Intelligent VLSI Design for Manufacturability

YANG, Haoyu

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
July 2020

Thesis Assessment Committee

Abstract

of thesis entitled:

Intelligent VLSI Design for Manufacturability

Submitted by YANG, Haoyu

for the degree of Doctor of Philosophy

at The Chinese University of Hong Kong in July 2020

With the technology node continuously shrinking down, modern VLSI designs are encountering great challenges. This thesis focuses on emerging AI solutions of various VLSI design and sign-off problems, covering layout printability estimation, layout sampling and layout generation.

Process variations caused by the limitations of the manufacturing system are one of the key challenges on chip manufacturing yields. Even with various resolution enhancement techniques, manufacturing defects are still likely to happen for some sensitive layout patterns, referred as hotspots. State-of-the-art solution to find hotspots is lithography simulation or chip inspection, which is expensive and time consuming. The extreme ultraviolet (EUV)-specific lithography hotspot detection, for example, usually requires manual examination on taped-out chips whose costs grow with inspection area. Therefore, we study and develop deep learning-based solutions for efficient hotspot detection, targeting at imbalanced dataset, learning algorithms, feature representations and model reliability.

Recent works show that learning-based hotspot detection solutions rely highly on the quality of reference layout libraries. We optimize pattern library by embedding the active learning engine into deep neural networks thus data sampling and incremental model training can be conducted alternatively. Guaranteed by the on-line property of stochastic gradient descent, we only need to finetune the neuron weights according to new labeled instances instead of training model from scratch in each iteration. A layout specific distance metric is to be designed to guarantee an efficient sampling procedure during training runtime.

VLSI layout patterns provide critical resources in various design for manufacturability (DFM) researches from mask optimization (mask pattern generation) to early technology node development (design pattern generation). We investigate the possibility of generative machine learning models being solution candidates for layout generation purposes. We develop the GAN-OPC family for intelligent mask optimization that promises efficient and high-quality mask generation. For the design pattern generation, we propose a transforming convolutional auto-encoder (TCAE) architecture that is designed to capture layout design rules, which contributes to DRC-clean test pattern generation.

With supporting experiments, we demonstrate the efficiency and effectiveness of intelligent algorithms and architectures being alternate solutions for DFM problems, which has the potential to push the industry forward under the high demand of manufacturing quantity and quality.

摘要

智能超大規模積體電路可製造性設計

隨著集成電路工藝節點的不斷進步，當代芯片設計正面著挑戰。本論文討論基於人工智能技術的集成電路後端設計解決方案。主要包括電路版圖可製造性（DFM）驗證，電路可製造性庫採樣與優化，以及集成電路版圖的生成。芯片製造系統的局限性會造成產出芯片的性能和質量的不確定性，這對集成電路生產線的產能造成了極大的影響。

儘管有諸多高階技術應用在芯片製造中，在某些類型的電路版圖中仍會出現不可預料的缺陷。我們通常把這類缺陷又稱為熱點。熱點的檢測通常依賴模擬和檢查。在極紫外光技術節點下的熱點檢測依賴流片結果的人工檢查，十分耗時昂貴。因此，在本論文中我們研究基於深度學習的高效解決方案，特別地，考慮數據的不平衡，特徵表示以及模型可靠性。考慮到基於深度學習的熱點檢測器特別依賴於參考版圖庫以及對版圖添加標籤所需要的昂貴代價。我們將主動學習引擎嵌入到傳統的熱點檢測流程中，使得版圖庫的優化採樣和模型的訓練能夠交替進行。這樣，熱點檢測框架能夠以最少的訓練數據集實現較高的熱點檢測精度。集成電路的版圖為可製造性的研究提供了重要的資源，從掩膜版的優化到早起工藝節點的研究。在這裡我們研究生成型機器學習模型在版圖生成時的應用。針對掩膜版優化問題，本文提出 GAN-OPC 系列框架，提供高效高質量的生成掩膜版。針對設計版圖的生成，我們提出了轉換卷積自編碼器（TCAE）架構。TCAE 通過抓取訓練集版圖中的設計規則，生成符合要求的測試圖形。在

支持有關理論的實驗中，我們證明了智慧型算法和架構作為 DFM 解決方案的有效性。

Acknowledgement

Firstly, I would like to have greatest thanks to my advisor, Professor Bei Yu, for his patience, guidance and support during the years. He inspires me to find and face cutting-edge research problems and develop emerging technologies to tackle these challenges. This thesis cannot be possible without his warm advice and continuous encouragements.

I also want to express my grateful thanks to rest of the committee members. In particular, I would like to thank Professor Irwin King for his technical comments and suggestions on mathematical formulations and terminologies. I would like to thank Professor Evangeline Young for her suggestions on the thesis structure and technique details on hotspot detection and mask optimization. I would like to thank Professor Yibo Lin for his thoughtful comments and suggestions on design pattern generation and thesis development. I would like to thank Professor David Pan for his detailed comments, helpful suggestions and proofread on the thesis.

Besides, I really appreciate Mr. Cyrus Tabery, Dr. Jing Su, Dr. Chenxi Lin and Dr. Yi Zou who taught me lithography during my internship at ASML. I would also like to thank Dr. Piyush Pathak, Dr. Frank Gennari and Mr. Ya-Chieh Lai for their advice and support during my internship at Cadence Design Systems.

My sincere thanks go to my colleagues in my group: Yuzhe Ma, Hao Geng, Tinghuan Chen, Ran Chen, Qi Sun, Lu Zhang, Wanli Chen, Leon He and Wei Li for our inspiring discussions and productive collaborations. I am also lucky to have worked with fellas at SHB913: Peishan Tu, Jordan Pui, Gengjie Chen, Haocheng Li, Jingsong Chen, Bentian Jiang, Xiaopeng Zhang. Except for academic discussions and collaborations, I will definitely miss all the joys and fun I had with them.

Last but not least, I would like to thank my parents Yajuan Zhang and Hongbin Yang for their endless love and being supportive from the very beginning, which have always been the source of power for me to achieve everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Moore's Law [75] 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. Before digging into these challenges and potential solutions, we will first introduce some basis of lithography process.

A classical lithography system consists of mainly five stages that include *source*, *condenser lens*, *mask*, *objective lens* and *wafer*, as shown in Figure 1.1. 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. The objective lens tries to collect diffraction information as much as possible for better transferred image quality. Because of the limited size of objective lens, higher order diffraction patterns will be discarded when forming the image on the wafer that results in a lower pattern fidelity [68]. Typically, to ensure the mask image can be transferred onto the

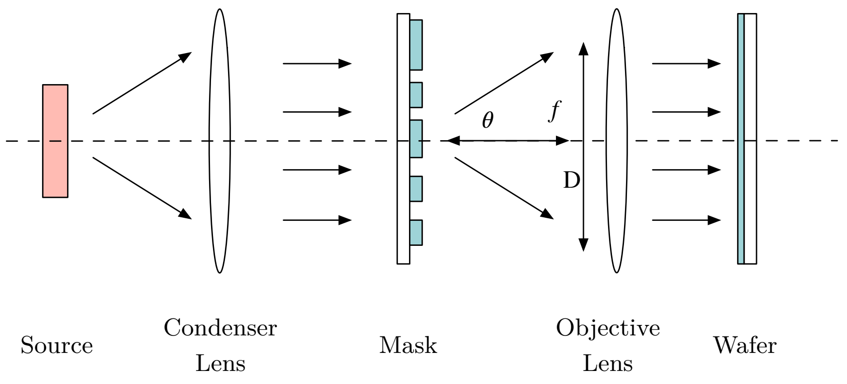


Figure 1.1: An example of a classical lithography system.

wafer as accurate as possible, at least the zero and ± 1st diffraction order should be captured by the objective lens. Accordingly, the smallest design pitch can be defined as Equation (1.1),

$$\frac{1}{p} \propto \frac{NA}{\lambda}, \tag{1.1}$$

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

$$NA = n \sin \theta_{\max} = \frac{D}{2f}, \tag{1.2}$$

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 [39].

Although researches have pushed higher NA design of lithography systems, diffraction information loss still causes mismatch between printed patterns on a wafer and the patterns in the design, which is well known as *lithography proximity effect*. A

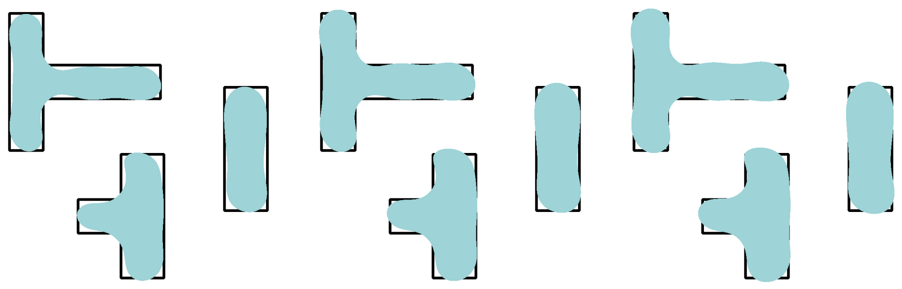


Figure 1.2: Lithography contour versus design target under process variations: (a) low dose at focus produces small contours; (b) normal dose at focus produces regular contours; (c) high dose at defocus produces larger contours.

mainstream solution is called resolution enhancement technique (RET) that includes multiple patterning lithography (MPL) [7, 60, 42, 56], sub-resolution assist feature (SRAF) [32, 109, 61] insertion and optical proximity correction (OPC) [18]. MPL attempts split designs into multiple masks to achieve higher resolution, while SRAF and OPC aim to compensate 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 1.2 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.

1.1 VLSI Manufacturing Challenges

Hotspot Detection

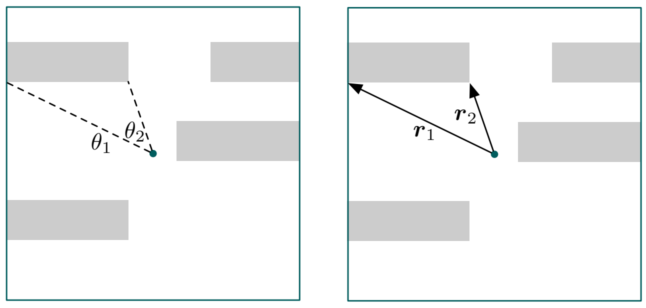
The fact of process variations have implied that even a design is fed into comprehensive and high-quality RET engines, it is inevitable to have unexpected manufacturing defects at certain design location(s), which causes fetal yield loss. In literature, we call these process week locations *hotspots* that can be the results of lithography [16], etch [88], planarization [82] or other manufacturing process variations. Efficient localization of hotspots in a design prior manufacturing is important to meet design closure requirements and avoid unnecessary overheads. However, hotspot detectors have to work at the context of known hotspot pattern libraries that are limited in both quantity and generality, which become major factors prevent efficient hotspot detector design.

Mask Optimization

RETs try to minimize the error when transferring a design onto silicon wafers. Mainstream solutions varies from lithography source configuration [95, 26], mask pattern optimization [125, 30, 109, 2], multiple patterning lithography [50, 42] and etc. Among the above, mask optimization is one of the most critic and inevitable stage in sign-off flows. It tweaks with features or contexts of design layout patterns to circumvent side effects from 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.

Early Technology Node Development

Early technology node development is another important topic when a new design technology (typically reflected as feature size shrinking down) comes right before flow deployment. At this phase, there are only limited designs available with non-optimally configured mask optimization recipes and design rule decks. Various new designs are hence necessary to make mature manufacturing technology nodes. However, such designs are usually in short supply due to the long design cycle at design houses.



(a) Angle (b) Radius

Figure 1.3: Improved tangent space with (a) angle and (b) radius.

1.2 Literature Review

1.2.1 Legacy Mask and Layout Optimization

Hotspot Detection

Hotspot Detection, as a special case of pattern recognition, is deemed to be efficiently solved with pattern/fuzzy matching technologies [101, 99, 66]. State-of-the-art solutions are sharing similar idea that patterns that are close to the data in an existing pattern library will be classified as hotspots. The procedure usually requires an compact layout representation that will be fed into some cluster engines.

One representative solution is using improved tangent space-based distance metric and hierarchical DBSCAN [110], where each polygon shape in a given layout clip is converted into a vector-based representation in terms of the relative angle (Figure 1.3(a)) and radius (Figure 1.3(b)) of all vertices and clip center. The distance of two clips will be evaluated by matching all polygons in the tangent space representation, which will be used to further formulate DBSCAN cluster. Other solutions include pure pattern matching in layout geometry space and its derivatives [104, 34].

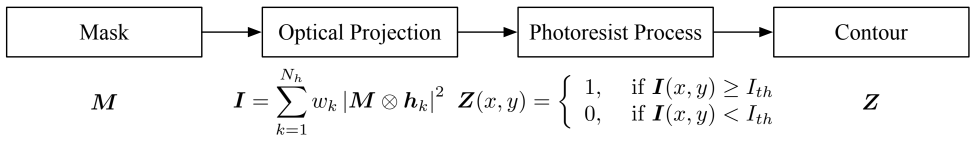

Mask
Optical Projection
Photoresist Process
Contour
M
$I = \sum_{k=1}^{N_h} w_k |M \otimes h_k|^2$
$Z(x,y) = \begin{cases} 1, & \text{if } I(x,y) \geq I_{th} \\ 0, & \text{if } I(x,y) < I_{th} \end{cases}$
Z

Figure 1.4: Continuous forward lithography modeling.

Mask Optimization

Model-based OPC and inverse lithography techniques (ILT) are two major solutions for layout optimization problems. The former conducts coarse-grained shape-level refinements on polygon edges that are divided into segments empirically. Optimization procedure requires interactively call of lithography engine to make compensation of contour errors with respect to design targets. An state-of-the-art model-based OPC solution is presented in [57], which adopts adaptive fragmentation method to make finer control of polygon corners. ILT, on the other hand, formulates the mask optimization problem with a non-linear objective that tries to minimize the error between the lithography contour and its corresponding target. MOSAIC [30] is a representative solution by building up a forward lithography process model (see Figure 1.4). The mask is hence updated by descending the gradient of contour error with respect to each mask pixels. To ensure the mask-to-contour mapping to be differentiable, sigmoid smoothness is applied on the binary mask and photoresist process.

1.2.2 Mask and Layout Learning

Recent advances of machine learning, especially deep learning, have brought opportunities to develop learning-based solutions for DFM challenges, which are associated with two major machine learning categorizes that are called *discriminative learning* and *generative learning*.

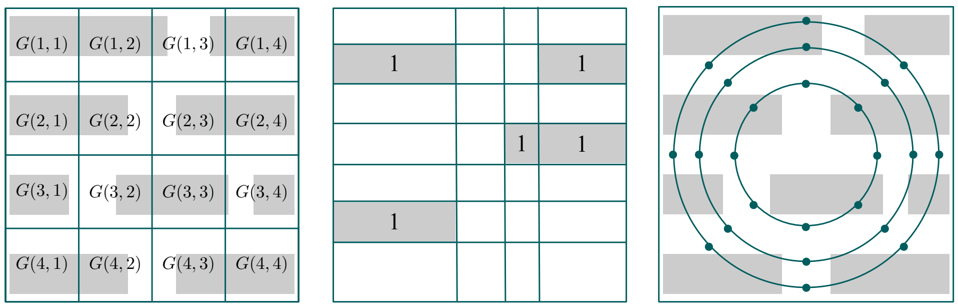
Discriminative learning targets at classification tasks that are either image level or pixel level. Usually, a Discriminative learning model can be abstracted as a conditional probability $P(Y = y|X = \boldsymbol{x}; \boldsymbol{w})$, where X is the random variable drawn from the data space, Y is the random variable drawn from the label space and $\boldsymbol{w}$ corresponds to model parameters. Researches on discriminative learning mainly focus on machine learning architecture and parameter training. Representative architectures range from simple logistic regression, support vector machines (SVMs) to complicated VGG [94], ResNet [44], MobileNet [49] and other convolutional neural networks, which are designed for certain application scenarios. Parameter training algorithms are developed for good convergence and model generality [35, 17, 54] that hence leads to high prediction accuracy.

Generative learning works in the opposite way to discriminative learning, where we are asked to obtain observations that belong to one or some categories, i.e. $P(X = \boldsymbol{x}|Y = y; \boldsymbol{w})$. Representative models are variational auto-encoder (VAE) [23] and generative adversarial networks (GAN) [38], which are specifically designed for image generation. Although neural network nature of these generative models make legacy architecture and training algorithms applicable [3], several derivatives are also explored for better performance [4, 64, 132].

The strict requirements and properties of DFM flows make these learning models, although powerful in classic computer vision tasks, not directly applicable, and hence motives continuous research of DFM-aware machine learning. These researches cover layout feature representations and learning models.

Layout Feature Engineering

Density-based Feature. Usually mask layouts with high pattern density show a higher risk of suffering defects, therefore it is reasonable to measure the mask printability via its local pattern density [70, 104]. As shown in Fig. 1.5(a), each layout clip is first divided into square grids and each grid $G(i, j)$ corresponds to a value $x_{i,j}$ reflects the


$G(1,1)$ $G(1,2)$ $G(1,3)$ $G(1,4)$
$G(2,1)$ $G(2,2)$ $G(2,3)$ $G(2,4)$
$G(3,1)$ $G(3,2)$ $G(3,3)$ $G(3,4)$
$G(4,1)$ $G(4,2)$ $G(4,3)$ $G(4,4)$
1
1
1
1
1

(a) Density (b) Topology (c) CCS

Figure 1.5: Recent research on layout feature engineering.

density information that is calculated through Equation (1.3).

$$x_{i,j} = \frac{A_M(i,j)}{A_G(i,j)}, \tag{1.3}$$

where $A_M(i,j)$ denotes the mask pattern area within $G(i,j)$ and $A_G(i,j)$ is the area of $G(i,j)$. Finally, the density information is flattened into a vector $\boldsymbol{x} = \{x_{11}, x_{12}, ..., x_{44}\}$ that is applicable for varies machine learning models.

Layout Topology. The topological representation of a layout clip contains the geometry relationships of the patterns (rectangles) within it [11, 128]. As shown in Fig. 1.5(b), all edges are extended to the border of the clip and cut the clip into grids, which are filled with either geometry or space. If we label each grid with 1 (for geometry) and 0 (for space), the original layout clip can then be converted into a binary matrix representation. The topology matrix itself has already been a good representation for pattern matching [11] and constructing the space of design layout configurations and sub-configurations [19]. Besides, by incorporating with design rules, the topological representation can be a effective layout feature for accurate hotspot detection [128]. Layout topology representation is also developed into squish pattern as an effective and efficient representation for layout matching [34].

Concentric Circle Area Sampling. It has been believed that layout hotspots are caused by optical proximity (diffraction) effect in the lithography process. Whether a pattern is problematic or not is determined by all the patterns within a square [101] or circular ambit [97]. In the $28nm$ technology node, the ambit area is approximately $1\mu m^2$ which corresponds to an $\mathbb{R}^{1000000}$ space with $1nm$ precision that makes it difficult to do quantitative analysis for OPC and hotspot detection. Concentric circle area sampling (CCAS) [72], developed from concentric square sampling (CSS) [40], has become an efficient feature extraction method because of being coherent with the fact that diffracted light propagates in a circular concentric scheme. As illustrated in Fig. 1.5(c), all the circles are concentrated on the center of the clip (or ambit). Eight points are evenly sampled from each circle and each point is labeled 0 or 1 based on whether it is located on the geometry or space. Although CCAS is originally designed for machine learning based OPC [72], it is also effective in hotspot detection [130].

Layout Learning Model

Boosting. Boosting is a family of ensemble machine learning methods which are able to build a strong classifier from a set of weak classifiers. [70] and [106] are two representative works for hotspot detection, where decision tree is chosen as the weak classifier. Decision tree works in the form of a flow chart where each non-leaf node splits the data by examining one attribute and leaf nodes predict the label. Information gain (IG) is utilized to determine the feature of each node, as shown in Equation (1.4).

$$\mathrm{IG}(P, f) = H(P) - H(P|f), \tag{1.4}$$

where $H(P)$ is the entropy related to feature f of the parent node and $H(P|f)$ is the weighted entropy of all children. Final prediction are made by a weighted combination of the results from all decision trees (weak classifiers). Very recently, [130] also investigated to use Naive Bayes as the weak learner to further improve the hotspot detection accuracy. The classifier is developed together with CCS features as follows

$$y_i^* = \arg\max_y p(y) \prod_{j=0}^{\lfloor \frac{n}{p} \rfloor - 1} p\left(x_i^{j \cdot p}, \ldots, x_i^{(j+1)\cdot p-1} | y\right), \tag{1.5}$$

where x_i's are encoded concentrated circles that are assumed to be independent.

SVM. SVM is deemed as one of the most powerful machine learning models with the problem formulation as follows.

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j k(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{1.6}$$

$$\text{s.t.} \sum_{i=1}^{N} \alpha_i d_i = 0, \tag{1.6a}$$

$$0 \le \alpha_i \le C, \;\; \forall i = 1, 2, \ldots, N, \tag{1.6b}$$

where α_i and d_i are the Lagrangian coefficient and the label that correspond to the instance $\boldsymbol{x}_i$. C is a manually chosen constant that controls the soft margin for non-separable datasets. After solving the problem, we can form the following classifier for hotspot detection.

$$d = \sum_{i=i}^{N_S} \alpha_i d_i k(\boldsymbol{x}_i, \boldsymbol{x}), \tag{1.7}$$

where $\boldsymbol{x}_i$s are support vectors from the training set that have non-zero Lagrangian coefficients. Because designers are free to choose different kernels, SVM provides more robust solutions for hotspot detection problems [128, 22, 21].

Artificial/Convolutional Neural Networks. ANN is a standard multilayer perceptron model that is able to fit highly nonlinear functions. An example of ANN is presented in Equations (1.8) and (1.9) with one input layer, one hidden layer and one output layer. The output layer generates the prediction scores or regression values.

The value of each node corresponds to the weighted sum of the nodes in previous layer, as shown in the following equations.

$$\boldsymbol{h} = f(\boldsymbol{W_1 x}), \tag{1.8}$$

$$\boldsymbol{y} = g(\boldsymbol{W_2 h}), \tag{1.9}$$

where f and g are activation functions that perform element wise operation on each neuron. If we map layout features to $\boldsymbol{x} = \{x_1, x_2, x_3, \dots\}$, the ANN can do either regression tasks (e.g. OPC [65]) or classification tasks (e.g. hotspot detection [22]). Due to the advantages of shared features, CNNs have become alternatives of ANNs as candidate layout learning models from hotspot detection [122, 124, 15] to mask optimization [2, 113].

1.3 Thesis Outline

In this dissertation, we will present the research of machine learning techniques on VLSI design for manufacturability problems, which, as outlined in Figure 1.6, includes lithography hotspot detection [120, 121], multi-layer hotspot detection [119], layout pattern sampling [114], mask optimization [112, 113] and test pattern generation [118]. These researches target at key challenges in DFM flows under advanced technology nodes.

In Chapter 2, we tackle the recent challenges of hotspot detection problems with discriminative learning models that cover: (1) standard lithography hotspot detection and (2) multi-layer hotspot detection. In the research of lithography hotspot detection, we develop efficient frequency-domain layout representation which introduces significant data compression that fits shallow convolutional neural networks but suffer little information loss. A theoretically supported biased learning algorithm is also developed to seek higher hotspot detection accuracy with limited false positive

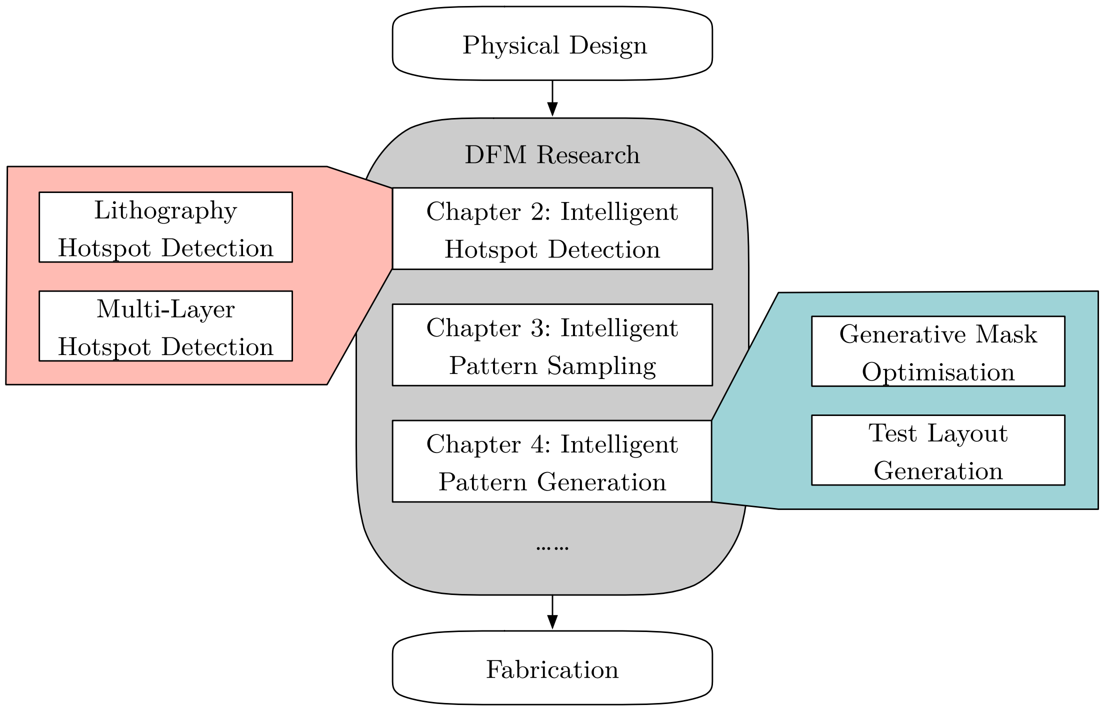

Physical Design
DFM Research
Lithography Hotspot Detection
Multi-Layer Hotspot Detection
Chapter 2: Intelligent Hotspot Detection
Chapter 3: Intelligent Pattern Sampling
Chapter 4: Intelligent Pattern Generation
Generative Mask Optimisation
Test Layout Generation
……
Fabrication

Figure 1.6: Bring artificial intelligence into back-end design flow.

penalty. In light of other hotspot detection tasks, e.g. CMP hotspot detection [27], where the detector is required to handle multiple layout layers, we introduce adaptive squish layout representation that fits well with state-of-the-art neural network models.

In Chapter 3, we discuss and analyze state-of-the-art physical verification and hotspot detection flows and show the importance of high quality small training datasets, considering (1) good distributed training set ensures better model convergence and generality and (2) labeled data instances are rare and costly to obtain at advanced technology node. We present an active learning based pattern sampling and hotspot detection flow that can efficiently update the training set on-the-fly and conduct detection task simultaneously.

In Chapter 4, we focus on pattern generation research that includes mask generation and design generation. Mask generation deals with mask optimization problems, which requires costly interaction with lithography simulation engines. We develop GAN-OPC family to instantly generate quasi-optimal masks, which takes much less legacy mask optimization time when producing final manufacturable masks. Design generation resolves the challenges of design shortage at the early stage of a technology node. We develop the TCAE framework that can capture simple design rules and generate DRC-clean massive designs with high diversity.

We summarize and conclude the thesis in Chapter 5.

Chapter 2

Intelligent Hotspot Detection

2.1 Introduction

Layout hotpot detection is one of the critical steps in modern integrated circuit design flow. It aims to find potential weak points (see Figure 2.1) in layouts before feeding them into manufacturing stage. These hotspots Classic solution is pattern simulation that can accurately predict hotspot regions incorporating with proper models, it is extremely computational costly.

To quickly and correctly recognize hotspots during physical verification, two ma-

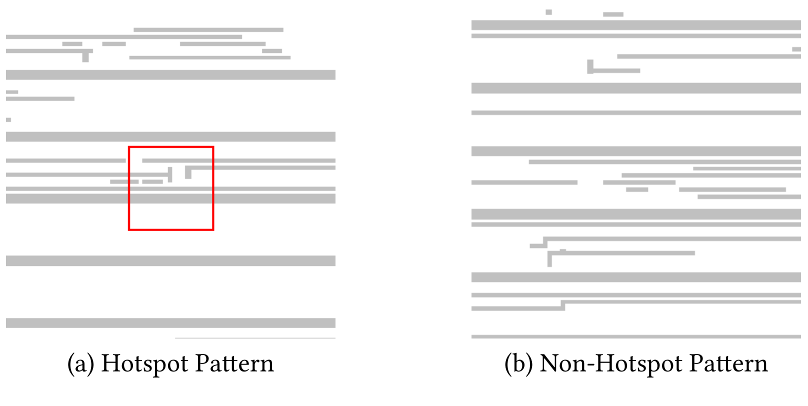

(a) Hotspot Pattern (b) Non-Hotspot Pattern

Figure 2.1: Layout design examples that contain hotspots and hotspot-free.

jor methodologies were heavily developed: pattern matching [104, 127, 110] and machine learning [128, 70, 130, 20, 22, 24]. In the pattern matching solutions, [104] facilitated the verification flow by integrating the density-based layout encoding, principle components analysis (PCA) and customized city-block distance. Yu *et al.* [127] developed a design rule-based pattern matching method to recognize hotspots. In [110], an improved tangent space-based distance metric was proposed to perform hotspot pattern analysis and classification. Although pattern matching is a direct and fast method to detect layout characteristics, it has a high error rate for unknown patterns. On the other hand, machine learning techniques are capable of learning hidden relations between layout patterns and their defect characteristics, and can greatly improve detection accuracy. Drmanac *et al.* [24] proposed a histogram-based layout representation and an unsupervised support vector machine (SVM) model to predict the variability of the lithography process. [20] incorporated hierarchical artificial neural networks (ANN) and SVM models to reduce the false alarm rates. Ding *et al.* [22] constructed a meta classifier with basic pattern matching and machine learning classifiers to achieve better detection performance. [128] presented a hotspot classification flow with a multi-kernel support vector machine and critical feature extraction. In [70], Adaboost and decision tree are adopted for fast hotspot detection. Very recently, Zhang *et al.* [130] achieved tremendous performance improvements on the ICCAD Contest 2012 benchmark suite [101] by applying an optimized concentric circle sampling (CCS) feature [72] and an online learning scheme.

However, there are several aspects that previous works do not take into account, especially when targeting a very large scale problem size. (1) **Scalability**: As integrated circuits develop to an ultra large scale, VLSI layout becomes more and more complicated and traditional machine learning techniques do not satisfy the scalability requirements for printability estimation of a large scale layout. That is, it may be hard for machine learning techniques to correctly model the characteristics of a large amount of layout patterns. (2) **Feature Representation**: The state-of-the-art layout feature extraction approaches, including density [70] and CCS [72], inevitably suffer from spatial information loss, because extracted feature elements are flattened into 1-D vectors, and ignore potential spatial relations. Because of the automatic feature learning technique and highly nonlinear neural networks, deep learning is highly successful in image classification tasks [55, 105]. Several attempts were made to detect layout hotspots using deep neural networks. [71, 93, 116, 117, 115] demonstrated that ordinary convolutional neural networks have potential on hotspot detection tasks. In this chapter, we will in detail discuss the layout-specific neural network architecture and algorithms on lithography (single layer) and multi-layer hotspot detection.

2.2 Lithography Hotspot Detection

2.2.1 Preliminaries and Problem Formulation

Designed layout patterns are transferred onto silicon wafers through a lithographic process, which involves a lot of variations. Some patterns are sensitive to lithographic process variations and may reduce the manufacturing yield due to potential open or short circuit failures. Layout patterns with a smaller process window and sensitive to process variations are defined as *lithography hotspots*.

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

Definition 2.1 (Accuracy[101]). *The ratio between the number of correctly predicted hotspot clips and the number of all real hotspot clips.*

Definition 2.2 (False Alarm[101])**.** *The number of non-hotspot clips that are predicted as hotspots by the classifier.*

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

Problem 2.1 (Hotspot Detection)**.** *Given a set of clips consisting of hotspot and non-hotspot patterns, the objective of hotspot detection is training a classifier that can maximize accuracy and minimize false alarms.*

2.2.2 Architectures and Algorithms

Feature Tensor Extraction

Finding a good feature representation is a key procedure in image classification tasks, and so is layout pattern classification. Local density extraction and concentric circle sampling were widely explored in previous hotspot detection and optical proximity correction (OPC) research [72], and were proved to be efficient on hotspot detection tasks because of the embedded lithographic prior knowledge. It is notable that layout hotspots are associated with light diffraction, therefore whether a layout pattern contains hotspots is not only determined by the pattern itself, but is also affected by the surrounding patterns. Therefore, to analyze clip characteristics, we mush be aware of the spatial relations of its local regions. However, all of these existing features finally are flattened into one dimensional vectors that limit hotspot detection accuracy due to a large amount of spatial information loss.

To address the above issue, we propose a feature tensor extraction method that provides a lower scale representation of the original clips while keeping the spatial information of the clips. After feature tensor extraction, each layout image $\boldsymbol{I}$ is converted into a hyper-image (image with a customized number of channels) $\boldsymbol{F}$ with

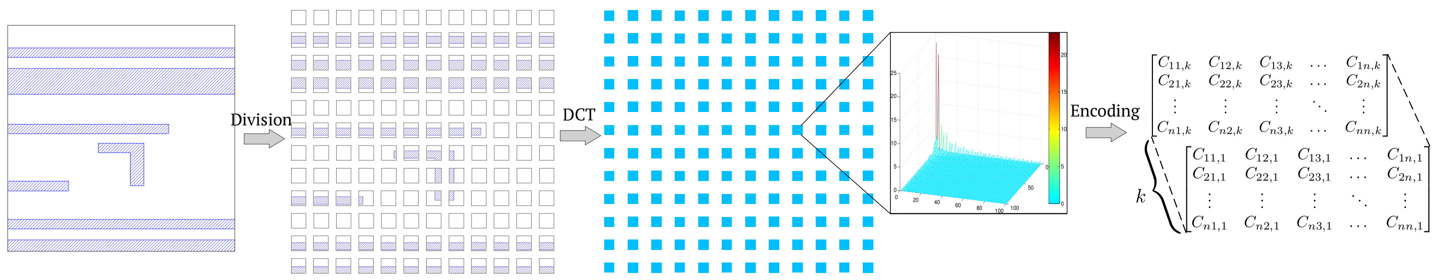

Division
DCT
Encoding
k
$C_{11,k}$ $C_{12,k}$ $C_{13,k}$ … $C_{1n,k}$
$C_{n1,k}$ $C_{n2,k}$ $C_{n3,k}$ … $C_{nn,k}$
$C_{11,1}$ $C_{12,1}$ $C_{13,1}$ … $C_{1n,1}$
$C_{n1,1}$ $C_{n2,1}$ $C_{n3,1}$ … $C_{nn,1}$

Figure 2.2: Feature tensor generation example ($n = 12$). The original clip ($1200 \times 1200\ nm^2$) is divided into 12×12 blocks and each block is converted to a 100×100 image representing a $100 \times 100\ nm^2$ sub region of the original clip. Feature tensor is then obtained by encoding on first k DCT coefficients of each block.

the following properties: (1) size of each channel is much smaller than $\boldsymbol{I}$ and (2) an approximation of $\boldsymbol{I}$ can be recovered from $\boldsymbol{F}$.

