Attacking a CNN-based Layout Hotspot Detector Using Group Gradient Method

Haoyu Yang
Chinese University of Hong Kong

Shifan Zhang
Chinese University of Hong Kong

Kang Liu
New York University

Siting Liu
Chinese University of Hong Kong

Benjamin Tan
New York University

Ramesh Karri
New York University

Siddharth Garg
New York University

Bei Yu
Chinese University of Hong Kong

Evangeline F.Y. Young
Chinese University of Hong Kong

Abstract

Deep neural networks are being used in disparate VLSI design automation tasks, including layout printability estimation, mask optimization, and routing congestion analysis. Preliminary results show the power of deep learning as an alternate solution in state-of-the-art design and sign-off flows. However, deep learning is vulnerable to adversarial attacks. In this paper, we examine the risk of state-of-the-art deep learning-based layout hotspot detectors under practical attack scenarios. We show that legacy gradient-based attacks do not adequately consider the design rule constraints. We present an innovative adversarial attack formulation to attack the layout clips and propose a fast group gradient method to solve it. Experiments show that the attack can deceive the deep neural networks using small perturbations in clips which preserve layout functionality while meeting the design rules. The source code is available at https://github.com/phdyang007/dlhsd/tree/dct_as_conv.

1 Introduction

Deep neural networks such as convolutional neural networks (CNNs) are being investigated for use in various VLSI design automation tasks, including layout printability estimation[1, 2], and mask optimization [3]. Instead of human-driven engineering of input features for prediction and classification tasks, CNNs can automatically discover/learn features for complex applications. Researchers have demonstrated the potential of CNNs as an alternative solution in state-of-the-art design and sign-off flows. In design-for-manufacturing, a CNN-based hotspot detector speeds-up the estimation of the risk of manufacturing a defective layout without going through time-consuming OPC and lithographic simulation. However, CNNs are *fragile*. CNN-based solutions are vulnerable to adversarial attacks [4–7], and this vulnerability has been extended to encompass CNN-based hotspot detectors in the VLSI context [8].

CNNs are designed as a multi-level stack of non-linear functions (e.g., ReLU and Softmax) [9–11] to learn a prior probability over the input space [7]. Hence the CNN might output unexpected probabilities on the region of the input space that has no training data. These "empty" regions are filled with objects using label-preserving transformations. These might be close to the original objects in the training data in terms of the pixel value. A trained CNN model may make incorrect predictions on an instance $\boldsymbol{x} + \boldsymbol{r}$ that is perturbed from the training data $\boldsymbol{x}$ by a tiny amount $\boldsymbol{r}$ with $||\boldsymbol{r}||_2 \leq \epsilon$. These *adversarial examples* can thus fool the CNN.

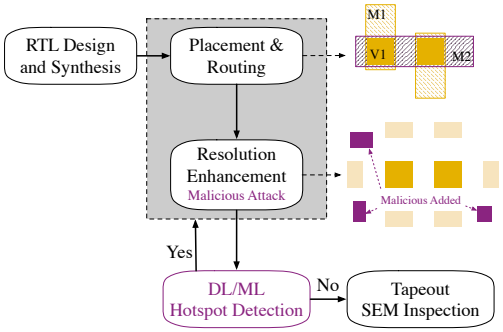


Figure 1: Malicious attack poses risks in chip design flow.

To robustify the CNN models, algorithms have been proposed to generate adversarial examples, allowing designers to probe and quantify robustness under adversarial settings [4–6]. These include L-BFGS attack [12], fast gradient sign methods (FGSM) [4], and basic iterative methods (BIM) [6]. In [12], the authors propose a framework to generate adversarial examples by solving an optimization problem using L-BFGS method [13] with line searching on constant parameters in its objective, and they also discuss the generality and root cause of adversarial examples. [4] introduces FGSM to generate adversarial examples in a single step gradient-based attack. This approach adds to the original input $\boldsymbol{x}$ a noise matrix that is generated from the sign of the gradient of classification loss with respect to input pixel values $\epsilon \text{sign}(\nabla_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, l))$, where l is the ground truth label of $\boldsymbol{x}$ and J_θ corresponds to the classification loss of the trained CNN. The procedure can be viewed as a one step gradient ascent. FGSM can be extended to target a given class by adjusting the perturbation term as $-\epsilon \text{sign}(\nabla_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, l'))$. l' is the target class [14]. BIM [6] is designed for scenarios when data are indirectly fed into a CNN by certain sensors. BIM forces the perturbed image $\boldsymbol{x}'$ to be in the L_∞ α-neighbourhood of $\boldsymbol{x}$ by clipping $\boldsymbol{x}'$ with $\text{clip}_{\boldsymbol{x},\alpha}(\boldsymbol{x}') = \min\{255, \boldsymbol{x} + \alpha, \max\{0, \boldsymbol{x} - \alpha, \boldsymbol{x}'\}\}$, where

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431571

$\boldsymbol{x}' = \boldsymbol{x} + \epsilon \text{sign}(\nabla_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, l'))$ is the ordinary FGSM. BIM can be modified to iterative least-likely class methods by replacing FGSM with attacking on a certain class.

