Detecting Multi-Layer Layout Hotspots with Adaptive Squish Patterns

Haoyu Yang
CSE Department, CUHK
hyyang@cse.cuhk.edu.hk

Piyush Pathak
Cadence Design Systems Inc.
ppathak@cadence.com

Frank Gennari
Cadence Design Systems Inc.
gennari@cadence.com

Ya-Chieh Lai
Cadence Design Systems Inc.
ylai@cadence.com

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

ABSTRACT

Layout hotpot detection is one of the critical steps in modern integrated circuit design flow. It aims to find potential weak points in layouts before feeding them into manufacturing stage. Rapid development of machine learning has made it a preferable alternative of traditional hotspot detection solutions. Recent researches range from layout feature extraction and learning model design. However, only single layer layout hotspots are considered in state-of-the-art hotspot detectors and certain defects such as metal-to-via failures are not naturally supported. In this paper, we propose an adaptive squish representation for multilayer layouts, which is storage efficient, lossless and compatible with deep neural networks. We conduct experiments on 14*nm* industrial designs with a metal layer and its two adjacent via layers that contain metal-to-via hotspots. Results show that the adaptive squish representation can achieve satisfactory hotspot detection accuracy by incorporating a medium-sized convolutional neural networks.

1 INTRODUCTION

Layout hotpot detection is one of the critical steps in modern integrated circuit design flow. It aims to find potential weak points in layouts before feeding them into manufacturing stage. Classic solutions include pattern matching, pattern simulation and machine learning. Pattern matching locates potential hotspot regions by finding same or similar patterns occurred in an existing hotspot pattern library [1–3]. However, pattern matching methods rely highly on the quality of the pattern library and will show weak performance on unseen designs. Although pattern simulation can accurately predict hotspot regions incorporating with proper models, it is extremely computational costly.

Machine learning, as a rapid developing field, has shown its advantage on solving DFM problems [4–8]. Recent study of machine learning on hotspot detection tasks cover layout feature representation [9–12] and learning model design [10–14]. Popular layout representations include density-based feature [11], pixel-based feature [7, 12], frequency domain feature [10, 15], concentric circle sampling (CCS) [13] and squish pattern [16]. Satisfactory hotspot prediction results can be achieved incorporating with certain machine learning models such as support vector machine [17], boosting [11, 13] and deep neural networks [7, 10, 12, 18]. However, all of these machine learning-based hotspot detectors only consider defects that could possibly occur in single metal layer, and certain hotspots like potential metal-to-via failures are not naturally supported or discussed.

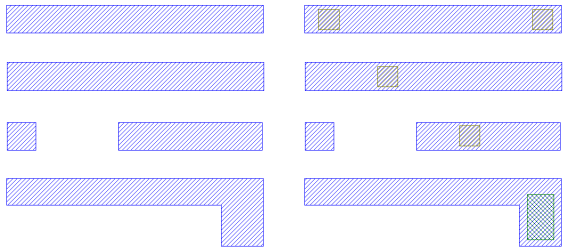

Figure 1: 14*nm* design examples of (a) single metal layer and (b) single metal layer with its upper and lower via.

Multilayer patterns are much more complicated from geometric point of view. As shown in Figure 1, via-introduced pattern variations make it more challenging when extracting layout features. State-of-the-art layout representations are more or less exhibiting minor drawbacks that might be amplified when applied in multilayer 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, benefiting 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. Data preprocessing is usually applied to make the feature

ASPDAC '19, January 21–24, 2019, Tokyo, Japan

ACM ISBN 978-1-4503-6007-4/19/01…$15.00
https://doi.org/10.1145/3287624.3288747

value of each instance fall into same scale. An example is that color images are usually subtracted by their mean pixel value of RGB channels before feeding them into learning machines, which has shown better model convergence and prediction performance [19]. Inspired by such observation, we propose an adaptive squish representation that attains the properties of original squish patterns and also ensures a fixed dimensionality for a given layout window size. Additionally, adaptive squish patterns promise to have smaller standard deviation in their geometric information that significantly benefits learning models.