Spectral analysis of mask patterns for wafer clustering was recently explored in literature [131, 91] and achieved good clustering performance. Inspired by that work, we express the sub-region as a finite combination of different frequency components. High sparsity of the discrete cosine transform (DCT) makes it preferable over other frequency representations in terms of spectral feature extraction, and it is consistent with the expected properties of the feature tensor.

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

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

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

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

where $B = \frac{N}{n}$ is sub-region size, (x, y) and (m, n) are original layout image and frequency domain indexes respectively. Particularly, the left-upper side of DCT coefficients in each block correspond to low frequency components, that contain high density information, as depicted in Figure 2.2.

Step 3: Flatten $\boldsymbol{D}_{i,j}$s into vectors in Zig-Zag form [103] with the larger index being higher frequency coefficients as follows.

$$\boldsymbol{C}^{*}_{i,j} = [\boldsymbol{D}_{i,j}(0,0), \boldsymbol{D}_{i,j}(0,1), \boldsymbol{D}_{i,j}(1,0), ..., \boldsymbol{D}_{i,j}(B,B)]^{\intercal}. \quad (2.1)$$

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

$$\boldsymbol{C}_{i,j} = \boldsymbol{C}^*_{i,j}[: k], \tag{2.2}$$

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

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} & \boldsymbol{C}_{13} & \dots & \boldsymbol{C}_{1n} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} & \boldsymbol{C}_{23} & \dots & \boldsymbol{C}_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{C}_{n1} & \boldsymbol{C}_{n2} & \boldsymbol{C}_{n3} & \dots & \boldsymbol{C}_{nn} \end{bmatrix}, \tag{2.3}$$

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

The nature of discrete cosine transform ensures that high frequency coefficients are near zero. As shown in Figure 2.2, large responses only present at the entries with smaller indexes, i.e. low frequency regions. Therefore, most information is kept even when a large amount of elements in $\boldsymbol{C}^*_{i,j}$ 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.

The Neural Network Basis

This section discusses the the convolutional neural network details. First, we introduce the basis and the architecture of convolutional neural networks. Then, we present a customized training procedure that looks for better trade-offs between ac-

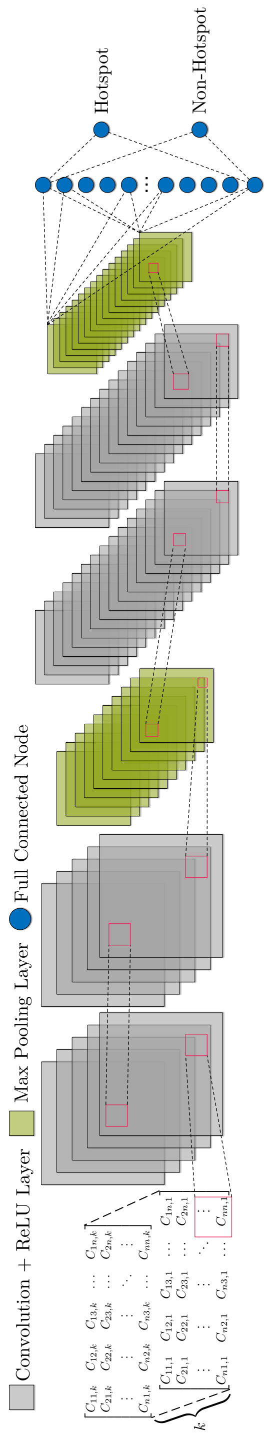


Figure 2.3: The proposed convolutional neural network structure.

curacy and false alarms. Finally, we list additional training and testing strategies.

To address the weak scalability of traditional machine learning techniques, we introduce the convolutional neural network (CNN) as preferred classifier. CNN is built with several convolution stages and fully connected layers, where convolution stages perform feature abstraction and fully connected (FC) layers generate the probability of testing instances drawn from each category (Figure 2.3).

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

$$\boldsymbol{F} \otimes \boldsymbol{K}(j,k) = \sum_{i=1}^{c} \sum_{m_0=1}^{m} \sum_{n_0=1}^{m} \boldsymbol{F}(i, j-m_0, k-n_0)\boldsymbol{K}(i, m_0, n_0), \tag{2.4}$$

where $\boldsymbol{F} \in \mathbb{R}^{c \times n \times n}$, and kernel $\boldsymbol{K} \in \mathbb{R}^{c \times m \times m}$. In this work, the convolution kernel size is set to 3×3 and the numbers of output feature maps in two convolution stages are 16 and 32 respectively. ReLU is an element-wise operation that follows each convolution layer as a replacement of the traditional sigmoid activation function. As shown in Equation (2.5), ReLU ensures that the network is nonlinear and sparse.

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0. \end{cases} \tag{2.5}$$

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

Table 2.1: Neural Network Configuration.

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

ing hotspot and non-hotspot. Detailed configurations are shown in Table 2.1.

Determining the gradient of each neuron and the parameter updating strategy in the neural network are two key mechanisms in the training procedure. Back-propagation [84] is widely applied to calculate gradients when training large neural networks. Each training instance $\boldsymbol{F}$ has a corresponding gradient set $\mathcal{G} = \{\boldsymbol{G}_1, \boldsymbol{G}_2, ..., \boldsymbol{G}_v\}$, where each element is a gradient matrix associated with a specific layer and v is the total layer number. All the neural network parameters are then updated with the obtained $\mathcal{G}$.

Stochastic gradient descent (SGD), where each training data instance is randomly presented to the machine learning model, has proved more efficient to train large data sets [31] than conventional batch learning, where a complete training set is presented to the model for each iteration. However, as a data set scales to the ultra large level, e.g. millions of instances, SGD has difficulty to efficiently utilize computation resources. Therefore, it takes a long time for the model to cover every instance in the training set. A compromise approach called mini-batch gradient descent (MGD) [37] can be applied where a group of instances are randomly picked to perform gradient descent. Additionally, MGD is naturally compatible with the online method allowing it to facilitate convergence and avoid large storage requirements for training ultra large instances.

However, for large nonlinear neural networks, back-propagation and MGD do not have rigorous convergence criteria. A fraction, empirically 25%, of training instances (validation set) is separated out and is never shown to the network for weight updating. We then test the trained model on the validation set every few iterations. When the test performance on the validation set does not show much variation or starts getting worse, the training procedure is considered to be converged. To make sure MGD reaches a more granular solution, we reduce the learning rate along with the training process.

Algorithm 2.1 Mini-batch Gradient Descent (MGD)

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

The details of MGD with learning rate decay are shown in **Algorithm** 2.1, where $\boldsymbol{W}$ is the neuron weights, λ is the learning rate, $\alpha \in (0, 1)$ is the decay factor, k is the decay step, $\boldsymbol{y}_h^*$ is the hotspot ground truth and $\boldsymbol{y}_n^*$ is the non-hotspot ground truth. MGD can be regarded as a function that returns the model with the best performance on the validation set. Indicator j will count up through iterations (line 4), and in each iteration, m training instances {$\boldsymbol{F}_1$, $\boldsymbol{F}_2$, ..., $\boldsymbol{F}_m$} are randomly sampled from the training set (line 5). Gradients of these training instances ($\mathcal{G}_i$) are calculated using

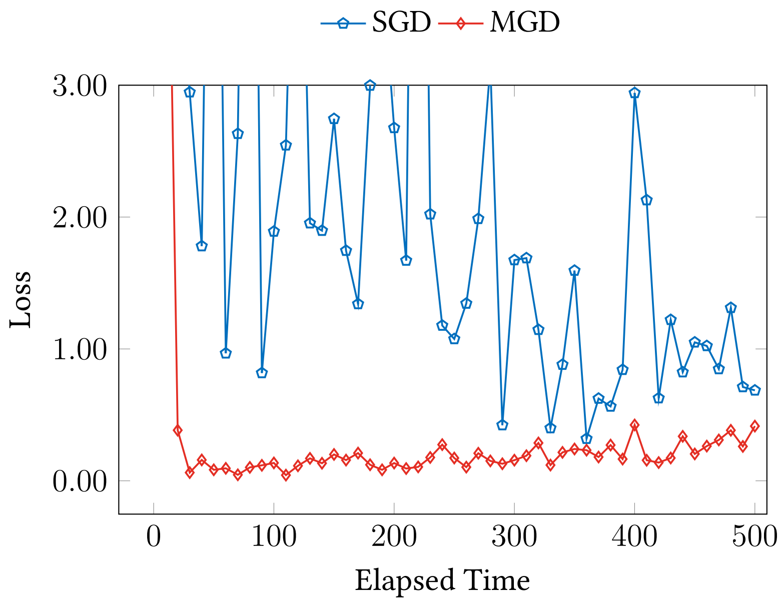
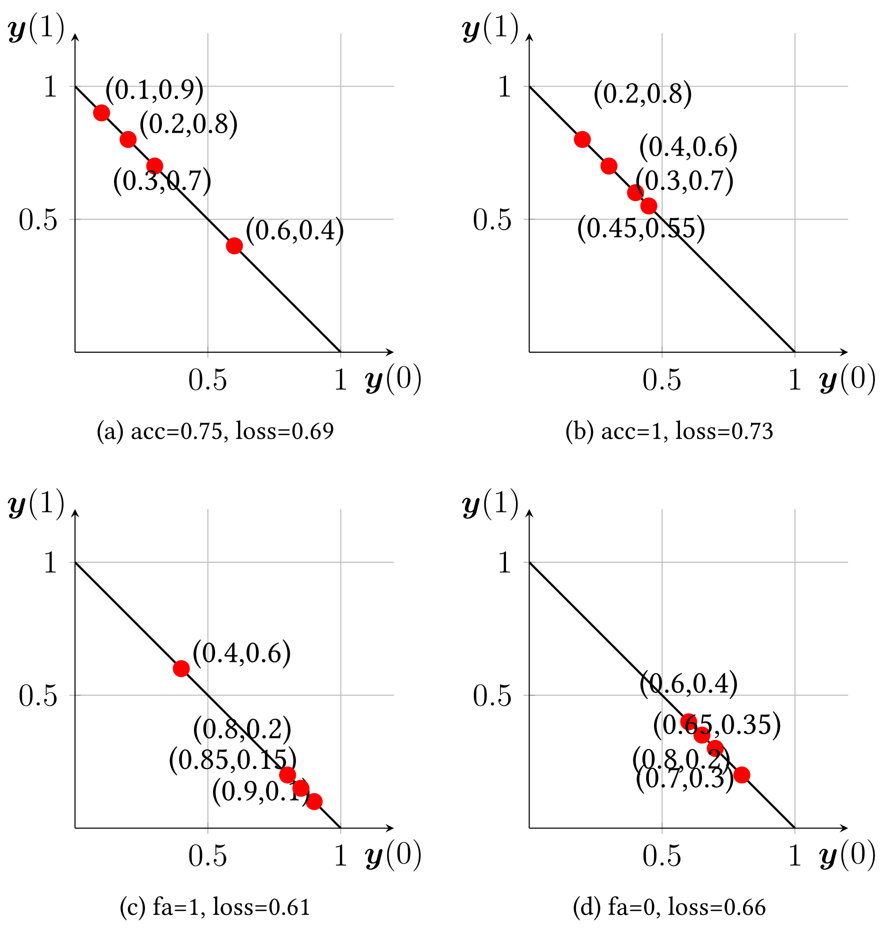


Figure 2.4: Stochastic gradient descent (SGD) v.s. Mini-batch gradient descent (MGD).

back-propagation (lines 6–8). Then neuron weights $\boldsymbol{W}$ are updated by subtracting the average gradient of sampled instances $\lambda\bar{\mathcal{G}}$ scaled by learning rate γ (line 14). When j is an integer multiple of k, λ is reduced to $\alpha\lambda$, i.e. the learning rate decays every k iterations (lines 11–13). At the end of MGD, a trained model that has satisfactory performance on validation set will be returned (line 15).

Although SGD has shown an advantage in emerging machine learning techniques, it cannot fully utilize GPU resources. MGD, on the other hand, is more compatible with parallel computing and can speed up training procedures. To evaluate the efficiency of MGD, we train the neural network on the `ICCAD` benchmark using MGD and SGD separately with the configuration in Table 2.3. The training procedure is shown in Figure 2.4, where the X-axis is the elapsed time (s) in the training procedure and the Y-axis is the cross-entropy loss on the validation set. The curve shows that MGD behaves much more stably than SGD, which indicates the neural network with the MGD learning strategy is more efficient and effective than conventional SGD.

Learning Towards Biased Target



(a) acc=0.75, loss=0.69

(b) acc=1, loss=0.73

(c) fa=1, loss=0.61

(d) fa=0, loss=0.66

Figure 2.5: Training loss versus accuracy.

Softmax cross entropy can provide speedup for back-propagation while attaining comparable network performance with the mean square error [37]. In a n-category classification task, the instance that belongs to class c has a ground truth $\boldsymbol{y}^* \in \mathbb{R}^n$ where $\boldsymbol{y}^*$ has the property that $\boldsymbol{y}^*(c) = 1$ and $\sum_{i=1}^{n} \boldsymbol{y}^*(i) = 1$. Each entry of $\boldsymbol{y}^*$ is regarded as the probability of the instance drawn from each category. The predicted label vector $\boldsymbol{y}$ by classifier is defined similarly.

In the task of hotspot detection, $\boldsymbol{y}^* = \boldsymbol{y}_n^* = [1, 0]$ and $\boldsymbol{y}^* = \boldsymbol{y}_h^* = [0, 1]$ are assigned as the ground truths for non-hotspot and hotspot. To generate loss with respect to ground truth, score $\boldsymbol{x} = [x_h, x_n]$ predicted by the neural network is scaled to a $(0, 1)$ interval by the softmax function shown in Equation (2.6).

$$\boldsymbol{y}(0) = \frac{\exp x_h}{\exp x_h + \exp x_n}, \quad \boldsymbol{y}(1) = \frac{\exp x_n}{\exp x_h + \exp x_n}, \tag{2.6}$$

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

$$l(\boldsymbol{y}, \boldsymbol{y}^*) = -(\boldsymbol{y}^*(0) \log \boldsymbol{y}(0) + \boldsymbol{y}^*(1) \log \boldsymbol{y}(1)). \tag{2.7}$$

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

$$\lim_{x \to 0} \ x \log x = 0. \tag{2.8}$$

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

$$\boldsymbol{F} \in \begin{cases} \mathcal{N}, & \text{if } \boldsymbol{y}(0) > 0.5, \\ \mathcal{H}, & \text{if } \boldsymbol{y}(1) > 0.5. \end{cases} \tag{2.9}$$

$$\boldsymbol{y}(0) + \boldsymbol{y}(1) = 1. \tag{2.10}$$

To improve the hotspot detection accuracy, a straightforward approach is shifting the decision boundary, as shown in Equation (2.11).

$$\boldsymbol{F} \in \begin{cases} \mathcal{N}, & \text{if } \boldsymbol{y}(0) > 0.5 + \lambda, \\ \mathcal{H}, & \text{if } \boldsymbol{y}(1) > 0.5 - \lambda, \end{cases} \tag{2.11}$$

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

The conventional training procedure applies ground truth label $\boldsymbol{y}_n^* = [1, 0]$ for non-hotspot instances and $\boldsymbol{y}_h^* = [0, 1]$ for hotspot instances. For non hotspot instances, the classifier is trained towards $\boldsymbol{y}_n^*$. If the training procedure meets the stop criteria, then for most non-hotspot clips, f will predict them to have a high probability, close to 1, to be non-hotspots. However, as can be seen in Equation (2.9), the instance would be predicted as a non-hotspot as long as the predicted probability is greater than 0.5. Thus, to some extent, the classifier is too confident as expected.

Intuitively, a too confident classifier is not necessary to give a good prediction performance and on the contrary, may induce more training pressure or even overfitting problem. We exemplify the case using a linear classifier. As illustrated in Figure 2.5, a more confident classifier results in an optimized loss, but cannot guarantee higher classification accuracy. Therefore, an assumption can be made that the hotspot detection accuracy can be further improved by sacrificing the training loss of non-hotspot samples. Meanwhile, the induced false alarm penalties are expected to be lower than directly shifting the decision boundary.

Assumption 2.1. *Given a pre-trained convolutional neural network model with ground truth $\boldsymbol{y}_n^* = [1, 0]$ and $\boldsymbol{y}_h^* = [0, 1]$ and hotspot detection accuracy a on a given test set. Fine-tune the network with $\boldsymbol{y}_n^\epsilon = [1 - \epsilon, \epsilon]$, $\epsilon \in [0, 0.5)$, we can obtain the hotspot detection accuracy a' and false alarm fs' of the new model. Shifting the decision boundary of the original model to reach the detection accuracy a', the corresponding false alarm is fs''. We have $a' \geq a$ and $fs'' \geq fs'$.*

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

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

$$\boldsymbol{x}_i = \boldsymbol{W}_l^{\intercal} O_{l-1}(\boldsymbol{F}_i), \tag{2.12}$$

where $\boldsymbol{W}_l$ is learned towards the target $\boldsymbol{y}_n^* = [1, 0]$. We will show in Equations (2.29)–(2.32) that training gradients of non-hotspot instances with biased labels are smaller than those without bias. Considering the fine-tune process with $\boldsymbol{y}_n^{\epsilon} = [1-\epsilon, \epsilon]$, training instances with predicted probability less than $1-\epsilon$ are only supposed to generate minor gradient to update the network, since they can not even make prominent difference with the target $\boldsymbol{y}_n$ when the stopping criteria is met. For those non-hotspot instances that have predicted probability in $(1-\epsilon, 1)$ (confident instances), neuron weights are updated along the gradient generated by confident instances.

Also, gradient vanishing theory [47] indicates a later layer in the neural network learns faster, therefore within a limited number of iterations, updates of the front layer can be ignored. We assume the layers before O_{l-1} are fixed for some iterations. Let the updated weight for the output layer be $\boldsymbol{W}_l'$, and the current output for confident instance $\boldsymbol{F}_c$ is,

$$\boldsymbol{x}_c' = \boldsymbol{W}_l'^{\intercal} O_{l-1}(\boldsymbol{F}_c). \tag{2.13}$$

Before adjusting the ground truth, we have

$$\boldsymbol{x}_c = \boldsymbol{W}_l^{\intercal} O_{l-1}(\boldsymbol{F}_c). \tag{2.14}$$

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

$$\boldsymbol{w} = \boldsymbol{W}_{l,1} - \boldsymbol{W}_{l,2}, \;\; \boldsymbol{w}' = \boldsymbol{W}_{l,1}' - \boldsymbol{W}_{l,2}'. \tag{2.15}$$

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

$$\begin{aligned}\boldsymbol{w}' &= \boldsymbol{w} - \alpha\nabla_{\boldsymbol{w}}(\boldsymbol{x}_c(0) - \boldsymbol{x}_c(1))(\nabla_{\boldsymbol{x}_c(0)}L_c - \nabla_{\boldsymbol{x}_c(1)}L_c) \\ &= \boldsymbol{w} - \alpha O_{l-1}(\boldsymbol{F}_c)(\nabla_{\boldsymbol{x}_c(0)}L_c - \nabla_{\boldsymbol{x}_c(1)}L_c), \end{aligned} \tag{2.16}$$

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

$$L_c = -\boldsymbol{y}_n^{\epsilon}(0)\log \boldsymbol{y}_c(0) - \boldsymbol{y}_n^{\epsilon}(1)\log \boldsymbol{y}_c(1), \tag{2.17}$$

$$\boldsymbol{y}_c(0) = \frac{\exp \boldsymbol{x}_c(0)}{\exp \boldsymbol{x}_c(0) + \exp \boldsymbol{x}_c(1)}, \tag{2.18}$$

$$\boldsymbol{y}_c(1) = \frac{\exp \boldsymbol{x}_c(1)}{\exp \boldsymbol{x}_c(0) + \exp \boldsymbol{x}_c(1)}. \tag{2.19}$$

Substitute Equation (2.18) and Equation (2.19) into Equation (2.17),

$$\begin{aligned}L_c = &-\boldsymbol{y}_n^{\epsilon}(0)\boldsymbol{x}_c(0) - \boldsymbol{y}_n^{\epsilon}(1)\boldsymbol{x}_c(1) \\ &+ \log(\exp \boldsymbol{x}_c(0) + \exp \boldsymbol{x}_c(1)),\end{aligned} \tag{2.20}$$

$$\nabla_{\boldsymbol{x}_c(0)}L_c = -\boldsymbol{y}_n^{\epsilon}(0) + \boldsymbol{y}_c(0), \tag{2.21}$$

$$\nabla_{\boldsymbol{x}_c(1)}L_c = -\boldsymbol{y}_n^{\epsilon}(1) + \boldsymbol{y}_c(1). \tag{2.22}$$

$$\nabla_{\boldsymbol{x}_c(0)}L_c - \nabla_{\boldsymbol{x}_c(1)}L_c > 0. \tag{2.23}$$

For hotspot instances:

$$\begin{aligned}&\boldsymbol{w}'^{\intercal}O_{l-1}(\boldsymbol{F}_h) \\ =&\boldsymbol{w}^{\intercal}O_{l-1}(\boldsymbol{F}_h) - \alpha O_{l-1}^{\intercal}(\boldsymbol{F}_c)(\nabla_{\boldsymbol{x}_c(0)}L_c - \nabla_{\boldsymbol{x}_c(1)}L_c)O_{l-1}(\boldsymbol{F}_h).\end{aligned} \tag{2.24}$$

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

$$\boldsymbol{w}^\intercal O_{l-1}(\boldsymbol{F}_h) > \boldsymbol{w}'^\intercal O_{l-1}(\boldsymbol{F}_h), \tag{2.25}$$

which indicates that

$$\boldsymbol{x}_h(0) - \boldsymbol{x}_h(1) > \boldsymbol{x}'_h(0) - \boldsymbol{x}'_h(1), \tag{2.26}$$

$$\Rightarrow \frac{\exp \boldsymbol{x}_h(1)}{\exp \boldsymbol{x}_h(0) + \exp \boldsymbol{x}_h(1)} < \frac{\exp \boldsymbol{x}'_h(1)}{\exp \boldsymbol{x}'_h(0) + \exp \boldsymbol{x}'_h(1)}. \tag{2.27}$$

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

From a physical point of view, the biased term ϵ can be regarded as a force "dragging" the decision boundary closer to non-hotspot instances. However, in the space defined by the pre-trained model, all the instances are located in different locations and have different distances to the decision boundary. For any non-hotspot instance, the loss with and without the biased term ϵ is given by Equations (2.28) and (2.29) respectively.

$$l = -\log \boldsymbol{y}(0), \tag{2.28}$$

$$\begin{aligned} l_b &= -(1-\epsilon)\log \boldsymbol{y}(0) - \epsilon \log \boldsymbol{y}(1) \\ &= -(1-\epsilon)\log \boldsymbol{y}(0) - \epsilon \log(1 - \boldsymbol{y}(0)). \end{aligned} \tag{2.29}$$

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

$$\frac{\partial l}{\partial \boldsymbol{y}(0)} = -\frac{1}{\boldsymbol{y}(0)}, \tag{2.30}$$

$$\frac{\partial l_b}{\partial \boldsymbol{y}(0)} = \frac{\epsilon + \boldsymbol{y}(0) - 1}{\boldsymbol{y}(0)(1 - \boldsymbol{y}(0))}. \tag{2.31}$$

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

$$|\frac{\partial l}{\partial \boldsymbol{y}(0)}| > |\frac{\partial l_b}{\partial \boldsymbol{y}(0)}|, \tag{2.32}$$

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

The bias term ϵ cannot be increased without limitations, because at some point, most of the non-hotspot patterns will cross the middle line, where the probability is 0.5, causing a significant increase of false alarms. Because the approach improves the performance of hotspot detection at the cost of confidence on non-hotspots, we call it *biased learning*. As the uncertainty exists for large CNN, a validation procedure is applied to decide when to stop biased learning. To sum up, biased learning is iteratively carrying out normal MGD with changed non-hotspot ground truths, as shown in Algorithm 2.2.

Algorithm 2.2 Biased-learning

```
Require: $\epsilon$, $\delta\epsilon$, $t$, $\boldsymbol{W}$, $\lambda$, $\alpha$, $k$, $\boldsymbol{y}_h^*$, $\boldsymbol{y}_n^*$;
1: $i \leftarrow 0$, $\epsilon \leftarrow 0$, $\boldsymbol{y}_h^* \leftarrow [0, 1]$;
2: while $i < t$ do
3:     $\boldsymbol{y}_n^* \leftarrow [1 - \epsilon, \epsilon]$;
4:     $f_\epsilon \leftarrow$ MGD($\boldsymbol{W}$, $\lambda$, $\alpha$, $k$, $\boldsymbol{y}_n^*$, $\boldsymbol{y}_h^*$);    ▷ Algorithm 2.1
5:     $i \leftarrow i + 1$, $\epsilon \leftarrow \epsilon + \delta\epsilon$;
6: end while
```

Here ϵ is the bias, $\delta\epsilon$ represents the bias step, and t is the maximum iteration of biased learning. In biased learning, the hotspot ground truth is fixed at $[0, 1]$ while the non-hotspot truth is $[1 - \epsilon, \epsilon]$. Initially, the normal MGD is applied with $\epsilon = 0$ (line 1). After getting the converged model, fine-tune it with ϵ updated by $\epsilon = \epsilon + \delta\epsilon$. Repeat the procedure until the framework reaches the maximum bias adjusting time t (lines 2–6).

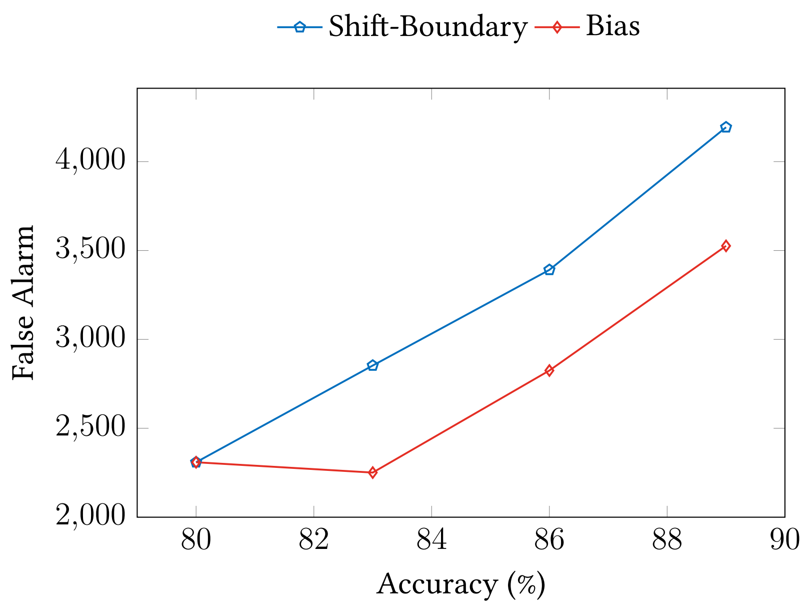


Figure 2.6: Bias learning shows a smaller false alarm penalty to obtain the same hotspot detection accuracy.

Assumption 2.1 shows that the biased learning algorithm can improve hotspot detection accuracy by taking advantage of the ReLU property. Because biased learning is applied through training, the false alarm penalty on the improvement of hotspot accuracy is expected to be limited. Here we evaluate the biased learning algorithm by training the neural network with $\epsilon = 0$ to obtain an initial model and fine-tuning with $\epsilon = 0.1, 0.2, 0.3$. Then we perform boundary shifting on the initial model to achieve the same test accuracy with three fine-tuned models. As shown in Figure 2.6, biased learning has 600 less false alarm penalties for the same improvement of hotspot detection accuracy, which demonstrates the validity of Assumption 2.1.

Batch Biased Learning

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

$$\epsilon(l) = \begin{cases} \frac{1}{1+\exp(\beta l)}, & \text{if } l \leq 0.3, \\ 0, & \text{if } l > 0.3, \end{cases} \tag{2.33}$$

where l is the training loss of the current instance or batch in terms of the unbiased ground truth and β is a manually determined hyper-parameter that controls how much the bias is affected by the loss. Because the training loss of the instance at the decision boundary is $-\log 0.5 \approx 0.3$, we set the bias function to take effect when $l \leq 0.3$. With the bias function, we can train the neural network in a single-round MGD and obtain a better model performance. Because $\epsilon(l)$ is fixed within each training step, no additional computing effort is required for back-propagation, as indicated by Equation (2.31).

The training procedure is summarized as Algorithm 2.3, where β is a hyper-parameter defined in Equation (2.33). Similar to MGD, we initialize the neural network (line 1) and update the weight until meeting the convergence condition (lines 2–16). Within each iteration, we first sample the same amount of hotspot and non-hotspot instances to make sure the training procedure is balanced (lines 3–4); we then calculate the average loss of non-hotspot instances to obtain the bias level and the biased ground truth (lines 5–6); the gradients of the hotspot and non-hotspot instances are calculated separately (lines 8–9); the rest of the steps are the normal weight update through back-propagation and learning rate decay (lines 11–15).

Algorithm 2.3 Batch Biased-learning

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

Data Augmentation and Ensemble Testing

Many studies have shown that data preprocessing provides a greater generalization ability of the deep learning model [116, 55, 94]. Observe that (1) the orientation of a clip does not affect its property under most illumination settings and (2) usually there are more non-hotspot clips than hotspot clips, we include several data augmentation techniques in the training stage accordingly.

- We randomly perform flipping (top-bottom transformation) and mirroring (left-right transformation) on each feature tensor along its last axis. It is easy to derive that the convolution operation is not flipping invariant, therefore random flipping and mirroring will introduce diversity to the dataset and increase the trained model's generalization ability (how well the model fits the testing data [129, 79]).

- We force the number of hotspot and non-hotspot instances to be equal in each mini-batch. As shown in the experiment of [116], a highly imbalanced dataset causes performance degradation. Sampling equal amount of instances from different categories is expected to benefit both the training progress and the model performance.

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

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

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

2.2.3 Experiments

Experimental Setup

We implement our deep biased learning framework in Python with the TensorFlow library [1], and test it on a platform with a Xeon E5 processor and Nvidia Graphic card. To fully evaluate the proposed framework, we employ four test cases. Because the individual test cases in the ICCAD 2012 contest [101] are not large to verify the scalability of our framework, we merge all the $28nm$ patterns into a unified test case `ICCAD`. Additionally, we adopt four more complicated industry test cases: `Industry0` – `Industry3`. The details for all test cases are listed in the Table 2.2.

Columns "Train HS#" and "Train NHS#" list the total number of hotspots and the total number of non-hotspots in the training set. Columns "Test NHS#" and "Test HS#" list the total number of hotspots and total number of non-hotspots in the testing set. Images in the testing set of `ICCAD` have a resolution of 3600×3600 which

Table 2.2: Benchmark Statistics

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

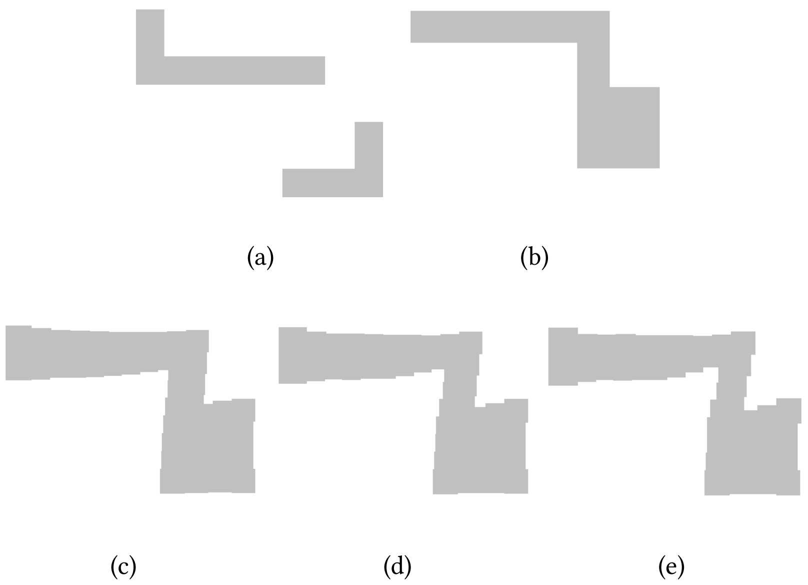


Figure 2.7: Benchmark examples. (a) and (b) are before-OPC patterns from `ICCAD` and `Industry0`, respectively. (b)–(d) correspond to `Industry1`-`Industry3`, which are from intermediate OPC results.. The pattern becomes more complicated after OPC and is more challenging for machine learning based hotspot detectors.