Although state-of-the-art attacks can fool mainstream image-classification CNNs trained on oft-used image data [15, 16], the attacks are formulated to manipulate individual pixels. At first glance, CNN-based hotspot detectors appear insulated from adversarial attacks, as pixel-by-pixel modifications are not meaningful in VLSI layouts due to design rules constraining valid design artifacts. However, [8] demonstrated that immunity to adversarial input attacks does not hold. By converting the pixel-based gradient method (PGM) into a "DRC-clean" version, they successfully fooled a layout hotspot detector by adding fraudulent sub-resolution assist features (SRAFs) in the original via-SRAF designs, posing risks in chip design flow (see Figure 1). They converted hotspot clips to images and divided them into blocks of pixels where the ascending gradients for pixels in a given block are summed to calculate the gradient for that block. Fraudulent SRAFs are inserted into blocks with large ascending summed gradients, considering blocks where SRAFs can be added without violating design rules. There are two aspects that might drop the success rate of the attack: (1) all pixels within the added SRAF patterns change in the *same* direction making the pixel-based gradient methods less optimal and (2) the attack flow *only* inserts SRAFs, whereas more effective attacks may involve SRAF removal.

To address emerging threats to deep learning in EDA, we investigate the risk to state-of-the-art deep learning-based layout hotspot detectors under practical attacking scheme. We propose an attack algorithm that is carefully designed for layout hotspot detectors, where we tackle the drawbacks of pixel-based gradient methods with a novel group gradient method (GGM). This new attack supports both insertion and removal of perturbed SRAF patterns. The proposed method is faster than prior approaches and offers a higher attack success rate due to the efficient group gradient ascension. Studying potential attacks will assist in the design of robust hotspot detectors. This paper makes three contributions:

- A demonstration that state-of-the-art gradient-based adversarial attack solutions are limited on layout hotspot detectors due to the existence of design rules, and that DRC-clean attacks result in sub-optimal adversarial examples.
- A novel fast group gradient method that significantly alleviates the optimality deviation problem of existing hotspot detector attacks as well as increases the solution space by allowing removal of SRAF patterns.
- A comprehensive evaluation on a state-of-the-art hotspot detector [1] for legacy node via patterns with model-based SRAFs, showing that the trained model is vulnerable to GGM generated adversarial layout samples.

The paper is organized as follows. Section 2 introduces concepts related to adversarial attacks on hotspot detectors. Section 3 presents our proposed method, algorithm, and discussions of the overall flow. Section 4 covers experimental work and results and Section 5 concludes the paper.

2 Preliminaries

In this section, we will introduce terminologies related to adversarial attacks and formulate this attack on a CNN-based hotspot detector.

2.1 Via Layout Hotspot Detection

As illustrated in Figure 1, learning-based layout hotspot detectors are used to facilitate the VLSI back-end design and sign-off flow by estimating layout printability without OPC and lithographic simulation. Learning model predicted hotspots will be fixed by feeding back to previous design stage, while hotspot free designs are going into followup manufacturing steps. As a case study, we focus on the CNN models that predict potential defects in via patterns which contains vias and model generated SRAFs. A trained CNN will output the probability of a via clip being a hotspot. *Clearly, missed detection will result in defects after tapeout, which significantly challenges yield and hence motivates the research on attacks of machine learning-based hotspot detectors.*

2.2 Adversarial Attack on a CNN-Based Hotspot Detector

Adversarial input perturbation attacks pose a challenge to the robustness of DL-based systems [4, 6, 12, 14]. Motivated by promising successes in incorporating machine learning throughout the CAD flow [17], recent work by Liu *et al.* [8] has shown that adversarial attacks extend beyond general-purpose image classification domains into more constrained, esoteric applications, including lithographic hotspot detection. This raises the need for robustness studies of DL, especially in constrained settings where perturbations need to be *"semantically meaningful"* [8].

In this work, we adopt a similar attack setting to [8], whereby a malicious designer attempts to exploit the novel attack vector in the supply chain [18] provided by incorporating DL-in-the-loop. By exploiting robustness shortcomings in DL-based hotspot detection, the malicious designer can sabotage the design flow, stealthily—their aim is to prepare hotspot-laden layouts that satisfy design rules but harbor lithographic defects. By carefully perturbing the layout, the malicious designer can make a hotspot design appear non-hotspot to the hotspot detector.

