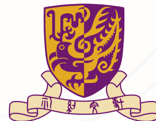
When Wafer Failure Pattern Classification Meets Few-shot Learning and Self-Supervised Learning

2021 INTERNATIONAL
CONFERENCE ON
COMPUTER-AIDED
DESIGN
ICCAD
40th Edition

Hao Geng[1], Fan Yang[2], Xuan Zeng[2], Bei Yu[1]

[1]The Chinese University of Hong Kong
[2]Fudan University
{hgeng,byu}@cse.cuhk.edu.hk

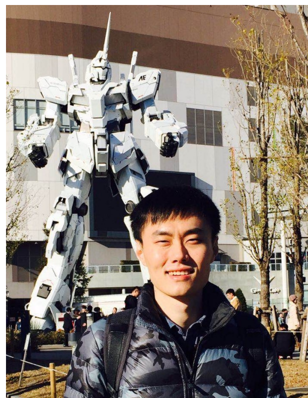
Nov. 1, 2021



Hao GENG

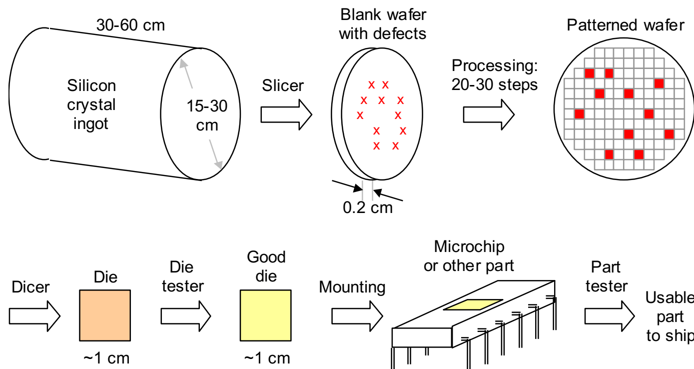
The Chinese University of Hong Kong (CUHK)
hgeng@cse.cuhk.edu.hk

Hao is a postdoctoral researcher at the Department of Computer Science and Engineering, the Chinese University of Hong Kong. Prior to that, he pursued his Ph.D. degree under the supervision of Prof. Bei YU in the same university. His research interests include design space exploration, machine learning, deep learning and the optimization methods with applications in VLSI CAD. He has received one best paper award nomination from ASPDAC 2019.



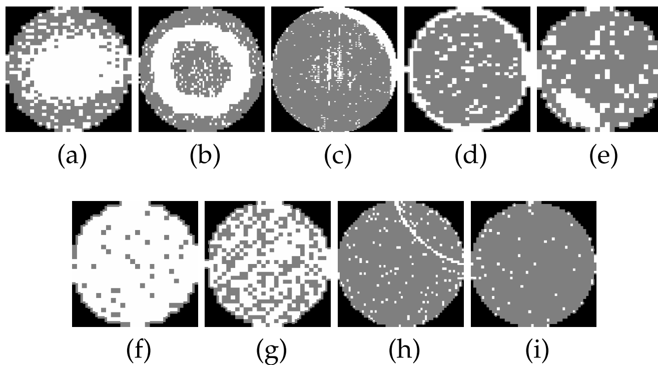
Backgrounds (i)

- Process technology nodes shrinks
- Increasingly complicated IC designs
- The increase of appearing probabilities of manufacturing process-based defects
- Wafer defects heavily affect product yield



Backgrounds (ii)

- Wafer map defect classification: locating defects at early fabrication stages
- To improve the yield with less human resource involved
- The wafer map can be obtained by chip probing
- Defective grains on a wafer map tend to converge into a certain distribution pattern



Prior Works

- Manually-crafted feature-driven:
  - Supervised Method [1] [2]: manually designed features (e.g., geometrical, gray, texture, and projection) + classifier (e.g., SVM)
  - Unsupervised Method [3]: manually designed features + clustering method
- Automatic feature extraction-based [4] [5]: exploiting deep learning model

[1] Wu et al., "Wafer map failure pattern recognition and similarity ranking for large-scale data sets," TSM, 2015.

[2] Yu et al., "Wafer map defect detection and recognition using joint local and nonlocal linear discriminant analysis," TSM, 2016.