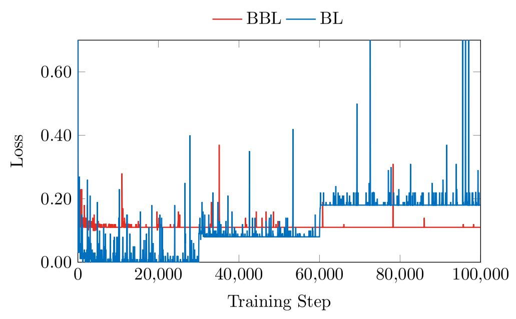
Table 2.3: Training Configurations

Configurations	Value
Optimizer	Adam [54]
Initial Learning Rate (λ)	0.001
Learning Rate Decay (α)	0.75
LR Decay Step (k)	10,000
Bias Function Coefficient (β)	6 and 43
Batch Size	32
Feature Tensor Channel	32

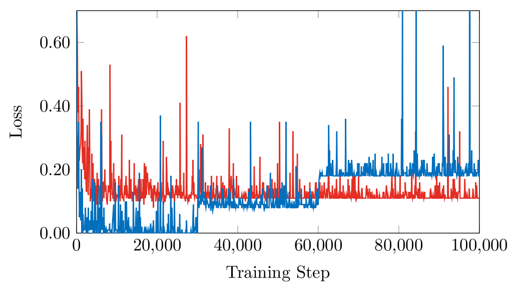
is larger than the images in industry benchmarks (1200×1200). Therefore, during the testing phase, each clip in `ICCAD` Testing Set is divided into nine 1200×1200 blocks before feeding into the testing flow. Mask images in the four benchmarks are from different OPC stages, and have different complexities, as shown in Figure 2.7. Note that classic hotspot detection problems aim to find and revise problematic designs at an early stage of the whole layout verification flow which corresponds to cases `ICCAD` benchmarks and `Industry0` that contains before-OPC patterns and are labeled based on the results of the entire layout verification flow, including OPC and lithography simulation. `Industry1`-`Industry3` are benchmark sets that contain layouts from intermediate OPC results which are labeled hotspot or non-hotspot based on the lithography simulation results of current OPC step. The motivation of introducing intermediate OPCed layouts is to show some potential of embedding efficient hotspot detectors into OPC engines and facilitate the procedure.

Model Training

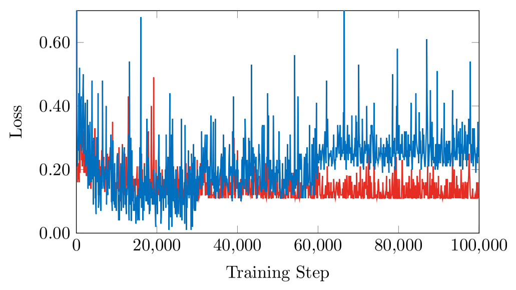
We train five individual models for each benchmark set following Algorithm 2.3. Table 2.3 lists the details of the training configurations. "Adam" is an improved optimizer to conduct a gradient descent proposed in [54] that is proved to converge faster.



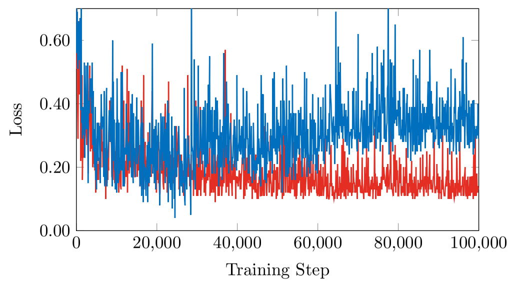
(a) `ICCAD`



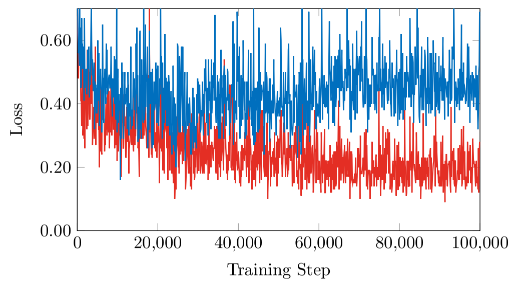
(b) `Industry0`



(c) `Industry1`



(d) `Industry2`



(e) `Industry3`

Figure 2.8: Visualize training of the batch biased learning (red) and the biased learning (blue). On each benchmarks, batch biased learning exhibits lower loss than the biased learning at convergence.

Parameters α and k denote that the learning rate (λ) drops to $\alpha\lambda$ every k steps. β is the coefficient that appears in Equation (2.33) and controls how much the bias is affected by the loss of a non-hotspot batch, therefore, it also controls the trade-offs between accuracy and false alarms. Here we use $\beta = 6$ for the datasets (`ICCAD`, `Industry0` and `Industry1`) with more regular patterns and $\beta = 43$ for the datasets (`Industry2` and `Industry3`) with complicated patterns. Because it takes more effort to fit the complicated patterns with the neural networks, we pick a larger β to avoid additional perturbation on the neural networks at early training stages (Figure 2.8). Feature Tensor Channel represents the number of remaining elements after dropping high frequency components in the feature tensor extraction procedure.

We visualize the training loss of Batch Biased Learning (BBL) and Biased Learning (BL) of five benchmarks in Figure 2.8, where ((a)), ((b)), ((c)), ((d)) and ((e)) correspond to `ICCAD`, `Industry0`, `Industry1`, `Industry2` and `Industry3`, respectively. The "Loss" is the average cross-entropy loss given by Equation (2.7) with respect to the unbiased ground truth. For the `ICCAD` and `Industry0` dataset, the batch biased learning converges quickly within 20,000 steps, while the biased learning requires manual adjustment of the bias and finally converges at a higher loss level. (see the blue curve in Fig. (a) and Fig. (b)). Also, for the benchmarks `Industry1-Industry3`, the batch biased learning (red curve) has better convergent result than the biased learning (blue).

Model Testing

To evaluate the effectiveness of the batch biased learning, we first compare the test results on four datasets (statistics are listed in Table 2.2) with our preliminary results in [121], as shown in Table 2.4 and Table 2.5. "Accu (%)" and "FA #" denote the hotspot detection accuracy (Definition 2.1) and the false alarm number (Definition 2.2). "FA (%)" is the percentage representation of false alarms that are coherent with "FA #".

Table 2.4: Performance comparison between the biased learning and the batch biased learning on target layouts.

Benchmarks	SPIE’ 17[116]+BL		SPIE’ 17[116]+BBL		BL[121]		BL[121]+AUG		BBL		BBL+AUG	
	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#
`ICCAD`	97.60	**2054**	98.04	3237	98.20	3413	**98.40**	3878	98.00	2745	**98.40**	3535
`Industry0`	98.73	316	**99.47**	469	97.35	**142**	99.26	246	97.66	136	99.36	387
Average	98.17	**1185**	98.76	1853	97.78	1778	98.83	2062	97.83	1441	**98.88**	1961
Ratio	0.993	**0.604**	0.999	0.945	0.989	0.906	0.999	1.052	0.989	0.735	**1.000**	1.000

Table 2.5: Performance comparison between the biased learning and the batch biased learning on OPCed layouts.

Benchmarks	SPIE’ 17[116]+BL		SPIE’ 17[116]+BBL		BL[121]		BL[121]+AUG		BBL		BBL+AUG	
	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#
`Industry1`	98.61	**311**	98.30	362	98.90	680	**99.00**	558	98.20	402	98.20	409
`Industry2`	94.62	**1156**	95.49	1550	93.60	2165	94.80	1483	94.70	1702	**95.50**	1691
`Industry3`	89.95	**2541**	**93.72**	4361	91.30	4196	91.30	3917	90.00	3840	91.40	3888
Average	94.39	**1336**	**95.84**	2091	94.60	2347	95.03	1986	94.30	1981	95.03	1996
Ratio	0.993	**0.669**	**1.008**	1.048	0.995	1.176	1.000	0.995	0.992	0.993	1.000	1.000

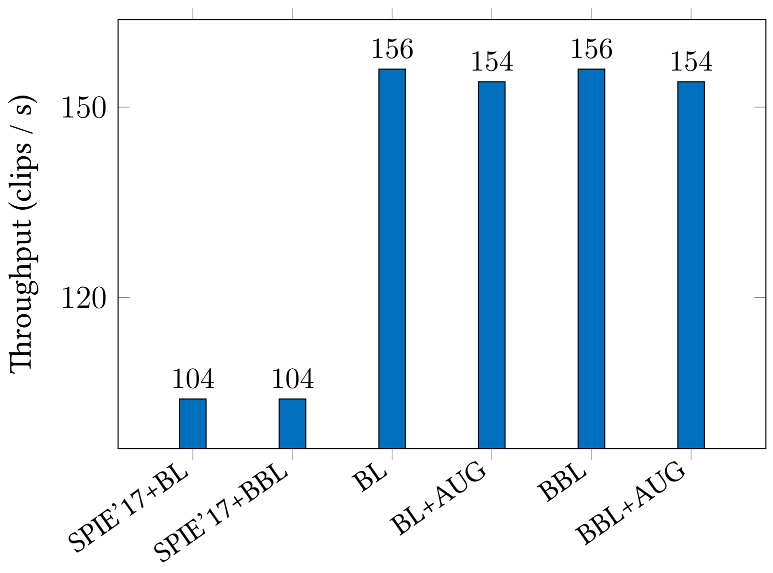


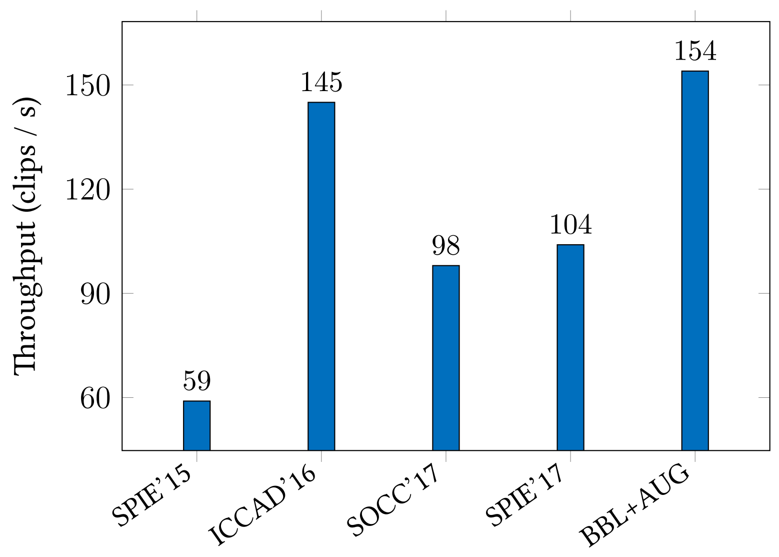
Figure 2.9: Throughput comparison of different neural network models.

"SPIE'17[116]+BL" corresponds to the results obtained by inserting biased learning (Algorithm 2.2) in SPIE'17[116] neural networks model; "SPIE'17[116]+BBL" lists the results obtained by embedding batch biased learning (Algorithm 2.3) in SPIE'17[116] neural networks model; "BL [121]" corresponds to the original biased learning algorithm (Algorithm 2.2) that is applied in our preliminary work; Column "BL [121] + AUG" contains the results obtained from the original biased learning and the data augmentation mentioned in Section 2.2.2; "BBL" lists the testing results of the batch biased learning as in **Algorithm** 2.3; "BBL + AUG" also includes the data augmentation in the batch biased learning procedure. Note that original SPIE'17 [116] also employs data augmentation on raw layout images.

In this paper, we propose a batch biased learning algorithm that can train the neural network in one round MGD and seek a better trade-off than the biased learning algorithm. Table 2.4 lists the testing results of target layouts `ICCAD` and `Industry0`, which show that BBL surpasses BL on both average detection accuracy and false alarm. With the aid of data augmentation, detection accuracy is further improved from 97.8% to 98.8% with ignorable false alarm penalty. For intermediate OPCed layouts `Industry1-3` in Table 2.5, BBL also significantly reduces average false alarm from 2347 to 1981 when achieving almost the same detection accuracy. We can also notice that for OPCed layouts with data augmentation, BBL does not exhibit as efficiently as on target layouts. One reason falls on the complexity of OPC patterns which induce more challenge on neural network training, when unbiased labels dominate the training procedure compared to BL which forces non-hotspot labels to be biased.

We also embed the proposed biased learning and batch biased learning algorithms into large neural networks designed in SPIE'17 [116] which takes raw layout images as input. Compared with original SPIE'17 [116] (as shown in Table 2.6 and Table 2.7), BL and BBL can significantly improve the hotspot detection performance. For target layouts, average detection accuracy increased from 97.63% to 98.17% and 98.76% with BL and BBL, respectively. BL also reduces the false alarm count from 1394 to 1185. For OPCed layouts, BL and BBL dramatically increase the detection accuracy from 88.17% to 94.39% and 95.84% respectively. In particular, BBL can further improve the hotspot detection accuracy when inserted in large SPIE'17 nets at relatively large false alarm penalty, because biased labels dominate during the training procedure of BBL for neural networks with larger capacity. Figure 2.9 illustrates the testing speed of different neural network models that correspond to Table 2.4 and Table 2.5. From the experimental results, we can clearly see that the proposed methods can achieve similar hotspot detection accuracy compared to larger neural network models but with much less computing costs (104 clips/s of deep networks in SPIE'17 v.s. 156 clips/s in our proposed network architecture). Although data augmentation and ensemble testing slightly induces computing costs, we can still observe great advantage of the proposed models over SPIE'17 nets.

We then compare the hotspot detection results with four state-of-the-art hotspot detectors in Table 2.6 and Table 2.7. "SPIE'15 [70]" is a traditional machine learning-



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

Figure 2.10: Throughput comparison with state-of-the-art hotspot detectors.

based hotspot detector that applies the density-based layout features and the AdaBoost [28]–DecisionTree model. “ICCAD'16 [130]” takes the lithographic properties into account during feature extraction and adopts the more robust Smooth Boosting [86] algorithm. “SPIE'17 [116]” is another deep learning solution for hotspot detection that takes the original layout image as input and contains more than 20 layers. “SOCC'17 [115]” employs a deep neural networks that replace all pooling layers with strided convolution layers and contains the same number of layers as SPIE'17. Overall, our framework performs better than traditional machine learning techniques (SPIE'15 [70] and ICCAD'16 [130]) with at least a 2% advantage for the detection accuracy (98.88% of BBL v.s. 96.89% of ICCAD'16) on target layouts. Traditional machine learning models are effective for the benchmarks with regular patterns (`ICCAD` and `Industry0`) with the highest detection accuracy of 97.7% on the `ICCAD` and 96.07% on the `Industry0` achieved by [130]. However, manually designed features, including the density-based features and the CCAS, have difficulties to grasp the attributes

Table 2.6: Performance comparison with state-of-the-art hotspot detectors on target layouts.

Benchmarks	SPIE'15[70]		ICCAD'16[130]		SOCC'17[115]		SPIE'17[116]		BBL+AUG	
	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#
ICCAD	84.20	2919	97.70	4497	96.90	**1960**	97.70	2703	**98.40**	3535
Industry0	93.63	**30**	96.07	1148	97.77	100	97.55	85	**99.36**	387
Average	88.92	1475	96.89	2823	97.34	**1030**	97.63	1394	**98.88**	1961
Ratio	0.899	0.752	0.980	1.439	0.984	**0.525**	0.987	0.711	**1.000**	1.000

Table 2.7: Performance comparison with state-of-the-art hotspot detectors on OPCed layouts.

Benchmarks	SPIE'15[70]		ICCAD'16[130]		SOCC'17[115]		SPIE'17[116]		BBL+AUG	
	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#	Accu (%)	FA#
Industry1	93.20	2204	89.90	1136	**98.50**	**318**	97.74	519	98.20	409
Industry2	44.80	1320	88.40	7402	91.1	**710**	89.62	760	**95.50**	1691
Industry3	44.00	3144	82.30	8609	84.01	**1628**	77.14	1966	**91.40**	3888
Average	60.67	2223	86.87	5716	91.20	**885**	88.17	1082	**95.03**	1996
Ratio	0.638	1.114	0.914	2.864	0.960	**0.444**	0.928	0.542	**1.000**	1.000

of the post-OPC mask layouts. For OPCed layouts, [70] suffers a large performance degradation with only approximately 45% detection accuracy on the most complicated case `Industry3`. Although prior knowledge and a more robust Smooth Boosting [86] algorithm are applied in [130], the hotspot detection accuracy inevitably drops around 8%. Deep learning solutions [115], [116] and BBL+AUG exhibit better performances with hotspot detection accuracy of 91.20%, 88.17% and 95.03% respectively. It is notable that the architecture of our framework contains significantly fewer layers than the framework in SPIE'17 and SOCC'17. Although the shallow architecture results in acceptable more false alarms than deep models, our framework can still offer significant higher accuracy on `ICCAD`, `Industry0`, `Industry2` and `Industry3`.

Although neural networks are not as computational efficient as traditional machine learning methods, with the aid of parallel computing units (e.g. GPU), we are able to achieve comparable and acceptable processing speed. Figure 2.10 presents the detecting speed of different hotspot detectors adopted in this work, where SPIE'15 and ICCAD'16 are tested on CPU only and neural network implementations are tested on graphic cards. Runtime reports show that although neural network models are not computational friendly due to the complicated convolutional operations, we are still able to complete the task with comparable and acceptable time, which also show the potential of enhancing the layout verification flow with dedicated computing units instead of CPU only.

2.2.4 Summary

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

2.3 Multi-Layer Hotspot Detection

2.3.1 Preliminaries and Problem Formulation

Squish Pattern

Multilayer patterns are much more complicated from geometric point of view. State-of-the-art layout representations are more or less exhibiting minor drawbacks that might be amplified when applied in multi-layer patterns. For example, density-based feature and CCS drop the spatial information of layouts, frequency domain features are computational costly if clip size is large and pixel-based representation is not storage friendly. *Squish patterns*, on the other hand, have been widely used in DFM tasks like pattern matching and pattern cataloging. The classic squish pattern [34] is a lossless layout representation that consists of layout topology and geometric information. As shown in Figure 2.11, a clip of layout is split into grids using a set of scan lines that cover all the shape edges. The topology of a given pattern can then be defined by a matrix $\boldsymbol{T}$ that has the same dimension as the pattern grids. Each entry

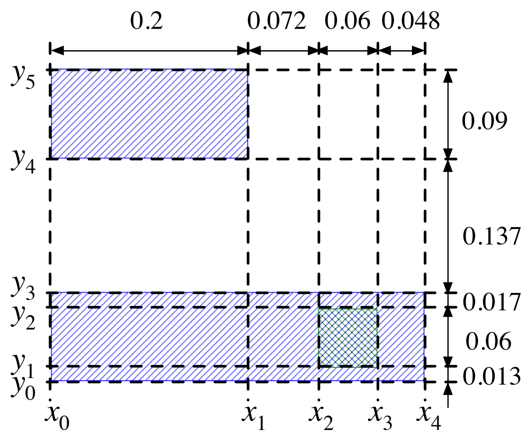


Figure 2.11: A simple multilayer pattern example with scan lines.

of $\boldsymbol{T} \in \mathbb{R}^{n_x \times n_y}$ is determined by Equation (2.34).

$$t_{ij} = \sum_{l=0}^{n-1} \alpha_l \cdot 2^l, \tag{2.34}$$

where l is the layer id and α_l indicates whether there are patterns in the corresponding grid, as in Equation (2.35).

$$\alpha_l = \begin{cases} 0, \text{given grid contains spacing in layer } l, \\ 1, \text{given grid contains geometry in layer } l. \end{cases} \tag{2.35}$$

We also need geometric information to define a pattern. Here we use two vectors $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$ to store the grid sizes in x and y directions, respectively. Each entry of the two vectors is defined in Equation (2.36) and Equation (2.37).

$$\delta_{x,i} = x_i - x_{i-1}, i = 1, 2, ..., n_x, \tag{2.36}$$

$$\delta_{y,i} = y_i - y_{i-1}, i = 1, 2, ..., n_y, \tag{2.37}$$

where x_is are the vertical scan line coordinates and y_is are horizontal scan line coordinates.

Therefore the pattern in Figure 2.11 can be represented as follows.

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \tag{2.38}$$

$$\boldsymbol{\delta}_x = \begin{bmatrix} 0.2 & 0.072 & 0.06 & 0.048 \end{bmatrix}, \tag{2.39}$$

$$\boldsymbol{\delta}_y = \begin{bmatrix} 0.013 & 0.06 & 0.017 & 0.137 & 0.09 \end{bmatrix}. \tag{2.40}$$

Above derivation shows that squish patterns benefit from two good properties: (1) Squish patterns are lossless representation and can be recovered to original layouts exactly; (2) Squish patterns store layout topologies and geometry information separately that make them storage efficient. However such representation still violates the requirements of most machine learning models, because squish patterns cannot guarantee a fixed size for a given layout window. Additionally, patterns with low complexity might have relatively larger geometry information per unit topology grid, which induces training bias for the learning models.

Problem Formulation

As hotspot detection problem, we still keep *accuracy* and *false alarms* as our evaluation metrics, with a new problem target as follows.

Problem 2.2 (Multi-Layer Hotspot Detection). *Given a set of clips consisting of hotspot and non-hotspot metal-via patterns, the objective of hotspot detection is training a classifier that can maximize accuracy and minimize false alarms.*

2.3.2 Architectures and Algorithms

This section will discuss the development of adaptive squish patterns and how them can be fed into convolutional neural networks.

Adaptive Squish Pattern

As we have discussed in Section 2.3.1, the dimensionality of squish topologies is not determined by the clip window size but the complexity of given patterns, which is not compatible with most machine learning models. Additionally, the large variation of pattern geometric information will induce more challenge on model convergence and generality, which can be explained by the relatively good behavior of the pixel-based images. Inspired by the benefits of data normalization computer vision task, here we propose an adaptive squish representation that derives from classic squish patterns. The main objectives are (1) **reducing the variance of pattern geometric information** (i.e. $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$) and (2) **extending original squish pattern into desired dimensions**. The basic idea is further adding more scan lines such that the topology dimensionality matches machine learning model requirements and the variance of $\boldsymbol{\delta}$s can be minimized.

Before discussing more details, we first introduce an operation

$$\boldsymbol{M}' = \texttt{RepeatElements}(\boldsymbol{M}, \boldsymbol{s}, a), \tag{2.41}$$

which duplicates the columns ($a = 0$) or rows ($a = 1$) of a matrix $\boldsymbol{M} \in \mathbb{R}^{a_1 \times a_2}$ by certain times such that the shape of the new matrix $\boldsymbol{M}'$ will be increased to a desired value. Equation (2.42) determines how the k^{th} column of $\boldsymbol{M}'$ is constructed when

$a = 0$.

$$\boldsymbol{m}'_k = \boldsymbol{m}_j, \forall \sum_{i=1}^{j-1} s_i < k \leq \sum_{i=1}^{j} s_i, \tag{2.42}$$

where $\boldsymbol{m}_j$ is the j^{th} column of $\boldsymbol{M}$ and $k = 1, 2, ..., \sum_{i=1}^{n} s_i$. Row duplication can be done similarly by

$$\texttt{RepeatElements}(\boldsymbol{M}, \boldsymbol{s}, 1) = \texttt{RepeatElements}(\boldsymbol{M}^\top, \boldsymbol{s}, 0)^\top. \tag{2.43}$$

For example, if we let $\boldsymbol{s} = \begin{bmatrix} 1 & 1 & 2 & 1 \end{bmatrix}^\top$ and $a = 0$, then the `RepeatElements` operation on the topology matrix of Figure 2.11 will result in

$$\boldsymbol{T}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 3 & 3 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{2.44}$$

We can also notice that `RepeatElements` is equivalent to get the topology matrix after adding additional scan lines that can evenly split existing grids. The number and the direction of additional scan lines are determined by $\boldsymbol{s}$ and a. Compared to zero-padding, `RepeatElements` extends a squish topology to a given size while keeping all the entries of the topology matrix to be informative. Now the problem becomes where and how many scan lines we should add, i.e. determining $\boldsymbol{s}$ for both x and y directions.

We denote the duplication vectors for both directions as $\boldsymbol{s}_x$ and $\boldsymbol{s}_y$ respectively. To ensure the layout represented by the squish pattern unchanged, we also need to scale and duplicate $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$ accordingly. Here we formulate the following problem to obtain satisfactory $\boldsymbol{s}_x$ and $\boldsymbol{s}_y$ to change the topology matrix to a desired size as well as attaining low variance $\boldsymbol{\delta_x}$ and $\boldsymbol{\delta_y}$. For simplicity, we discard x and y subscription and use unified symbols in following discussion and assume the desired total number of scan lines in one direction is d. The geometry information before and after scaling are denoted as $\boldsymbol{\delta}$ and $\boldsymbol{\delta}'$.

$$\begin{aligned}
\min_{\boldsymbol{s}} \quad & ||\boldsymbol{\delta}'||_\infty && \text{(2.45a)}\\
\text{s.t.} \quad & \delta'_i = \delta_i / s_i, \forall i, && \text{(2.45b)}\\
& s_i \in \mathbb{Z}^+, \forall i, && \text{(2.45c)}\\
& \sum_i s_i = d. && \text{(2.45d)}
\end{aligned}$$

The problem in Formula (2.45) aims to add more scan lines in the original squish pattern such that the grids are split into given number of pieces. The objective ensures the variance of the geometric vectors are minimized. Although this problem is non-convex and hard to solve, we can still observe the basic idea beneath this problem is adding scan lines to split large grids. Here we will propose two algorithms that promise an approximate solution of Formula (2.45).

Algorithm 2.4 Obtaining adaptive squish patterns with a greedy procedure.

```
Require: $\boldsymbol{T}$, $\boldsymbol{\delta}$, $a$, $d_0$, $d$;
Ensure: $\boldsymbol{T}$, $\boldsymbol{\delta}$;
1: while $d_0 < d$ do
2:     $\boldsymbol{s} \leftarrow \mathbf{1} \in \mathbb{R}^{d_0}$, $i \leftarrow \arg\max_i \{\delta_i | i = 1, 2, ..., d_0 - 1\}$;
3:     $s_i \leftarrow 2$, $\delta_i \leftarrow \delta_i / 2, \forall i$;
4:     $\boldsymbol{\delta} \leftarrow$ RepeatElements($\boldsymbol{\delta}, \boldsymbol{s}, 1$);
5:     $\boldsymbol{T} \leftarrow$ RepeatElements($\boldsymbol{T}, \boldsymbol{s}, a$);
6:     $d_0 \leftarrow d_0 + 1$;
7: end while
```

Algorithm 2.4 circumvents Formula (2.45) and directly targets at obtaining adaptive squish patterns where scaling and duplication are conducted in serial. It requires inputs of original squish topology matrix, geometry information vector $\boldsymbol{\delta}$ with respect to a given direction a, the current and the desired total number of scan lines d_0 and d along that direction. The algorithm will continuously add scan lines until d_0 reaches d. In each iteration, we first find the index i corresponding to the largest value in $\boldsymbol{\delta}$ (line 2), then we build a split vector $\boldsymbol{s}$ that has the same size as $\boldsymbol{\delta}$ and reduce the largest value in $\boldsymbol{\delta}$ by a half (line 3), and then both $\boldsymbol{\delta}$ and $\boldsymbol{T}$ will be updated with `RepeatElements` according to current $\boldsymbol{s}$ indicating a scan line is added at location i (lines 4–5) followed by the update of d_0.

Algorithm 2.5 Deriving an approximate solution of Formula (2.45) that will be used for generating adaptive squish patterns.

```
Require: $\boldsymbol{\delta}$, $d_0$, $d$;
Ensure: $\boldsymbol{s}$;
 1: $l \leftarrow \sum_i \delta_i$;
 2: $t \leftarrow l/(d-1)$;
 3: $s_i \leftarrow \max\{1, \texttt{int}(\delta_i/t)\}, \forall i$;
 4: while $\sum_i s_i < d-1$ do
 5:     $\delta'_i \leftarrow \delta_i/s_i, \forall i$;
 6:     $i \leftarrow \arg\max_i\{\delta_i | i = 1, 2, ..., d_0 - 1\}$;
 7:     $s_i \leftarrow s_i + 1$;
 8: end while
 9: $\delta_i \leftarrow \delta_i/s_i, \forall i$;
10: $\boldsymbol{\delta} \leftarrow \texttt{RepeatElements}(\boldsymbol{\delta}, \boldsymbol{s}, 1)$;
11: $\boldsymbol{T} \leftarrow \texttt{RepeatElements}(\boldsymbol{T}, \boldsymbol{s}, a)$;
```

Algorithm 2.5 does not generate the adaptive squish patterns on-the-fly and we target at Formula (2.45) itself for an optimal or suboptimal $\boldsymbol{s}$. Therefore, only $\boldsymbol{\delta}$, d_0 and d are required. The first step is to calculate the mean t for all the δ_is which will be used as a criteria to determine $\boldsymbol{s}$ (lines 1–2). We then obtain an approximate $\boldsymbol{s}$ according to the ratio between current $\boldsymbol{\delta}$ and t (line 3). We add additional scan lines at index i where δ_i is still the largest among all the entries in $\boldsymbol{\delta}$ until the requirement of the number of scan lines is met (lines 4–8). A few steps are required to get the adaptive squish pattern, including update the $\boldsymbol{\delta}$ (line 9) and duplicate entries in $\boldsymbol{\delta}$ (line 10) and

$\boldsymbol{T}$ (line 11). We will show later in the experiment that Algorithm 2.5 actually performs better than Algorithm 2.4, which can be explained by the fact that the ideal objective value in Formula (2.45) is the mean of δ_is that is targeted by Algorithm 2.5.

To make the adaptive squish patterns compatible with convolutional neural networks, we package $\boldsymbol{T}$, $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$ into a 3D tensor $\boldsymbol{S} \in \mathbb{R}^{a_1 \times a_2 \times 3}$ that is defined below,

$$\boldsymbol{S}[:, :, 0] = \boldsymbol{T}, \tag{2.46}$$

$$\boldsymbol{S}[:, :, 1] = \texttt{RepeatElements}(\boldsymbol{\delta}_x^\top, [a_1], 1), \tag{2.47}$$

$$\boldsymbol{S}[:, :, 2] = \texttt{RepeatElements}(\boldsymbol{\delta}_y, [a_2], 0). \tag{2.48}$$

The Network Architecture and Detection Flow

ResNet [44] has shown significant better training convergence and model generality. It resembles regular convolutional neural networks with additional shortcut links connecting the input and the output of each convolution stage, which is a group of convolution layers as designed in VGG [94]. We therefore integrate ResNet blocks with a regular CNN similar to the structure used in [117]. Although additional ResNet blocks introduce more computational efforts, we will also show later that our network exhibits similar throughput compared to [117]. Both trainging and testing layouts are cataloged with legacy squish patterns that will be used to derive adaptive squish patterns. We train machine learning models with adaptive squish dataset and select the best model with standard cross-validation. Finally, the machine learning model will categorize testing data into hotspot and nonhotspot.

2.3.3 Experiments

The framework is implemented using Python 2.7 with `Tensorflow` library [1]. All experiments are conducted on a platform with NVIDIA Tesla P100 accelerator.

Table 2.8: Benchmark statistics.

	Train	Test	Image	Squish
Hotspot	3073	6015	320×320	64×64×3
Nonhotspot	973197	1457830		

The Dataset

To verify the effectiveness and the performance of our multilayer hotspot detection flow, we adopt an industry 14nm benchmark layout that contains at most 3 layers that cover metal 4, via 3 and via 4. The statistics are listed in Table 2.8. Columns "Train" and "Test" indicate the training set and testing set respectively. Columns "Image" and "Squish" denotes the data dimensionality that is used for image-based detection and adaptive squish pattern-based detection. Rows "Hotspot" and "Nonhotspot" correspond to the number of hotspot and non-hotspot clips. We can notice that the benchmark is consistent with regular layout designs where hotspot patterns/locations are extremely rare, which brings much more challenges to machine learning engines [117]. To address this concern, we apply a training technique that during training, we manually force each mini-batch contains same number of hotspot and nonhotspot samples.

Effectiveness of Adaptive Squish Patterns

In this experiment, we show how adaptive squish patterns behave with two approximate solutions. We train the neural networks using two types of squish patterns obtained from Algorithm 2.4 and Algorithm 2.5 respectively. The initial learning rate is set to be 0.001 and decays by 0.7 every 2000 steps. The batch size is 64 and each batch contains 32 hotspot samples and 32 non-hotspot samples. Neuron weights are initialized with xavier approach [35] and are optimized with Adam optimizer on softmax cross entropy loss. We also apply weight normalization on weights in all convolution

Table 2.9: Result comparison of two adaptive squish solutions and a baseline CNN with image-based inputs.

Item	JM3 [117]	Algorithm 2.4	Algorithm 2.5
Accuracy (%)	98.87	97.51	**99.24**
False Alarm Rate (%)	4.81	5.05	**4.52**
Hit	5947	5865	**5969**
False Alarm	70193	73645	**65926**
Precision (%)	7.81	7.38	**8.30**

layers and fully connected layers with a coefficient of 0.001. The trained model is selected based on standard cross validation with a validation set contains 500 hotspot samples from the training set. The maximum training step is set to 10000. Table 2.9 lists the detailed hotspot detection results. It can be obviously seen that Algorithm 2.5 outperforms Algorithm 2.4 from both hotspot detection accuracy (by 1.7%) and false alarm (by 7719 clips), which can be explained by the fact that Algorithm 2.5 achieves much lower variance of geometric vector values.

Result Comparison with A State-of-the-art CNN Solution

In this experiment, we train another baseline CNN model using the neural networks proposed in [117]. We train the neural network with exactly the same strategy as discussed in previous section. The only difference is that the inputs become image/pixel-based representation with a resolution of $1nm$ per pixel, and the input size changes to 320×320 accordingly. As can be seen in the column "JM3 [117]" in Table 2.9, the image-based input behaves better than Algorithm 2.4 but worse than Algorithm 2.5. Although the hit count in [121] is quite close to the result of Algorithm 2.5, there are still significantly larger amount of false alarms.