Adversarial attacks on a hotspot detector are either white-box or black-box depending on how much information is accessible to an attacker. In a *white-box attack*, the attacker has access to details of the trained CNN including the training set, network architecture, hyper-parameters, and neuron weights. In a *black-box attack*, the attacker can only query the trained detector and access the prediction.

Here we focus on white-box attacks as transferability of adversarial examples make them applicable to black-box attacks [19]. The attack objectives include making the CNN predict a sample of class A as class $\bar{A}$ or predict $\bar{A}$ as A. We call these *false-negative* and *false-positive* attacks, respectively. We focus on true-positive attacks that make CNNs predict hotspots as non-hotspots, resulting in a yield loss. Thus, an ideal adversarially perturbed version of an original layout instance should: (1) *be as close to the original instance as possible (for stealthiness/imperceptibility)*, (2) *be in the same class as the original instance*, (3) *make the trained CNN incorrectly classify the adversarial version* and (4) *retain circuit functionality and satisfy design rules*. Otherwise, the adversarial attacks can be discovered using design rule checks and functional analysis.

2.3 Attack Metrics

We define the following terms to evaluate an attack method.

Definition 1 (Accuracy [20]). The ratio between the number of successfully predicted hotspot clips and the total number of hotspot clips.

Definition 2 (Adversarial Layout Example). A hotspot layout clip that can not be correctly identified by a trained neural network model due to small DRC-clean and function preserving perturbations.

With the above definitions, we formulate the problem of attacking a CNN-based hotspot detector as follows.

Problem 1 (AttackHSD). Given a set of hotspot patterns and a CNN-based hotspot detector (under white-box settings, i.e., with the CNN architecture and neuron weights accessible), the objective of AttackHSD is to make the minimal DRC-clean SRAF perturbations on hotspot patterns such that they are still hotspots and the detection accuracy on the perturbed hotspot dataset is minimized.

3 Algorithms

In this section, we will discuss the non-optimality of the pixel-based gradient method (PGM) and show that the group gradient method alleviates them.

3.1 Pixel-based Gradient Method (PGM) is not Optimal

PGM for creating adversarial examples [4, 5, 14] focus on individual pixels as attack targets. Using an example of 2-class classification problem, the objectives are:

$$\min \ ||\boldsymbol{R}||_F^2, \tag{1a}$$

$$\text{s.t. } f(\boldsymbol{X}+\boldsymbol{R};\boldsymbol{W}) < 0, \tag{1b}$$

$$f(\boldsymbol{X};\boldsymbol{W}) > 0, \tag{1c}$$

where $\boldsymbol{X}$ is the original image, $\boldsymbol{R}$ corresponds to adversarial perturbation and f is some trained binary classifier parameterized with $\boldsymbol{W}$. The sign of f indicates the class its input belongs to. f can be the value difference between the softmax output of a 2-class neural network. PGM attack applies gradient descent over the loss with respect to the incorrect label, i.e.,

$$\boldsymbol{R} = -\gamma \frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}}, \tag{2}$$

$$\boldsymbol{X} = \boldsymbol{X} + \boldsymbol{R}, \tag{3}$$

where γ is a small coefficient that controls the perturbation level on each pixel. Equation (2) and Equation (3) can be solved iteratively until the trained model makes a wrong prediction. In the VLSI context, gradient-based methods cannot generate adversarial samples directly due to the DRC violations that they engender. Liu *et al.* [8] tackle this by scanning over layout clips with valid SRAF patterns. The gradients of all pixels in a valid SRAF region are summed as the criteria to pick candidate perturbations $\boldsymbol{R}_i$'s with i in Equation (4):

$$i = \arg\max_k \sum_{(x,y)\in\mathcal{R}_k} \frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}(x,y)}, \tag{4}$$

where $\mathcal{R}_k$ corresponds to the SRAF region in the perturbation matrix $\boldsymbol{R}_k$. (x, y) is the coordinate representing each pixel. Multiple i's might be selected according to the sum of gradient value in a descending order. Within each inserted SRAF, it is inevitable that some of the pixels in that SRAF do not contribute towards flipping the label. This is because an adversarial sample might change to non-adversarial after DRC legalization. Also, Equation (4) is not designed to "remove" SRAFs as an attack option. In light of these shortcomings, we propose the group gradient method for efficient and effective adversarial layout sample generation.

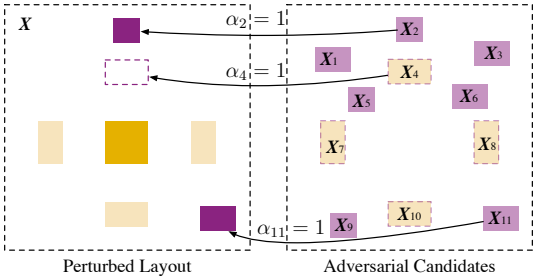


Figure 2: Illustration of the proposed attack scheme as in Formula (5), with a solution of $\alpha_2 = 1, \alpha_4 = 1$ and $\alpha_{11} = 1$.

