begin{aligned}\min_G \max_D \mathbb{E}_{\boldsymbol{Z}_t\sim\mathcal{Z}}[1-\log(D(\boldsymbol{Z}_t,G(\boldsymbol{Z}_t)))+||\boldsymbol{M}^*-G(\boldsymbol{Z}_t)||_n^n] \\ +\mathbb{E}_{\boldsymbol{Z}_t\sim\mathcal{Z}}[\log(D(\boldsymbol{Z}_t,\boldsymbol{M}^*))],\end{aligned} \tag{37}$$

where $\mathcal{Z}$ denotes the distribution of target design layouts, $G(\cdot)$ and $D(\cdot)$ represent the generator and the discriminator in GAN-OPC architecture, $\boldsymbol{Z}_t$ denotes the target design layout and $\boldsymbol{M}^*$ is the golden mask for $\boldsymbol{Z}_t$.

Previous analysis shows that the generator and the discriminator have different objectives, therefore the two sub-networks are trained alternatively, as shown in algorithm 4. In each training iteration, we sample a mini-batch of target images (line 2); Gradients of both the generator and the discriminator are initialized to zero (line 3); A feed forward calculation is performed on each sampled instances (lines 4–

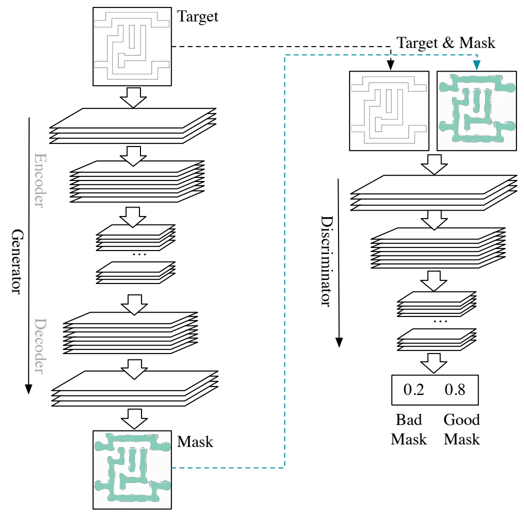


Fig. 21: The proposed GAN-OPC architecture.

5); The groundtruth mask of each sampled target image is obtained from OPC tools (line 6); We calculate the loss of the generator and the discriminator on each instance in the mini-batch (lines 7–8); We obtain the accumulated gradient of losses with respect to neuron parameters (lines 9–10); Finally the generator and the discriminator are updated by descending their mini-batch gradients (lines 11–12). Note that in Algorithm 4 we convert the min-max problem in Equation (37) into two minimization problems such that gradient ascending operations are no longer required to update neuron weights.

Algorithm 4 GAN-OPC Training

```
1:  for number of training iterations do
2:      Sample $m$ target clips $\mathcal{Z} \leftarrow \{\boldsymbol{Z}_{t,1}, \boldsymbol{Z}_{t,2}, \ldots, \boldsymbol{Z}_{t,m}\}$;
3:      $\Delta\boldsymbol{W}_g \leftarrow \mathbf{0}, \Delta\boldsymbol{W}_d \leftarrow \mathbf{0}$;
4:      for each $\boldsymbol{Z}_t \in \mathcal{Z}$ do
5:          $\boldsymbol{M} \leftarrow G(\boldsymbol{Z}_t; \boldsymbol{W}_g)$;
6:          $\boldsymbol{M}^* \leftarrow$ Groundtruth mask of $\boldsymbol{Z}_t$;
7:          $l_g \leftarrow -\log(D(\boldsymbol{Z}_t, \boldsymbol{M})) + \alpha||\boldsymbol{M}^* - \boldsymbol{M}||_2^2$;
8:          $l_d \leftarrow \log(D(\boldsymbol{Z}_t, \boldsymbol{M})) - \log(D(\boldsymbol{Z}_t, \boldsymbol{M}^*))$;
9:          $\Delta\boldsymbol{W}_g \leftarrow \Delta\boldsymbol{W}_g + \frac{\partial l_g}{\partial \boldsymbol{W}_g}; \Delta\boldsymbol{W}_d \leftarrow \Delta\boldsymbol{W}_d + \frac{\partial l_d}{\partial \boldsymbol{W}_g}$;
10:     end for
11:     $\boldsymbol{W}_g \leftarrow \boldsymbol{W}_g - \frac{\lambda}{m}\Delta\boldsymbol{W}_g; \boldsymbol{W}_d \leftarrow \boldsymbol{W}_d - \frac{\lambda}{m}\Delta\boldsymbol{W}_d$;
12: end for
```