It can be observed that pixel-based layout is actually a special case of adaptive squish pattern. If we chose d to be the same as clip size in terms of nm, the matrix

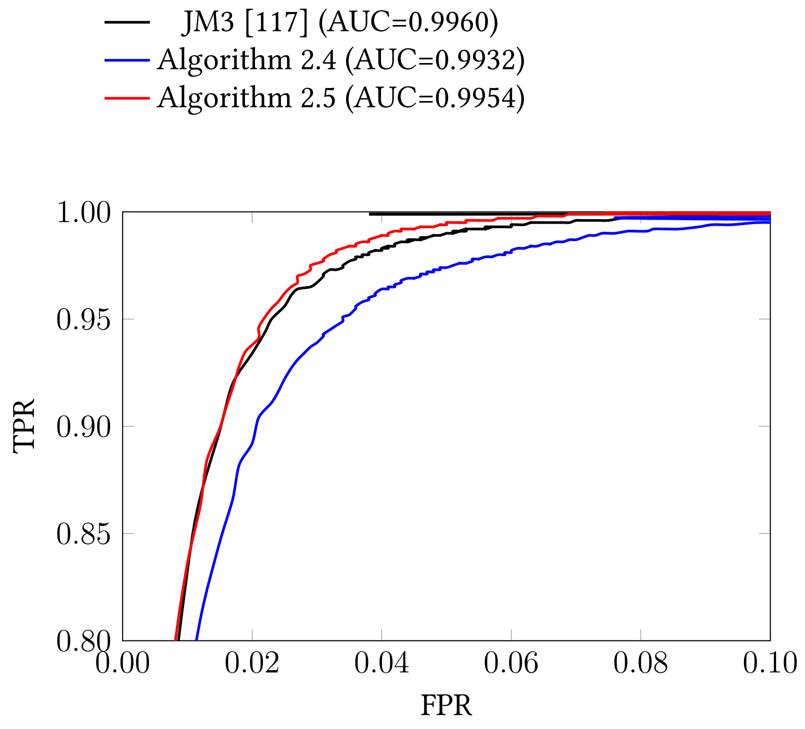


Figure 2.12: Partial receiver operating characteristics of three hotspot detectors.

that represents a clip image will be the optimal solution of Formula (2.45) where all the entries of $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_x$ are one with zero variance. However, larger input size brings computational cost, requires more storage and equips with redundant information that are not training friendly.

We also depict the part of the receiver operating characteristic (ROC) of three models in Figure 2.12. We can observe that three models show good quality in terms of area under ROC curve (AUC) with Algorithm 2.4 slightly weaker than image-based solution and Algorithm 2.5, which is coherent with the detection results listed in Table 2.9. [117] exhibits even better than Algorithm 2.5 in terms of AUC. By analyzing the detailed data, however, we find that most of the AUC advantages of [117] come from the region where the decision threshold is above 0.9. That means the model of [117] shows higher confidence on hotspot patterns that can be correctly predicted by both classifiers, which explains why [117] does not behave as good as Algorithm 2.5

in final prediction results as in Table 2.9.

2.3.4 Summary

This section introduces an adaptive layout squish representation which is lossless, storage friendly, compatible with neural networks and naturally support multilayer patterns. Such representation is applied in a medium sized convolutional neural networks with ResNet blocks. To the best of our knowledge, this is first time multi-layer layout hotspots are considered in hotspot detector design. Experimental results show that our framework outperforms a state-of-the-art CNN-based hotspot detector on both accuracy and false alarm. Future research on reducing false alarms will be explored to enable the framework to face more challenging EUV design flow.

Chapter 3

Intelligent Pattern Sampling

3.1 Introduction

Last chapter shows that although pattern matching-based methods and machine learning-based methods exhibit different functionalities, they all rely highly on the quality of reference layout libraries. For example, in-cluster pattern variance directly affects pattern matching results and pattern diversity contributes to the generality of trained machine learning models. Layout pattern sampling problems are addressed by several works that are, to some extent, related to clustering approaches. Representative methods include clustering on frequency domain [131, 92], Bayesian clustering [73], and clustering based on layout topology [13, 14, 92, 41]. However, sampling and hotspot detection are mostly conducted exclusively which ignores the beneath integrity between them.

In this chapter, we will introduce an active learning-based framework that can bridge the gap between the layout pattern sampling procedure and the hotspot detection problem. Active learning targets at machine learning problems with massive data that is costly and time consuming to label. A major step of active learning is querying instances to determine whether the instance should be labeled and added

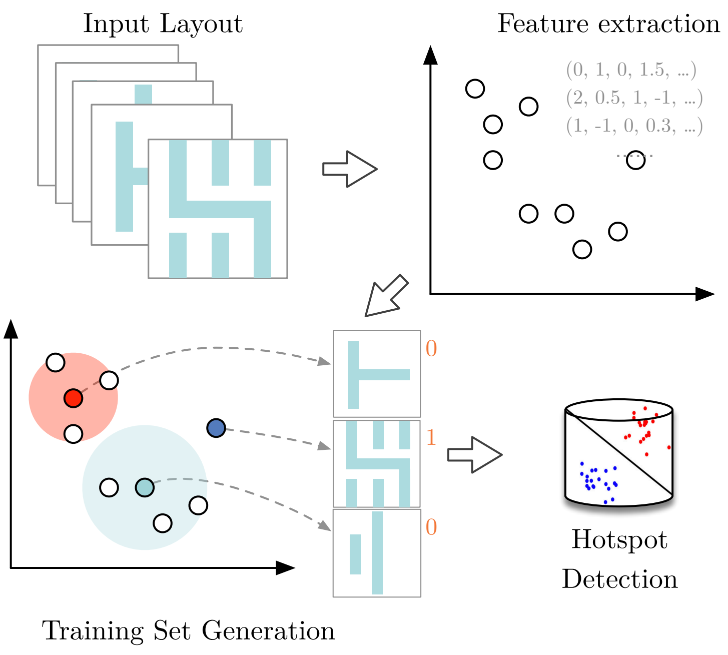


Figure 3.1: A conventional process of layout hotspot training set and detection model generation.

into the training set from a perspective of machine learning model generality [87]. Representative querying strategies include uncertainty sampling (US [59]), query by committee (QBC [29]), and expected model change (EMC [9]). US aims to find the instances which the prediction model is most uncertain about and have the posterior probability around 0.5, QBC selects instances based on the disagreement among multiple classifiers and EMC labels most influential data in terms of the existing model. A common idea behind these strategies is labeling instances that are hardly distinguished by the classifier. However, there are several drawbacks of existing active learning strategies: (1) only one sample is selected in each iteration in most active learning flows which is lacking in efficiency; (2) machine learning models have to be retrained from raw state once the training set is updated; (3) training set diversity is not considered in sampling flow which might cause serious overfitting problem [87, 98, 85]. Although Kullback–Leibler (KL) divergence [58] on posterior probabilities of unlabeled samples can be applied for diversity analysis [12], the effectiveness is limited on binary classification problems.

To address these concerns, we propose a batch mode active learning method that considers both model uncertainty and training set diversity for a better hotspot detection results. We embed the active learning engine into deep neural networks thus data sampling and incremental model training can be conducted alternatively. Guaranteed by the on-line property of stochastic gradient descent, we only need to finetune the neuron weights according to new labeled instances instead of training model from scratch in each iteration. The rapid development of deep neural networks also makes it possible to learn representative features from raw image, which becomes another reason that we pick CNN as our learning engine. We take advantage of this characteristic and construct a diversity matrix of automatically learned features which will contribute as a partial criterion for data sampling in each iteration.

3.2 Diversity-Aware Layout Pattern Sampling

3.2.1 Preliminaries and Problem Formulation

This section introduces some terminologies and related problem formulation. Throughout this section, scalers are written as lowercase letters (e.g. x), vectors are bold lowercase letters (e.g. $\boldsymbol{x}$) and matrices are represented as bold uppercase letters (e.g. $\boldsymbol{X}$). Particularly, we use $J_n(\cdot)$ to represent the Bessel function of the first kind of order n. All layout images are with a resolution of $1nm$/pixel. The framework evaluation metrics are defined as follows.

Definition 3.1 (Hit). *A hit is defined as when the detector reports hotspot on a clip of which at least one defect occurs at the core region. We also denote the ratio between number of hits and total hotpsot clips as detection accuracy.*

Here the *detector* in this paper refers to a hotspot detector with its input being a layout clip and its output being a label indicating whether the clip contains manufacturing issues.

Definition 3.2 (Extra). *An extra is defined as when the detector reports hotspot on a clip of which no defect occurs at the core region.*

Definition 3.3 (Litho-clip). *A litho-clip is a pattern in the training set or a pattern that is labeled hotspot by the machine learning model. The count of litho-clips reflects the lithography simulation overhead.*

According to the evaluation metrics above, we define the problem of layout pattern sampling and hotspot detection (PSHD) as follows.

Problem 3.1 (PSHD). *Given a layout design, the objective of PSHD is sampling representative clips that will generalize the hotspot pattern space and maximize the machine learning model generality, i.e., maximizing the detection accuracy while minimizing the number of litho-clips.*

From the definition we can observe that our problem formulation differs from traditional hotspot detection in previous works. Here we are dealing with a practical application, where no training set and testing set are given in advance. In short, we seek to conduct full chip hotspot detection with smallest lithography simulation overhead. Accordingly, the input of our framework will be **full chip layout** design and the outputs include a **training set** (labeled dataset), a **trained model** and an **unlabeled dataset**. The evaluation metrics of such framework include overall hotspot detection accuracy (see Equation (3.1))

$$Acc = \frac{HS_{Train} + Hits}{HS_{Total}}, \tag{3.1}$$

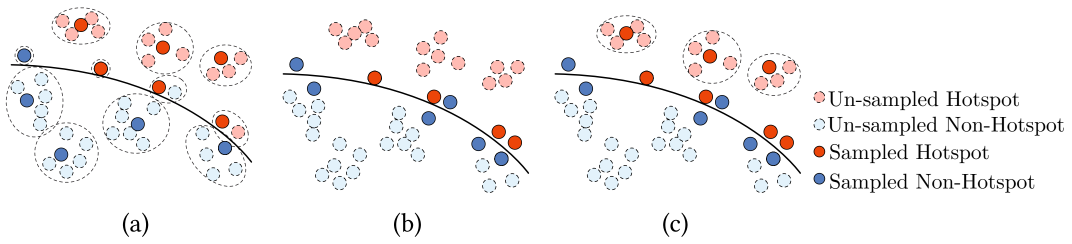


Figure 3.2: Visualization of different layout pattern sampling methods: (a) Pattern matching; (b) Conventional active learning; (c) Proposed layout pattern sampling.

where $HS_{Train}, Hits$ and HS_{Total} denote hotspot number in the training set, correctly predicted hotspot number in the unlabeled dataset and total hotspot number, and the lithography overhead as in Equation (3.2),

$$Litho = Tr + FA, \tag{3.2}$$

where Tr represents the total number of clips in the training set and FA is the detection false alarm in the unlabeled dataset.

3.2.2 Architectures and Algorithms

In this section, we will discuss the details of our pattern sampling and hotspot detection flow, including initial training set selection, batch active sampling algorithm, and some mathematical analysis.

Because it is extremely costly to label layout clips, our flow aims to sample as small number of clips as possible while ensuring good machine learning model generality. Conventional layout pattern sampling methods conduct clustering on layout clips and obtain representative patterns based on the results of pattern matching or clustering. Although the clustering can effectively reduce the sample number, it does not consider the behavior or requirement of, especially, machine learning-based hotspot detectors. As shown in Figure 3.2(a), pattern matching collects a lot of less critical patterns that lie far from the decision boundary while ignoring important patterns. In conventional active learning-based sampling (see Figure 3.2(b)), prediction uncertainty of each clip is included in the selection criteria. That is, patterns with posterior probability around 0.5 will be sampled with higher priority. However, in a layout pattern sampling and hotspot detection task, we care more about hotspot regions. Therefore, apart from considering diversity of training instances, we tend to select clips with higher probability being hotspot in our sampling approach, as illustrated in Figure 3.2(c).

The Flow Summary

The proposed layout pattern sampling and hotspot detection flow is illustrated in Figure 3.3. To analyze the printability of a full chip design, we dispatch the layout into clips based on the lithography proximity effect analysis, such that the whole chip is covered by the core region of each clip that contains enough information to conduct printability estimation. And then, the training set and the machine learning model will be updated until convergence, when all the clips will be either labeled or dropped. Finally, the full chip hotspot detection will be conducted on the dropped clips with the final learning model. We go through the framework details in the rest of this section.

Diversity Aware Batch Sampling

Discriminative machine learning models are usually designed to find the optimal hyperplane that separates the whole data space. The quality of a model is measured by its generative loss which is associated with the prediction error on the future instances. In this section, we will discuss an instance selection policy considering both *model uncertainty* and *data diversity*, thus the selected instances are expected to contribute most on the trained model generality. The uncertainty describes how confident of the

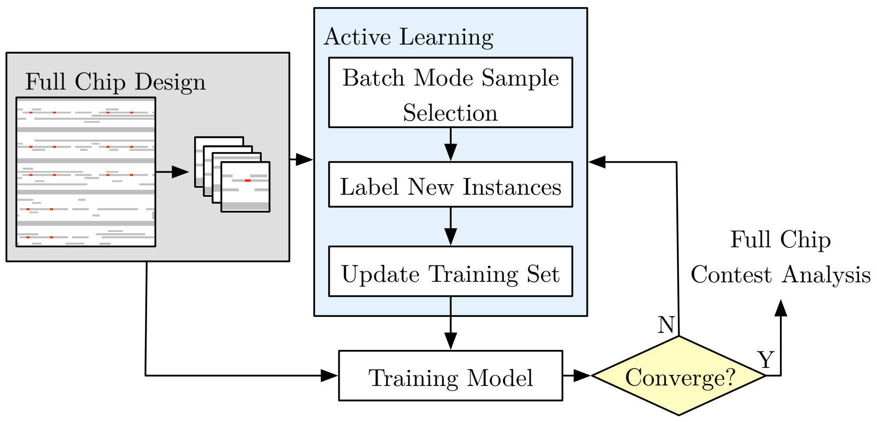


Figure 3.3: Pattern sampling and hotspot detection flow.

classifier when recognizing new instances. A model is uncertain on a given instance if the prediction probability draws around 0.5 according to the posterior distribution or the instance is too close to the hypothesis plane in the feature space. Data diversity corresponds to the instance distribution in the data set. The underneath idea is to label instances into training set such that the training set entropy is maximized. Most active learning algorithms, such as US [59], QBC [29] and EMC [9], are designed to pick one instance in each iteration, which is not efficient as problem sizes grow. Even these methods are applied for batch selection, samples touch the selection criteria are labeled into train set, when redundant instance samples are more likely to be chosen. Here we consider a **batch selection** mechanism that takes both model uncertainty and data diversity into account.

Suppose we have a training set $\mathcal{L}_t$ and an unlabeled set $\mathcal{U}_t$ at time t. Let $\boldsymbol{w}_t$ be the classifier parameters trained on $\mathcal{L}_t$. The objective is to select a batch $\mathcal{B}$ with k points from $\mathcal{U}_t$ so that the future learner $\boldsymbol{w}_{k+1}$, trained on $\mathcal{L}_t \cup \mathcal{B}$, has maximum generalization capability. Let $\mathcal{Y} = \{0, 1\}$ be the set of possible classes in the problem. For a given unlabeled layout clip $\boldsymbol{x}_i$, we denote the related posterior probability as $p(y|\boldsymbol{x}_i; \boldsymbol{w}_t)$. Usually, the uncertainty of the unlabeled instance $\boldsymbol{x}_i$ is defined as the entropy of the predicted probabilities, as shown in Equation (3.3).

$$c(i) = -\sum_{j \in \mathcal{Y}} p(y = j|\boldsymbol{x}_i; \boldsymbol{w}_t) \log p(y = j|\boldsymbol{x}_i; \boldsymbol{w}_t). \tag{3.3}$$

However, in the layout pattern sampling problems, problematic instances are of more interests. We therefore pick a simple but more practical representation of $c(i)$,

$$c(i) = p(y = 1|\boldsymbol{x}_i; \boldsymbol{w}_t), \tag{3.4}$$

which corresponds to the probability of a given instance being hotspot. Usually, the redundancy between unlabeled points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ can be calculated through KL divergence, which measures how two training instances differ from each other in a statistic point of view. In the domain of layout hotspot detection, however, we are dealing with yes or no problem, which is less informative for diversity analysis. To benefit the layout analysis problem, we use inner-product of two instances in the normalized feature space as shown in Equation (3.5).

$$E(i, j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j. \tag{3.5}$$

We can further formulate the diversity matrix $\boldsymbol{D} \in \mathbb{R}^{n \times n}$, whose entries are defined by Equation (3.5).

Given the matrix $\boldsymbol{D}$, the batch mode active learning problem is shown in mathematical formulation (3.6), where the objective is to select a batch of points with high aggregate uncertainty scores and high divergences among the samples.

$$\min_{\boldsymbol{m}} \quad \boldsymbol{m}^\top \boldsymbol{D}\boldsymbol{m}, \tag{3.6a}$$

$$\text{s.t.} \quad m_i \in \{0,1\}, \ \forall i, \tag{3.6b}$$

$$\sum_i m_i = k, \ \forall i. \tag{3.6c}$$

Here k is the number of patterns that will be selected into the training set, m_i is a binary variable, and $m_i = 1$ if pattern $\boldsymbol{x}_i$ is selected in the batch $\mathcal{B}$. It should be noted that Formula (3.6) is binary quadratic programming, which is NP-hard. We relax the integer constrains and derive the following problem,

$$\min_{\boldsymbol{m}} \quad \boldsymbol{m}^\top \boldsymbol{D}\boldsymbol{m}, \tag{3.7a}$$

$$\text{s.t.} \quad m_i \in [0,1], \quad \forall i, \tag{3.7b}$$

$$\sum_i m_i = k, \ \forall i, \tag{3.7c}$$

which is a standard quadratic programming problem and can be solved efficiently. It can be seen here one advantage of the proposed distance metric over KL-divergence and Euclidean distance is that Equation (3.5) ensures the objectives of (3.6) and (3.7) to be convex by $\boldsymbol{D} \succeq 0$. Finally, the integer solution can be recovered by picking k largest entries in $\boldsymbol{m}$.

Layout Pattern Sampling and Hotspot Detection

The rapid development of deep neural networks makes it possible to learn representative features from raw image and complete effective classification jobs. Therefore, in this project, we pick up a well designed shallow convolutional neural networks from [121] as the preferred machine learning model which will be embedded into the active learning flow. In particular, features obtained from the fully-connected layers are fed into Equation (3.5) to calculate the divergence matrix. Most neural networks are trained with mini-batch gradient descent (MGD), where a random small batch of training samples are fed into the neural networks to update neuron weights. The online property of MGD makes it easier to update the model on new instances without retraining the model from scratch compared to traditional support vector machine or logistic regression. It should be noted that the proposed classification driven active learning flow is very general that it can be plugged into any incremental hotspot detectors.

In most cases $\boldsymbol{D}$ will be extremely large, especially for EUV specific layers, which makes Formula (3.7) hard to solve. We therefore stochastically sample a subset $\hat{\mathcal{U}_t} \subseteq \mathcal{U}_t$ before entering the quadratic programming phase to further reduce the computational cost. Finally, the neural network can be accordingly updated as $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \frac{\partial l}{\partial \boldsymbol{w}_t}$. Here α denotes the updating rate and l is the average cross-entropy loss of sampled instances, defined as follows:

$$l = \frac{1}{k} \sum_{i=1}^{k} \log p(y_i = 1 | \boldsymbol{x}_i; \boldsymbol{w}_t). \tag{3.8}$$

It should be noted that although the neural networks may need multiple iterations to finish training, the computational cost is much less than training from a raw model. [121] has shown that biased label is able to provide better trade-offs on hotspot detection problem during the fine-tune procedure. However, by our observation, stepped bias significantly disturbs the pre-trained model. We therefore improves this technique by letting the bias change linearly along with the training step.

Algorithm 3.1 presents the details of the layout pattern sampling flow. The algorithm requires an initial training set $\mathcal{L} = \mathcal{L}_0$ with labeled patterns, an unlabeled pattern pool $\mathcal{U} = \mathcal{U}_0$, number of patterns to be queried n and a standard deviation σ used to initialize the machine learning models (lines 1–2); we first train an initial

Algorithm 3.1 Batch Active Sampling

```
Require: $\mathcal{L}_0, \mathcal{U}_0, n, \sigma$.
Ensure: $\boldsymbol{w}, \mathcal{D}$.
 1: Initialize $\boldsymbol{w} \sim \mathcal{N}(0, \sigma)$;
 2: $\mathcal{L} \leftarrow \mathcal{L}_0, \mathcal{U} \leftarrow \mathcal{U}_0, \mathcal{D} \leftarrow \emptyset$;
 3: $\boldsymbol{w} \leftarrow$Train the machine learning model based on $\mathcal{L}$.
 4: while $\mathcal{U} \neq \emptyset$ do
 5:     $\hat{\mathcal{U}} \leftarrow$ Sample $n$ instances with highest probability (predicted with current $\boldsymbol{w}$)
    being hotspot from $\mathcal{U}$;
 6:     $\mathcal{U} \leftarrow \mathcal{U} \backslash \hat{\mathcal{U}}$;
 7:     $\mathcal{B} \leftarrow$Select $k$ instances by solving Problem (3.7);
 8:     $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{B}$;
 9:     $\mathcal{D} \leftarrow \hat{\mathcal{U}} \backslash \mathcal{B} \cup \mathcal{D}$;
10:     $\boldsymbol{w} \leftarrow$Update machine learning model based on $\mathcal{L}$;
11: end while
12: return $\boldsymbol{w}, \mathcal{D}$.
```

machine learning model based on $\mathcal{L}_0$ (line 3). In each sampling iteration, we fetch n instances from $\mathcal{U}$ without replacement and form a query set $\hat{\mathcal{U}}$ (lines 5–6); k instances are sampled into a set $\mathcal{B}$ by solving Problem (3.7) (line 7); then k instances will be added up to the training set and rest $n - k$ instances will be dropped and accordingly, new training set $\mathcal{L}$, discarded set $\mathcal{D}$ and the machine learning model are updated (lines 7–9). The algorithm ends when the unlabeled instance pool is empty and returns the trained model and remaining unlabeled patterns to be verified by the machine learning model.

Initial Training Set Generation

Note that Algorithm 3.1 requires an initial labeled dataset $\mathcal{L}_0$ to obtain a pre-trained model that will be used to extract features for future layout patterns. Thus, $\mathcal{L}_0$ is critical on the performance of the whole flow. Because it is almost impossible to know which pattern is more likely to have defects at beginning and hotspots are fetal but rare in layout spaces, we select the initial training set through analyzing the distribution of the unlabeled dataset, assuming that hotspots occur with lowest posterior probabilities.

Details of initial training set generation can be found in Algorithm 3.2 that takes the unlabeled data set $\mathcal{U}$, the number of principle components n_c and the number of initial sampled instances n_i as inputs. We first convert layout patterns into frequency domain via feature tensor extraction [121] (line 1) followed by one step principle component analysis for further dimensionality reduction (line 2). A general Gaussian model is established by calculating the mean and the covariance (line 3). We then calculate the posterior probabilities of the unlabeled data set based on the estimated mean and covariance (lines 4–6). Finally we sample n_i instances with smallest posterior probabilities to form the initial training set (lines 7–9).

Algorithm 3.2 Initial Sampling

```
Require: $\mathcal{U}, n_c, n, n_i$.
Ensure: $\mathcal{L}_0$,$\mathcal{U}_0$.
1: $\mathcal{D} \leftarrow$FeatureTensorExtraction($\mathcal{U}$);
2: $\mathcal{F} \leftarrow$PCA($\mathcal{D}, n_c$);
3: $\boldsymbol{\mu} \leftarrow \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{f_i}$,$\Sigma \leftarrow$ cov($\mathcal{F}$);
4: for $f_i \in \mathcal{F}, i = 1, 2, ..., n$ do
5:     $p_i = P(f_i; \boldsymbol{\mu}, \Sigma)$;
6: end for
7: $\mathcal{I}_0 \leftarrow$Get the indices of $n_i$ patterns with smallest posterior probabilities;
8: $\mathcal{L}_0 \leftarrow \mathcal{U}_{\mathcal{I}_0}$,$\mathcal{U}_0 \leftarrow \mathcal{U} \backslash \mathcal{L}_0$;
9: return $\mathcal{L}_0, \mathcal{U}_0$.
```

Algorithm Analysis

In this section, we will discuss and analyze some technique details of our proposed framework. As described in previous section, we relax the integer constraints when solving the sampling problem Equation (3.6). Because each queried instance will be sampled or dropped by solving the problem in Equation (3.7), the entries of the optimal solution will be rounded into binary values. Here we will analyze the loss of optimality of problem Equation (3.7) when reconstructing an integer solution as the sampling choice, as claimed in Theorem 3.1.

Theorem 3.1. *Let $\boldsymbol{m}$ be the optimal solution of Problem* (3.7) *that is binarized into $\boldsymbol{m}_b$ by setting k largest entries to 1 and rest $n-k$ entries to 0, then*

$$f(\boldsymbol{m}) \leq f(\boldsymbol{m}_b) \leq 2f(\boldsymbol{m}) + 2\lambda_n(k - \frac{k^2}{n}), \tag{3.9}$$

where $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{x}$, n is total number of instances in each query iteration, k is the number of instances that will be sampled into training set and $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of $\boldsymbol{D}$.

Proof. $f(\boldsymbol{m}) \leq f(\boldsymbol{m}_b)$ is trivial, and we will show that $f(\boldsymbol{m}_b) \leq 2f(\boldsymbol{m}) + 2\lambda_n(k - \frac{k^2}{n})$. According to Equation (3.5), the distance matrix $\boldsymbol{D}$ can be written as $\boldsymbol{D} = \boldsymbol{F}^\top \boldsymbol{F}$, where each column of $\boldsymbol{F}$ is the feature vector of each queried instance, thus $\boldsymbol{D} \succeq \boldsymbol{0}$ and f is convex. By definition,

$$\frac{1}{2}f(\boldsymbol{m}) + \frac{1}{2}f(\boldsymbol{m}_b - \boldsymbol{m}) \geq f(\frac{1}{2}\boldsymbol{m}_b), \tag{3.10}$$

i.e.

$$\frac{1}{2}\boldsymbol{m}_b^\top \boldsymbol{D}\boldsymbol{m}_b - \boldsymbol{m}^\top \boldsymbol{D}\boldsymbol{m} \leq (\boldsymbol{m}_b - \boldsymbol{m})^\top \boldsymbol{D}(\boldsymbol{m}_b - \boldsymbol{m}). \tag{3.11}$$

By Rayleigh-Ritz theorem [36],

$$(\boldsymbol{m}_b - \boldsymbol{m})^\top \boldsymbol{D}(\boldsymbol{m}_b - \boldsymbol{m}) \leq \lambda_n ||\boldsymbol{m}_b - \boldsymbol{m}||_2^2. \tag{3.12}$$

Claim that

$$\begin{aligned} ||\boldsymbol{m}_b - \boldsymbol{m}||_2^2 &\le \max_{\boldsymbol{y}\in\mathcal{T}} ||\boldsymbol{m}_b - \boldsymbol{y}||_2^2 \\ &= \max_{\boldsymbol{y}\in\mathcal{T}} \min_{\boldsymbol{x}\in\mathcal{T}_b, \boldsymbol{y}\in\mathcal{T}} ||\boldsymbol{x} - \boldsymbol{y}||_2^2, \end{aligned} \tag{3.13}$$

where $\mathcal{T} = \{\boldsymbol{x} \in \mathbb{R}^n | \sum_1^n x_i = k, x_i \in [0,1], \forall i\}$ and $\mathcal{T}_b = \{\boldsymbol{x} \in \mathbb{R}^n | \sum_1^n x_i = k, x_i \in \{0,1\}, \forall i\}$. Without loss of generality, we assume all the entries of a given $\boldsymbol{y}$ are placed in an order

$$1 \ge y_{\delta_1} \ge y_{\delta_2} \ge \cdots \ge y_{\delta_n} \ge 0, \tag{3.14}$$

thus according to the rounding strategy, $\boldsymbol{m}_b$ is defined as follows,

$$m_{b,i} = \begin{cases} 1, & \forall i \in \{\delta_1, \delta_2, \ldots, \delta_k\}, \\ 0, & \text{otherwise}. \end{cases} \tag{3.15}$$

Then as claimed in Equation (3.13):

$$\begin{aligned} ||\boldsymbol{m}_b - \boldsymbol{y}||_2^2 &= \sum_{i=1}^{k} (1 - y_{\delta_i})^2 + \sum_{i=k+1}^{n} y_{\delta_i}^2 \\ &= k + \sum_{i=1}^{n} y_{\delta_i}^2 - 2\sum_{i=1}^{k} y_{\delta_i} \\ &\le k + \sum_{i=1}^{n} y_{\delta_i}^2 - 2\sum_{i=1}^{k} y_{\eta_i} \\ &= ||\boldsymbol{x} - \boldsymbol{y}||_2^2, \forall \boldsymbol{x} \in \mathcal{T}_b, \boldsymbol{y} \in \mathcal{T}, \end{aligned} \tag{3.16}$$

Consider a right pyramids with a regular base, which has its apex at the origin, C_n^k edges defined by the vectors defined in $\mathcal{T}_b$ and a regular polygon base A lies in the hyperplane $\sum_{i=1}^n x_i = k$. As shown in Figure 3.4, $||\boldsymbol{m}_b - \boldsymbol{m}||_2^2$ reaches its maximum

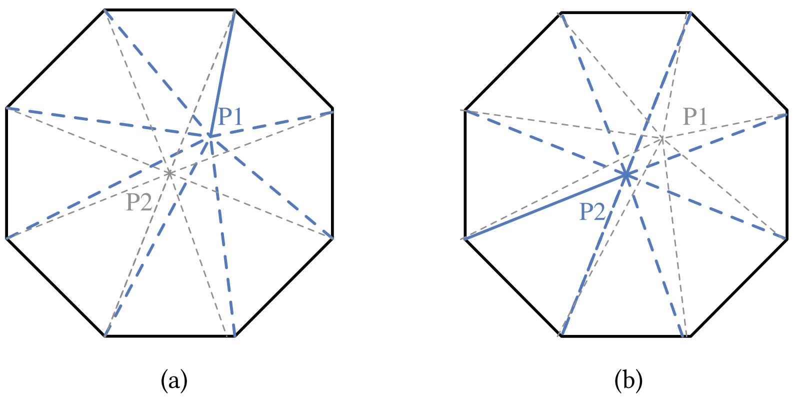


Figure 3.4: Geometric view of $\boldsymbol{m}_b - \boldsymbol{m}$. P_1 and P_2 denote the end points of $\boldsymbol{m}$, and in particular, P_2 lies in the center of the base polygon. The solid segments in each figure represents $||\boldsymbol{m}_b - \boldsymbol{m}||_2^2$ for a given $\boldsymbol{m}$.

value when $\boldsymbol{m}$ lies in the center of the pyramid base. That is $\boldsymbol{m} = [\frac{k}{n}, \frac{k}{n}, \dots, \frac{k}{n}]^\top$, and

$$\begin{aligned}\max_{\boldsymbol{m}_b, \boldsymbol{m}} ||\boldsymbol{m}_b - \boldsymbol{m}||_2^2 &= k(1 - \frac{k}{n})^2 + (n-k)(\frac{k}{n})^2 \\ &= k - \frac{k^2}{n},\end{aligned} \tag{3.17}$$

which, combined with Equation (3.12), justifies the theorem. □

Theorem 3.1 provides a theoretical guidance on choosing proper n and k in the batch sampling procedure, which can also be intuitively explained by the fact that if we sampling all or one instances in each querying iteration, we have no risk on the integer relaxation error, however, at the cost of diversity loss.

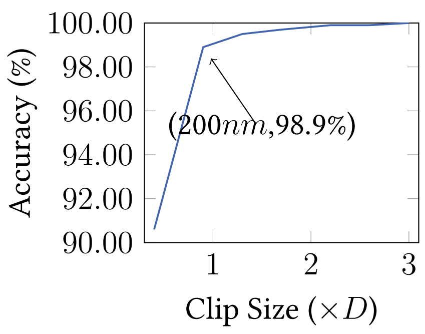
3.2.3 Experiments

Our layout pattern sampling and hotspot detection flow is tested on two industrial benchmark sets: ICCAD12 [101] and ICCAD16 [100].