ResNet resolves the gradient vanishing problem by introducing shortcut links between convolution stages in traditional convolutional neural networks. It has shown great success in both classic image classification tasks and DFM applications [20, 21]. In this paper, we apply the adaptive squish patterns on a convolutional neural network with ResNet blocks. The reduced neural network architecture and efficient layout representation show significant discriminate power on detecting multilayer hotspots. Rest of the paper is organized as follows.

Section 2 introduces related concepts and terminologies, Section 3 describes the details of our framework including the extraction of adaptive squish patterns and the neural network architecture, Section 4 shows the experimental results and Section 5 concludes this paper.

2 PRELIMINARIES

In this section, we will introduce basic terminologies and concepts related to this work.

2.1 Evaluation Metrics

We adopt the same evaluation metrics as introduced in [22] which is applied in most recent hotspot detection works.

Definition 1 (Hit)**.** The total number of correctly predicted hotspots is called *hit*. The ratio between hit and total number of hotspots is defined as *accuracy*.

Definition 2 (False Alarm)**.** The number of nonhotspot locations that are reported as hotspots by the hotspot detector. We denote the ratio between false alarm and total number of nonhotspots as *false alarm rate*.

Definition 3 (Precision)**.** The ratio between hit and total number of predicted positive samples.

2.2 Squish Pattern

The classic squish pattern [16] is a lossless layout representation that consists of layout topology and geometric information. As shown in Figure 2, 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 of $\boldsymbol{T} \in \mathbb{R}^{n_x \times n_y}$ is determined by Equation (1).

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

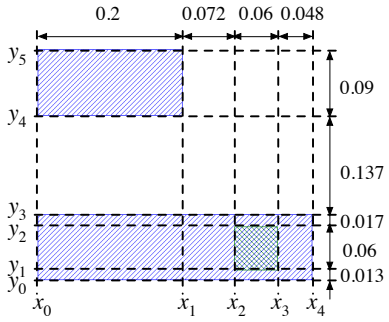

Figure 2: A simple multilayer pattern example with scan lines.

where l is the layer id and α_l indicates whether there are patterns in the corresponding grid, as in Equation (2).

$$\alpha_l = \begin{cases} 0, \text{given grid contains spacing in layer } l, \\ 1, \text{given grid contains geometry in layer } l. \end{cases} \tag{2}$$

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 (3) and Equation (4).

$$\delta_{x,i} = x_i - x_{i-1}, i = 1, 2, ..., n_x, \tag{3}$$

$$\delta_{y,i} = y_i - y_{i-1}, i = 1, 2, ..., n_y, \tag{4}$$

where x_is are the vertical scan line coordinates and y_is are horizontal scan line coordinates. Therefore the pattern in Figure 2 can be represented as follows.

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

$$\boldsymbol{\delta_x} = \begin{bmatrix} 0.2 & 0.072 & 0.06 & 0.048 \end{bmatrix},$$

$$\boldsymbol{\delta_y} = \begin{bmatrix} 0.013 & 0.06 & 0.017 & 0.137 & 0.09 \end{bmatrix}.$$

3 THE ALGORITHM

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

3.1 Adaptive Squish Pattern

As we have discussed in Section 2, 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 $\boldsymbol{M'}$ =RepeatElements($\boldsymbol{M}$, $\boldsymbol{s}$, a), 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 (5) determines how the k^{th} column of $\boldsymbol{M'}$ is constructed when $a = 0$.

$$\boldsymbol{m}'_k = \boldsymbol{m}_j, \forall \sum_{i=1}^{j-1} s_i < k \leq \sum_{i=1}^{j} s_i, \tag{5}$$

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

$$\begin{aligned} &\texttt{RepeatElements}(\boldsymbol{M}, \boldsymbol{s}, 1) \\ =&\texttt{RepeatElements}(\boldsymbol{M}^\top, \boldsymbol{s}, 0)^\top. \end{aligned} \tag{6}$$

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 will result in

$$\boldsymbol{T'} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 3 & 3 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{7}$$

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 $\boldsymbol{\delta'}$.

$$\min_{\boldsymbol{s}} ||\boldsymbol{\delta'}||_\infty \tag{8a}$$

$$\text{s.t. } \delta'_i = \delta_i / s_i, \forall i, \tag{8b}$$

$$s_i \in \mathbb{Z}^+, \forall i, \tag{8c}$$

$$\sum_i s_i = d. \tag{8d}$$

The problem in Formula (8) 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 (8).