ILT-guided Pretraining. Although with OPC-oriented techniques, GAN is able to obtain a fairly good performance and training behavior, it is still a great challenge to train the complicated GAN model with satisfactory convergence. Observing that ILT and neural network training stage share similar gradient descent techniques, we develop an ILT-guided pre-training method to initialize the generator, after which the alternative mini-batch gradient descent is discussed as a training strategy of GAN optimization. The main objective in ILT is minimizing the lithography error through gradient descent.

$$E = ||\boldsymbol{Z}_t - \boldsymbol{Z}||_2^2, \tag{38}$$

where $\boldsymbol{Z}_t$ is the target and $\boldsymbol{Z}$ is the wafer image of a given mask. Because mask and wafer images are regarded as continuously valued matrices in the ILT-based optimization flow, we apply translated sigmoid functions to make the pixel values close to either 0 or 1.

$$\boldsymbol{Z} = \frac{1}{1+\exp[-\alpha \times (\boldsymbol{I} - \boldsymbol{I}_{th})]}, \tag{39}$$

$$\boldsymbol{M}_b = \frac{1}{1+\exp(-\beta \times \boldsymbol{M})}, \tag{40}$$

where $\boldsymbol{I}_{th}$ is the binarization threshold, $\boldsymbol{M}_b$ is the incompletely binarized mask, while α and β control the steepness of relaxed images.

Considering the lithography behavior we discussed in Section 2.1.1, we can also derive the gradient representation as follows,

$$\begin{aligned}\frac{\partial E}{\partial \boldsymbol{M}} =& 2\alpha\beta \times \boldsymbol{M}_b \odot (1-\boldsymbol{M}_b) \odot \\ & (((\boldsymbol{Z}-\boldsymbol{Z}_t) \odot \boldsymbol{Z} \odot (1-\boldsymbol{Z}) \odot (\boldsymbol{M}_b \otimes \boldsymbol{H}^*)) \otimes \boldsymbol{H} + \\ & ((\boldsymbol{Z}-\boldsymbol{Z}_t) \odot \boldsymbol{Z} \odot (1-\boldsymbol{Z}) \odot (\boldsymbol{M}_b \otimes \boldsymbol{H})) \otimes \boldsymbol{H}^*),\end{aligned} \tag{41}$$

where $\boldsymbol{H}^*$ is the conjugate matrix of the original lithography kernel $\boldsymbol{H}$ and $\odot$ is the operator for element-wise product. In traditional ILT flow, the mask can be optimized by iteratively descending the gradient until E is below a threshold.

The objective of the mask optimization problem indicates the generator is the most critical component in GAN. Observing that both ILT and neural network optimization share similar gradient descent procedures, we propose a jointed training algorithm that takes advantage of the ILT engine, as depicted in Figure 22(b). We initialize the generator with lithography-guided pre-training to make it converge well in the GAN optimization flow thereafter. The key step of neural network training is back-propagating the training error from the output layer to the input layer while neural weights are updated as follows,

$$\boldsymbol{W}_g = \boldsymbol{W}_g - \frac{\lambda}{m}\Delta \boldsymbol{W}_g, \tag{42}$$

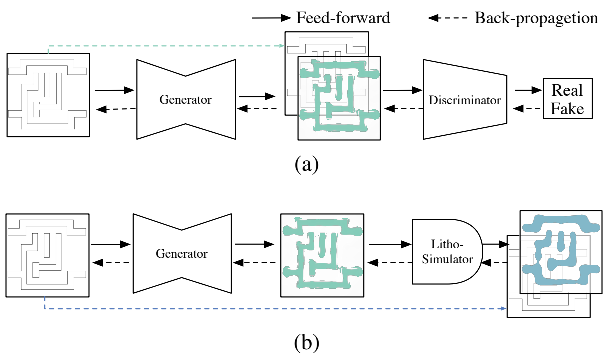


Fig. 22: (a) GAN-OPC training and (b) ILT-guided pre-training.

where $\Delta \boldsymbol{W}_g$ is accumulated gradient of a mini-batch of instances and m is the mini-batch instance count. Because Equation (42) is naturally compatible with ILT, if we create a link between the generator and ILT engine, the wafer image error can be back-propagated directly to the generator as presented in Figure 22.