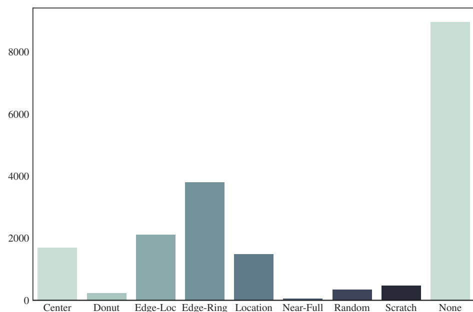
[3] Alawieh et al., "Identifying wafer-level systematic failure patterns via unsupervised learning," TCAD, 2017.

[4] Nakazawa et al., "Wafer map defect pattern classification and image retrieval using convolutional neural network," TSM, 2018.

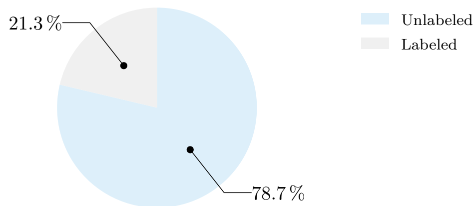
[5] Alawieh et al., "Wafer map defect patterns classification using deep selective learning," DAC, 2020.

Existing Issues

- ☹ Manually inspection is time-consuming
- ☹ Imbalanced distribution issue
- ☹ Unlabeled wafer maps are seldom exploited



(a)



(b)

Issues in an example dataset: `WM-811K` [1]

[1] "WM-811K," https://www.kaggle.com/qingyi/wm811k-wafer-map.

Defects Distribution

Labeled vs Unlabeled

21.3%

78.7%

Wafer Maps for Training

Wafer Defect Pattern Classifier

Embedding Space

Edgering

Center

Nearfull

Wafer Maps for Querying

Our Solution: The few shot learner (i)

The few shot learner

- To learn representations that generalize well to the minority failure pattern types where few wafer images are available.
- In one training batch, the wafer map embeddings are trained to predict the labels of $N_Q \times N_C$ query wafer maps conditioned on $N_S \times N_C$ support wafer maps using a certain classifier.

- A training set of N labeled examples $\mathcal{D}_{train} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
- A support set $\mathcal{D}_s$ and a query set $\mathcal{D}_q$ are sampled from the $\mathcal{D}_{train}$ per training batch
- $\mathcal{D}_k$: a subset of $\mathcal{D}_s$ labeled with wafer defect type k
- $N_C (\leq K)$: the number of classes per batch
- N_S: the number of support wafer examples per class (N_S is usually small)
- N_Q: the number of query examples for each class

Our Solution: The few shot learner (ii)

- A Prototypical few-shot learning learner with the backbone $f(\cdot;\phi)$
- Prototypical network computes a vector representation $\boldsymbol{c}_k \in \mathbb{R}^M$ (termed as `prototype`) of the central of each class
- Each `prototype` is computed by taking the mean of the embedded support wafer map vectors belonging to the associated defect type:

$$\boldsymbol{c}_k := \frac{1}{|\mathcal{D}_k|} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_k} f(\boldsymbol{x}_i; \phi) \tag{1}$$

  - A training set of N labeled examples $\mathcal{D}_{train} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$
  - $\mathcal{D}_k$: a subset of $\mathcal{D}_s$ labeled with wafer defect type k

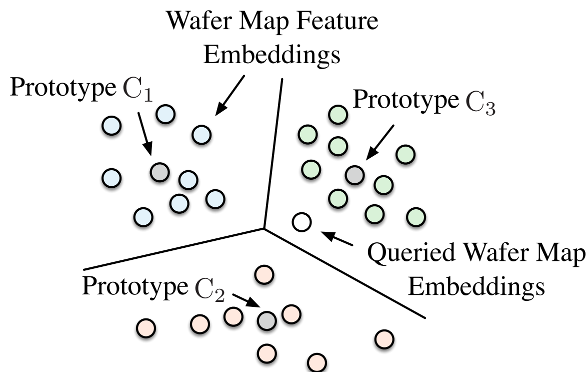
Given a distance function $d : \mathbb{R}^M \times \mathbb{R}^M \to [0, +\infty)$, the definition of the empirical loss function is:

$$\begin{aligned} \ell(\hat{y}, y) &:= -\log p(y = k \mid \boldsymbol{x}, \boldsymbol{\phi}) \\ &= -\log \frac{\exp\left(-d\left(f(\boldsymbol{x}; \boldsymbol{\phi}), \mathbf{c}_k\right)\right)}{\sum_{k'=1}^{N_C} \exp\left(-d\left(f(\boldsymbol{x}; \boldsymbol{\phi}), \mathbf{c}_{k'}\right)\right)}. \end{aligned} \tag{2}$$