Algorithm 1 Obtaining adaptive squish patterns with a greedy procedure.

Input: $\boldsymbol{T}$, $\boldsymbol{\delta}$, a, d_0, d;
Output: $\boldsymbol{T}$, $\boldsymbol{\delta}$;

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

Algorithm 1 circumvents Formula (8) 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 Deriving an approximate solution of Formula (8) that will be used for generating adaptive squish patterns.

Input: $\boldsymbol{\delta}$, d_0, d;
Output: $\boldsymbol{s}$;

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: $\quad \delta'_i \leftarrow \delta_i / s_i, \forall i$;
6: $\quad i \leftarrow \arg\max_i \{\delta_i | i = 1, 2, ..., d_0 - 1\}$;
7: $\quad s_i \leftarrow s_i + 1$;
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 does not generate the adaptive squish patterns on-the-fly and we target at Formula (8) itself for an optimal or sub-optimal $\boldsymbol{s}$. 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

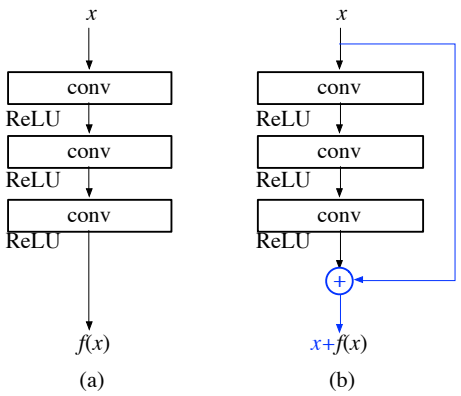

Figure 3: CNN vs. ResNet CNN.

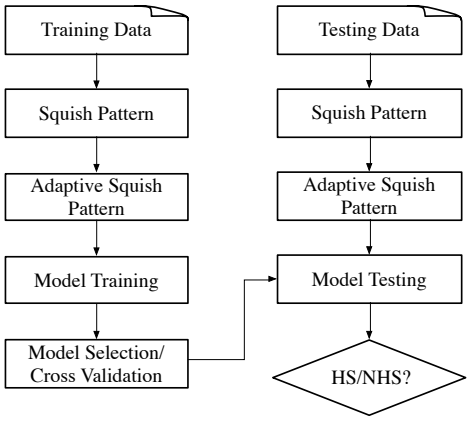

Figure 4: Multilayer hotspot detection flow.

Algorithm 2 actually performs better than Algorithm 1, which can be explained by the fact that the ideal objective value in Formula (8) is the mean of δ_is that is targeted by Algorithm 2.

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,

$$\boldsymbol{S}[:,:,0] = \boldsymbol{T}, \tag{9}$$

$$\boldsymbol{S}[:,:,1] = \texttt{RepeatElements}(\boldsymbol{\delta}_x^{\top}, [a_1], 1), \tag{10}$$

$$\boldsymbol{S}[:,:,2] = \texttt{RepeatElements}(\boldsymbol{\delta}_y, [a_2], 0). \tag{11}$$

3.2 The Neural Network Architecture

3.2.1 ResNet Block. ResNet has shown significant better training convergence and model generality. It resembles regular convolutional neural networks except additional shortcut links connecting the input and the output of each convolution stage, which is a group of convolution layers as designed in VGG [23]. Visualization of ResNet and legacy convolution stage are shown in Figure 3, where we can see that in a ResNet block, the input of a convolution stage is added up to the output of that stage and it is the addends goes into the next layer. While in conventional CNN, all layers are connected in serial. One advantage of the ResNet architecture is that gradients can not only be propagated back through regular convolution layers but can also jump over the whole convolution stage.

3.2.2 Our Network Architecture. The detailed network configurations are listed in Table 1 where we also list one state-of-the-art architecture used for image-based hotspot detection for comparison. Columns "Layer" list layer types and ID. Columns "Filter" and "Stride" define the size and the scan step of convolution and pooling layers. Columns "Output" list the output diminsionality of current layer. Yang *et al.* [12] use a reference input size of 320×320 which is exactly the same as the benchmark clip size with 1*nm* resolution. The output sizes of our architecture are derived with an input size of 64×64×3 that corresponds to the adaptive squish pattern size. Columns "Parameter" show the number of trainable parameters of each layer that reflects the neural network capacity. Although our network has 20× more trainable parameters than [12], most of the paramteres come from more feature maps within same layers, which can be efficiently distributed into GPU cores in parallel. We will also show later that our network exhibits similar throughput compared to [12].