The generator pre-training phase is detailed in Algorithm 5. In each pre-training iteration, we sample a mini-batch of target layouts (line 2) and initialize the gradients of the generator $\Delta \boldsymbol{W}_g$ to zero (line 3); The mini-batch is fed into the generator to obtain generated masks (lines 5). Each generated mask is loaded into the lithography engine to obtain a wafer image (line 6); The quality of wafer image is estimated by Equation (38) (lines 7); We calculate the gradient of lithography error E with respect to the neural networks parameter $\boldsymbol{W}_g$ through the chain rule, i.e., $\frac{\partial E}{\partial \boldsymbol{M}} \frac{\partial \boldsymbol{M}}{\partial \boldsymbol{W}_g}$ (line 8) ; Finally, $\boldsymbol{W}_g$ is updated following the gradient descent procedure (line 10).

Algorithm 5 ILT-guided Pre-training

```
1: for number of pre-training iterations do
2:     Sample $m$ target clips $\mathcal{Z} \leftarrow \{\boldsymbol{Z}_{t,1}, \boldsymbol{Z}_{t,2}, \dots, \boldsymbol{Z}_{t,m}\}$;
3:     $\Delta \boldsymbol{W}_g \leftarrow 0$;
4:     for each $\boldsymbol{Z}_t \in \mathcal{Z}$ do
5:         $\boldsymbol{M} \leftarrow G(\boldsymbol{Z}_t; \boldsymbol{W}_g)$;
6:         $\boldsymbol{Z} \leftarrow \texttt{LithoSim}(\boldsymbol{M})$
7:         $E \leftarrow ||\boldsymbol{Z} - \boldsymbol{Z}_t||_2^2$;
8:         $\Delta \boldsymbol{W}_g \leftarrow \Delta \boldsymbol{W}_g + \frac{\partial E}{\partial \boldsymbol{M}} \frac{\partial \boldsymbol{M}}{\partial \boldsymbol{W}_g}$;        ▷ Equation (41)
9:     end for
10:    $\boldsymbol{W}_g \leftarrow \boldsymbol{W}_g - \frac{\lambda}{m} \Delta \boldsymbol{W}_g$;        ▷ Equation (42)
11: end for
```

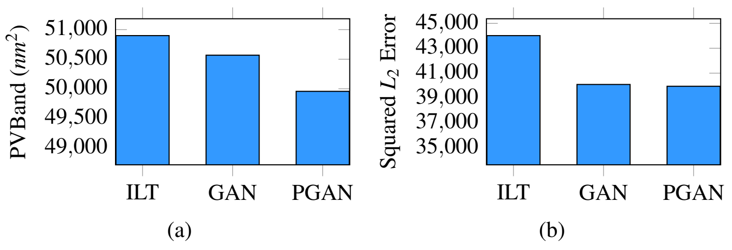


Fig. 23: Result comparison with a vanilla ILT engine.

Experiments. In this experiment, we optimize the ten layout masks in ICCAD 2013 contest benchmark [60] and compare the results with previous work. The quantitative results are illustrated in Figure 23.

Note that all the GAN-OPC and PGAN-OPC results are refined by an ILT engine which generates final masks to obtain wafer images. Column "L_2" is the squared L_2 error between the wafer image and the target image under the nominal condition. Column "PVB" denotes the process variation band under $\pm$ 2% dose error. It is notable that GAN-OPC significantly reduces squared L_2 error of wafer images under the nominal condition by 9% and with the ILT-guided pre-training, squared L_2 error is slightly improved and PVB is further reduced by 1%. Because we only focus on the optimization flow under the nominal condition and no PVB factors are considered, our method only achieves comparable PVB areas.

5 Pattern Generation

VLSI layout patterns provide critical resources in various design for manufacturability (DFM) researches, from (1) early technology node development to (2) back-end design and sign-off flow [61]. The former includes the perfection of design rules, OPC recipes, lithography models, and so on. The latter covers, but is not limited to, layout hotspot detection and correction [40, 53, 31, 39, 62, 7, 43]. However, layout pattern libraries are sometimes not large and diverse enough for DFM research/solutions due to the long logic-to-chip design cycle. Even some test layouts can be synthesized within a short period, they are usually restricted by certain design rules and the generated pattern diversity is limited [63, 21].