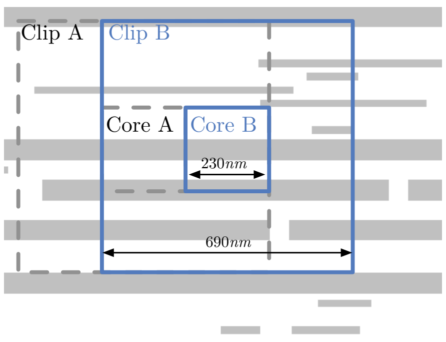
Table 3.1 lists the benchmark details. To verify the efficiency of our proposed method on EUV oriented designs, we shrink ICCAD16 layouts to reach a CD under $7nm$ technology node as indicated in the column "CD (nm)". Columns "HS #" and "NHS #" are numbers of hotspot and non-hotspot clips in each benchmark and "Tech (nm)" is the technology nodes of each design. ICCAD12 contains five benchmark sets from [101] with labels which can be directly input to our flow. To verify the scalability of our algorithm, we also merge all $28nm$ designs from ICCAD12 into a much larger dataset ICCAD12-28 with 3728 hotspot patterns and 0.15M non-hotspot patterns. Very recently, Reddy *et al.* [80] find that the designs from ICCAD12 may lack truly never seen before and hard to classify patterns, and hence the dataset is less effective as an evaluation of machine learning models. Derived from designs in ICCAD12, they synthesized a new challenging benchmark suit that contains both $28nm$ and $32nm$ designs with $\sim$40% hotspot patterns. Statistics of the new benchmark suit are shown as ICCAD19 in Table 3.1. ICCAD16 contains four layouts that are original designed for fuzzy matching tasks. To locate defects in those layouts, we apply ASML Tachyon optical proximity correction (OPC) and layout manufacturability checker (LMC) tools on scaled layouts using EUV lithography models for $7nm$

Table 3.1: Benchmark Details

Benchmarks	CD (nm)	HS #	NHS #	Tech (nm)
ICCAD12-1	45	325	5019	32
ICCAD12-2	45	672	46583	28
ICCAD12-3	45	2717	50976	28
ICCAD12-4	45	272	36342	28
ICCAD12-5	45	67	22043	28
ICCAD12-28	45	3728	155944	28
ICCAD19	-	65778	163680	28/32
ICCAD16-1	16	0	63	7
ICCAD16-2	16	56	967	7
ICCAD16-3	16	1100	3916	7
ICCAD16-4	16	157	1678	7



(a) Influence of clip size



(b) Clip-based scan

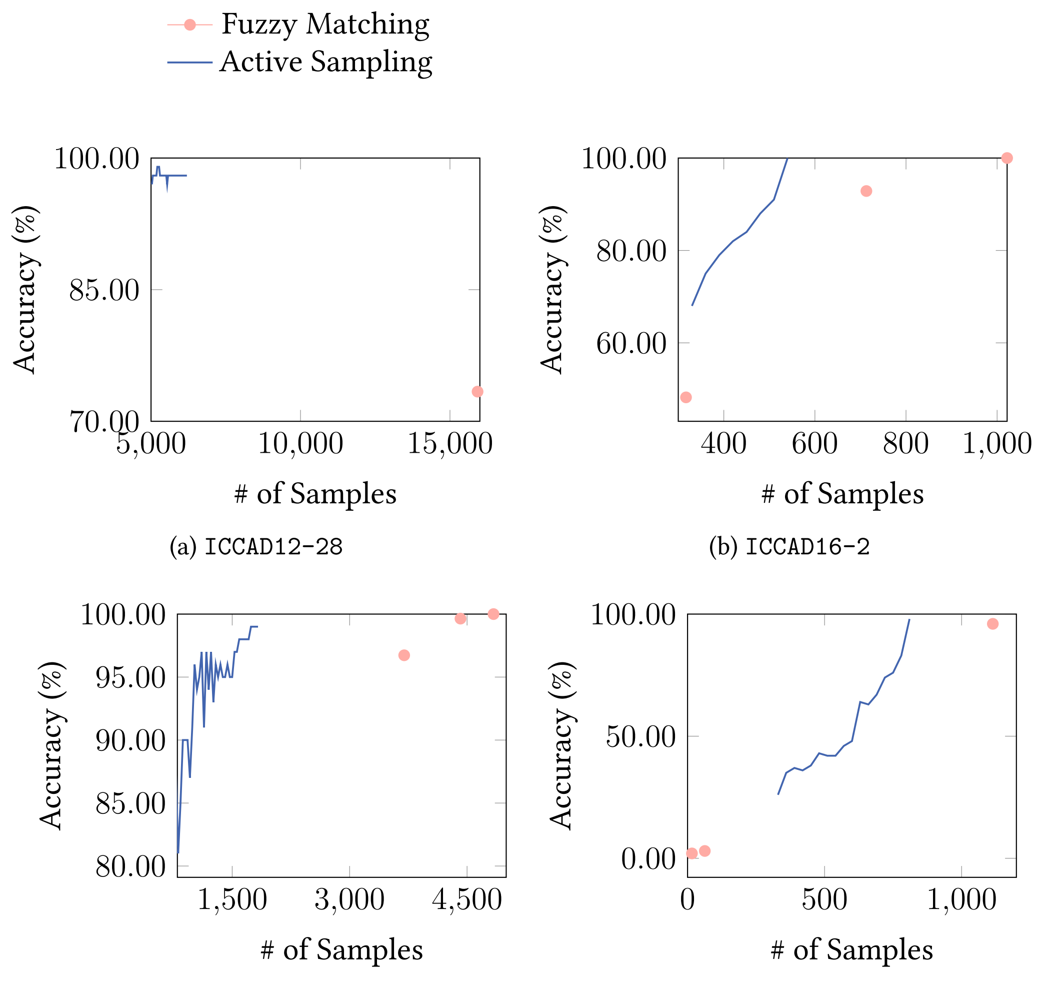
Figure 3.5: Dispatching layouts based on estimated D. Because there is no spacing and overlapping between adjacent core regions of adjacent clips, each layout is fully scanned in the sampling and detection flow. Particularly, exact matching has a detection rate of 98.9% with the clip size in the original contest setting.

metal layer. In the LMC stage, we only consider three types of defects that are edge placement error, bridge and neck which contribute most to circuit failures. Then all the locations where edge placement error, bridging and necking occur are marked as defects. To perform efficient and parallel testing, clip-based scan is usually applied in classic hotspot detection flow, where the clip size and scanning stride are empirically determined according to the optical diameter (D=230nm) under given lithography specifications. Figure 3.5(a) shows that fail detected hotspot count of exact pattern matching reduces to zero as clip size increases to around $3 \times D = 690nm$ that will be chosen as the clip size in our experiment. It should also be noted that, original `ICCAD16` contains predefined marker layer covering a fraction of the layouts. To conduct full chip detection, we manually replaced the original marker layer with a new layer that contains uniformly distributed markers that cover whole layouts.

According to the estimated D of $7nm$ EUV lithography system, we adopt an overlapped dispatching method that covers the whole layout with reasonably small clip size that contains enough information to determine whether the center core region is hotspot or not. Figure 3.5(b) illustrates the details of the dispatching procedure. We use a $690nm \times 690nm$ sliding window to scan the whole layout with scanning stride being $\frac{1}{3}$ of the clip size, which ensures that center 230×230 core regions of each clips are exactly covering the whole chip. Note that to ensure that clips contain more than 96% information to estimate the printability of their core region, the smallest distance from the core boundary to the clip boundary is intentionally selected as $230nm$. Furthermore, each clip will be marked as hotspot clip if defects occur at its core region as shown in Figure 3.5(b). Statistics of `ICCAD16` benchmarks are also listed in columns "HS #" and "NHS #". We can notice that the smallest layout `ICCAD16-1` is defect-free, therefore the case `ICCAD16-1` is ignored in following experiments. For the rest of `ICCAD16` cases, `ICCAD16-2` has 56 hotspots out of 1023 clips, `ICCAD16-3` has 1100 hotspots out of 5016 clips and `ICCAD16-4` has 157 hotspots out of 1835 clips. It should be noted that although layout `ICCAD16-4` is much larger than other cases. However, it is much more regular and a large fraction of the patterns violate EUV direct print metal layer design. We therefore only extract clips from DRC-clean regions and that is why the total clip count is less than `ICCAD16-3`. We can also see the out of expected behaviors on `ICCAD16-4` in the experiments in the following sections.

To accommodate the shallow neural networks and the computational requirements, we conduct feature tensor extraction [121] on each clips in all benchmark cases that convert layout images into reduced frequency domain.

In the first experiment, we will compare the batch active sampling method with fuzzy matching under different area constraints. The procedures of Algorithm 3.1 on four benchmark sets are depicted in Figure 3.6, where the x-axis represents the total number of patterns sampled into training set and the y-axis denotes the detection accuracy. According to the analysis, we avoid choosing the $\frac{k}{n}$ that results in big rounding error (i.e. 0.5). Considering that sample count also affects the training performance and the lithography simulation overhead, we pick $k = 30, n = 90$ in all benchmarks. On the other hand, because `ICCAD12/19` are much larger benchmark



(a) `ICCAD12-28` (b) `ICCAD16-2`

(c) `ICCAD16-3` (d) `ICCAD16-4`

Figure 3.6: Learning model performance v.s. sampling count. The blue curve is the reference performance obtained from fuzzy matching with different area constrains reflected as different sampling count. The red curve shows the sampling results based on Algorithm 3.1.

Table 3.2: Full chip pattern sampling and hotspot detection on ICCAD12 benchmarks.

Benchmarks	PM-exact [14]		PM-a95^{12} [14]		PM-a90^{12} [14]		PM-e2^{2} [14]		FT [91]		Greedy [130]		Ours	
	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#
ICCAD12-1	100.0	3839	93.17	732	81.66	398	100.0	3828	68.31	1226	99.08	1140	99.70	1436
ICCAD12-2	100.0	31812	99.54	13091	93.84	5720	100.0	32601	5.95	3024	2.36	1036	97.42	2648
ICCAD12-3	100.0	31513	98.54	20904	91.79	9335	100.0	30769	93.95	9244	96.22	10579	96.55	4484
ICCAD12-4	100.0	29438	90.46	4527	62.00	1411	100.0	28067	8.82	5511	82.35	2723	95.64	6842
ICCAD12-5	100.0	14988	91.20	3045	84.91	1204	100.0	14849	8.15	1227	91.30	1486	94.12	1227
ICCAD12-28	100.0	127746	96.83	38879	73.38	15923	100.0	124320	32.14	20000	24.57	26945	96.99	8210
ICCAD19	99.93	157238	-	-	-	-	-	-	63.66	65168	82.01	86917	96.30	69743
Average	99.99	56653	-	-	-	-	-	-	29.38	14986	68.27	18689	96.67	13512
Ratio	1.034	4.193	-	-	-	-	-	-	0.304	1.109	0.768	1.020	1.000	1.000

1 Experiments on all ICCAD12 cases are conducted on the center 600×600 region of each clip because the area constrained fuzzy matching cannot be finished within one week using original clip size of 1200×1200.
2 Experiments on ICCAD19 case cannot finish with given time limit even on the center 600×600 region of each clip.

Table 3.3: Full chip pattern sampling and hotspot detection on ICCAD16 benchmarks.

Benchmarks	PM-exact [14]		PM-a95 [14]		PM-a90 [14]		PM-e2 [14]		FT [91]		Greedy [130]		Ours	
	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#	Acc (%)	Litho#
ICCAD16-2	100.0	1022	92.86	717	48.21	328	100.0	1022	91.07	782	51.79	475	100.0	881
ICCAD16-3	100.0	4838	99.64	4420	96.73	3717	99.91	4777	86.18	1854	73.82	2496	99.74	3589
ICCAD16-4	95.54	1134	2.55	65	1.91	20	78.34	842	50.32	573	50.32	668	98.09	1608
Average	98.51	2325	65.01	1730	48.95	1343	92.75	2199	77.13	983	58.64	1213	99.27	2026
Ratio	0.992	1.147	0.655	0.854	0.493	0.663	0.934	1.085	0.777	0.485	0.591	0.599	1.000	1.000

Table 3.4: Result comparison with conventional active learning solutions on ICCAD12 benchmarks.

Benchmarks	US [59]		EMC [9]		Ours	
	Acc (%)	Litho	Acc (%)	Litho	Acc (%)	Litho
ICCAD12-1	98.46	788	98.15	1542	99.70	1436
ICCAD12-2	2.340	1601	2.330	2812	97.42	2648
ICCAD12-3	85.50	2938	75.12	3229	96.55	4484
ICCAD12-4	73.72	2913	90.28	4798	95.64	6842
ICCAD12-5	76.47	908	81.71	2325	94.12	1227
ICCAD12-28	95.92	8642	96.33	11010	96.99	8210
ICCAD19	74.24	32958	69.14	30299	96.30	69743
Average	71.54	7296	72.93	8055	96.67	13512
Ratio	0.684	0.540	0.754	0.596	1.000	1.000

sets that contain more than $150,000$ clips, early stopping is applied in the batch active sampling procedure, where we pick the maximum sampling number to be 5% of total instance number in each dataset.

The discrete dots in Figure 3.6 correspond to fuzzy matching results with area constraints 90%, 95% and 100%, respectively. It can be seen that our batch sampling converges at a reasonably high detection accuracy on both DUV and EUV specific layers while requiring much less training instances than exact pattern matching. In other words, our proposed method can significantly reduce lithography simulation overhead. Particularly for the case ICCAD12, exact matching samples more than 10^5 clips among the whole data set, while our method achieves similar results with only 6200 clips. Because our framework adopts CNN as the learning engine, uncertainty behavior will be introduced by weight initialization and batching sampling. However, with the help of good weight initialization, reasonable sampling ratio in diversity-aware sampling (see Section 3.2.2) and dynamic learning rate, we are able to attain a ± 5 hits variation cross multiple runs.

Table 3.5: Result comparison with conventional active learning solutions on ICCAD16 benchmarks.

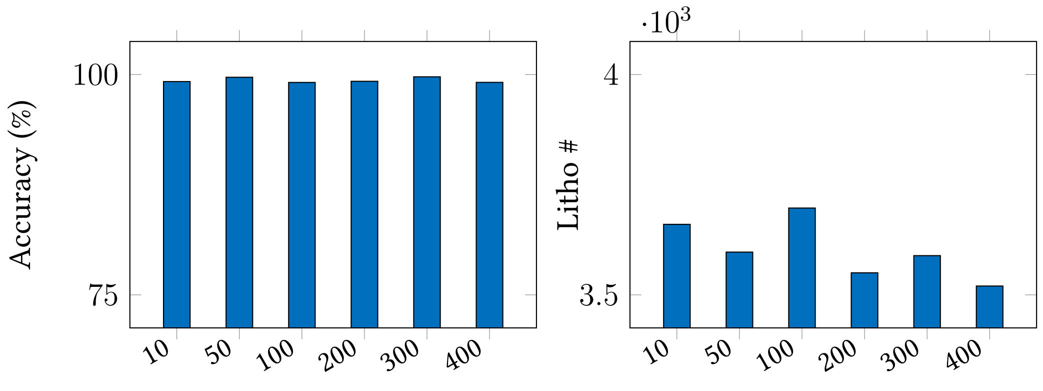
Benchmarks	US [59]		EMC [9]		Ours	
	Acc (%)	Litho	Acc (%)	Litho	Acc (%)	Litho
ICCAD16-2	71.43	543	73.21	546	100.0	881
ICCAD16-3	84.60	1939	87.30	1958	99.74	3589
ICCAD16-4	66.03	870	57.69	875	98.08	1608
Average	74.02	1117	72.73	1126	99.27	2026
Ratio	0.746	0.551	0.733	0.556	1.000	1.000

We compare the sampling results on `ICCAD12/16/19` with exact/fuzzy matching methods and two recent sampling methods, as listed in Table 3.2 and Table 3.3. Columns “PM-exact”, “PM-a95”, “PM-a90”, “PM-e2” correspond to the results derived from pattern matching using a state-of-the-art pattern analysis tool [14], where “PM-exact” denotes only exactly same patterns can be clustered together, “PM-a95” and “PM-a90” refer to any clips that satisfy 95% and 90% area constraints are clustered together and “PM-e2” groups clips with less than $2nm$ edge displacements. Here the area and edge constraints are defined following [100]. Column “FT” lists the result of clustering on frequency domain of layout patterns that is similar to the flow proposed in [91]. Column “Litho#” denotes the number of clips being labeled according to the lithography simulation results, including the clips sampled into training sets and all the detection extras.

“PM-exact”, as the reference method, shows 100% accuracy on all test cases except for `ICCAD16-4`. According to the lithography simulation results of the layout in `ICCAD16-4`, we notice all defects appear at the patterns belong to a different design space, which possibly makes the lithography model and optical proximity analysis inaccurate. Therefore, we observe minor prediction error and extra on `ICCAD16-4`. The result also shows exact pattern matching can achieve extremely high verification accuracy, however, at the cost of simulating and labeling a large fraction clip patterns in the whole dataset. On the contrary, the proposed batch sampling method achieves similar detection accuracy querying only 7% of total layout clips for `ICCAD-12` and 300 less clips on average for `ICCAD-16` compared to the best baseline "PM-exact". In particular, our method shows even higher detection accuracy on `ICCAD16-4` than exact pattern matching because of the effective gaussian initial sampling, which also demonstrates our assumption holds in Algorithm 3.2. It should be noted that it is normal that the instances in a training set is not completely separable, which explains why our method behaves even better than exact pattern matching. We can also observe that our algorithm behaves much better on `ICCAD12` than `ICCAD16`, which can be explained by the fact that `ICCAD12` contains more than $10\times$ sample candidates that fits our algorithm better.

For three fuzzy matching options, varies area or edge constraints offers different level of trade-offs between verification performance and lithography overhead. "PM-a95" and "PM-e2" can still maintain good prediction accuracy on `ICCAD16-2` and `ICCAD16-3` with slightly less litho count, but the total number of labeled instances is still much larger than our method. Moreover, fuzzy matching fails to extract problematic instances on a more difficult testcase `ICCAD16-4` with looser constraints that they all reach less than 50% prediction accuracy. Experiments on `ICCAD19` also manifest the scalability issue of fuzzy matching solutions. As can be seen, area or edge displacement constrained fuzzy matching cannot be done within days when the design pool is extremely large and with complicated patterns. It should be noted that we did not further narrow down the matching window, because matching on regions much smaller than the estimated optical diameter does not make sense for hotspot detection purpose.

[91] proposes to use frequency domain representation to sample layout patterns with similar property and detect hotspots. Here we conduct additional experiments



(a) Accuracy (%) (b) Litho #

Figure 3.7: Influence of initial sampling size.

by clustering layout clips based on their Fourier Transform results. Clips closest to a cluster center will be selected as the representative clip that indicates the property of the whole cluster. By the results in the column "FT", we can observe that with similar sampling number, batch active sampling exhibits much better than frequency domain clustering. We also conduct an experiment using greedy sampling method [130] where each instance being predicted as hotspot will be incrementally added into the training set. Although greedy sampling method can successfully select partial hotspot clips in some test cases, the performance is highly affected by the initial learning model, where prediction error will be gradually amplified in the greedy sampling procedure. As listed in "Greedy", the greedy method [130] only achieves $\sim$60% average detection accuracy on ICCAD16, $\sim$74% average detection accuracy on ICCAD12, and $\sim$80% on ICCAD19.

To show that the proposed sampling solution is efficiently and carefully designed for layout printabilility estimation purpose, we also compare the pattern sampling and hotspot detection results with two popular active learning solutions US [59] and EMC [9]. Both algorithms are implemented and integrated into our deep learning framework. As shown in Table 3.4 and Table 3.5, our method exhibits obvious ad-

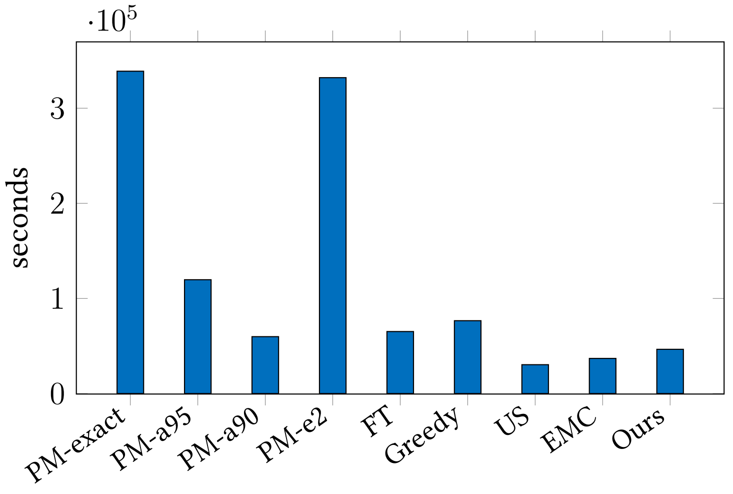


Figure 3.8: Runtime comparison among different solutions. “PM-xx”s are conducted on 10-core Intel E7-4830v2 with 512GB memory. “FT”, “Greedy”, “US”, “EMC” and “Ours” are deep learning based flows that are tested on a GPU platform with one GeForce GTX 1080Ti, one Intel i9-7900X and 64GB memory.

vantage on detection accuracy on `ICCAD12`, `ICCAD16` and `ICCAD19` cases over US and EMC, and the weak accuracy behavior makes the lithography overhead advantage of US and EMC trivial. We can also observe that the performance gap is narrowed on the `ICCAD12`. It can be explained by that the `ICCAD12` benchmark is not a full chip design and are delivered with clips that are diverse in hotspot and non-hotspot patterns. Thus, most sampling methods will work and the deep learning model dominates the performance. Different trade-offs of US and EMC on `ICCAD12` is a reflection of sampling mechanisms in two methods. Although US and EMC behave reasonably good on `ICCAD12-28`, their drawbacks can be easily seen when patterns are becoming complicated, which is consistent with the results on `ICCAD19` (72.4% of US and 69.14% of EMC compared to 96.3% of our approach). We did not implement QBC [29] in this work, because QBC requires ensemble learning engines which is not directly compatible with our deep learning-based framework.

Intuitively, the final prediction results are to be also sensitive to the number of instances sampled in the initial sampling stage. In this experiment, we study the influence of different initial sampling sizes. Here we use ICCAD16-3 as an example and conduct our PSHD flow with different initial sampling size, as shown in Figure 3.7. It can be seen that initial sampling size does not cause much variations in prediction accuracy due to our diversity-aware sampling strategy. The only difference comes from lithography overhead that a reasonably large initial sampling size will efficiently reduce the lithography overhead.

In real backend layout verification flow, almost all the computation and runtime overhead come from lithography simulation. Here we assume a $10s$ penalty on each litho-clip in our framework as in [130]. The overall runtime is then evaluated by the summation of simulation penalty and PSHD overhead. As shown in Figure 3.8, the proposed framework is much more efficient than existing solutions without any performance degradation.

3.2.4 Summary

A layout pattern sampling and hotspot detection flow is proposed to adaptively sample layout patterns into a pattern library that is used to train a machine learning model for layout hotspot detection. The diversity-aware batch sampling and the interactive optimization of learning model can efficiently select interesting patterns and ensure a better model generality. Experiments show that the proposed framework is able to achieve similar detection accuracy requiring much smaller number of labeled patterns, which reduces lithography simulation overhead by a significant amount.

Chapter 4

Intelligent Pattern Generation

4.1 Introduction

VLSI layout patterns provide critic resources in various design for manufacturability (DFM) researches, from (1) early technology node development to (2) back-end design and sign-off flows [97]. The former includes perfection of design rules [108], OPC recipes [97], lithography models [62] and so on, which require design patterns. The latter covers, but is not limited to, layout hotspot detection and correction [117, 53, 120, 93, 119, 32, 115], linking to both design and mask patterns.

Design patterns usually come from long logic-to-chip design cycle that results in shortage of pattern libraries for DFM researches/solutions. 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 [114, 62]. One state-of-the-art industrial pattern generation solution is migration from older technology node designs [133], where seed patterns are selected from the process weak points in older designs and shrunk by a scale factor to match current design rules. However, such flow is no longer applicable when there are large gaps between design nodes. An example is that $7nm$ EUV designs contain all unidirectional shapes while patterns are still 2D in previous design nodes [10]. Monte Carlo shape generation is another approach that is typically applied industry-wide to generate metal layer patterns for lithography and OPC research. In this method, shapes are randomly created and placed on a given region according to certain geometry constraints, which, to some extent, limits the pattern library diversity created with the flow [111].

Mask patterns are generated with mask optimization flows, aiming at compensating lithography proximity effects when transferring design patterns on silicon wafers. Mask optimization methodologies include model-based techniques [5, 57, 96, 125, 76, 126, 96, 6] and inverse lithography-based technique (ILT) [77, 30, 67, 107]. In model-based OPC flows, pattern edges are fractured into segments which are then shifted/corrected according to mathematical models. A high printability mask can then be obtained with sub-resolution assist features (SRAF) [102]. Awad *et al.* [5] propose a pattern fidelity aware mask optimization algorithm that optimizes core polygons by simultaneously shifting adjacent segmentation. Su *et al.* [96] significantly accelerate the OPC flow by extracting representative process corners while maintaining a good wafer image quality. However, model-based OPC flows are highly restricted by their solution space and hence lacking in reliability for complicated designs. On the other hand, ILTs minimize the error between the wafer image and the target with lithography constraints. Because ILTs conduct pixel-based optimization on layout masks, they are expected to offer better lithography contour quality, which although comes with additional challenges on mask manufacturability problems including manufacturing cost and mask design rules. Recently, Ma *et al.* [67] adopt ILT to simultaneously perform mask optimization and layout decomposition that brings a better solution of multiple patterning mask design. Although the model-based method and the ILT-based method behave well on a variety of designs, they take the wafer image as a mask update criterion in each iteration of the OPC process. In other works, multiple rounds of lithography simulation are indispensable in the

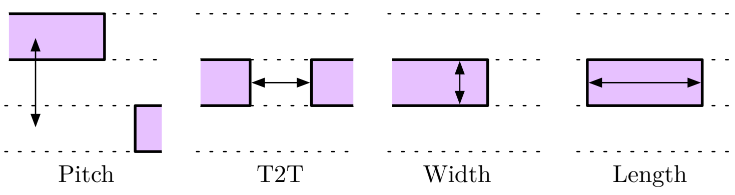


Figure 4.1: Layout geometry and critical dimensions.

optimization flow which is drastically time consuming.

Generative machine learning models have been widely studied on computer vision related tasks like image synthesis [38, 64] and scene transformation [132, 74] by extracting and understanding feature distributions. In EDA area, such architecture and its variations have also been successfully applied in lithography simulation [123], mask optimization [113, 33, 2] and on-chip sensor placement [63]. In this chapter, we will introduce two series of generative machine learning models for pattern generation: the TCAE family for design pattern generation and the GAN-OPC family for mask optimization.

4.2 Test Layout Generation

4.2.1 Preliminaries and Problem Formulation

In this section, we introduce some geometry concepts to accommodate layout design rules and the pattern generation flow. Because in this paper we focus on $7nm$ EUV metal layer designs which contains only unidirectional on-track shapes, following terms are accordingly adopted to quantize pattern shapes and positions, as depicted in Figure 4.1. *Pitch*, denoted as p, measures the distance between two adjacent tracks that contain shapes. *T2T*, denoted as t, measures the line-end-to-line-end distance between two adjacent shapes in a track. Wire *length* l and *width* w measure the shape size along and against the design track, respectively. In order for a pattern to be DRC-clean, these measurements will be constrained by some design rules.

All shape edges in a fixed-size window are aligned with x-axis and y-axis. If we extend all horizontal and vertical edges infinitely into scan lines, more non-overlapping scan lines always come with more complex patterns. We hence define the complexity of a layout pattern as follows.

Definition 4.1 (Pattern Complexity). *The complexity of a pattern in x and y directions (denoted as c_x and c_y) are defined as the number of scan lines subtracted by one along x-axis and y-axis, respectively.*

We also introduce the concept of *pattern diversity* (denoted as H) to measure how are the pattern complexities distributed in a given library. A larger H implies the library contains patterns that are more evenly distributed, as in the following definition.

Definition 4.2 (Pattern Diversity). *The diversity of a pattern library is given by the Shannon Entropy [89] of the pattern complexity sampled from the library, as shown in Equation* (4.1),

$$H = -\sum_i \sum_j P(c_{xi}, c_{yj}) \log P(c_{xi}, c_{yj}), \tag{4.1}$$

where $P(c_{xi}, c_{yj})$ is the probability of a pattern sampled from the library has complexities of c_{xi} and c_{yj} in x and y directions respectively.

With above definitions, the pattern generation problem can be formulated as follows.

Problem 4.1 (Pattern Generation). *Given a set of layout design rules, the objective of pattern generation is to generate a pattern library such that the pattern diversity and the number of unique DRC-clean patterns in the library is maximized.*

4.2.2 Architectures and Algorithms

Generating layout patterns is extremely challenging for learning machines as design rules are usually not friendly to most machine learning models. We therefore simplify the problem into two stages with the aid of squish patterns [34]. We first deal with the topology generation problem and then establish a linear system to finalize the pattern generation flow with proper geometry vectors.

Squish Pattern Simplifies Pattern Generation

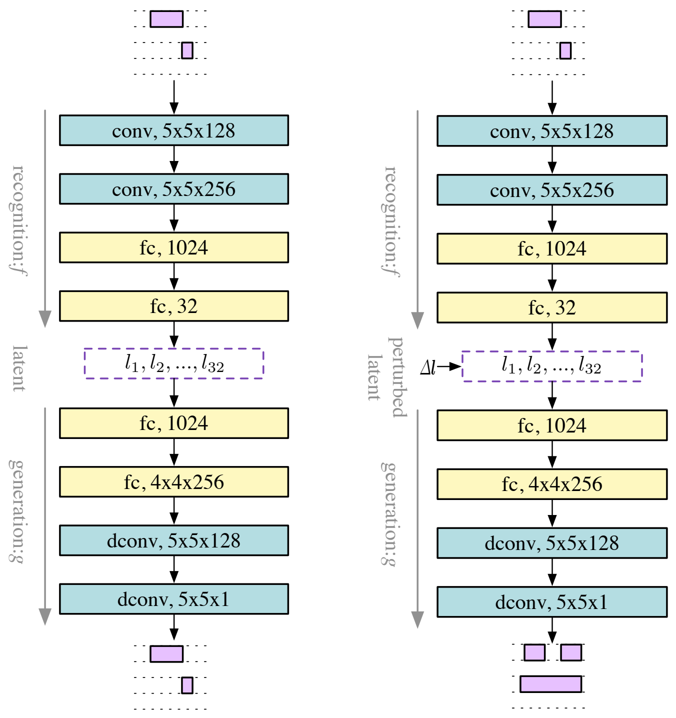
Let recall that squish pattern is a scan line-based representation that each layout clip is cut into grids aligned at all shape edges, as illustrated in Figure 2.11. From the squish pattern extraction procedure, we can also easily obtain the following lemma.

Lemma 4.1. *Squish pattern representation is lossless.*

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 supports neural networks and other machine learning models. (2) Squish patterns are naturally compatible with the simplified pattern generation flow that will be discussed in following sections.

Topology Generation

In this paper, we propose a TCAE architecture that aims at efficient pattern topology $\boldsymbol{T}$ generation. The TCAE is derived from original transforming auto-encoders (TAEs) [46] which are a group of densely connected auto-encoders and each individual, referred as a capsule, is targeting on certain image-to-image transformations. Each capsule employs a recognition unit and a generation unit to capture a pose position and re-synthesize the translated object, respectively. The translation is defined on a



(a) Training (b) Testing

Figure 4.2: Architecture of transforming convolutional auto-encoder in (a) training phase and (b) testing phase.

regular coordination system that will be added up to the original pose position before fed into the generation unit. In the training phase, neuron weights are updated by backpropagating the differences between the output image and the actual translated image given the translation information. After the neural network is trained, it takes inputs of an image and translation information and outputs the image with desired shifts. Such architecture agrees with the pattern generation tasks in the following aspects. (1) Feature instantiation attains data set domain properties. (2) All capsules contribute together to produce variations of any input objects. Although such architecture can capture the feature characteristics from original object pixel intensities, 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.*

Observe that each capsule unit functions similarly as independent receptive field in convolutional layers, we develop the TCAE architecture for feature learning and pattern reconstruction, as shown in Figure 4.2. 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 (4.2).

$$\boldsymbol{l} = f(\boldsymbol{T}; \boldsymbol{W}_f), \tag{4.2}$$

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 an 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 [25] that cast the pattern object from the latent vector space back to the original pattern space, as in Equation (4.3).

$$\boldsymbol{T}' = g(\boldsymbol{l} + \Delta\boldsymbol{l}; \boldsymbol{W}_g), \tag{4.3}$$

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.

$$\min_{\boldsymbol{W}_f, \boldsymbol{W}_g} ||\boldsymbol{T} - \boldsymbol{T}'||_2, \text{ s.t. } \Delta\boldsymbol{l} = \boldsymbol{0}. \tag{4.4}$$

The TCAE-based framework differs from TAEs in the following aspects.

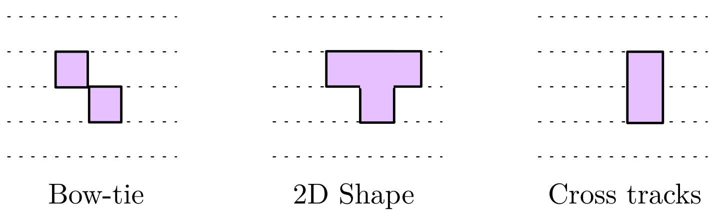


Figure 4.3: Illegal topology examples.

- We replace the group capsules with a simpler latent vector that contains feature nodes that connect to different receptive fields in the early convolutional layers and hence can represent certain part-whole feature instantiation.
- The transformation in our framework applies directly on the latent vector space that promises a much larger diversity of the generated patterns compared to the limited transformation on the coordinate system only in TAEs.
- Identity mapping in the training phase helps the TCAE capture the design rule properties of existing patterns.

Once the TCAE is trained, we can adopt the flow in Figure 4.2(b) 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 extract latent vector instantiations of existing topologies. Perturbation on the latent vector space is expected to expand the existing pattern library with legal topologies. We claim the following assumption.

Assumption 4.1. *The latent vector space offers richer information and each latent vector node represents patten geometry locally or globally, e.g., some vector nodes may control shape sizes while some others may control shape positions.*

A straightforward perturbation approach is combining existing topologies in the latent vector space, which is expected to fill certain region of a given pattern library.

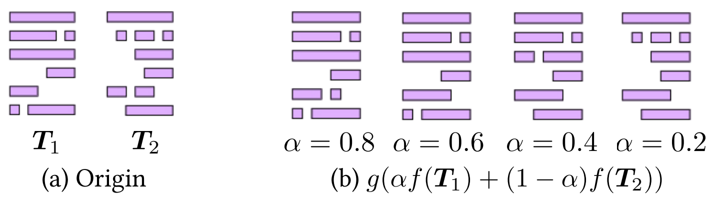


Figure 4.4: Combination of existing patterns with latent vectors.

The combination rule can be defined as Equation (4.5).

$$\boldsymbol{T}_g = g(\sum_i \alpha_i f(\boldsymbol{T}_i)), \tag{4.5}$$

where $0 < \alpha_i < 1, \forall i$ are combination coefficients and satisfy $\sum_i \alpha_i = 1$. However, the diversity topologies generated by such approach might be limited by the existing pattern complexity. We randomly pick two patterns from existing designs and combine them in the latent vector space using TCAE. As shown in Figure 4.4, even we adjust the combination coefficient with a large step, there are still repeating topologies in the reconstructed topology set.

Another approach is introducing random perturbations in the latent vectors. We take Assumption 4.1 into consideration when generating perturbation vectors to avoid illegal topologies as many as possible. We introduce the concept of feature sensitivity $\boldsymbol{s}$ that statistically defines how easily an legal topology can be transformed to illegal when manipulating the latent vector node with everything else unchanged.