- $p(y = k \mid \boldsymbol{x}, \boldsymbol{\phi})$: the softmax function over squared Euclidean distances to the prototypes in the embedding space.
- The query wafer defect map is classified based on $p(y = k \mid \boldsymbol{x}, \boldsymbol{\phi})$

The loss of few-shot learning ℓ_{few} for one batch is minimizing the empirical loss $\ell(\hat{y}_j, y_j)$ on the whole query set along with a suitable regularization r:

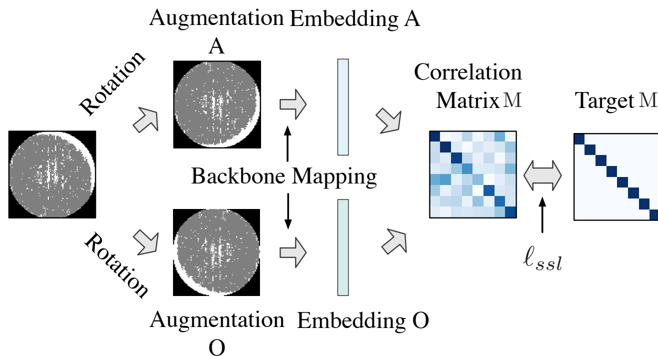
$$\ell_{few} := \sum_{\substack{j=1, \\ (\boldsymbol{x}_j, y_j) \in \mathcal{D}_q}}^{N_Q \times N_C} \ell\left(\hat{y}_j, y_j\right) + r. \tag{3}$$



The illustration of the Prototypical network-based 8-shot learner.

The self-supervised learner

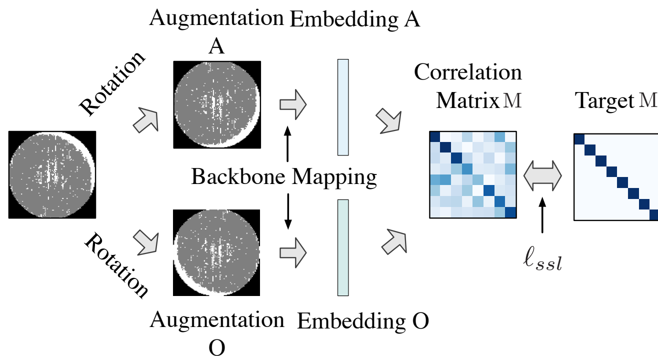
- Making full use of unlabeled wafer images
- Operating on joint embeddings of input wafer image augmentations.



Our Solution: The self-supervised learner (ii)

Components

- Data augmentation module (e.g., rotation, top-bottom and left-right flipping)
- Backbone network
- The self-supervised loss ℓ_{ssl}



Our Solution: The self-supervised learner (iii)

- The self-supervised loss ℓ_{ssl}:

$$\ell_{ssl} := \sum_{i} (1 - \boldsymbol{M}_{ii})^2 + \lambda \quad \sum_{i} \sum_{j \neq j} \boldsymbol{M}_{ij}^2, \tag{4}$$

  - λ: a positive constant
  - $\boldsymbol{M}$: the correlation matrix which computes the correlations between the outputs of the two augmented views along the batch dimension
  - $\boldsymbol{M}_{ij}$ in $\boldsymbol{M}$:

$$\boldsymbol{M}_{ij} := \frac{\sum_b e^o_{b,i} z^a_{b,j}}{\sqrt{\sum_b \left(e^o_{b,i}\right)^2} \sqrt{\sum_b \left(e^a_{b,j}\right)^2}} \tag{5}$$

    * e^o and e^a: latent embeddings of two views

Our Solution: The overall architecture design

Stem
4 Inception-A
Reduction-A
3 Inception-C
7 Inception-B
Reduction-B
Average Pooling
Max Pooling
Convolution

Inception A Block
Input Feature Maps
AvgPooling
1 × 1 (S)
1 × 1 (S)
1 × 1 (S)
3 × 3(S)
3 × 3(S)
3 × 3(S)
Concat
Output Feature Maps

Reduction A Block
Input Feature Maps
MaxPooling
3 × 3 (2V)
1 × 1 (S)
3 × 3 (S)
3 × 3 (2V)
Concat
Output Feature Maps

Reduction B Block
Input Feature Maps
MaxPooling
1 × 1 (S)
1 × 1 (S)
3 × 3 (2V)
1 × 7 (S)
7 × 1 (S)
3 × 3 (2V)
Concat
Output Feature Maps