5.1 Machine Learning for Layout Pattern Generation

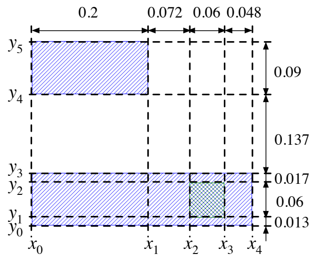
The development of generative machine learning techniques offers the opportunities to generate desired layouts efficiently. Representative works are [64, 65, 66]. DeePattern [64] presents a transforming convolutional auto-encoder (TCAE) architecture for 1D pattern generation. TCAE consists of a recognition unit and a generation unit. The recognition unit is built with stacked convolutional layers that convert layout pattern topologies into latent vectors that allow perturbation for certain transformations. The generation unit is expected to capture layout spatial information (e.g. track and wire direction) well and convert latent vectors back to legal pattern topologies with corresponding deconvolution operations. The work of [65] investigates the possibility of the variational auto-encoder for 2D pattern generation. The techniques in [66] take the VAE in [65] as the backbone and push the technique to advanced technology node with transfer learning and channel-wise attention.

5.2 Case Study

5.2.1 DeePattern [64]

Squish Pattern. Squish pattern is a scan line-based representation that each layout clip is cut into grids aligned at all shape edges, as shown in Figure 24. The squish pattern representation of a given layout clip consists of a topology matrix $\boldsymbol{T}$ and two vectors $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$ that contain geometry information in x and y directions. Each entry of $\boldsymbol{T}$ is either zero or one which indicates shape or space respectively. The geometric information describes the size of each grid. For example, the pattern in Figure 24 can be accordingly expressed as in the right side of Figure 24. Here x_is and y_is are the locations of vertical and horizontal scan lines respectively, and the pattern complexity is accordingly given by $c_x = 4$ and $c_y = 3$. Canonically, (x_0, y_0) is the coordinate of the bottom left corner of the pattern and $x_i < x_{i+1}, y_i < y_{i+1}$. Now the problem becomes generating legal topologies and solving associated $\boldsymbol{\delta}_x$s and $\boldsymbol{\delta}_y$s that are much easier than directly generating DRC-clean patterns. The advantages of squish patterns are two-fold: (1) Squish patterns are storage-efficient and support neural networks and other machine learning models. (2) Squish patterns are naturally compatible with the simplified pattern generation flow that will be discussed in the following sections.

TCAE for Topology Generation. Here the TCAE architecture aims at efficient pattern topology $\boldsymbol{T}$ generation. The TCAE is derived from original transforming auto-encoders (TAEs) [67] which are a group of densely connected auto-encoders and each individual, referred to as a capsule, is targeting certain image-to-image transformations. TAEs cannot be directly applied for pattern generation due to the fact that *transformations are restricted by layout design rules and only very simple pose transformations are supported by original TAEs, which does not satisfy our pattern generation objectives*. Therefore we develop the TCAE architecture for feature learning and pattern reconstruction, as shown in Figure 25. The detection unit in TCAE consists of multiple convolutional layers for hierarchical feature extraction, followed by several densely connected layers as an instantiation of the input pattern in the latent vector space, as in Equation (43).

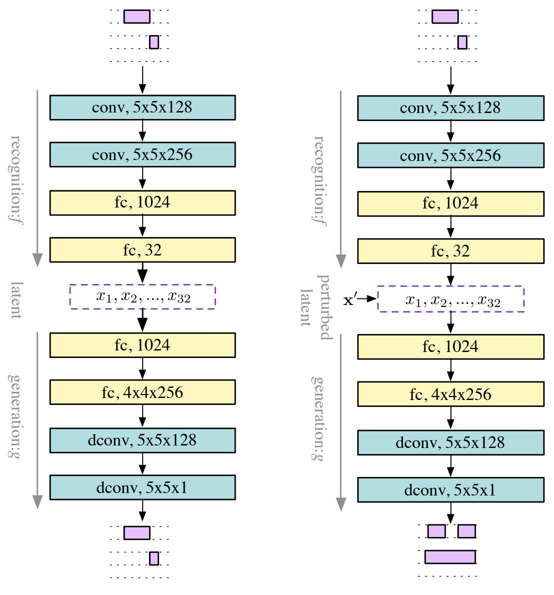


$$\boldsymbol{T} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

$$\boldsymbol{\delta}_x = \begin{bmatrix} x_1 - x_0 & x_2 - x_1 & x_3 - x_2 & x_4 - x_3 \end{bmatrix}$$

$$\boldsymbol{\delta}_y = \begin{bmatrix} y_1 - y_0 & y_2 - y_1 & y_3 - y_2 \end{bmatrix}$$

Fig. 24: Squish pattern generation.



(a) Training (b) Testing

Fig. 25: Architecture of transforming convolutional auto-encoder in (a) training phase and (b) testing phase.

$$\boldsymbol{l} = f(\boldsymbol{T}; \boldsymbol{W}_f), \tag{43}$$

where $\boldsymbol{l}$ is the latent vector, $\boldsymbol{T}$ represents the input topology and $\boldsymbol{W}_f$ contains all the trainable parameters associated with the recognition unit. The latent vector works similarly as a group of capsule units with each node being a low-level feature representation. We will show each latent vector node contributes to pattern shape globally or locally in the experiment section.