3.2 Group Gradient Attack is Our Proposal

The group gradient attack targets layouts with adversarial DRC-clean SRAF patterns, as shown in Formula (5).

$$\min_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\alpha}) = ||\sum_i \alpha_i \boldsymbol{X}_i||_F^2, \tag{5a}$$

$$\text{s.t. } f(\boldsymbol{X} + \sum_i \alpha_i \boldsymbol{X}_i; \boldsymbol{W}) < 0, \tag{5b}$$

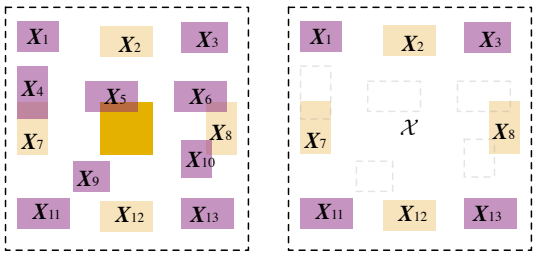
$$\alpha_i + \alpha_j \le 1, \forall i,j \in \mathcal{C}, \tag{5c}$$

$$\alpha_i \in \{0,1\}, \forall i, \tag{5d}$$

where $\boldsymbol{X}_i$'s are candidate SRAFs that can be added to or subtracted from the original input $\boldsymbol{X}$, $\boldsymbol{\alpha}$ contains a set of coefficients that control whether a perturbation SRAF will be selected for adversarial sample generation and $\mathcal{C}$ is the conflict set determined by design rules. Similar to PGM adversarial attack objectives, Equation (5a) tries to minimize the perturbation as much as possible when generating adversarial examples. We also have Equation (5b) to ensure an adversarial sample can successfully fool the neural networks defined as $f(\cdot;\boldsymbol{W})$. Equation (5c) guarantees the perturbed SRAFs will not violate design rules with existing SRAFs, while Equation (5d) makes the problem formulation practical, i.e., there will be either a shape (represented as 1) or spacing (represented as 0) in the layout. We visualize the perturbation scheme in Figure 2 with each perturbation matrix $\boldsymbol{X}_i$ containing one SRAF shape.

3.2.1 Generate Candidates for Perturbation. Before deriving a mathematical solution for this formulation, we introduce the perturbation candidate generation flow that produces well-distributed and compliant SRAF insertion candidates. From Formula (5), a good adversarial sample generator relies on SRAF perturbation candidates. We propose Algorithm 1 to generate DRC-clean perturbation candidates that are evenly distributed around via shapes (illustrated in Figure 3). The algorithm has two parts: (1) Identify SRAFs in $\boldsymbol{X}$ as removal candidates (lines 1–8); (2) Scan over the clip and check for insertion candidates (lines 12–17) that will not intersect with each other or existing shapes in $\boldsymbol{X}$. These are added into $\mathcal{X}$ (lines 19–21).

3.2.2 Numerical Optimization. Formula (5) is not easy to solve as it is non-convex, discrete, and has a complex objective function and constraints. To apply suitable numerical solutions, we relax



(a) Mass Candidates (b) Legal Candidates

Figure 3: Visualization of perturbation candidate generation $\mathcal{X} = \{X_1, X_2, X_3, X_7, X_8, X_{11}, X_{12}, X_{13}\}$. Due to design rule violation with existing shapes $\{X_4, X_5, X_6, X_9, X_{10}\}$ will not be included in the perturbation candidate set $\mathcal{X}$.

Algorithm 1 Generate Candidate for Perturbation

Input: $\boldsymbol{X}$, design rule, search step s;
Output: $\mathcal{X}$;
1: $\mathcal{X} \leftarrow \emptyset$;
2: Generate a set of all shapes $\mathcal{S}$ in $\boldsymbol{X}$;
3: **for** S in $\mathcal{S}$ **do**
4: **if** S is a SRAF **then**
5: $\boldsymbol{X}_t \leftarrow \boldsymbol{X}$ with all shapes removed except S;
6: $\mathcal{X} = \mathcal{X} \cup \{-\boldsymbol{X}_t\}$;
7: $x \leftarrow 0, y \leftarrow 0$;
8: **for** $x < \boldsymbol{X}$.shape[1] **do**
9: **for** $y < \boldsymbol{X}$.shape[0] **do**
10: $h \leftarrow \mathcal{U}(40,90)$;
11: **if** $h > 40$ **then**
12: $w \leftarrow 40$;
13: **else**
14: $w \leftarrow \mathcal{U}(40,90)$;
15: $\boldsymbol{X}_t \leftarrow$ generate a matrix with size $\boldsymbol{X}$.shape[1] $\times$ $\boldsymbol{X}$.shape[0] and contains solo rectangle shape defined by (x, y) and $(x + h, y + w)$;
16: **if** $\boldsymbol{X} + \boldsymbol{X}_t$ is DRC clean **then**
17: $\boldsymbol{X} \leftarrow \boldsymbol{X} + \boldsymbol{X}_t$; $\mathcal{X} = \mathcal{X} \cup \{\boldsymbol{X}_t\}$;
18: $y \leftarrow y + s$;
19: $x \leftarrow x + s$;
20: **return** $\mathcal{X}$;