3.3 Multilayer Hotspot Detection Flow

Our multilayer hotspot detection flow is summarized in Figure 4. 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.

4 EXPERIMENTAL RESULTS

The framework is implemented using Python 2.7 with `Tensorflow` library [24]. All experiments are conducted on a platform with NVIDIA Tesla P100 accelerator.

4.1 The Dataset

To verify the effectiveness and the performance of our multilayer hotspot detection flow, we adopt an industry 14*nm* benchmark layout that contains at most 3 layers that cover metal 4, via 3 and via 4. The statistics are listed in Table 2. 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 [12]. 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.

4.2 Effectiveness of Adaptive Squish Patterns

In this experiment, we show how adaptive squish patterns behave with two approximate solutions. We train the neural networks

Table 1: Neural networks configuration details.

JM3 [12]					Ours				
Layer	Filter	Stride	Output	Parameter	Layer	Filter	Stride	Output	Parameter
conv1-1	3×3×4	2	160×160×4	36	conv1-1	5×5×128	2	32×32×128	9600
conv1-2	3×3×4	2	80×80×4	144	conv1-2	5×5×128	1	32×32×128	409600
conv2-1	3×3×8	1	80×80×8	288	conv1-3	5×5×128	1	32×32×128	409600
conv2-2	3×3×8	1	80×80×8	576	conv1-4	5×5×128	1	32×32×128	409600
conv2-3	3×3×8	1	80×80×8	576	conv2-1	5×5×256	2	16×16×256	819200
pool2	2×2	2	40×40×8		conv2-2	5×5×256	1	16×16×256	1638400
conv3-1	3×3×16	1	40×40×16	1152	conv2-3	5×5×256	1	16×16×256	1638400
conv3-2	3×3×16	1	40×40×16	2304	conv2-4	5×5×256	1	16×16×256	1638400
conv3-3	3×3×16	1	40×40×16	2304	conv3-1	5×5×512	2	8×8×512	3276800
pool3	2×2	2	20×20×16		conv3-2	5×5×512	1	8×8×512	6553600
conv4-1	3×3×32	1	20×20×32	4608	conv3-3	5×5×512	1	8×8×512	6553600
conv4-2	3×3×32	1	20×20×32	9216	conv3-4	5×5×512	1	8×8×512	6553600
conv4-3	3×3×32	1	20×20×32	9216	conv4-1	5×5×1024	2	4×4×1024	13107200
pool4	2×2	2	10×10×32						
conv5-1	3×3×32	1	10×10×32	9216					
conv5-2	3×3×32	1	10×10×32	9216					
conv5-3	3×3×32	1	10×10×32	9216					
pool5	2×2	2	5×5×32						
fc1			2048	1638400	fc1			1024	16777216
fc2			512	1048576	fc2			2	2048
fc3			2	1024					
Summary				2746068					59796864

Table 2: Benchmark statistics.

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

Table 3: Result comparison of two adaptive squish solutions and a baseline CNN with image-based inputs.

Item	JM3 [12]	Algorithm 1	Algorithm 2
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**

using two types of squish patterns obtained from Algorithm 1 and Algorithm 2 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 [25] and are optimized with Adam optimizer on softmax cross entropy loss. We also apply weight normalization on weights in all convolution 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 3 lists the detailed hotspot detection results. It can be obviously seen that Algorithm 2 outperforms Algorithm 1 from both hotspot detection accuracy (by 1.7%) and false alarm (by 7719 clips), which can be explained by the fact that Algorithm 2 achieves much lower variance of geometric vector values.

4.3 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 [12]. The detailed architecture is listed in Table 1. 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 [12]" in Table 3, the image-based input behaves better than Algorithm 1 but worse than Algorithm 2. Although the hit count in [10] is quite close to the result of Algorithm 2, there are still significantly larger amount of false alarms.