The generation unit contains deconvolutional layers [68] that cast the pattern object from the latent vector space back to the original pattern space, as in Equation (44).

$$\boldsymbol{T}' = g(\boldsymbol{l} + \Delta\boldsymbol{l}; \boldsymbol{W}_g), \tag{44}$$

where $\Delta\boldsymbol{l}$ is the perturbation applied on the latent vector that allows inputs to conduct transformations. During training, we force the TCAE to learn an identity mapping with the following objectives.

$$\min_{\boldsymbol{W}_f, \boldsymbol{W}_g} ||\boldsymbol{T} - \boldsymbol{T}'||_2, \text{ s.t. } \Delta\boldsymbol{l} = \boldsymbol{0}. \tag{45}$$

Once the TCAE is trained, we can apply the flow in Figure 25b to generate pattern topologies from perturbed latent vector space of existing layout patterns. During the inference phase, we feed a group of squish topologies into the trained recognition unit that extracts latent vector instantiations of existing topologies. Perturbation on the latent vector space is expected to expand the existing pattern library with legal topologies.

The method introduces random perturbations in the latent vectors. We introduce the concept of feature sensitivity $\boldsymbol{s}$ that statistically defines how easily a legal topology can be transformed to illegal when manipulating the latent vector node with everything else unchanged.

Definition 1 (Feature Sensitivity) *Let* $\boldsymbol{l} = \begin{bmatrix} l_1 & l_2 & \dots & l_n \end{bmatrix}^\top$ *be the output of the layer associated with the latent vector space. The sensitivity* s_i *of a latent vector node* l_i *is defined as the probability of reconstructed pattern being invalid when a perturbation* $\Delta l_i \in [-t, t]$ *is added up on* l_i *with everything else unchanged.*

It can be seen from Definition 1 that a larger s_i indicates the corresponding latent vector node l_i is more likely to create invalid topologies if a large perturbation is applied. We, therefore, avoid manipulating such nodes when sampling random perturbation vectors from a Gaussian distribution. The s_is are estimated following Algorithm 6, which requires a set of legal topologies and trained TCAE. The sensitivity of each latent vector node is estimated individually (lines 1–2). We first obtain the latent vectors of all topologies in $\mathcal{T}$ and feed them into the reconstruction unit along with certain perturbations on one latent vector node (lines 3–5). Reconstructed patterns are appended in the corresponding set $\mathcal{R}_i$ (line 6). The sensitivity of the latent vector node i is given by the fraction of invalid topologies in $\mathcal{R}_i$ (line 8).

After we get the estimated sensitivity of all latent vector nodes, we are able to sample perturbation vectors whose elements are sampled independently from $\mathcal{N}(0, \frac{1}{s_i})$. These perturbation vectors will be added up to the latent vectors of existing pattern topologies to formulate perturbed latent vectors which will be fed into the generation unit to construct new topologies. That is, illegal topologies can be filtered out by checking whether shapes appear at any two adjacent tracks.

Algorithm 6 Estimating feature sensitivity. $\mathcal{T} = \{\boldsymbol{T}_1, \boldsymbol{T}_2, ..., \boldsymbol{T}_N\}$ is a set of valid pattern topologies, f and g are trained recognition unit and generation unit respectively, t determines the perturbation range, and $\boldsymbol{s}$ is the estimated feature sensitivity.

```
Require: $\mathcal{T}, f, g, t$.
Ensure: $\boldsymbol{s}$.
 1: $\mathcal{R}_i \leftarrow \emptyset, \forall i = 1, 2, ..., N$;
 2: for $i = 1, 2, ..., n$ do
 3:     for $\lambda = -t : t$ do
 4:         $\Delta \boldsymbol{l} \leftarrow \boldsymbol{0}, \Delta l_i \leftarrow \lambda$;
 5:         $\mathcal{T}_i \leftarrow g(f(\mathcal{T}) + \Delta \boldsymbol{l})$;
 6:         $\mathcal{R}_i \leftarrow \mathcal{R}_i + \mathcal{T}_i$;
 7:     end for
 8:     $s_i \leftarrow$ fraction of invalid topologies in $\mathcal{R}_i$;
 9: end for
10: return $\boldsymbol{s}$.
```

Table 8: Statistics of generated patterns.