Equation (5d) into Equation (6) and make Formula (5) continuous,

$$0 \le \alpha_i \le 1, \forall i. \tag{6}$$

Using design rule constraints when generating candidates for perturbation ensures an adversarial example that is DRC-clean, allowing Equation (5c) to be trivial. This simplifies Formula (5),

$$\min_{\boldsymbol{\alpha}} \mathcal{L}_{\text{cont}}(\boldsymbol{\alpha}) = || \sum_i \alpha_i \boldsymbol{X}_i ||_F^2, \tag{7a}$$

$$\text{s.t. } f(\boldsymbol{X} + \sum_i \alpha_i \boldsymbol{X}_i; \boldsymbol{W}) < 0, \tag{7b}$$

$$0 \le \alpha_i \le 1, \forall i, \tag{7c}$$

which requires approximately $\mathcal{O}(n^2)$ FLOPs and $\mathcal{O}(n^2)$ MEMOPs with $|| \sum_i \alpha_i \boldsymbol{X}_i ||_F^2$ as objectives. Here n is the dimension of an input layout image and usually exceeds 1,000 to have enough information for printability estimation [20, 21]. This brings much more overhead than a medium sized CNN.

Because the goal is to generate adversarial examples, Equation (5b) is somehow playing a more important role than Equation (5a), which leaves space to circumvent the complicated objectives. We hence replace Equation (5a) with an approximation in Formula (8), which reduces the computational overhead of the objectives knowing the number of $\boldsymbol{X}_i$'s is around 100 with controllable error induced from relaxation from Equation (7a) to Equation (8a), as claimed in the following theorem.

$$\min_{\boldsymbol{\alpha}} \mathcal{L}_{\text{sim}}(\boldsymbol{\alpha}) = ||\boldsymbol{\alpha}||_2^2, \tag{8a}$$

$$\text{s.t. } f(\boldsymbol{X} + \sum_i \alpha_i \boldsymbol{X}_i; \boldsymbol{W}) < 0, \tag{8b}$$

$$0 \le \alpha_i \le 1, \forall i, \tag{8c}$$

Theorem 1. *Let $\boldsymbol{\alpha}^*_{cont}$ and $\boldsymbol{\alpha}^*_{sim}$ be the optimal solution of Problem (7) and Problem (8), respectively, then we have,*

$$\mathcal{L}_{cont}(\boldsymbol{\alpha}^*_{cont}) \le \mathcal{L}_{cont}(\boldsymbol{\alpha}^*_{sim}), \tag{9}$$

and,

$$\begin{aligned} &\mathcal{L}_{cont}(\boldsymbol{\alpha}^*_{sim}) - \mathcal{L}_{cont}(\boldsymbol{\alpha}^*_{cont}) \\ &\le ||\boldsymbol{\alpha}^*_{sim}||_0^2 \cdot ||\boldsymbol{X}_\delta||_F^2 - ||\boldsymbol{\alpha}^*_{cont}||_0^2 \cdot ||\boldsymbol{X}_\xi||_F^2, \end{aligned} \tag{10}$$

where $\delta = \text{argmax}_i |\boldsymbol{e}^\top \boldsymbol{X}_i \boldsymbol{e}|$ and $\xi = \text{argmin}_i |\boldsymbol{e}^\top \boldsymbol{X}_i \boldsymbol{e}|$.

The non-convexity and non-linearity of f makes it impossible to obtain a closed form solution of Formula (8). We adopt Lagrangian relaxation by embedding Equation (5b) into Equation (8a) yielding,

$$\min_{\boldsymbol{\alpha}} \mathcal{L}_{\text{lag}}(\boldsymbol{\alpha}, \lambda) = ||\boldsymbol{\alpha}||_2^2 + \lambda f(\boldsymbol{X} + \sum_i \alpha_i \boldsymbol{X}_i; \boldsymbol{W}), \tag{11a}$$

$$\text{s.t. } \lambda \ge 0, 0 \le \alpha_i \le 1, \forall i, \tag{11b}$$

which can be solved by descending the gradient of $\mathcal{L}_{\text{lag}}$ with respect to $\boldsymbol{\alpha}$ and λ. To ensure Equation (6) will always hold during optimization, we introduce a sigmoid that forces α_i's inside $[0, 1]$

$$\alpha_i = \frac{1}{1 + e^{-\beta_i}}, \beta_i \in \mathbb{R}, \forall i. \tag{12}$$