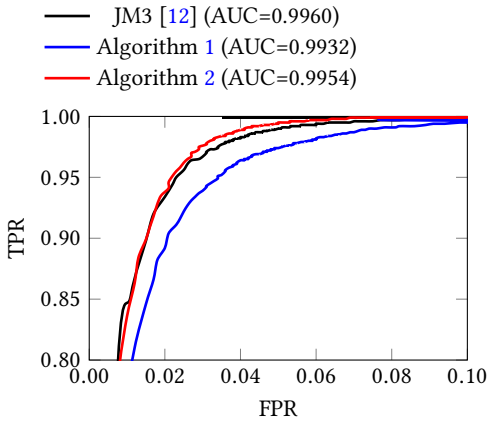

Figure 5: Partial receiver operating characteristics of three hotspot detectors.

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 that represents a clip image will be the optimal solution of Formula (8) 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 5. We can observe that three models show good quality in terms of area under ROC curve (AUC) with Algorithm 1 slightly weaker than image-based solution and Algorithm 2, which is coherent with the detection results listed in Table 3. [12] exhibits even better than Algorithm 2 in terms of AUC. By analyzing the detailed data, however, we find that most of the AUC advantages of [12] come from the region where the decision threshold is above 0.9. That means the model of [12] shows higher confidence on hotspot patterns that can be correctly predicted by both classifiers, which explains why [12] does not behave as good as Algorithm 2 in final prediction results as in Table 3.

5 CONCLUSION

In this paper, we propose a 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 multilayer 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.

ACKNOWLEDGEMENTS

This work is supported in part by The Research Grants Council of Hong Kong SAR (Project No. CUHK24209017).

REFERENCES

[1] 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)*, 2017, pp. 64:1–64:6.
[2] 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)*, vol. 33, no. 11, pp. 1671–1680, 2014.
[3] 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)*, 2017, pp. 63:1–63:6.
[4] B. Yu, D. Z. Pan, T. Matsunawa, and X. Zeng, "Machine learning and pattern matching in physical design," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2015, pp. 286–293.
[5] B. Yu, J.-R. Gao, D. Ding, X. Zeng, and D. Z. Pan, "Accurate lithography hotspot detection based on principal component analysis-support vector machine classifier with hierarchical data clustering," *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, vol. 14, no. 1, p. 011003, 2015.
[6] T. Matsunawa, B. Yu, and D. Z. Pan, "Laplacian eigenmaps and bayesian clustering based layout pattern sampling and its applications to hotspot detection and OPC," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2016, pp. 679–684.
[7] M. Shin and J.-H. Lee, "Accurate lithography hotspot detection using deep convolutional neural networks," *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)*, vol. 15, no. 4, p. 043507, 2016.
[8] 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)*, 2018, pp. 131:1–131:6.
[9] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical bayes model," in *Proceedings of SPIE*, vol. 9426, 2015.
[10] 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)*, 2017, pp. 62:1–62:6.
[11] 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*, vol. 9427, 2015.
[12] 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)*, vol. 16, no. 3, p. 033504, 2017.
[13] 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)*, 2016, pp. 47:1–47:8.
[14] 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)*, 2011, pp. 775–780.
[15] W. Zhang, X. Li, S. Saxena, A. Strojwas, and R. Rutenbar, "Automatic clustering of wafer spatial signatures," in *ACM/IEEE Design Automation Conference (DAC)*, 2013, pp. 71:1–71:6.
[16] F. E. Gennari and Y.-C. Lai, "Topology design using squish patterns," Sep. 9 2014, US Patent 8,832,621.
[17] 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)*, 2012, pp. 263–270.
[18] H. Yang, Y. Lin, B. Yu, and E. F. Young, "Lithography hotspot detection: From shallow to deep learning," in *IEEE International System-on-Chip Conference (SOCC)*, 2017, pp. 233–238.
[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Conference on Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.
[20] 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)*, 2016, pp. 770–778.
[21] Y. Lin, Y. Watanabe, T. Kimura, T. Matsunawa, S. Nojima, M. Li, and D. Z. Pan, "Data efficient lithography modeling with residual neural networks and transfer learning," in *ACM International Symposium on Physical Design (ISPD)*, 2018, pp. 82–89.
[22] 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)*, 2012, pp. 349–350.
[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations (ICLR)*, 2015.
[24] 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)*, 2016, pp. 265–283.
[25] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 9, 2010, pp. 249–256.