Method	Pattern #	Pattern Diversity (H)
Existing Design	-	3.101
Industry Tool	55408	1.642
DCGAN	1	0
TCAE	**286898**	**3.337**

Experiments. In the experiment, we make use of the flow above to augment the pattern space. Pattern library statistics are listed in Table 8. Column “Method” denotes the approach used to generate layout patterns, column “Pattern #” denotes the number of DRC clean patterns that are different from others, and column “Pattern Diversity” corresponds to the Shannon Entropy of each pattern library in terms of pattern complexity. Row “TCAE” corresponds to the details of 1M patterns generated by perturbing the features of 1000 patterns in existing design with Gaussian noise. “Industry Tool” shows the cataloged results of a test layout generated from a state-of-the-art industry layout generator. The test layout has similar total chip area (10000 μm^2) as “TCAE” (14807 μm^2). We also implement a DCGAN [24] that has similar number of trainable parameters as the TCAE designed in this framework. 1M patterns are generated by feeding random latent vectors in the trained generator networks. “Existing Design” lists the statistics of a pattern library extracted from an industry layout.

6 Conclusion

In this chapter, we survey recent progress and challenges of machine learning solutions on a variety of VLSI DFM problems, ranging from lithography modeling, lithography hotspot detection, mask optimization to layout generation. These are all critical phases in VLSI design and sign-off flows and seriously affect chip design cycles. We hope the investigation of machine learning techniques in these problems would offer alternate solutions to traditional DFM flows and hence enable faster design closure. Follow-up researches are necessary to prototype these techniques. Future directions will include, but are not limited to, problem dedicated learning model and algorithm design to meet industrial requirements and constraints, efficient machine learning framework plugins to assist traditional design flows, massive data generation to allow better training convergence and model generality, etc.

References

1. Chris Mack. *Fundamental Principles of Optical Lithography: The Science of Microfabrication*. John Wiley & Sons, 2008.
2. John E. Greivenkamp. *Field Guide to Geometrical Optics*. SPIE Press Bellingham, WA, 2004.
3. Shayak Banerjee, Kanak B. Agarwal, and Michael Orshansky. Simultaneous OPC and decomposition for double exposure lithography. In *Proceedings of SPIE*, volume 7973, 2011.
4. Xiaohai Li, Gerry Luk-Pat, Chris Cork, Levi Barnes, and Kevin Lucas. Double-patterning-friendly OPC. In *Proceedings of SPIE*, volume 7274, 2009.
5. Mohit Gupta, Kwangok Jeong, and Andrew B Kahng. Timing yield-aware color reassignment and detailed placement perturbation for double patterning lithography. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 607–614, 2009.
6. Jian Kuang, Wing-Kai Chow, and Evangeline F. Y. Young. Triple patterning lithography aware optimization for standard cell based design. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 108–115, 2014.
7. Hao Geng, Haoyu Yang, Yuzhe Ma, Joydeep Mitra, and Bei Yu. SRAF insertion via supervised dictionary learning. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 406–411, 2019.
8. Xiaoqing Xu, Tetsuaki Matsunawa, Shigeki Nojima, Chikaaki Kodama, Toshiya Kotani, and David Z. Pan. A machine learning based framework for sub-resolution assist feature generation. In *ACM International Symposium on Physical Design (ISPD)*, pages 161–168, 2016.
9. Timothy Lin, Frederic Robert, Amandine Borjon, Gordon Russell, Catherine Martinelli, Andrew Moore, and Yves Rody. Sraf placement and sizing using inverse lithography technology. In *Optical Microlithography XX*, volume 6520, page 65202A. International Society for Optics and Photonics, 2007.
10. Nicolas Bailey Cobb. *Fast optical and process proximity correction algorithms for integrated circuit manufacturing*. PhD thesis, University of California at Berkeley, 1998.
11. Jhih-Rong Gao, Xiaoqing Xu, Bei Yu, and David Z. Pan. MOSAIC: Mask optimizing solution with process window aware inverse correction. In *ACM/IEEE Design Automation Conference (DAC)*, pages 52:1–52:6, 2014.
12. Haoyu Yang, Shuhe Li, Yuzhe Ma, Bei Yu, and Evangeline F.Y. Young. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets. In *ACM/IEEE Design Automation Conference (DAC)*, pages 131:1–131:6, 2018.
13. Yu-Hsuan Su, Yu-Chen Huang, Liang-Chun Tsai, Yao-Wen Chang, and Shayak Banerjee. Fast lithographic mask optimization considering process variation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 35(8):1345–1357, 2016.
14. Guojin Chen, Ziyang Yu, Hongduo Liu, Yuzhe Ma, and Bei Yu. DevelSet: Deep neural level set for instant mask optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