Definition 4.3 (Feature Sensitivity). *Let* $\boldsymbol{l} = \begin{bmatrix} l_1 & l_2 & \ldots & l_n \end{bmatrix}^\top$ *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 4.3 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 4.1, 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 perturbation 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).

Algorithm 4.1 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 $\boldsymbol{s}$.
```

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. Because we focus on EUV metal layers in this work, a topology is illegal if and only if it contains any patterns in Figure 4.3. That is, illegal topologies can be filtered out by checking whether shapes appear at any two adjacent tracks.

Why Not Variational Auto-Encoders?

The encode-decoder architecture for data generation will possibly remind readers a very similar generative model called variational auto-encoder (VAE) [23], which shares almost the same architecture as TCAE. A VAE also consists of an encoder (recognition unit) and a decoder (reconstruction unit) which are trained with the following loss function:

$$\begin{aligned}\mathcal{L}(\boldsymbol{W}_e, \boldsymbol{W}_d) = &- \mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{W}_e}(\boldsymbol{z}|\boldsymbol{x})}(\log p_{\boldsymbol{W}_d}(\boldsymbol{x}|\boldsymbol{z})) \\ &+ \mathbb{KL}(q_{\boldsymbol{W}_e}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})),\end{aligned} \tag{4.6}$$

where the first term aims to train the neural network to generate latent vectors $\boldsymbol{z}$ that can be reconstructed into data instances following the distribution of the training dataset, and the second term makes $\boldsymbol{z}$ follow a distribution $p(\boldsymbol{z})$ via K-L Divergence. Usually, $p(\boldsymbol{z})$ is specified as normal distribution $\mathcal{N}(0, 1)$. In the inference stage, the decoder can create data instances following the training set by taking inputs of $\boldsymbol{z} \sim p(\boldsymbol{z})$. Equation (4.6) tells us that a VAE focuses on the generation of some data that follows certain (known) distribution, which shares the same beneath idead as GANs. However, in the task of test layout pattern generation, we expect generation of patterns that are not exist before, which explains why TCAEs are trained with identity mapping without any perturbation information.

Legal Pattern Assessment

To generate DRC clean patterns, we need legal $\boldsymbol{\delta}_x$s and $\boldsymbol{\delta}_y$s of all generated topologies. We first detect all critical dimensions listed in Figure 4.1 in each valid topology and then formulate a linear system combining all associated constraints, as in Formula

(4.7).

$$y_{i+1} - y_i = \frac{p}{2}, \qquad \forall i, \tag{4.7a}$$

$$x_i - x_j = t_{\text{min}}, \qquad \forall (i,j) \in \mathcal{C}_{T2T}, \tag{4.7b}$$

$$x_i - x_j = l_{\text{min}}, \qquad \forall (i,j) \in \mathcal{C}_{W}, \tag{4.7c}$$

$$x_{i+1} - x_i > 0, \qquad \forall i, \tag{4.7d}$$

$$x_{\max} - x_0 = d_x, y_{\max} - y_0 = d_y. \tag{4.7e}$$

where d_x, d_y, t_{min} and l_{min} denote clip width, clip height, minimum tip-to-tip distance and wire length (refer to Figure 4.1), respectively. Each index pair $(i,j) \in \mathcal{C}_{T2T}$ indicates that there exists at least one tip-to-tip pattern at scan lines x_i and x_j in the clip. $\mathcal{C}_W$ is defined similarly for wire patterns. x_0 and y_0 define the origin of each clip that can be any value and do not affect pattern complexities. Certain constraint values correspond to the minimum critical dimensions when no defects are found in EUV simulation under a given process window [43]. Note that the system in Equation (4.7) can be efficiently solved with vast linear programming algorithms or numerical methods. Because, as discussed previously, all shapes will occupy the entire track in y direction, pitch and wire width are both covered by fixed track width and so is $\boldsymbol{\delta}_y$. We only need to consider constraints on $\boldsymbol{\delta}_x$ track by track. With the aid of squish representation, the problems of finding line-end-to-line-end patterns (for T2T constraints) and floating wires (for in-clip wire length constraints) become finding $\underbrace{\text{100...001}}_{\text{continuous zeros}}$ and $\underbrace{\text{011...110}}_{\text{continuous ones}}$ respectively. A solution of $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$ in together with the associated topology matrix $\boldsymbol{T}$ formulates a complete squish pattern representation.

Overall Flow

We summarize the pattern generation flow as in Figure 4.5, where key steps include squish pattern extraction, topology generation and pattern generation. In the topol-

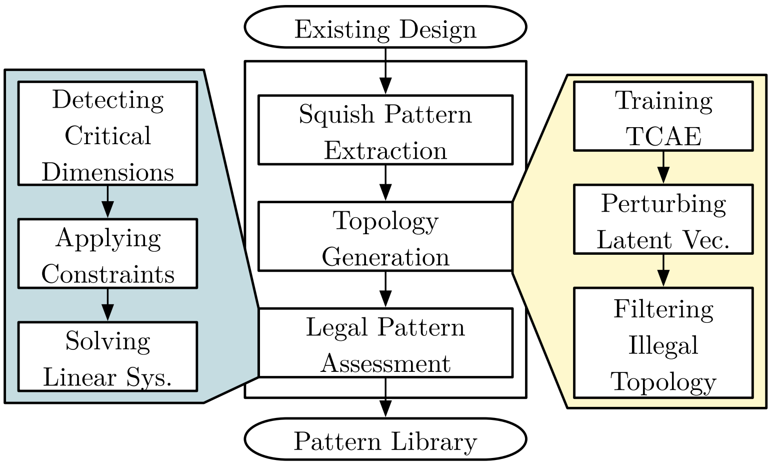


Figure 4.5: TCAE flow.

ogy generation phase, we force the TCAE to learn an identity mapping that can capture simple but important design rules. Such strategy also allows us to create a large fraction of new legal topologies by perturbing the latent vectors. In the final pattern generation stage, we search critical dimensions that are defined in the design rules in all generated topologies and formulate corresponding linear systems to obtain legal $\boldsymbol{\delta}_x$s and $\boldsymbol{\delta}_y$s.

4.2.3 Experiments

The Dataset and Configurations

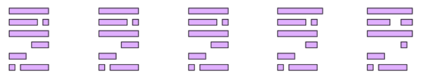
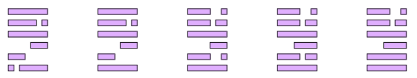
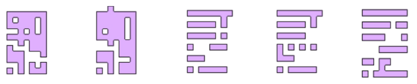
We implement the pattern generation flow using `Python` and `Tensorflow` [1] library. The framework is tested on a platform with one Tesla P100 Graphic Card. We adopt five industry benchmark groups that contains metal-2 layout clips under $7nm$ EUV design node. The clip size is $192 \times 192nm^2$ and the corresponding squish topology size is zero-padded to 24×24 that will be the input size of the neural networks. The initial learning rate is set to 0.001 and decays by 0.7 every 2000 iterations. The maximum number of training steps is 10000 with a mini-batch size of 64. All neuron weights are initialized with Xavier [35] initializer and regularized with l_2 regularizer.

The regularization coefficients for convolution layers are 0.001 and we chose 0.01 for densely connected layers. No data augmentation strategies are employed during training and the model at last training step is picked during inference stage. Note that we mark topologies with $c_x > 12$ and $c_y > 12$ as illegal such that the linear systems associated to legal topologies always *admit at least one* solution under the given window size, which ensures the quantity and quality of the generated pattern libraries. We adopt industrial solver when generating geometry information with Equation (4.7) for new topologies and only one solution is kept for each topology. Regarding the GAN component, the generator is initialized with Xavier without any regularization and the discriminator is l_2 regularized with coefficient of 0.01. The learning rate for GAN is set to be 0.001 and decayed by 0.05 every 10,000 iterations.

Understanding Features in TCAE

In the first experiment, we study the relationship between auto-learned features and human understandable layout space. Because the TCAE is trained to reconstruct input topologies as accurate as possible, feature vectors derived from the latent vector layer must attain all geometry informations that include wire tracks, line-end alignments, tip-to-tip distances, shape directions and so on. To show exactly how these auto-learned features affect the topology space, we conduct simple transformations on each individual entry of the latent vector and keep everything else unchanged. The transformed feature vectors are then fed into the reconstruction unit for topology reconstruction. We visualize part of the reconstructed patterns in Table 4.1, where each row corresponds to the transformation on certain nodes in the latent vector with everything else unchanged. In our case, a small perturbation is added up to a specific entry. We can easily observe that some features extend or pull back line-ends, some features create or destroy geometries and some features controls the directions of shapes. Unlike traditional design rules, auto-learned features control layout patterns

Table 4.1: Visualizing how are convolutional features reflected in original topology space.

Transformations	Reconstructed Topologies
Extend or pull back line-ends	
Create or destroy shapes	
Control shape directions	

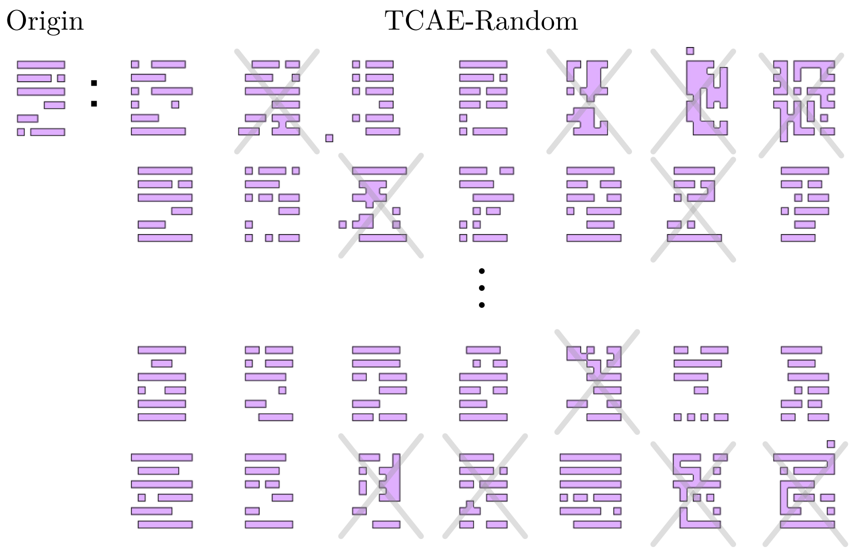
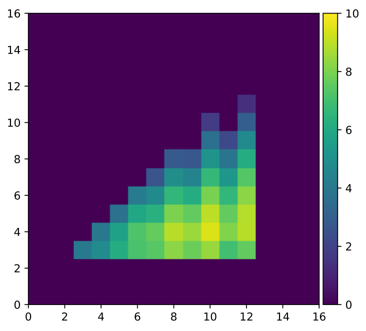
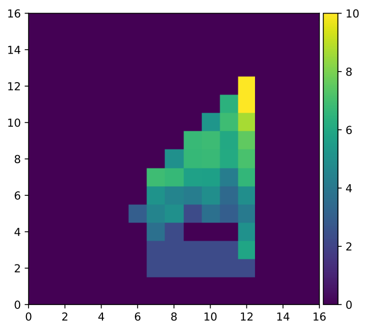


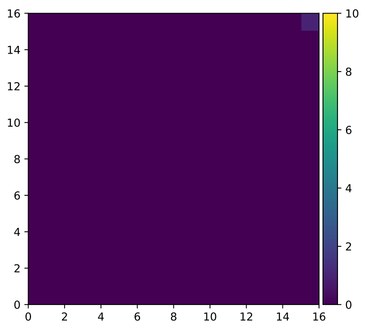
Figure 4.6: Contribution of Gaussian perturbation on topology reconstruction. 1000 topologies (∼400 legal) are created from one topology randomly picked from the existing pattern library.

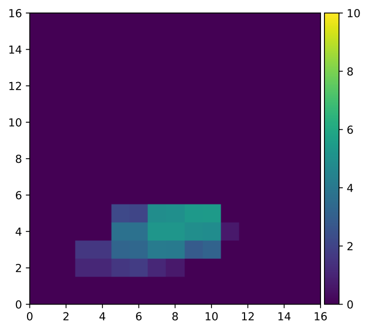
in a more global point of view that some feature determines spacing, wire length and sometimes geometry direction as a whole.

Here we show how random perturbations on the latent vectors contribute to topology generation. We randomly take one topology from the training set and obtain its feature vector through the trained encoder network. 1000 noise vectors sampled


(a) Existing Design


(b) Industry Tool


(c) DCGAN


(d) TCAE-Combine

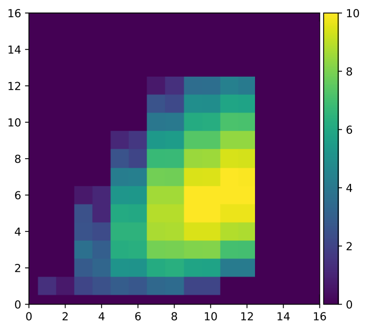

(e) TCAE-Random

Figure 4.7: Visualization of the distribution of layout libraries: (a) Existing layout pattern dataset. (b) Industrial layout generator; (c) Patterns generated by DCGAN; (d) Patterns generated by TCAE-Combine; (e) Patterns generated by TCAE-Random.

from Gaussian are added up to the feature vector before it is fed into the decoder network for pattern reconstruction. We visualize the generated topologies in Figure 4.6. We can observe that perturbations create significantly amount of new topologies where a large fraction of them are consistent with EUV pattern rules (e.g. no bow-tie shapes and unique direction single track polygons). Such results also show that deconvolution layers have the ability to learn simple design rules during training. On the contrary, no valid topology will be generated if the noise are directly applied on pattern space.

Table 4.2: Statistics of generated patterns.

Method	Pattern #	Pattern Diversity (H)
Existing Design	-	3.101
Industry Tool	55408	1.642
DCGAN	$\leq$10	0
TCAE-Combine	1738	2.665
TCAE-Random	**286898**	**3.337**

Evaluation of TCAE

We have shown the manipulation in the latent vector space can generate new topologies. In the last experiment, we will make use of the flow above to augment the pattern space. Pattern library statistics are listed in Table 4.2, where we randomly pick one pattern group to show the advantage of TCAE-Random. 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-Random" corresponds to the details of 1M patterns generated by perturbing the features of 1000 patterns in existing design with Gaussian noise. Row "TCAE-Combine" represents patterns generated from 1M different combinations of 10 test layout clip features. "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-Random" (14807 μm^2). We also implement a DCGAN [78] that has similar number of trainable parameters as the TCAE designed in this paper. 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. Perturbation with Gaussian exhibits greatest pattern generation power with around 30% generated patterns are unique and DRC clean. Combination of patterns in feature space shows much less unique pattern count because the generation procedure are restricted by existing pattern space. Combine more patterns will not affect the result much which will significantly reduce the count of DRC clean patterns if any two candidate patterns contain unaligned wires. Most GAN generated patterns fail with bow-tie or 2D wires even the training procedure has reached the equilibrium point because it is very hard to learn layout track information with randomly generated latent vectors.

Figure 4.7 compares the distributions of generated patterns and existing layout data set with similar pattern count, where x-axis and y-axis denote pattern complexity in each direction and the heatmap value is the total count (`log`-scale) of the pattern with that complexity. We employ *Pattern Diversity* to measure the pattern library distribution. We observe that Random perturbation can efficiently expand the weakly distributed pattern library (large fraction of patterns falls in certain complexities) with $H = 3.337$ while the industrial layout generator are still weakly distributed with $H = 1.642$ compared to existing designs.

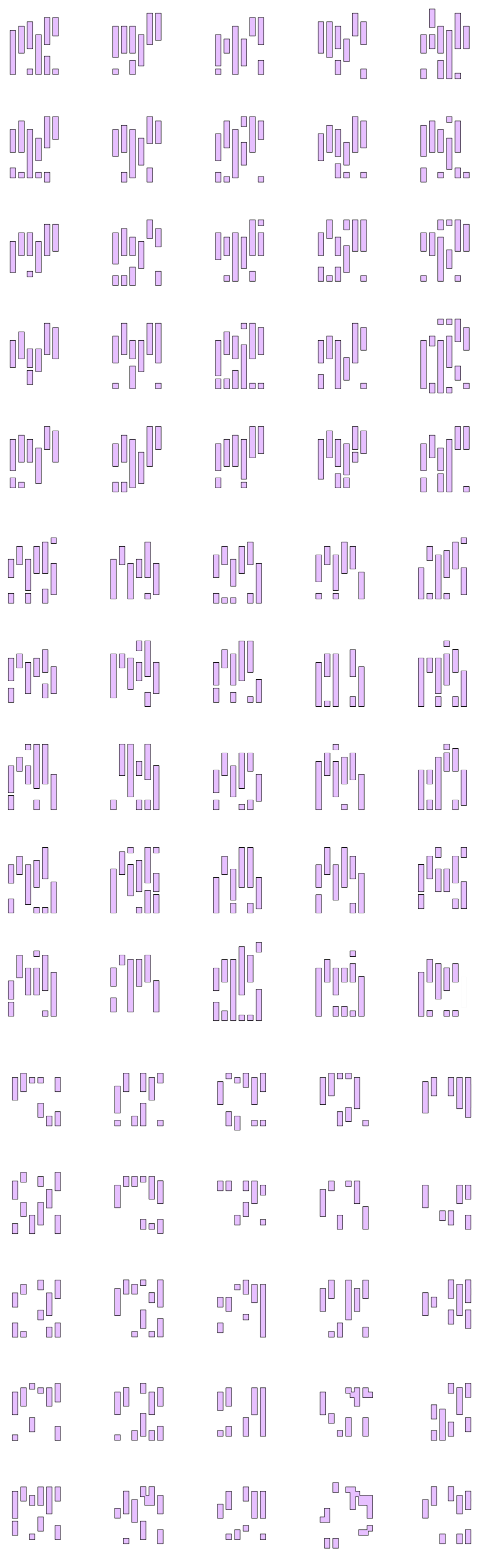
Evaluation of GAN-Guided TCAE

We will evaluate the performance of G-TCAE in two aspects: (1) massive pattern generation and (2) context-specific pattern generation.

In the *massive pattern generation* experiments, we train the TCAE model with the same settings as previous experiments. We dump out perturbation vectors that are used to create new valid patterns in TCAE-test phase that will serve as training source of the GAN. 1M of patterns are then created with GAN-generated perturbations. The results are listed in Table 4.3, where columns "Pattern Diversity (H)" are calculated dataset pattern diversity in terms of Equation (4.1), columns "Pattern #" are unique DRC-clean pattern count amount 1M generated patterns. column "Benchmarks" lists five benchmark groups `directprint1`-`directprint5`, column "Training Set" corre-

Table 4.3: Result comparison between TCAE and G-TCAE.

Benchmarks	Training Set	TCAE		G-TCAE	
	Pattern Diversity (H)	Pattern #	Pattern Diversity (H)	Pattern #	Pattern Diversity (H)
`directprint1`	2.28	472259	**3.48**	**491901**	3.41
`directprint2`	2.29	452892	3.67	**494784**	**3.72**
`directprint3`	3.16	300019	3.83	**313679**	**3.84**
`directprint4`	3.18	346233	**3.69**	**382069**	**3.69**
`directprint5`	3.62	408738	3.83	**413060**	**3.84**
Average	2.91	396028	**3.70**	**419098**	**3.70**
Ratio	0.786	1.000	**1.000**	**1.058**	**1.000**



(a) Low c_x and c_y

(b) Midium c_x and c_y

(c) High c_x and c_y

Figure 4.8: Context specific pattern generation for different complexities (c_x's and c_y's).

sponds to the statistics of the data used to training G-TCAE, column "TCAE" lists the performance of the original TCAE framework on five benchmarks, column "G-TCAE" corresponds to the performance of the proposed G-TCAE framework.

From the table we can observe that both "TCAE" and "G-TCAE" successfully enlarges the layout pattern space by increasing the pattern diversity from 2.91 to 3.70, which gives the validness of TCAE-family. By comparing the results of G-TCAE and TCAE, we show that G-TCAE offers much batter unique DRC-clean pattern count, with the help of GAN that transforms random perturbation vectors into design-rule-preserving vectors. On average, G-TCAE offers $\sim 5.8\%$ more DRC-clean patterns than TCAE. We also observe that G-TCAE exhibits similar pattern diversity compared to TCAE, which can be explained by the fact that the GAN component is trained with perturbation vectors that are used for TCAE pattern generation. Similar pattern diversity will hence be expected when the GAN is trained well to optimal states ($p_x = p_{\text{data}}$).

Regarding the *context-specific pattern generation*, we choose `directprint1` as an example for performance evaluation. The first step still comes with training of TCAE, after which latent vectors are divided into groups according to their pattern complexities. The GAN is then trained with these different latent vector groups and yields new vectors for certain complexity generation. We visualize our results in Figure 4.8, that contains mixes of low, medium and high pattern complexities in x and y directions.

4.2.4 Summary

In this section, we address the pattern library requirements in DFM flows/researches under advanced technology nodes. We propose a transforming convolutional auto-encoder framework that can capture layout design rule characteristics. We show individual element in latent vector instantiation contributes to form the pattern space locally or globally, which inspires a pattern generation flow with perturbation of the latent vector space. For the perspective of massive diverse DRC-clean pattern generation and context specific pattern generation, we also propose the GAN-guided TCAE framework that further enhances the performance and functionality of TCAE-family. The experimental results show that our framework outperforms a state-of-the-art industrial layout generation tool in terms of pattern library diversity, which is promising to facilitate early technology node development and the back-end and sign-off flows.

4.3 Generative Mask Optimization

4.3.1 Preliminaries and Problem Formulation

In this section, we will discuss some preliminaries of mask optimization and the generative adversarial nets. Major math symbols with their descriptions are listed in Table 4.4. In order to avoid confusion, all the norms $||\cdot||$ are calculated with respect to flattened vectors.

Hopkins theory of the partial coherence imaging system has been widely applied to mathematically analyze the mask behavior of lithography [48]. Because the Hopkins diffraction model is complex and not computational-friendly, [18] adopts the singular value decomposition (SVD) to approximate the original model with a weighted summation of coherent systems.

$$\boldsymbol{I} = \sum_{k=1}^{N^2} w_k |\boldsymbol{M} \otimes \boldsymbol{h}_k|^2, \tag{4.8}$$

where $\boldsymbol{h}_k$ and w_k are the k^{th} kernel and its weight. As suggested in [30], we pick the

Table 4.4: Symbols and notations used throughout the section.

Symbols	Description
$\boldsymbol{Z}_t$	Matrix representing the target layout pattern
$\boldsymbol{M}$	Matrix representing the mask pattern
$\boldsymbol{I}$	Matrix representing the aerial image
I_{th}	Threshold used for constant threshold resist model
$\boldsymbol{Z}$	Matrix representing the wafer image
$\boldsymbol{G}(\cdot)$	The generator function
$\boldsymbol{D}(\cdot)$	The discriminator function
$\boldsymbol{h}_k$	The k^{th} convolution kernel of the simple lithography model
w_k	The coefficient associated with $\boldsymbol{h}_k$
$\|\| \cdot \|\|_2$	L_2 norm, calculated with respect to flattened vectors
$\mathbb{E}$	Expectation
$\boldsymbol{x} \sim p$	Random vector $\boldsymbol{x}$ with its elements drawn from distribution p
$\otimes$	Matrix convolution
$\odot$	Entrywise product

N_h^{th} order approximation to the system. Equation (4.8) becomes,

$$\boldsymbol{I} = \sum_{k=1}^{N_h} w_k |\boldsymbol{M} \otimes \boldsymbol{h}_k|^2. \tag{4.9}$$

The lithography intensity corresponds to the exposure level on the photo resist that controls the final wafer image. In real design and sign-off flow, it is far from enough to use constant threshold model to analyze resist images, especially for advanced technology node. For the methodology verification purpose only and for simplicity, we still adopt the simple constant threshold resist model throughout the experiments, which is consistent with the ICCAD 2013 CAD contest settings [8]. In the constant threshold resist model [52], only area with intensity greater than a certain threshold will contribute to the final wafer image, as shown in Equation (4.10).

$$\boldsymbol{Z}(x, y) = \begin{cases} 1, & \text{if } \boldsymbol{I}(x, y) \geq I_{th}, \\ 0, & \text{if } \boldsymbol{I}(x, y) < I_{th}. \end{cases} \tag{4.10}$$

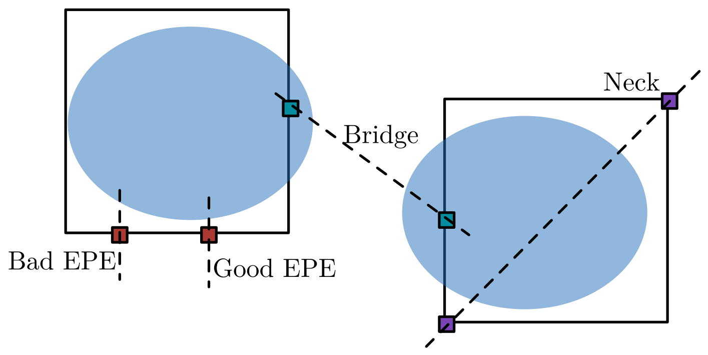


Figure 4.9: Different types of defects. Same lithography images result in different EPE violation counts due to different choices of measurement points. Some defects are not detectable through merely checking edge placement errors.

Mask quality is evaluated through the fidelity of its wafer image with respect to the target image. Edge placement error (EPE), bridge and neck are three main types of defect detectors that are adopted in a layout printability estimation flow. As shown in Figure 4.9, EPE measures horizontal or vertical distances from given points (i.e. EPE measurement sites) on target edges to lithography contours. Neck detector checks the error of critical dimensions of lithography contours compared to target patterns, while bridge detector aims to find unexpected short of wires. Note that unlike EPE violations, bridge and neck defects can appear in any directions. Because EPE violations could happen with good critical dimension and neck or bridge occurs with small EPE, none of these defect types individually can be an ideal representation of mask printability. Considering the objective of mask optimization is to make sure the remaining patterns after lithography process are as close as target patterns, we pick the squared L_2 error as the metric of lithography quality since a smaller L_2 indicates a better wafer image quality.

Definition 4.4 (Squared L_2 Error)**.** *Let $\boldsymbol{Z}_t$ and $\boldsymbol{Z}$ as target image and wafer image respectively, the squared L_2 error of $\boldsymbol{Z}$ is given by $||\boldsymbol{Z}_t - \boldsymbol{Z}||_2^2$.*

In real manufacturing scenario, lithography conditions (e.g. focus, dose) are usually not fixed as we expected, which results in variations of wafer images. To measure the robustness of the designed mask, process variation (PV) bands are proposed [81]. The mathematical definition can be found as follows.

Definition 4.5 (PV Bands). *Given the lithography simulation contours under a set of process conditions, the PV Bands is the area among all the contours under these conditions.*

In this work, for simplicity, we use the XOR between the innermost and the outermost images as an approximation of the PV Bands. Following above terminologies, we define the mask optimization problem as follows.

Problem 4.2 (Mask Optimization). *Given a target image $\boldsymbol{Z}_t$, the objective of the problem is generating the corresponding mask $\boldsymbol{M}$ such that remaining patterns $\boldsymbol{Z}$ after lithography process is as close as $\boldsymbol{Z}_t$ or, in other word, minimizing the squared L_2 error of lithography images.*

4.3.2 Architectures and Algorithms

GAN-OPC

A classical GAN architecture comprises a generator and a discriminator. The generator accepts random vectors $\boldsymbol{z} \sim p_z$ as the input and generates samples $\boldsymbol{G}(\boldsymbol{z}; \boldsymbol{W}_g)$ that follows some distribution p_g, where $\boldsymbol{G}$ is a convolutional neural networks parameterized by $\boldsymbol{W}_g$. The discriminator acts as a classifier that distinguishes $\boldsymbol{G}(\boldsymbol{z}; \boldsymbol{W}_g)$ and the instance drawn from a data distribution p_d. The output $\boldsymbol{D}(\boldsymbol{x}; \boldsymbol{W}_d)$ represents the probabilities of $\boldsymbol{x}$ drawn from p_d and p_g. It should be noted that the original settings are not well suitable for the mask optimization problem. In this section, we will introduce the details of our framework including OPC-oriented GAN architecture and advanced training strategies.

From the previous discussion we can notice that the generator learns a distribution of a given dataset, which is originally designed as a mapping function $\boldsymbol{G} : p_z \rightarrow$

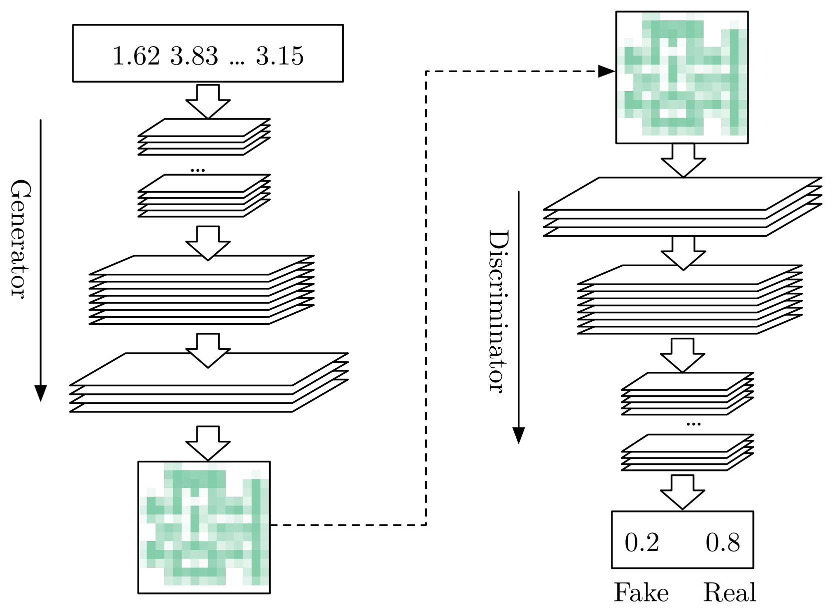


Figure 4.10: Conventional GAN architecture.

p_g, where p_z is a distribution that input vectors are drawn and p_g denotes the distribution of the training set. The objective of the generator is to generate samples that deceive the discriminator as much as possible, as in Equation (4.11):

$$\max \mathbb{E}_{\boldsymbol{z}\sim p_z}[\log(\boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z})))], \tag{4.11}$$

which maximizes the log-likelihood of the discriminator giving predictions that generated samples are real. Correspondingly, the generator comprises a deconvolutional architecture that casts 1D vectors back to 2D images through stacked deconvolution operations, as shown in Figure 4.10. Our framework, however, is expected to perform mask optimization on given target circuit patterns and obviously violates the deconvolutional architecture. To resolve this problem, we design a generator based on auto-encoder [69] which consists of an encoder and a decoder subnets. As depicted in Figure 4.11, the encoder is a stacked convolutional architecture that performs hierarchical layout feature abstractions and the decoder operates in an opposite way that predicts the pixel-based mask correction with respect to the target based on key features obtained from the encoder.

The discriminator is usually an ordinary convolutional neural networks that perform classification to distinguish the generated samples from the given data samples as shown in Equation (4.12):

$$\max \mathbb{E}_{\boldsymbol{x}\sim p_d}[\log(\boldsymbol{D}(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_z}[\log(1-\boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z})))]. \tag{4.12}$$

In this work, the discriminator predicts whether an input instance is the generated mask $\boldsymbol{M}$ or the reference mask $\boldsymbol{M}^*$ which is the ground truth OPC'ed mask generated by a state-of-the-art academic OPC tool [30]. However, the discriminator in Equation (4.12) is necessary but not sufficient to ensure generator to obtain a high quality mask (Figure 4.10). Consider a set of target patterns $\mathcal{Z} = \{\boldsymbol{Z}_{t,i}, i = 1, 2, \dots, N\}$ and a corresponding reference mask set $\mathcal{M} = \{\boldsymbol{M}_i^*, i = 1, 2, \dots, N\}$. Without loss of generality, we use $\boldsymbol{Z}_{t,1}$ in the following analysis. Suppose the above GAN structure has enough capacity to be well trained, the generator outputs a mask $\boldsymbol{G}(\boldsymbol{Z}_{t,1})$ that optimizes the objective function as in Equation (4.11). Observe that $\log(\boldsymbol{D}(\boldsymbol{G}(\boldsymbol{Z}_{t,1})))$ reaches its maximum value as long as

$$\boldsymbol{G}(\boldsymbol{Z}_{t,1}) = \boldsymbol{M}_i^*, \forall i = 1, 2, \dots, N. \tag{4.13}$$

Therefore, a one-to-one mapping between the target and the reference mask cannot be guaranteed with current objectives. To address above concerns, we adopt a classification scheme that predicts positive or negative labels on target-mask pairs that inputs of the discriminator will be either $(\boldsymbol{Z}_t, \boldsymbol{G}(\boldsymbol{Z}_t))$ or $(\boldsymbol{Z}_t, \boldsymbol{M}^*)$, as illustrated in Figure 4.11. Claim that $\boldsymbol{G}(\boldsymbol{Z}_t) \approx \boldsymbol{M}^*$ at convergence with new discriminator. We still assume enough model capacity and training time for convergence. The discriminator now performs prediction on target-mask pairs instead of masks. Because only

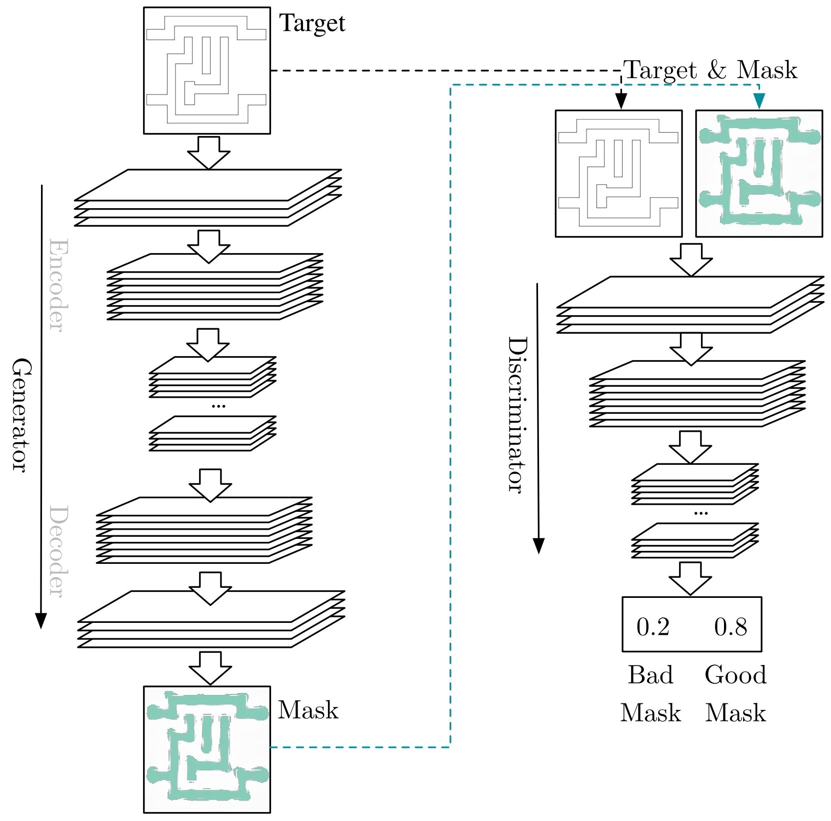


Figure 4.11: The proposed GAN-OPC architecture.

pairs $\{\boldsymbol{Z}_{t,i}, \boldsymbol{M}_i^*\}$ are labeled as data, the generator can deceive the discriminator if and only if $\boldsymbol{G}(\boldsymbol{Z}_{t,i}) \approx \boldsymbol{M}_i^*, \forall i = 1, 2, \ldots, N$, where N is the total number of training instances.

Based on the OPC-oriented GAN architecture in our framework, we tweak the objectives of $\boldsymbol{G}$ as follows,

$$\max_{\boldsymbol{G}} \mathbb{E}_{\boldsymbol{Z}_t \sim \mathcal{Z}}[\log(\boldsymbol{D}(\boldsymbol{Z}_t, \boldsymbol{G}(\boldsymbol{Z}_t)))], \tag{4.14}$$

and for the discriminator $\boldsymbol{D}$ we have,

$$\max_{\boldsymbol{D}} \mathbb{E}_{\boldsymbol{Z}_t \sim \mathcal{Z}}[\log(\boldsymbol{D}(\boldsymbol{Z}_t, \boldsymbol{M}^*))] + \mathbb{E}_{\boldsymbol{Z}_t \sim \mathcal{Z}}[1 - \log(\boldsymbol{D}(\boldsymbol{Z}_t, \boldsymbol{G}(\boldsymbol{Z}_t)))]. \tag{4.15}$$

In addition to facilitate the training procedure, we minimize the differences between generated masks and reference masks when updating the generator as in Equation (4.16).

$$\min_{\boldsymbol{G}} \mathbb{E}_{\boldsymbol{Z}_t \sim \mathcal{Z}} ||\boldsymbol{M}^* - \boldsymbol{G}(\boldsymbol{Z}_t)||_n, \tag{4.16}$$

where $||\cdot||_n$ denotes the l_n norm. Combining (4.14), (4.15) and (4.16), the objective of our GAN model becomes

$$\begin{aligned}\min_{\boldsymbol{G}} \max_{\boldsymbol{D}} \ & \mathbb{E}_{\boldsymbol{Z}_t \sim \mathcal{Z}}[1 - \log(\boldsymbol{D}(\boldsymbol{Z}_t, \boldsymbol{G}(\boldsymbol{Z}_t))) + ||\boldsymbol{M}^* - \boldsymbol{G}(\boldsymbol{Z}_t)||_n^n] \\ & + \mathbb{E}_{\boldsymbol{Z}_t \sim \mathcal{Z}}[\log(\boldsymbol{D}(\boldsymbol{Z}_t, \boldsymbol{M}^*))]. \end{aligned} \tag{4.17}$$

Previous analysis shows that the generator and the discriminator have different objectives, therefore the two sub-networks are trained alternatively, as shown in Figure 4.12(a) and Algorithm 4.2. 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 4–5); 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 4.2 we convert the min-max problem in Equation (4.17) into two minimization problems such that gradient ascending operations are no longer required to update neuron weights. Algorithm 4.2 differs from traditional GAN optimization flow on the following aspects. (1) The generator plays as a mapping function from target to mask instead of merely a distribution, therefore the gradient of L_2 loss is back-propagated along with the information from the discriminator. (2) The dis-

Algorithm 4.2 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:     $\Delta\boldsymbol{W}_g \leftarrow \boldsymbol{0}, \Delta\boldsymbol{W}_d \leftarrow \boldsymbol{0}$;
4:     for each $\boldsymbol{Z}_t \in \mathcal{Z}$ do
5:         $\boldsymbol{M} \leftarrow \boldsymbol{G}(\boldsymbol{Z}_t; \boldsymbol{W}_g)$;
6:         $\boldsymbol{M}^* \leftarrow$ Groundtruth mask of $\boldsymbol{Z}_t$;
7:         $l_g \leftarrow -\log(\boldsymbol{D}(\boldsymbol{Z}_t, \boldsymbol{M})) + \alpha||\boldsymbol{M}^* - \boldsymbol{M}||_2^2$;
8:         $l_d \leftarrow \log(\boldsymbol{D}(\boldsymbol{Z}_t, \boldsymbol{M})) - \log(\boldsymbol{D}(\boldsymbol{Z}_t, \boldsymbol{M}^*))$;
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
```