The variables can be updated as follows,

$$\begin{aligned} \beta_i^{(t+1)} &= \beta_i^{(t)} - \frac{\partial \mathcal{L}_{\text{lag}}^{(t)}}{\partial \alpha_i^{(t)}} \frac{\partial \alpha_i^{(t)}}{\partial \beta_i^{(t)}} \\ &= \beta_i^{(t)} - (2\alpha_i^{(t)} + \lambda \frac{\partial f}{\partial \alpha_i^{(t)}}) \alpha_i^{(t)} (1 - \alpha_i^{(t)}), \forall i, \end{aligned} \tag{13}$$

$$\lambda^{(t+1)} = \lambda^{(t)} - f(\boldsymbol{X} + \sum_i \alpha_i^{(t)} \boldsymbol{X}_i; \boldsymbol{W}), \tag{14}$$

where $\cdot^{(t)}$ refers to the variable in the t-th updating step.

Since we consider all pixels in each perturbation candidate region during optimization, we call this adversarial sample generation method the *group gradient method (GGM)*. We present the adversarial via layout generation flow in Algorithm 2. Given a hotspot pattern $\boldsymbol{X}$ that is correctly identified by f, we generate a set of candidate SRAF perturbations (insertion and removal) (line 1). Decision variables $\boldsymbol{\beta}$ and λ are initialized for good convergence (line 2) and are updated with gradient descent (lines 3–16). When f starts to produce negative predictions, we generate the adversarial sample $\boldsymbol{X}'$ by gradually

adding perturbations with largest $\alpha_k^{(t)}$ to $\boldsymbol{X}$ until f makes a wrong prediction on the perturbed $\boldsymbol{X}'$ (lines 4–15).

Algorithm 2 Group Gradient Method

Input: A trained neural network model $f(\cdot;\boldsymbol{W})$, a hotspot pattern $\boldsymbol{X}$ that is correctly identified by f, maximum number of SRAFs can be changed $s_{\max}$, maximum optimization iteration $t_{\max}$;
Output: An adversarial pattern $\boldsymbol{X}'$;

1: Generate SRAF perturbation candidates $\mathcal{X}=\{\boldsymbol{X}_1, \boldsymbol{X}_2, ..., \boldsymbol{X}_m\}$ from Algorithm 1;
2: Initialize $\boldsymbol{\beta} \in \mathbb{R}^m$ and $\lambda \in \mathbb{R}^+$; ▷ According to (11) & (12)
3: **for** $t = 1, 2, ..., t_{\max}$ **do**
4: **if** $f(\boldsymbol{X} + \sum_i \alpha_i^{(t)} \boldsymbol{X}_i; \boldsymbol{W}) < 0$ **then**
5: $\boldsymbol{X}' \leftarrow \boldsymbol{X}$;
6: $\mathcal{A}^{(t)} \leftarrow \{\alpha_i^{(t)}, i = 1, 2, ...m\}$;
7: **for** $s = 1, 2, ..., s_{\max}$ **do**
8: $k \leftarrow \arg\max \mathcal{A}^{(t)}$;
9: $\mathcal{A}^{(t)} = \mathcal{A}^{(t)} / \{\alpha_k^{(t)}\}$;
10: $\boldsymbol{X}' \leftarrow \boldsymbol{X}' + \boldsymbol{X}_k$;
11: **if** $f(\boldsymbol{X}') < 0$ **then**
12: **return** $\boldsymbol{X}'$;
13: Update $\boldsymbol{\beta}$ and λ; ▷ According to (13) & (14);

3.2.3 Discussion. In the group gradient method, Algorithms 1 and 2 work together to generate adversarial examples with minor changes to original SRAF-via patterns. The initial values of $\boldsymbol{\beta}$ and λ significantly affect the convergence of Algorithm 2. Because the perturbation is expected to be included in $\boldsymbol{X}$ gradually, $\boldsymbol{\alpha}$ should be close to zero at the beginning of the optimization flow. It should be also noted that we do not conduct explict rounding of $\boldsymbol{\alpha}$ when determining the final perturbation candidates, because we will gradually add these perturbations in the design until the prediction label flips. Therefore, we require $\boldsymbol{\beta}$ to be initialized to a negative value while making sure that it does not slow down convergence by being too negative for small gradients of the sigmoid function. The initialization of $\boldsymbol{\alpha}$ almost guarantees that $f(\boldsymbol{X} + \sum_i \alpha_i \boldsymbol{X}_i; \boldsymbol{W}) \approx f(\boldsymbol{X}; \boldsymbol{W})$ is positive at the beginning, which hence leads to $\lambda^{(t+1)} < \lambda^{(t)}$ when t is small. Since Equation (13) has shown the risk of failure when λ is too small to propagate the gradient of f back to $\boldsymbol{\beta}$, it is recommended to have λ initialized with a relatively large positive value. This makes Equation (11a) more like a general minimization problem with $\mathcal{L}_2$ regularization instead of the method of Lagrangian multipliers. Theorem 1 also tells us that the relaxation gap from $\mathcal{L}_{\text{cont}}$ to $\mathcal{L}_{\text{sim}}$ will be efficiently narrowed if (1) we insert as few SRAFs as possible, according to Equation (10) (2) inserted perturbations are with fewer shape variations.