15. Xu Ma and Gonzalo R Arce. *Computational lithography*, volume 77. John Wiley & Sons, 2011.
16. Yuki Watanabe, Taiki Kimura, Tetsuaki Matsunawa, and Shigeki Nojima. Accurate lithography simulation model based on convolutional neural networks. In *Proceedings of SPIE*, volume 10454, page 104540I, 2017.
17. Wei Ye, Mohamed Baker Alawieh, Yibo Lin, and David Z Pan. LithoGAN: End-to-end lithography modeling with generative adversarial networks. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.
18. Guojin Chen, Wanli Chen, Qi Sun, Yuzhe Ma, Haoyu Yang, and Bei Yu. DAMO: Deep agile mask optimization for full chip scale. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.
19. Wei Ye, Mohamed Baker Alawieh, Yuki Watanabe, Shigeki Nojima, Yibo Lin, and David Z Pan. TEMPO: Fast mask topography effect modeling with deep learning. In *ACM International Symposium on Physical Design (ISPD)*, pages 127–134, 2020.
20. Seongbo Shim, Suhyeong Choi, and Youngsoo Shin. Machine learning-based 3d resist model. In *Proceedings of SPIE*, volume 10147, page 101471D. International Society for Optics and Photonics, 2017.
21. Yibo Lin, Meng Li, Yuki Watanabe, Taiki Kimura, Tetsuaki Matsunawa, Shigeki Nojima, and David Z Pan. Data efficient lithography modeling with transfer learning and active data selection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2019.
22. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
23. Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 3–11. Springer, 2018.
24. Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2016.
25. Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8798–8807, 2018.
26. Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.
27. Andres J. Torres. ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 349–350, 2012.
28. Tetsuaki Matsunawa, Jhih-Rong Gao, Bei Yu, and David Z. Pan. A new lithography hotspot detection framework based on AdaBoost classifier and simplified feature extraction. In *Proceedings of SPIE*, volume 9427, 2015.
29. Hang Zhang, Bei Yu, and Evangeline F. Y. Young. Enabling online learning in lithography hotspot detection with information-theoretic feature optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 47:1–47:8, 2016.
30. Yen-Ting Yu, Geng-He Lin, Iris Hui-Ru Jiang, and Charles Chiang. Machine-learning-based hotspot detection using topological classification and critical feature extraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 34(3):460–470, 2015.
31. Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, Bei Yu, and Evangeline F. Y. Young. Layout hotspot detection with feature tensor generation and deep biased learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 38(6):1175–1187, 2019.
32. Yiyang Jiang, Fan Yang, Hengliang Zhu, Bei Yu, Dian Zhou, and Xuan Zeng. Efficient layout hotspot detection via binarized residual neural network. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