criminator functions as an alternative of ILT engine that determines only the quality of generated masks without any calibration operations. Besides, our combined input ensures that the discriminator will make positive prediction if and only if the generated mask is much close to the ground truth, which also helps train the generator better.

ILT-guided Pre-training

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.

$$E = ||\boldsymbol{Z}_t - \boldsymbol{Z}||_2^2, \tag{4.18}$$

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.

$$\boldsymbol{Z} = \frac{1}{1+\exp[-\alpha \times (\boldsymbol{I} - \boldsymbol{I}_{th})]}, \tag{4.19}$$

$$\boldsymbol{M}_b = \frac{1}{1+\exp(-\beta \times \boldsymbol{M})}, \tag{4.20}$$

where $\boldsymbol{I}_{th}$ is the threshold matrix in the constant resist model with all the entries being I_{th}, $\boldsymbol{M}_b$ is the incompletely binarized mask, while α and β control the steepness of relaxed images.

Combine Equations (4.8)–(4.10), Equations (4.18)–(4.20) and the analysis in [77], we can derive the gradient representation as follows,

$$\begin{aligned}\frac{\partial E}{\partial \boldsymbol{M}} =& 2\alpha\beta \times \boldsymbol{M}_b \odot (1-\boldsymbol{M}_b) \odot \\ & (((\boldsymbol{Z}-\boldsymbol{Z}_t) \odot \boldsymbol{Z} \odot (1-\boldsymbol{Z}) \odot (\boldsymbol{M}_b \otimes \boldsymbol{H}^*)) \otimes \boldsymbol{H} + \\ & ((\boldsymbol{Z}-\boldsymbol{Z}_t) \odot \boldsymbol{Z} \odot (1-\boldsymbol{Z}) \odot (\boldsymbol{M}_b \otimes \boldsymbol{H})) \otimes \boldsymbol{H}^*),\end{aligned} \tag{4.21}$$

where $\boldsymbol{H}^*$ is the conjugate matrix of the original lithography kernel $\boldsymbol{H}$. In traditional ILT flow, the mask can be optimized through iteratively descending the gradient until E is below a threshold.

The objective of 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 procedure, we propose a jointed training algorithm that takes advantages of ILT engine, as depicted in Figure 4.12(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-

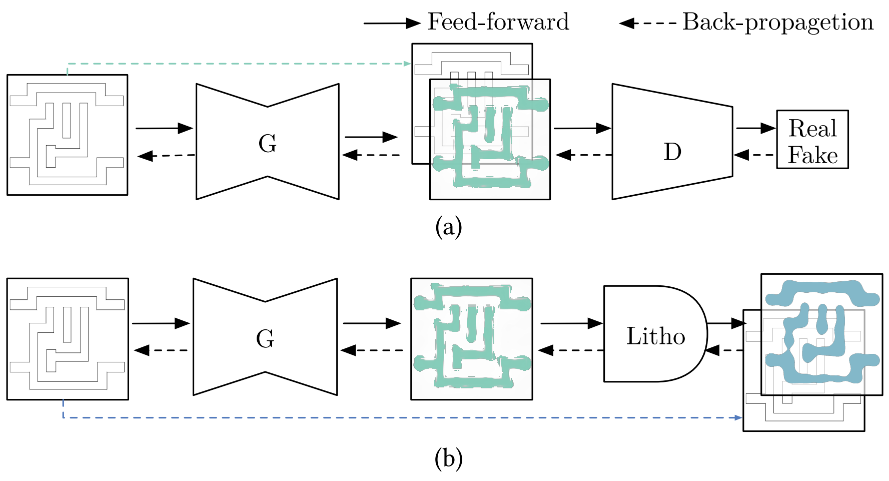


Figure 4.12: (a) GAN-OPC training and (b) ILT-guided pre-training.

propagating the training error from the output layer to the input layer while neural weights are updated as follows,

$$\boldsymbol{W}_g = \boldsymbol{W}_g - \frac{\lambda}{m}\Delta\boldsymbol{W}_g, \tag{4.22}$$

where $\Delta\boldsymbol{W}_g$ is accumulated gradient of a mini-batch of instances and m is the mini-batch instance count. Because Equation (4.22) 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 4.12.

The generator pre-training phase is detailed in Algorithm 4.3. 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 (4.18) (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., $\dfrac{\partial E}{\partial \boldsymbol{M}}\dfrac{\partial \boldsymbol{M}}{\partial \boldsymbol{W}_g}$ (line 8) ; Finally, $\boldsymbol{W}_g$ is updated following the gradient descent procedure (line 10).

Algorithm 4.3 ILT-guided Pre-training

```
1: for number of pre-training iterations do
2:     Sample $m$ target clips $\mathcal{Z} \leftarrow \{\boldsymbol{Z}_{t,1}, \boldsymbol{Z}_{t,2}, \ldots, \boldsymbol{Z}_{t,m}\}$;
3:     $\Delta \boldsymbol{W}_g \leftarrow 0$;
4:     for each $\boldsymbol{Z}_t \in \mathcal{Z}$ do
5:         $\boldsymbol{M} \leftarrow \boldsymbol{G}(\boldsymbol{Z}_t; \boldsymbol{W}_g)$;
6:         $\boldsymbol{Z} \leftarrow \texttt{LithoSim}(\boldsymbol{M})$                ▷ Equations (4.9)–(4.10)
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 (4.21)
9:     end for
10:    $\boldsymbol{W}_g \leftarrow \boldsymbol{W}_g - \frac{\lambda}{m} \Delta \boldsymbol{W}_g$;                ▷ Equation (4.22)
11: end for
```

Enhanced GAN-OPC Framework

In this section, we will introduce the enhanced GAN-OPC framework, which significantly improves the training efficiency in a more elegant way compared to pretraining with ILT engine. The enhanced GAN-OPC framework includes a U-Net structure that allows gradients to be easily back-propagated to early layers and a sub-pixel super-resolution architecture for better generated mask quality.

We have noticed the GANs are typically deeper than traditional neural networks, which brings more challenges due to a longer gradient back-propagation path. A common solution is creating shortcut links that allow addition or stacking of feature maps in different layers [83, 45, 51], such that gradients can be more efficiently back-propagated from output layer to early layers. Here we enhance our generator design with a U-Net-like structure where intermediate feature maps in the encoder are stacked at corresponding layers in the decoder, as shown in Figure 4.13. Such architecture has two good properties: (1) The inevitable information loss in strided convolution layer can be drastically reduced. (2) The gradient vanishing problem can be alleviated with multiple shortcut links bypassing intermediate feature maps.

In previous designs, low level features in intermediate generator layers are cast

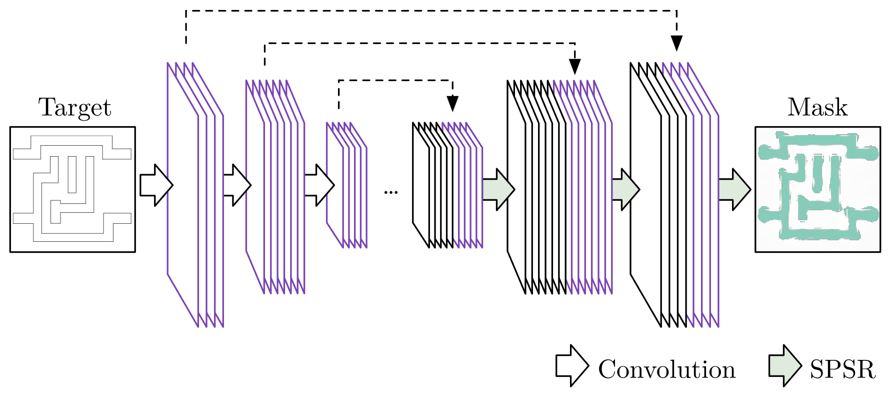


Figure 4.13: New generator architecture with concatenation of intermediate feature maps and an SPSR structure.

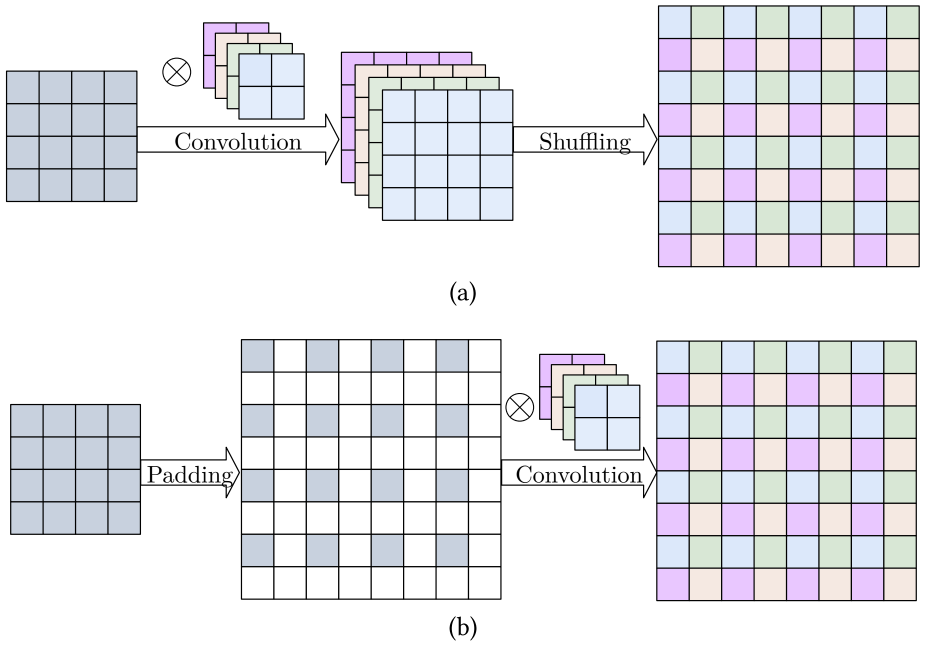


Figure 4.14: Visualization of (a) standard deconvolution operation and (b) SPSR.

back to mask images by standard strided deconvolution operation that can be visualized as in Figure 4.14(a). In detail, zeros are inserted among existing pixels such that the output dimension after a convolution operation reaches the desired value. However, such mechanism requires multiple convolution operations on high resolution space which is not computational efficient and might induce additional noises.

SPSR [90] is another upsampling solution that has been widely used in super-resolution tasks. It conducts convolution operations in lower resolution space and generates additional feature maps such that the number of feature map entries matches the desired size of target image, as shown in Figure 4.14(b). The major step of SPSR is called periodic shuffling that casts a tensor with shape $H \times W \times r^2C$ into shape $rH \times rW \times C$ as defined in Equation (4.23).

$$
\begin{aligned}
t^{hr}_{i,j,k} &= t^{lr}_{i',j',k'}, && (4.23\text{a})\\
i' &= \lfloor \frac{i}{r} \rfloor, && (4.23\text{b})\\
j' &= \lfloor \frac{j}{r} \rfloor, && (4.23\text{c})\\
k' &= C \cdot r \cdot \text{mod}(j, r) + C \cdot \text{mod}(i, r) + k, && (4.23\text{d})
\end{aligned}
$$

where $\lfloor \cdot \rfloor$ is the math floor operator, $\text{mod}(x, y)$ finds the remainder of x divided by y, $t^{hr}_{i,j,k}$ and $t^{lr}_{i',j',k'}$ denotes the (i, j, k) and (i', j', k') entry of high resolution images (or feature maps) and low resolution images (or feature maps) respectively. It should be noted that Equation (4.23) represents only a reshape operation which is still differentiable as other convolution layers. SPSR has several advantages compared to Figure 4.14(a). (1) SPSR is ideally r^2 times faster than the strided deconvolution operation. As shown in Figure 4.14, same convolution kernels have to scan over a r^2 larger feature maps in traditional deconvolution layers to achieve same output tensor size as SPSR layers. (2) SPSR layers reduce noises in generated masks by a significant amount, as can be seen in Figure 4.15. Such results can be explained by the fact that explicit interpolations are removed in SPSR structure, where the upscaling and rendering are automatically learned during the network training. Traditional deconvolution layers, on the other hand, have to apply padding or zero insertion to increase the feature map size before feeding them into next level convolution layer for rendering, which in turn results in noises (as empty dots) in the generated masks

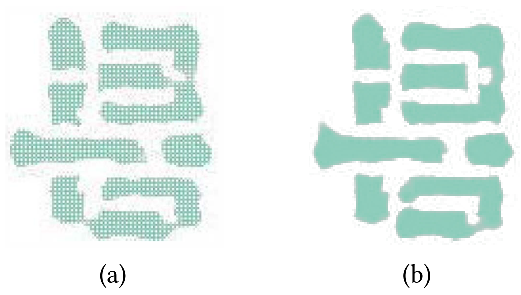


Figure 4.15: Patterns generated from (a) deconvolution layers and (b) SPSR layers.

because it is hard for limited number of convolution layers to smooth such noise. On the contrary, SPSR directly organizes the low resolution feature maps into the high resolution space, where every pixels are informative, compared to manually inserted zeros in deconvolution layers.

We follow the basic convolutional auto-encoder architecture for the generator design with additional shortcut links for U-Net feature map sharing and SPSR layers for upscaling. The detailed architecture can be found in Table 4.5, where column "Layer" includes layer types and layer ID, columns "Filter" and "Stride" list configurations for convolution layers, "Output" lists the output tensor shape of corresponding layers and "Parameter" represents the total number of trainable parameters of a given layer. The proposed generator architecture contains five regular convolution layers for feature extraction and five SPSR layers for mask image construction. It should be noted that the input tensor of the i^{th} SPSR layer has $2\times$ channel numbers as the output tensor of the $(i-1)^{\text{th}}$ SPSR layer, because of the existence of U-Net concatenation.

The discriminator design is detailed in Table 4.6. The neural network architecture resembles VGG [94] with more layers and smaller kernels. "repeat2" and "repeat3" indicate two and three consecutive convolution layers with the same configurations. We replace all the pooling layers with strided convolution layers to attain information as much as possible. Three densely connected layers are connected following the last convolution layer for final class prediction. The total number of trainable parameters

Table 4.5: The generator configuration.

Layer	Filter	Stride	Output	Parameter
conv-1	5×5×16	2	128×128×16	400
conv-2	5×5×64	2	64×64×64	25600
conv-3	5×5×128	2	32×32×128	204800
conv-4	5×5×512	2	16×16×512	1638400
conv-5	5×5×1024	2	8×8×1024	13107200
spsr-5	3×3×2048	1	16×16×512	18874368
spsr-4	3×3×512	1	32×32×128	4718952
spsr-3	3×3×256	1	64×64×64	589824
spsr-2	3×3×64	1	128×128×16	73728
spsr-1	3×3×4	1	256×256×1	1152
Summary	-	-	-	39234064

Table 4.6: The discriminator configuration.

Layer	Filter	Stride	Output	Parameter
repeat2-1	3×3×64	1	256×256×64	38016
conv-1	3×3×64	2	128×128×64	36864
repeat2-2	3×3×128	1	128×128×128	221184
conv-2	3×3×128	2	64×64×128	147456
repeat3-1	3×3×256	1	64×64×256	1474560
conv-3	3×3×256	2	32×32×256	589824
repeat3-2	3×3×512	1	32×32×512	5898240
conv-4	3×3×512	2	16×16×512	2359296
repeat3-3	3×3×512	1	16×16×512	7077888
conv-5	3×3×512	2	8×8×512	2359296
fc-1	-	-	2048	67108864
fc-2	-	-	512	1048576
fc-3	-	-	2	1024
Summary	-	-	-	88361088

of the discriminator are intentionally designed much larger than the generator (88M vs. 39M) in case of model collapsing.

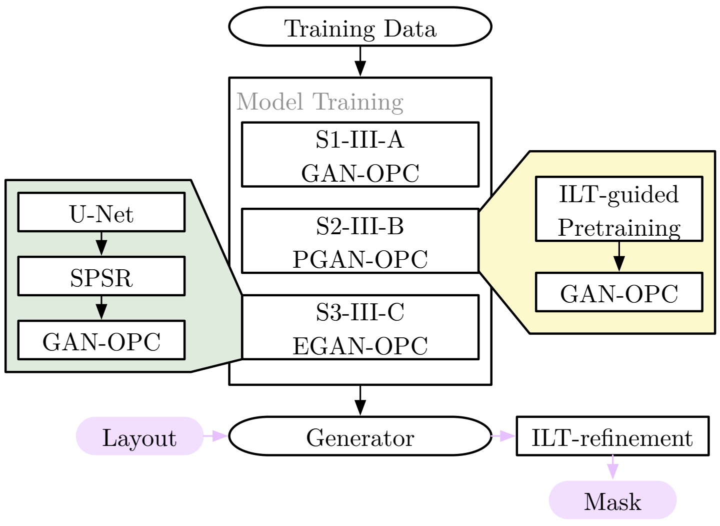


Figure 4.16: The framework summary.

Overall Flow and Discussion

The proposed GAN-OPC family is summarized in Figure 4.16. With the given training data that includes target patterns and mask patterns, we propose three alternative solutions to obtain a trained generator that include direct GAN-OPC proposed in Section 4.3.2, GAN-OPC with ILT-guided pretraining as in Section 4.3.2 and the enhanced GAN-OPC solution with U-Net and SPSR techniques as in Section 4.3.2.

Although ILT engines are, to some extent, suffering mask manufacturability (e.g violation of mask notch rule and mask spacing rule [6, 96] which are not considered in this work) and runtime issues compared to traditional model-based OPC, our framework still takes advantage of such methodology with the following reasons. Our framework is built upon the structure of conditional GAN that learns a pixel-to-pixel mapping from the target pattern to the OPCed pattern. The optimization scheme is in a continuous form that compensation patterns can appear in any shapes and any places within the clip. Thus the patterns generated by GAN are inconsistent with the model-based OPC results (e.g. [57, 96]) where compensations are made by moving polygon segments inward or outward. However, we observe that the mask patterns are naturally compatible with the process of ILT, which becomes one reason that we choose ILT for our refinement tasks. As can be seen in previous works [30, 67], ILT is associated with a highly non-convex optimization problems that means the mask initialization affects the final results. The ILT refinement results outperform direct ILT optimization and also experimentally demonstrate the effectiveness of the proposed GAN-OPC framework. Another reason that we choose ILT is that theoretically and intuitively ILT provides a larger solution space in mask optimization problems and tends to offer better mask quality. There are two major advantages of our proposed framework:

- Compared to ILT itself, the GAN-OPC family offers a better starting point for ILT optimization that promises faster convergence and better mask quality.
- Compared to model-based OPC, the proposed framework attains good properties of ILT, i.e., a lager solution space that has the potential to generate better pattern compensation for better mask printability.

Although we did not consider the mask notch and spacing rule in our framework, it is straightforward to conduct mask manufacturability rule check on the generated masks, and fix the violated region by making minor pixel changes in the generated masks. Actually, if the ground truth masks used for training can meet the mask manufacturability requirements, the GAN-OPC framework is supposed to capture these rules during training, because the discriminator is specifically designed to tell whether the generated masks are good or not. Here a “good” mask refers to the mask that has good printability and good manufacturability.

4.3.3 Experiments

The generative adversarial network for mask optimization is implemented based on `Tensorflow` [1] library and tested on single Nvidia Titan X. The lithography engine

Table 4.7: The design rules used.

Item	Min Size (nm)
M1 Critical Dimension	80
Pitch	140
Tip to tip distance	60

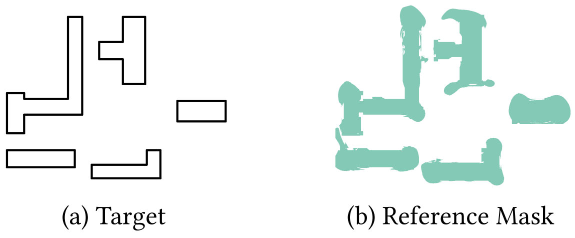


Figure 4.17: An example of a target and a reference mask pair.

is based on the `lithosim_v4` package from ICCAD 2013 CAD Contest [8], which also provides ten industrial M1 designs on $32nm$ design node. We pick $N_h = 24, \alpha = 50$ and $\beta = 4$ for the lithography simulation procedure. The ILT refinement will be stopped if the average gradient per pixel as calculated in Equation (4.21) is smaller than 5×10^{-4}. Related parameters are chosen according to the experimental results on one test case. The OPC framework applies $8nm$ resolution during the initial mask generation stage and $1nm$ resolution for refinement.

Synthesizing Training Data

As a type of deep neural networks, GAN can be hardly well trained with only ten instances. To verify our framework, we synthesize a training layout library with 4000

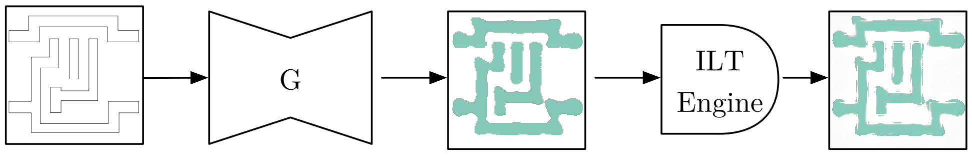


Figure 4.18: GAN-OPC flow: generator inference and ILT refinement.

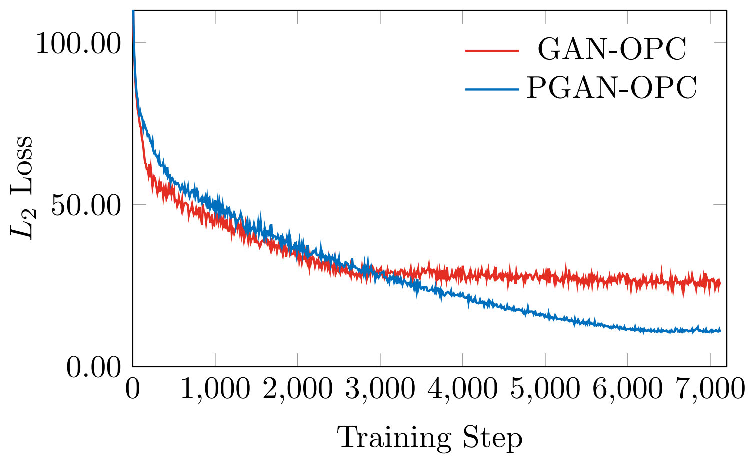


Figure 4.19: Training curves of GAN-OPC and PGAN-OPC.

instances based on the design specifications from existing $32nm$ M1 layout topologies. We adjust the wire sizes to make sure the shapes in synthesized layouts are similar to those in the given benchmark. To generate experimental cells, all the shapes are randomly placed together based on simple design rules, as detailed in Table 4.7. An example of such synthesized target-reference mask pair can be found in Figure 4.17. In addition, most generative models have shown obvious weakness in image details, which makes it extremely hard to optimize images with size 2048×2048. Therefore, we perform 8×8 average pooling on layout images before feeding them into the neural networks. In the generation stage we adopt simple linear interpolation to convert the layout images back to their original resolution.

Evaluation of GAN-OPC and PGAN-OPC

The proposed GAN-OPC flow is illustrated in Figure 4.18, where we first feed target patterns into the generator and obtain the quasi-optimal masks, followed by refinement through an ILT engine. In the first experiment, to verify the effectiveness of ILT-guided pre-training algorithm, we record training behaviors of two GANs which are denoted by GAN-OPC and PGAN-OPC. Here “GAN-OPC” and “PGAN-OPC” denote

Table 4.8: Comparison with state-of-the-art.

Benchmarks	ILT [30]		Model-Based [57]		GAN-OPC		PGAN-OPC		EGAN-OPC	
	L2	PVB (nm^2)	L2	PVB (nm^2)	L2	PVB (nm^2)	L2	PVB (nm^2)	L2	PVB (nm^2)
1	49893	65534	53816	66218	54970	64163	52570	56267	55425	58043
2	50369	48230	41382	53434	46445	56731	42253	50822	40211	53020
3	81007	108608	79255	146776	88899	84308	83663	94498	93090	75644
4	20044	28285	21717	33266	18290	29245	19965	28957	22877	26401
5	44656	58835	48858	65631	42835	59727	44733	59328	42650	59765
6	57375	48739	46320	62068	44313	52627	46062	52845	39776	54878
7	37221	43490	31898	51069	24481	47652	26438	47981	22761	49156
8	19782	22846	23312	25898	17399	23769	17690	23564	16296	24441
9	55399	66331	55684	75387	53637	66766	56125	65417	52157	66492
10	24381	18097	19722	18536	9677	20693	9990	19893	9765	21338
Average	44012.7	50899.5	42196.4	59828.3	40094.6	50568.1	39948.9	49957.2	39500.8	48917.8
Ratio	1.0	1.0	0.959	1.175	0.911	0.993	0.908	0.981	**0.898**	**0.961**

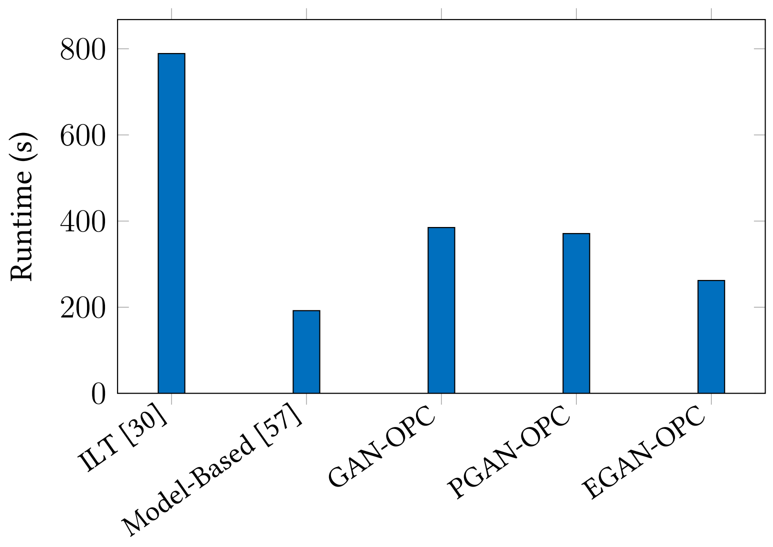


Figure 4.20: Average runtime comparison of different methods.

GAN-OPC flow without generator pre-training and GAN-OPC flow with ILT-guided pre-training, respectively. “ILT” corresponds to `MOSAIC_fast` in [30]. The training procedure is depicted in Figure 4.19, where x-axis indicates training steps and y-axis is L_2 loss between generator outputs and ground truth masks, as in Equation (4.16).

The training time for both GAN and PGAN are around 10 hours on our platform. Although L_2 loss of GAN-OPC drops slightly faster before 3000 iterations, the training curve shows that PGAN-OPC is a more stable training procedure and converges to a lower loss. Besides, it takes much more efforts for GAN-OPC to search a direction to descending the gradient fast, while the training loss of PGAN-OPC drops smoothly and converges at a lower L_2 loss than GAN-OPC, which indicates ILT-guided pre-training indeed facilitates mask-optimization-oriented GAN training flow. We will also show that PGAN-OPC exhibits better mask optimization results in the following section.

In the second experiment, we optimize the ten layout masks in ICCAD 2013 contest benchmark [8] and compare the results with previous work, as listed in Table 4.8. Here the wafer images are calculated from the simulation tool (`lithosim_v4`) in the

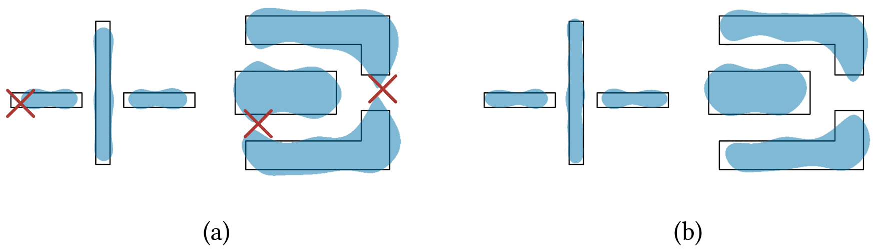


Figure 4.21: Some wafer image details of (a) ILT [30] and (b) PGAN-OPC.

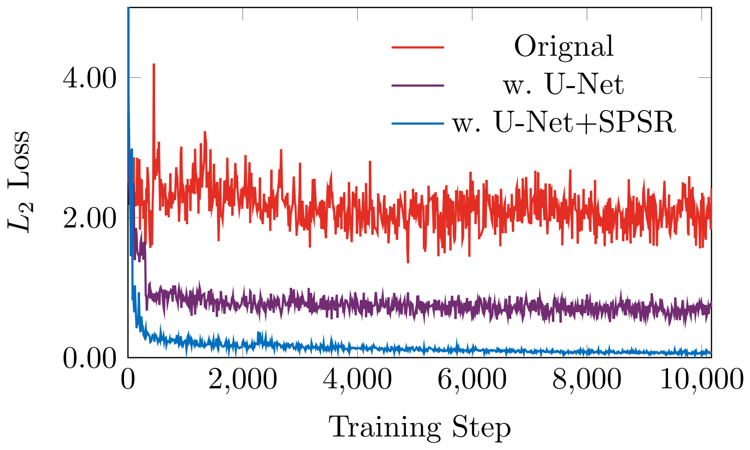


Figure 4.22: Training behavior of the EGAN-OPC framework with faster and better convergence.

contest [8]. 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 nominal condition. Column “PVB” denotes the contour area variations under $\pm$ 2% dose error and defocus range of $\pm 25nm$ settings as in the contest. 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%. We also compared our work with one academic state-of-the-art Model-based OPC engine [57], which exhibits larger L_2 error (42196.4) and worse PVB (59828.3nm^2) compared to GAN-OPCs.