3.3 Overall Flow

Figure 4 depicts the flow of the group gradient method, where a hotspot pattern combined with its perturbation candidates feed into the trained CNN model. The optimization engine back-propagates the error gradients to the combination coefficients β_i's which finally guide the insertion of perturbation candidates. This flow has three advantages compared to the PGM approach [8]. (1) Candidate perturbations are generated by scanning over the entire clip ensuring a comprehensive solution space. (2) GGM optimizes toward DRC-clean

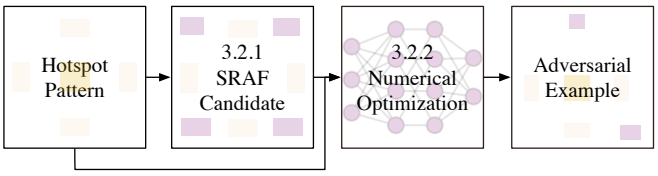
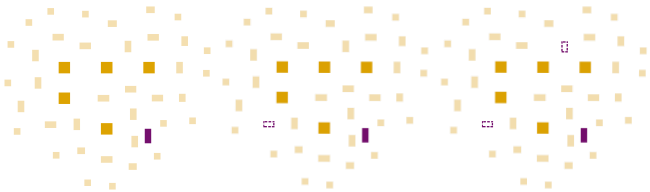


Figure 4: Overall flow of the proposed GGM.



(a) Step-1 ($f = 0.4984$) **(b) Step-2** ($f = 0.0835$) **(c) Step-3** ($f = -1.043$)

Figure 5: Step view of adversarial layout generation.

perturbation circumventing post-processing and potential deviation from optimality. (3) Gradient back-propagation and perturbation candidate determination steps make the framework robust when more changes are used to create adversarial layout examples.

4 Experimental Results

In this section, we present the results and some ablation studies that show effectiveness and efficiency of the group gradient method.

We implement GGM and PGM using `Python` and `Tensorflow` [22]. We test these methods on an Intel platform with GTX 1080 Ti graphics processing unit. For the via hotspot data, we use legacy node via designs that are verified and simulated using Mentor Graphics `Calibre` Design For Manufacturability tool suite [23]. We generate adversarial examples based on four groups of 400 hotspot clips and a CNN model trained on 34356 via layout clips with 2454 hotspots. We use the same CNN architecture and training strategies as in [20]. When generating an adversarial example, we initialize β_i's with -10 and λ with 10^5 for reasons discussed in Section 3.2.3. We set $t_{\max}$ to 6 and $s_{\max}$ to $\{1, 2, 3\}$ to evaluate the performance of the method under different perturbation tolerances.

We launch adversarial attacks on four hotspot groups using GGM. Results for the three SRAF perturbation constraints $s_{\max}$ are listed in Table 1. Column "ID" shows the four groups of hotspot patterns. Column "GGM" corresponds to results using GGM and column "PGM" refers to PGM in [8]. Minor column "Acc (%)" lists the hotspot detection accuracy (definition 1) and minor column "AE" corresponds to the number of adversarial layout examples produced (definition 2).

GGM can deceive the trained CNN model easily by altering the original hotspot layout with at most one SRAF. The detection accuracy drops from 83.94% to 61.81%. In contrast, using PGM [8] drops the detection accuracy by ~ 10% and generates only half as many adversarial examples as the GGM. This is because PGM is non-optimal when legalizing DRC-clear SRAFs according to pixel-based gradients. When we gradually increase the allowable perturbations from $s_{\max} = 1$ to $s_{\max} = 3$, GGM and PGM exhibit an increased attack success rate. However, GGM is superior as it results in > 10% more accuracy drop relative to PGM. Figure 5 shows the step-by-step

Table 1: Group gradient method (GGM) vs. state-of-the-art pixel gradient method (PGM [8]).