33. Binwu Zhu, Ran Chen, Xinyun Zhang, Fan Yang, Xuan Zeng, Bei Yu, and Martin DF Wong. Hotspot detection via multi-task learning and transformer encoder. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2021.
34. Hao Geng, Haoyu Yang, Lu Zhang, Jin Miao, Fan Yang, Xuan Zeng, and Bei Yu. Hotspot detection via attention-based deep layout metric learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021.
35. Ran Chen, Wei Zhong, Haoyu Yang, Hao Geng, Xuan Zeng, and Bei Yu. Faster region-based hotspot detection. In *ACM/IEEE Design Automation Conference (DAC)*, 2019.
36. Ying Chen, Yibo Lin, Tianyang Gai, Yajuan Su, Yayi Wei, and David Z Pan. Semi-supervised hotspot detection with self-paced multi-task learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2019.
37. Hang Zhang, Fengyuan Zhu, Haocheng Li, Evangeline F. Y. Young, and Bei Yu. Bilinear lithography hotspot detection. In *ACM International Symposium on Physical Design (ISPD)*, pages 7–14, 2017.
38. Fan Yang, Subarna Sinha, Charles C. Chiang, Xuan Zeng, and Dian Zhou. Improved tangent space based distance metric for lithographic hotspot classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 36(9):1545–1556, 2017.
39. Moojoon Shin and Jee-Hyong Lee. Accurate lithography hotspot detection using deep convolutional neural networks. *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, 15(4):043507, 2016.
40. Haoyu Yang, Luyang Luo, Jing Su, Chenxi Lin, and Bei Yu. Imbalance aware lithography hotspot detection: a deep learning approach. *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, 16(3):033504, 2017.
41. Gregory K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics (TCE)*, 38(1):xviii–xxxiv, 1992.
42. Haoyu Yang, Luyang Luo, Jing Su, Chenxi Lin, and Bei Yu. Imbalance aware lithography hotspot detection: A deep learning approach. In *SPIE Advanced Lithography*, volume 10148, 2017.
43. Haoyu Yang, Yajun Lin, Bei Yu, and Evangeline F. Y. Young. Lithography hotspot detection: From shallow to deep learning. In *IEEE International System-on-Chip Conference (SOCC)*, pages 233–238, 2017.
44. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.
45. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.
46. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Conference on Neural Information Processing Systems (NIPS)*, pages 91–99, 2015.
47. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, pages 21–37, 2016.
48. Zhiyang Song, Xu Ma, Jie Gao, Jie Wang, Yanqiu Li, and Gonzalo R Arce. Inverse lithography source optimization via compressive sensing. *Optics express*, 22(12):14180–14198, 2014.
49. Andreas Erdmann, Tim Fuehner, Thomas Schnattinger, and Bernd Tollkuehn. Toward automatic mask and source optimization for optical lithography. In *Optical Microlithography XVII*, volume 5377, pages 646–657. International Society for Optics and Photonics, 2004.
50. Peng Yu, Sean X. Shi, and David Z. Pan. Process variation aware OPC with variational lithography modeling. In *ACM/IEEE Design Automation Conference (DAC)*, pages 785–790, 2006.
51. Mohamed Baker Alawieh, Yibo Lin, Zaiwei Zhang, Meng Li, Qixing Huang, and David Z Pan. GAN-SRAF: Sub-resolution assist feature generation using conditional generative adversarial networks. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.
52. Shiyan Hu and Jiang Hu. Pattern sensitive placement for manufacturability. In *ACM International Symposium on Physical Design (ISPD)*, pages 27–34, 2007.

53. Bentian Jiang, Hang Zhang, Jinglei Yang, and Evangeline FY Young. A fast machine learning-based mask printability predictor for OPC acceleration. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 412–419, 2019.
54. Tetsuaki Matsunawa, Bei Yu, and David Z. Pan. Optical proximity correction with hierarchical bayes model. *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, 15(2):021009, 2016.
55. Haoyu Yang, Shuhe Li, Zihao Deng, Yuzhe Ma, Bei Yu, and Evangeline FY Young. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2019.
56. Tetsuaki Matsunawa, Bei Yu, and David Z. Pan. Optical proximity correction with hierarchical bayes model. In *Proceedings of SPIE*, volume 9426, 2015.
57. Mehrdad J. Gangeh, Ahmed K. Farahat, Ali Ghodsi, and Mohamed S. Kamel. Supervised dictionary learning and sparse representation-a review. *arXiv preprint arXiv:1502.05928*, 2015.
58. Robert Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society: Series B*, 58:267–288, 1996.
59. Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.
60. Shayak Banerjee, Zhuo Li, and Sani R. Nassif. ICCAD-2013 CAD contest in mask optimization and benchmark suite. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 271–274, 2013.
61. Cyrus Tabery, Yi Zou, Vincent Arnoux, Praveen Raghavan, Ryoung-han Kim, Michel Côté, Luca Mattii, Ya-Chieh Lai, and Philippe Hurat. In-design and signoff lithography physical analysis for 7/5nm. In *SPIE Advanced Lithography*, volume 10147, 2017.
62. Haoyu Yang, Piyush Pathak, Frank Gennari, Ya-Chieh Lai, and Bei Yu. Detecting multi-layer layout hotspots with adaptive squish patterns. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 299–304, 2019.
63. Haoyu Yang, Shuhe Li, Cyrus Tabery, Bingqing Lin, and Bei Yu. Bridging the gap between layout pattern sampling and hotspot detection via batch active sampling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2020.
64. Haoyu Yang, Piyush Pathak, Frank Gennari, Ya-Chieh Lai, and Bei Yu. Deepattern: Layout pattern generation with transforming convolutional auto-encoder. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.
65. Xiaopeng Zhang, James Shiely, and Evangeline FY Young. Layout pattern generation and legalization with generative learning models. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.
66. Xiaopeng Zhang, Haoyu Yang, and Evangeline FY Young. Attentional transfer is all you need: Technology-aware layout pattern generation. In *ACM/IEEE Design Automation Conference (DAC)*, pages 169–174. IEEE, 2021.
67. Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International Conference on Artificial Neural Networks (ICANN)*, pages 44–51, 2011.
68. Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2018.