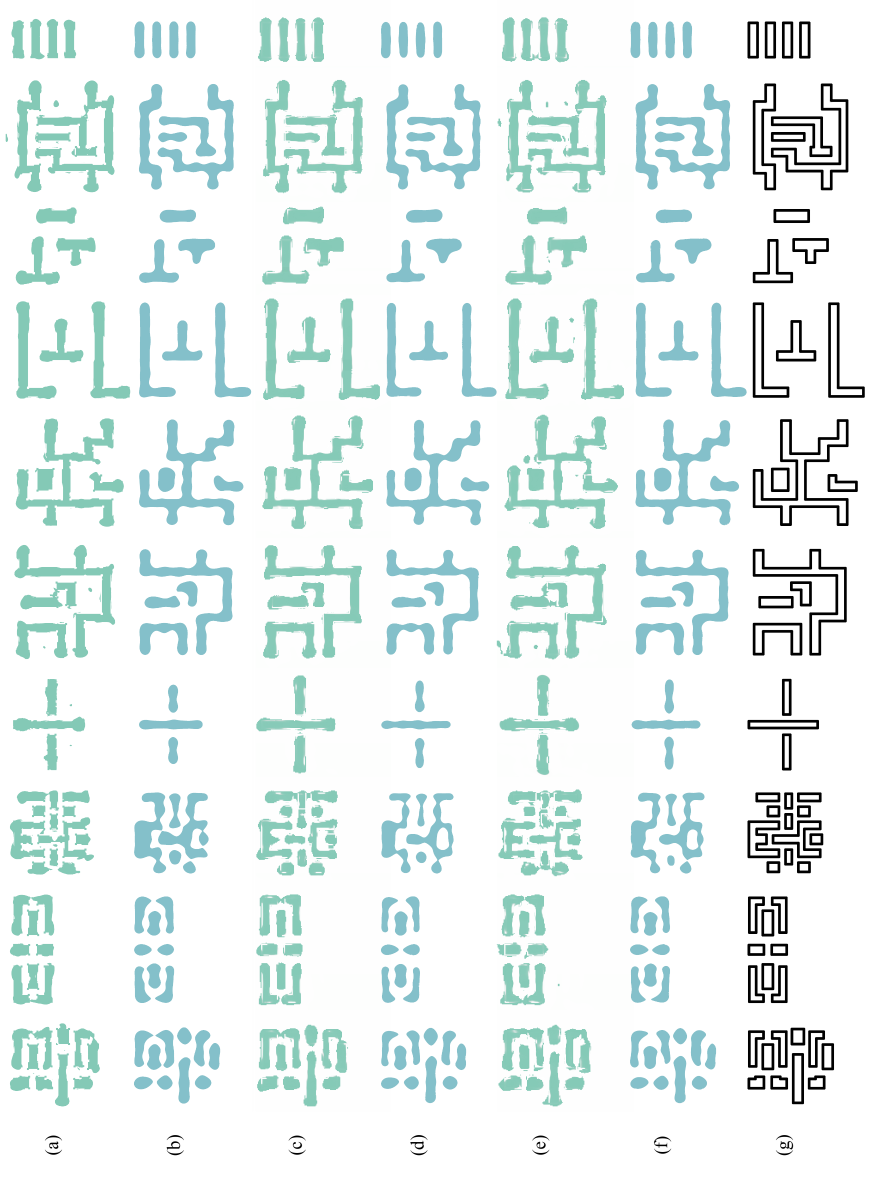


Figure 4.23: Result visualization of PGAN-OPC, Enhanced GAN-OPC and ILT. Columns correspond to ten test cases from ICCAD 2013 CAD contest. Rows from top to bottom are: (a) masks of [30]; (b) wafer images by masks of [30]; (c) masks of PGAN-OPC; (d) wafer images by masks of PGAN-OPC; (e) masks of Enhanced GAN-OPC; (f) wafer images by masks of Enhanced GAN-OPC; (g) target patterns.

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 among ten test cases. Additionally, feed-forward computation of GAN only takes $0.2s$ for each image which is ignorable, therefore runtime of our flow is almost determined by ILT refinements. Runtime of different frameworks are illustrated in Figure 4.20. Items “ILT”, “Model-Based”“GAN-OPC”, “PGAN-OPC” list the average mask optimization time of [30], [57], GAN-OPC and PGAN-OPC, respectively. For most benchmark cases, GAN-OPC and PGAN-OPC show a earlier stop at a smaller L_2 error and, on average, reduce the optimization runtime by more than 50%. We also observe that model-based OPC engine shows advantages on execution time at the cost of wafer image quality as well as PVB area, as shown in Table 4.8. For most test cases, [30] exhibits a smaller PV band area possibly because the printed images are more likely to have large wafer image CD and shorter wire length, which makes masks suffer less proximity effects while inducing bridge or line-end pull back defects, as shown in Figure 4.21.

Evaluation of Enhanced GAN-OPC

Here we show the effectiveness and the efficiency of the Enhanced GAN-OPC (EGAN-OPC) framework. In the first experiment, we illustrate the training behavior of PGAN-OPC and the EGAN-OPC frameworks as shown in Figure 4.22. Red curve stands for the original PGAN-OPC model, which fluctuates fiercely around a large value. Dark curve refers to the results with U-Net generator. Blue curve represents the complete version of enhanced GAN model with both U-net structure and the embedded SPSR structure. It’s encouraging to see that U-net alone can already ensure a good convergence in terms of L_2 loss. As we have pointed out in algorithm section, such structure attains the neural network capacity with significantly lower computational cost, which is consistent with the trends of L_2 error during training.

In the second experiment, we compare the mask optimization results of the EGAN-OPC with original GAN-OPC and PGAN-OPC, as depicted in Figure 4.23. The quantitative results can also be found in column "EGAN-OPC" of Table 4.8. EGAN-OPC outperforms PGAN-OPC and GAN-OPC on most test cases with better L_2 error (39500 vs. 39948) and smaller PVB area (48917nm^2 vs. 49957nm^2) with only 70% average runtime of PGAN-OPC (see Figure 4.20), which demonstrates the efficiency of EGAN-OPC framework. It should be also noted that EGAN-OPC can be trained end-to-end without any interaction with the lithography engine which induces a large amount of computational cost in PGAN-OPC.

On the Scalability of GAN-OPC Family

In order to verify the scalability of our frameworks, we conduct further experiments on 10 additional testcases that contain more patterns and larger total pattern areas. Similar to [96], these 10 testcases are created from the original IBM benchmarks with additional geometries. The results of one example can be found in Figure 4.24. It can be seen that our framework generalizes to more complex patterns. We also visualize the ILT convergence in terms of different mask initialization in Figure 4.25. Here we use testcase 18 as an example. It can be seen that ILT converges much faster when using the mask initialized by EGAN-OPC as input, with only ignorable PV Band penalty. We did not compare the performance with model-based OPC as the binary release in [57] encounters unknown failure on the new benchmarks.

We list the detailed optimization results in Table 4.9, where columns are defined exactly the same as Table 4.8. It can be seen that GAN-OPC exhibits trade-offs on nominal image quality and PVB compared to pure ILT, while both PGAN-OPC and EGAN-OPC show significant advantages on L_2 error (86105.7 v.s. 90486.3) with similar or slightly better PVB (108690.7nm^2 v.s.109842.7nm^2). Besides, competitive results of our framework are also achieved with shorter optimization time thanks to the

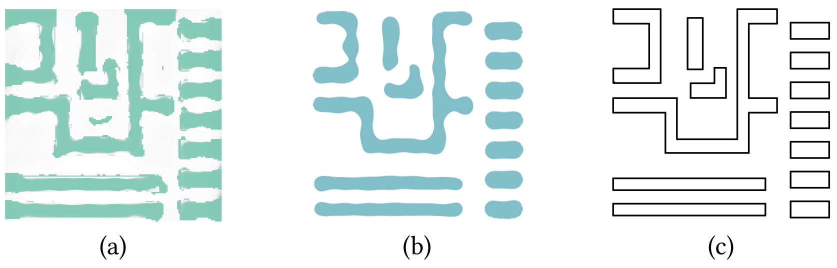


Figure 4.24: A larger-case example of (a) a mask pattern, (b) its wafer image and (c) the corresponding target pattern.

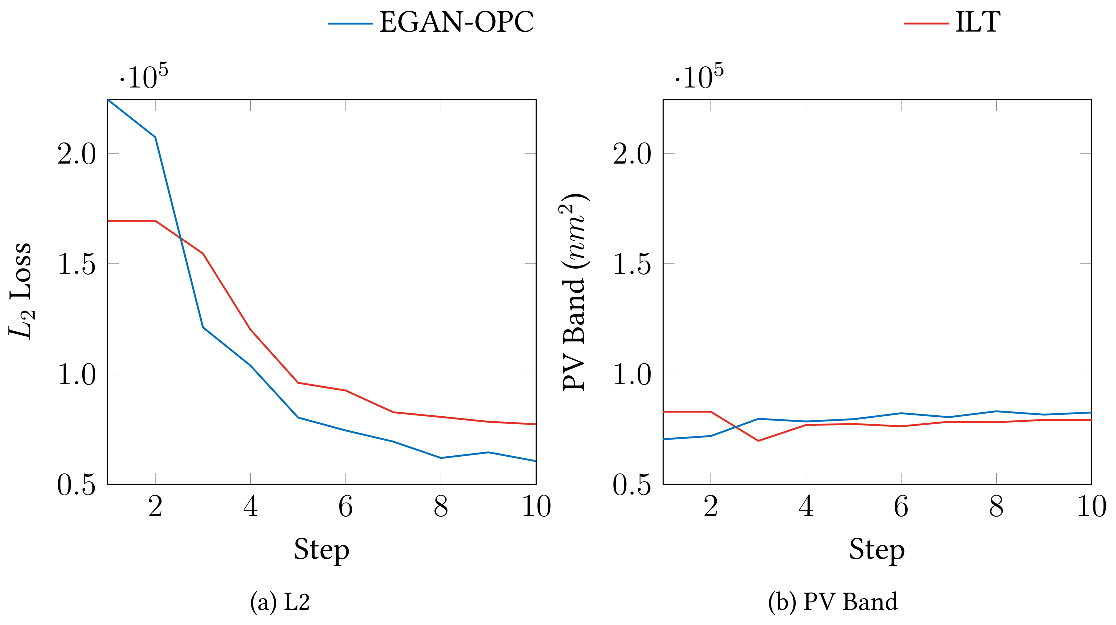


Figure 4.25: Visualization of convergence during ILT refinement.

good initialization offered by the generator, as shown in Figure 4.26.

4.3.4 Summary

In this section, we have proposed a GAN-based mask optimization flow that takes target circuit patterns as input and generates quasi-optimal masks for further ILT refinement. We analyze the specialty of mask optimization problem and design OPC-

Table 4.9: Experiments on larger benchmarks.

Benchmarks	ILT [30]		GAN-OPC		PGAN-OPC		EGAN-OPC	
	L2	PVB (nm^2)	L2	PVB (nm^2)	L2	PVB (nm^2)	L2	PVB (nm^2)
11	94792	125578	90547	137107	89229	121276	93704	119928
12	94604	128306	97969	132929	93454	127551	96634	122060
13	136861	160327	137981	175390	134397	155511	131220	153527
14	69090	79337	62582	94562	58776	85644	57329	85560
15	96961	116039	102759	126157	99830	116728	99926	113971
16	98159	115107	98070	123707	96335	113981	93755	111186
17	79192	91989	76807	98744	71522	94841	70864	94877
18	65572	81503	63573	93219	60372	83718	58383	83568
19	107095	121922	103753	136493	105973	122770	102994	122371
20	62537	78319	61524	90514	57086	81285	56248	79859
Average	90486.3	109842.7	89556.5	120882.2	86697.4	110330.5	86105.7	108690.7
Ratio	1.00	1.00	0.990	1.101	0.958	1.004	**0.952**	**0.990**

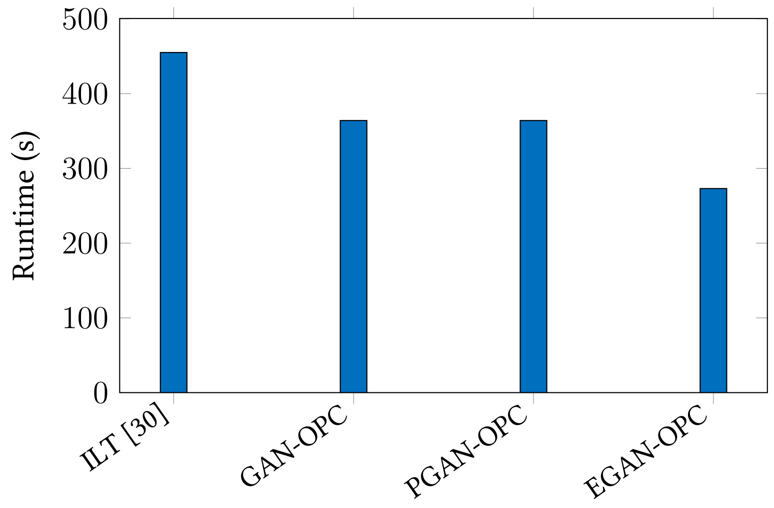


Figure 4.26: Average runtime comparison on larger benchmarks.

oriented training objectives of GAN. Inspired by the observation that ILT procedure resembles gradient descent in back-propagation, we develop an ILT-guided pre-training algorithm that initializes the generator with intermediate ILT results, which significantly facilitates the training procedure. We also enhance the GAN-OPC flow by integrating U-Net and SPSR layers in the generator that ensures better model convergence and mask quality. Experimental results show that our framework not only accelerates ILT but also has the potential to generate better masks through offering better starting points in ILT flow.

Chapter 5

Conclusion and Future Works

In this thesis, we have proposed and discussed a series of machine learning/deep learning solutions for critic challenges in modern chip design for manufacturibility flows. The major contributions include:

- In Chapter 2, we deal with challenging layout hotspot detection problem with dedicated discriminative deep learning technologies. In Section 2.2, we proposed an efficient frequency domain layout representation that compresses layouts into neural network compatible format with minor information loss. Considering the main objectives in hotspot detection problems, we also introduced a provably good training algorithm called *biased-learning* that seeks better trade-off between detection accuracy and false alarms. The compact representation, the efficient learning algorithm and the equipped shallow neural networks contribute to the state-of-the-art machine learning-based hotspot detection solution. In light of non-lithographic layout hotspot detection, in Section 2.3, we study the the defect type of metal-to-via failure, where an adaptive squish representation is proposed for multi-layer layouts. The adaptive squish pattern ensures an evenly distributed layout feature representation which is necessary for neural network applications, while attains the advantage of original squish pattern that is lossless and highly compressed.

- In Chapter 3, we view the hotspot detection challenges from a deeper perspective and we discussed the importance and the overhead obtaining a high quality layout library. In Section 3.2, we formulate a pattern sampling and hotspot detection problem that seeks to optimize the training set (labeled layout pattern library) and the machine learning model simultaneously. We propose an active learning-based solution that considers both the posterior probabilities of pattern instances and the pattern library diversity, which lead to better model generality. The proposed sample-to-sample distance makes the diversity-aware sampling problem convex and hence provides a provable integer relaxation bound that can guide batch active learning configurations. Experiments show the proposed flow can effectively achieve high hotspot detection accuracy with a much smaller group of labeled pattern library.

- In Chapter 4, we discuss the layout pattern generation with generative machine learning models. Section 4.2 covers design pattern generation to benefit early technology node developments. We proposed a TCAE family with a hybrid GAN-TCAE architecture, where the TCAE targets to capture simple design rules and the relationships between auto-learned latent features and layout geometric properties. The GAN, on the other hand, learns legal perturbations that can be applied on latent vectors and generate new layout topology. Different configurations of data flow in GAN-TCAE enables two major functionalities of the framework: massive pattern generation and context-specific pattern generation. Section 4.3 deals with mask pattern generation, which is a key step in chip manufacturing flows. We proposed GAN-OPC family to facilitate ILT-based mask optimization flow. The generator is trained to offer better initialization for legacy OPC engines that can effectively narrows iterations required to generate an optimal mask.

Researches in this thesis have demonstrate the importance and the potential of emerging machine learning techniques being an alternative solution in VLSI back-end design and sign-off flows. New solutions always come along with new challenges:

- In the long logic-to-chip design cycle, there are many more challenges we are facing today to keep the technology on track with Moore's Law. Bringing emerging techniques like deep learning and AI to deal with these challenges is and will be of great interests in both academic and industrial research. Example topics and open problems include (1) how to fix lithography hotspots, (2) how machine learning assist placement and routing in physical design, etc.
- Another research direction comes from machine learning solution itself. An example is the robustness and reliability of machine learning models. Most vision-based deep learning frameworks suffer from heavy performance degradation when exposed to adversarial examples, so will the deep learning models in VLSI design. How to identify adversarial examples in layouts and make the model free from adversarial attacks will be a continuously open topic in future researches.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al. TensorFlow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016.

[2] M. B. Alawieh, Y. Lin, Z. Zhang, M. Li, Q. Huang, and D. Z. Pan. GAN-SRAF: Sub-resolution assist feature generation using conditional generative adversarial networks. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[3] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[4] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning (ICML)*, pages 214–223, 2017.

[5] A. Awad, A. Takahashi, S. Tanaka, and C. Kodama. A fast process variation and pattern fidelity aware mask optimization algorithm. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 238–245, 2014.

[6] A. Awad, A. Takahashi, S. Tanaka, and C. Kodama. A fast process-variation-aware mask optimization algorithm with a novel intensity modeling. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 25(3):998–1011, 2017.

[7] S. Banerjee, K. B. Agarwal, and M. Orshansky. Simultaneous OPC and decomposition for double exposure lithography. In *Proceedings of SPIE*, volume 7973, 2011.

[8] S. Banerjee, Z. Li, and S. 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.

[9] W. Cai, Y. Zhang, and J. Zhou. Maximizing expected model change for active learning in regression. In *IEEE International Conference on Data Mining (ICDM)*, pages 51–60, 2013.

[10] J. P. Cain, M. Fakhry, P. Pathak, J. Sweis, F. E. Gennari, and Y.-C. Lai. Pattern-based analytics to estimate and track yield risk of designs down to 7nm. In *SPIE Advanced Lithography*, volume 10148, 2017.

[11] J. P. Cain, Y.-C. Lai, F. Gennari, and J. Sweis. Methodology for analyzing and quantifying design style changes and complexity using topological patterns. In *Proceedings of SPIE*, volume 9781, 2016.

[12] S. Chakraborty, V. Balasubramanian, Q. Sun, S. Panchanathan, and J. Ye. Active batch selection via convex relaxations with guaranteed solution bounds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(10):1945–1958, 2015.

[13] W.-C. Chang, I. H.-R. Jiang, Y.-T. Yu, and W.-F. Liu. iClaire: A fast and general layout pattern classification algorithm. In *ACM/IEEE Design Automation Conference (DAC)*, pages 64:1–64:6, 2017.

[14] K.-J. Chen, Y.-K. Chuang, B.-Y. Yu, and S.-Y. Fang. Minimizing cluster number with clip shifting in hotspot pattern classification. In *ACM/IEEE Design Automation Conference (DAC)*, pages 63:1–63:6, 2017.

[15] Y. Chen, Y. Lin, T. Gai, Y. Su, Y. Wei, and D. 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.

[16] P. Chien and M. Chen. Proximity effects in submicron optical lithography. In *Optical microlithography VI*, volume 772, pages 35–41. International Society for Optics and Photonics, 1987.

[17] A. Choromanska, M. Henaff, M. Mathieu, G. Ben Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 192–204, 2015.

[18] N. B. Cobb. *Fast optical and process proximity correction algorithms for integrated circuit manufacturing*. PhD thesis, University of California at Berkeley, 1998.

[19] V. Dai, E. K. C. Teoh, J. Xu, and B. Rangarajan. Optimization of complex high-dimensional layout configurations for IC physical designs using graph search, data analytics, and machine learning. In *SPIE Advanced Lithography*, volume 10148, 2017.

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

[21] D. Ding, X. Wu, J. Ghosh, and D. Z. Pan. Machine learning based lithographic hotspot detection with critical-feature extraction and classification. In *IEEE International Conference on IC Design and Technology (ICICDT)*, pages 219–222, 2009.

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

[23] C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

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

[25] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2018.

[26] A. Erdmann, T. Fuehner, T. Schnattinger, and B. 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.

[27] L. Francisco, R. Mao, U. Katakamsetty, P. Verma, and R. Pack. Multilayer cmp hotspot modeling through deep learning. In *Design-Process-Technology Co-optimization for Manufacturability XIII*, volume 10962, page 109620U. International Society for Optics and Photonics, 2019.

[28] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[29] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2):133–168, 1997.

[30] J.-R. Gao, X. Xu, B. Yu, and D. 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.

[31] W. A. Gardner. Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. *Signal Processing*, 6(2):113–133, 1984.

[32] H. Geng, H. Yang, Y. Ma, J. Mitra, and B. Yu. SRAF insertion via supervised dictionary learning. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 406–411, 2019.

[33] H. Geng, H. Yang, B. Yu, X. Li, and X. Zeng. Sparse VLSI layout feature extraction: A dictionary learning approach. In *IEEE Annual Symposium on VLSI (ISVLSI)*, pages 488–493, 2018.

[34] F. E. Gennari and Y.-C. Lai. Topology design using squish patterns, Sept. 9 2014. US Patent 8,832,621.

[35] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9, pages 249–256, 2010.

[36] G. H. Golub and C. F. Van Loan. *Matrix Computations*, volume 3. JHU Press, 2012.

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

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Conference on Neural Information Processing Systems (NIPS)*, pages 2672–2680. 2014.

[39] J. E. Greivenkamp. *Field Guide to Geometrical Optics*. SPIE Press Bellingham, WA, 2004.

[40] A. Gu and A. Zakhor. Optical proximity correction with linear regression. *IEEE Transactions on Semiconductor Manufacturing (TSM)*, 21(2):263–271, 2008.

[41] J. Guo, F. Yang, S. Sinha, C. Chiang, and X. Zeng. Improved tangent space based distance metric for accurate lithographic hotspot classification. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1173–1178, 2012.

[42] M. Gupta, K. Jeong, and A. 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.

[43] P. Gupta. What is process window? *ACM SIGDA Newsletter*, 40(8):1–1, 2010.

[44] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[46] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In *International Conference on Artificial Neural Networks (ICANN)*, pages 44–51, 2011.

[47] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[48] H. Hopkins. The concept of partial coherence in optics. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 208, pages 263–277. The Royal Society, 1951.

[49] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[50] S. Hu and J. Hu. Pattern sensitive placement for manufacturability. In *ACM International Symposium on Physical Design (ISPD)*, pages 27–34, 2007.

[51] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

[52] W.-C. Huang, C.-H. Lin, C.-C. Kuo, C. Huang, J. Lin, J.-H. Chen, R.-G. Liu, Y. C. Ku, and B.-J. Lin. Two threshold resist models for optical proximity correction. In *Optical Microlithography XVII*, volume 5377, pages 1536–1544. International Society for Optics and Photonics, 2004.

[53] B. Jiang, H. Zhang, J. Yang, and E. F. 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] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

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

[56] J. Kuang, W.-K. Chow, and E. 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.

[57] J. Kuang, W.-K. Chow, and E. F. Y. Young. A robust approach for process variation aware mask optimization. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, pages 1591–1594, 2015.

[58] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[59] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *ACM SIGIR Conference*, pages 3–12, 1994.

[60] X. Li, G. Luk-Pat, C. Cork, L. Barnes, and K. Lucas. Double-patterning-friendly OPC. In *Proceedings of SPIE*, volume 7274, 2009.

[61] T. Lin, F. Robert, A. Borjon, G. Russell, C. Martinelli, A. Moore, and Y. Rody. Sraf placement and sizing using inverse lithography technology. In *Optical Microlithography XX*, volume 6520, page 65202A. International Society for Optics and Photonics, 2007.

[62] Y. Lin, M. Li, Y. Watanabe, T. Kimura, T. Matsunawa, S. Nojima, and D. 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.

[63] J. Liu, Y. Ding, J. Yang, U. Schlichtmann, and Y. Shi. Generative adversarial network based scalable on-chip noise sensor placement. In *IEEE International System-on-Chip Conference (SOCC)*, pages 239–242, 2017.

[64] M.-Y. Liu and O. Tuzel. Coupled generative adversarial networks. In *Conference on Neural Information Processing Systems (NIPS)*, pages 469–477, 2016.

[65] R. Luo. Optical proximity correction using a multilayer perceptron neural network. *Journal of Optics*, 15(7):075708, 2013.

[66] N. Ma, J. Ghan, S. Mishra, C. Spanos, K. Poolla, N. Rodriguez, and L. Capodieci. Automatic hotspot classification using pattern-based clustering. In *Proceedings of SPIE*, volume 6925, 2008.

[67] Y. Ma, J.-R. Gao, J. Kuang, J. Miao, and B. Yu. A unified framework for simultaneous layout decomposition and mask optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 81–88, 2017.

[68] C. Mack. *Fundamental Principles of Optical Lithography: The Science of Microfabrication*. John Wiley & Sons, 2008.

[69] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks (ICANN)*, pages 52–59, 2011.

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

[71] T. Matsunawa, S. Nojima, and T. Kotani. Automatic layout feature extraction for lithography hotspot detection based on deep neural network. In *SPIE Advanced Lithography*, volume 9781, 2016.

[72] T. Matsunawa, B. Yu, and D. Z. Pan. Optical proximity correction with hierarchical bayes model. In *Proceedings of SPIE*, volume 9426, 2015.

[73] T. Matsunawa, B. Yu, and D. Z. Pan. Laplacian eigenmaps-and bayesian clustering-based layout pattern sampling and its applications to hotspot detection and optical proximity correction. *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, 15(4):043504–043504, 2016.

[74] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[75] G. E. Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp. 114 ff. *IEEE solid-state circuits society newsletter*, 11(3):33–35, 2006.

[76] J.-S. Park, C.-H. Park, S.-U. Rhie, Y.-H. Kim, M.-H. Yoo, J.-T. Kong, H.-W. Kim, and S.-I. Yoo. An efficient rule-based OPC approach using a DRC tool for 0.18 μm ASIC. In *IEEE International Symposium on Quality Electronic Design (ISQED)*, pages 81–85, 2000.

[77] A. Poonawala and P. Milanfar. Mask design for optical microlithography–an inverse imaging problem. *IEEE Transactions on Image Processing*, 16(3):774–788, 2007.

[78] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[79] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. On the expressive power of deep neural networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[80] G. R. Reddy, K. Madkour, and Y. Makris. Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[81] J. A. T. Robles. Integrated circuit layout design methodology with process variation bands, Aug. 5 2014. US Patent 8,799,830.

[82] N. Rodriguez, L. Song, S. Shroff, K. H. Chen, T. Smith, and W. Luo. Hotspot prevention using cmp model in design implementation flow. In *IEEE International Symposium on Quality Electronic Design (ISQED)*, pages 365–368. IEEE, 2008.

[83] O. Ronneberger, P. Fischer, and T. 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.

[84] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[85] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *International Conference on Machine Learning (ICML)*, pages 839–846, 2000.

[86] R. A. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4:633–648, 2003.

[87] B. Settles. Active learning literature survey. Technical report, 2010.

[88] S. Shang, Y. Granik, and M. Niehoff. Etch proximity correction by integrated model-based retargeting and opc flow. In *Photomask Technology 2007*, volume 6730, page 67302G. International Society for Optics and Photonics, 2007.

[89] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[90] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1874–1883, 2016.

[91] S. Shim, W. Chung, and Y. Shin. Synthesis of lithography test patterns through topology-oriented pattern extraction and classification. In *Proceedings of SPIE*, volume 9053, 2014.

[92] S. Shim and Y. Shin. Topology-oriented pattern extraction and classification for synthesizing lithography test patterns. *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, 14(1):013503–013503, 2015.

[93] M. Shin and J.-H. Lee. Accurate lithography hotspot detection using deep convolutional neural networks. *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, 15(4):043507, 2016.

[94] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015.

[95] Z. Song, X. Ma, J. Gao, J. Wang, Y. Li, and G. R. Arce. Inverse lithography source optimization via compressive sensing. *Optics express*, 22(12):14180–14198, 2014.

[96] Y.-H. Su, Y.-C. Huang, L.-C. Tsai, Y.-W. Chang, and S. 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.

[97] C. Tabery, Y. Zou, V. Arnoux, P. Raghavan, R.-h. Kim, M. Côté, L. Mattii, Y.-C. Lai, and P. Hurat. In-design and signoff lithography physical analysis for 7/5nm. In *SPIE Advanced Lithography*, volume 10147, 2017.

[98] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2(Nov.):45–66, 2001.

[99] R. O. Topaloglu. ICCAD-2016 CAD contest in pattern classification for integrated circuit design space analysis and benchmark suite. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 41:1–41:4, 2016.

[100] R. O. Topaloglu. ICCAD-2016 CAD contest in pattern classification for integrated circuit design space analysis and benchmark suite. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 41:1–41:4, 2016.

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

[102] R. Viswanathan, J. T. Azpiroz, and P. Selvam. Process optimization through model based SRAF printing prediction. In *SPIE Advanced Lithography*, volume 8326, 2012.

[103] G. K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics (TCE)*, 38(1):xviii–xxxiv, 1992.

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

[105] T. Xiao, H. Li, W. Ouyang, and X. Wang. Learning deep feature representations with domain guided dropout for person re-identification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1249–1258, 2016.

[106] Z. Xiao, Y. Du, H. Tian, M. D. F. Wong, H. Yi, H.-S. P. Wong, and H. Zhang. Directed self-assembly (DSA) template pattern verification. In *ACM/IEEE Design Automation Conference (DAC)*, pages 55:1–55:6, 2014.

[107] W. Xiong, J. Zhang, Y. Wang, Z. Yu, and M.-C. Tsai. A gradient-based inverse lithography technology for double-dipole lithography. In *International Conference on Simulation of Semiconductor Processes and Devices*, pages 1–4. IEEE, 2009.

[108] J. Xu, K. N. Krishnamoorthy, E. Teoh, V. Dai, L. Capodieci, J. Sweis, and Y.-C. Lai. Design layout analysis and dfm optimization using topological patterns. In *Proceedings of SPIE*, volume 9427, 2015.

[109] X. Xu, T. Matsunawa, S. Nojima, C. Kodama, T. Kotani, and D. 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.

[110] F. Yang, S. Sinha, C. C. Chiang, X. Zeng, and D. Zhou. Improved tangent space based distance metric for lithographic hotspot classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 36(9):1545–1556, 2017.

[111] H. Yang, W. Chen, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu. Automatic layout generation with applications in machine learning engine evaluation. *arXiv preprint arXiv:1912.05796*, 2019.

[112] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Young. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2019.

[113] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets. In *ACM/IEEE Design Automation Conference (DAC)*, pages 131:1–131:6, 2018.

[114] H. Yang, S. Li, C. Tabery, B. Lin, and B. 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.

[115] H. Yang, Y. Lin, B. Yu, and E. F. Y. Young. Lithography hotspot detection: From shallow to deep learning. In *IEEE International System-on-Chip Conference (SOCC)*, pages 233–238, 2017.

[116] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu. Imbalance aware lithography hotspot detection: A deep learning approach. In *SPIE Advanced Lithography*, volume 10148, 2017.

[117] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu. Imbalance aware lithography hotspot detection: a deep learning approach. *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, 16(3):033504, 2017.

[118] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu. Deepattern: Layout pattern generation with transforming convolutional auto-encoder. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[119] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. 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.

[120] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. 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.

[121] H. Yang, J. Su, Y. Zou, B. Yu, and E. F. Y. Young. Layout hotspot detection with feature tensor generation and deep biased learning. In *ACM/IEEE Design Automation Conference (DAC)*, pages 62:1–62:6, 2017.

[122] W. Ye, M. B. Alawieh, M. Li, Y. Lin, and D. Z. Pan. Litho-GPA: Gaussian process assurance for lithography hotspot detection. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, pages 54–59. IEEE, 2019.

[123] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan. LithoGAN: End-to-end lithography modeling with generative adversarial networks. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[124] W. Ye, Y. Lin, M. Li, Q. Liu, and D. Z. Pan. LithoROC: lithography hotspot detection with explicit ROC optimization. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 292–298, 2019.

[125] P. Yu, S. X. Shi, and D. Z. Pan. Process variation aware OPC with variational lithography modeling. In *ACM/IEEE Design Automation Conference (DAC)*, pages 785–790, 2006.

[126] P. Yu, S. X. Shi, and D. Z. Pan. True process variation aware optical proximity correction with variational lithography modeling and model calibration. *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, 6(3):031004–031004, 2007.

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

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

[129] C. Zhang, Q. Liao, A. Rakhlin, K. Sridharan, B. Miranda, N. Golowich, and T. Poggio. Theory of deep learning III: Generalization properties of SGD. Technical report, Center for Brains, Minds and Machines (CBMM), 2017.

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

[131] W. Zhang, X. Li, S. Saxena, A. Strojwas, and R. Rutenbar. Automatic clustering of wafer spatial signatures. In *ACM/IEEE Design Automation Conference (DAC)*, pages 71:1–71:6, 2013.

[132] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[133] L. Zhuang, J. Xu, M. Tsai, Q. W. Liu, E. Yang, Y. Zhang, J. Sweis, C. Lai, and H. Ding. A novel methodology of process weak-point identification to accelerate process development and yield ramp-up. In *IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 852–855, 2016.