ID	Origin	GGM ($s_{\max}=1$)		PGM ($s_{\max}=1$)		GGM ($s_{\max}=2$)		PGM ($s_{\max}=2$)		GGM ($s_{\max}=3$)		PGM ($s_{\max}=3$)	
	Acc (%)	Acc (%)	AE	Acc (%)	AE	Acc (%)	AE	Acc (%)	AE	Acc (%)	AE	Acc (%)	AE
1	81.50	61.50	73	71.00	38	51.50	110	64.00	63	46.25	131	56.25	93
2	89.25	64.50	92	78.75	39	52.25	137	66.75	87	45.75	161	59.50	114
3	85.25	64.75	78	74.50	40	53.75	119	65.25	76	47.00	146	55.00	117
4	79.75	56.50	91	69.50	40	47.75	125	60.50	74	44.00	140	53.50	101
Avg.	83.94	**61.81**	**83.50**	73.44	39.25	**51.31**	**122.80**	64.13	75.00	**45.75**	**144.50**	56.06	106.3

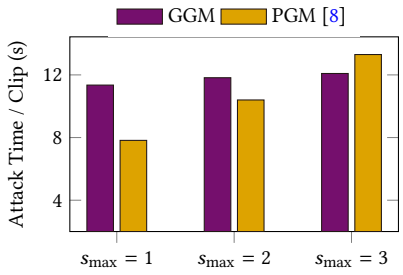


Figure 6: Runtime comparison: GGM vs PGM.

generation of an adversarial via layout where SRAFs in the original design are perturbed according to optimized β_i's.

Figure 6 summarizes the efficiency of GGM and PGM using average generation time per clip. PGM is faster than GGM when $s_{\max}=1$ with adversarial example generation time of 7.83s per clip compared to 11.35s per clip for GGM. However, GGM exhibits robustness with increasing $s_{\max}$. This is because the overall runtime overhead in GGM is dominated by updating $\boldsymbol{\beta}$ and λ, which is upper-bounded by $t_{\max}$. PGM, however, requires more efforts to search for effective perturbation locations, whose computational overhead scales up prominently.

5 Conclusion

In this paper, we examine the risks of deep learning-based lithography hotspot detectors assuming a practical adversarial attack scenario. We explain that adversarial example generation employing a conventional pixel-based gradient method deviates from the optimal when making legal perturbations. To present an efficient and successful attack, we recommend the group gradient method that makes DRC clean perturbations by solving an unconstrained optimization problem with an objective function that is differentiable. This can produce successful attacks with minimal perturbations. Experimental results show the superiority of the attack solution over the pixel gradient method and confirm the susceptibility of state-of-the-art CNN-based hotspot detectors. We expect this study will spur research in defenses against adversarial layout examples culminating in robust machine learning solutions in VLSI design and sign-off flow.

6 Acknowledgment

This work is partially supported by The Research Grants Council of Hong Kong SAR (No. CUHK14209420).

References

[1] 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 TCAD*, vol. 38, no. 6, pp. 1175–1187, 2019.
[2] W. Ye, Y. Lin, M. Li, Q. Liu, and D. Z. Pan, "LithoROC: lithography hotspot detection with explicit ROC optimization," in *Proc. ASPDAC*, 2019, pp. 292–298.
[3] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. DAC*, 2018, pp. 131:1–131:6.
[4] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. ICLR*. IEEE, 2015, pp. 1–11.
[5] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proc. CVPR*, 2016, pp. 2574–2582.
[6] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proc. ICLR*, 2017.
[7] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
[8] K. Liu, H. Yang, Y. Ma, B. Tan, B. Yu, E. F. Y. Young, R. Karri, and S. Garg, "Adversarial Perturbation Attacks on ML-Based CAD: A Case Study on CNN-Based Lithographic Hotspot Detection," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 5, Aug. 2020.
[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. CVPR*, 2015, pp. 1–9.
[10] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. ECCV*, 2014, pp. 818–833.
[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. ICML*, 2010, pp. 807–814.
[12] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.
[13] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.
[14] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.
[15] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.
[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
[17] A. B. Kahng, "Machine Learning Applications in Physical Design: Recent Results and Directions," in *International Symposium on Physical Design - ISPD '18*. Monterey, California, USA: ACM, 2018, pp. 68–73.
[18] K. Basu, S. M. Saeed, C. Pilato, M. Ashraf, M. T. Nabeel, K. Chakrabarty, and R. Karri, "CAD-Base: An Attack Vector into the Electronics Supply Chain," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, no. 4, pp. 38:1–38:30, Apr. 2019.
[19] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.
[20] 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 *Proc. DAC*, 2017, pp. 62:1–62:6.
[21] A. J. Torres, "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite," in *Proc. ICCAD*, 2012, pp. 349–350.
[22] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, 2016, pp. 265–283.
[23] Mentor Graphics, "Calibre verification user's manual," 2008.