Donut
Center
Scratch
Edgeloc
Labeled Wafer Maps

Unlabeled Wafer Maps

Backbone Network

Branch I: Few-shot Classification Loss

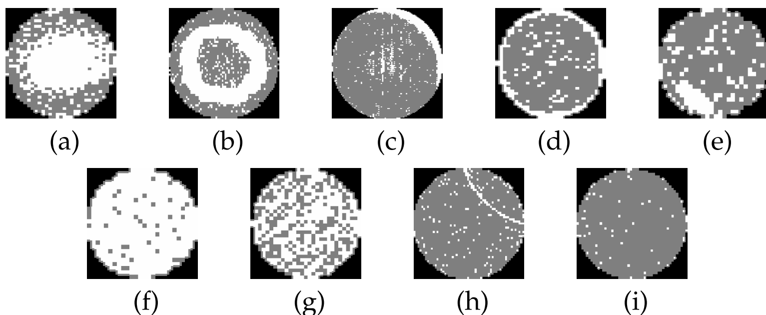
Branch II: Self-supervised Loss

Inception B Block
Input Feature Maps
AvgPooling
1 × 1 (S)
1 × 1 (S)
1 × 1 (S)
1 × 1 (S)
1 × 7 (S)
1 × 7 (S)
1 × 7 (S)
7 × 1 (S)
1 × 7 (S)
7 × 1 (S)
Concat
Output Feature Maps

Inception C Block
Input Feature Maps
AvgPooling
1 × 1 (S)
1 × 1 (S)
1 × 1 (S)
1 × 1 (S)
1 × 3 (S)
3 × 1 (S)
1 × 3 (S)
3 × 1 (S)
1 × 3 (S)
3 × 1 (S)
Concat
Output Feature Maps

Stem Block
MaxPooling (2V)
1 × 1 (S)
3 × 3 (V)
3 × 3 (V)
Concat
Concat
3 × 3 (2V)
3 × 3 (V)
3 × 3 (S)
Concat
3 × 3 (2V)
1 × 1 (S)
7 × 1 (S)
1 × 7 (S)
3 × 3 (V)
MaxPooling (2V)

- A platform with a Xeon Silver 4114 CPU processor and Nvidia TITAN Xp Graphic card
- Industry Benchmark Suite: `WM-811K`



9 kinds of wafer map pattens: (a) Center; (b) Donut; (c) Edge-Loc; (d) Edge-Ring; (e) Location; (f) Near-Full; (g) Random; (h) Scratch; (i) None (the defect-free).

- Benchmark Statistics:

Table: Benchmark Statistics

Categories	Training Set		Testing Set	
	#	Percent (%)	#	Percent (%)
Center	2576	2.48	1718	2.48
Donut	333	0.32	222	0.32
Edge-Loc	3113	2.99	2076	3.00
Edge-Ring	5808	5.60	3872	5.60
Location	2155	2.08	1438	2.08
Near-Full	89	0.09	60	0.09
Random	519	0.50	347	0.50
Scratch	715	0.69	478	0.69
None	88459	85.25	58972	85.25
Total	103767	100	69183	100

Comparison with state of the arts (i)

- Metrics: Precision, Recall, F_1 score

Table: Comparison with state of the arts

Defect Pattern	TSM'15 [1]			DAC'20 [2]			Ours		
	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
Center	0.661	0.861	0.748	**0.949**	0.942	**0.945**	0.736	**0.950**	0.830
Donut	0.729	0.459	0.564	0.798	0.748	0.772	**0.806**	**0.842**	**0.824**
Edge-Loc	0.453	0.577	0.507	**0.739**	0.690	0.714	0.647	**0.802**	**0.716**
Edge-Ring	0.611	0.908	0.731	0.992	0.950	**0.970**	**0.992**	0.921	0.955
Location	0.533	0.346	0.420	0.191	0.627	0.293	**0.605**	**0.720**	**0.658**
Near-Full	0.254	**0.867**	0.392	0.697	0.383	0.495	**0.810**	**0.867**	**0.840**
Random	0.412	0.101	0.162	0.608	0.553	0.579	**0.816**	**0.652**	**0.724**
Scratch	**0.835**	0.339	0.482	0.127	0.287	0.176	0.474	**0.701**	**0.565**
None	0.973	0.940	0.956	0.985	0.927	0.955	**0.986**	**0.967**	**0.977**
Macro-average	0.607	0.600	0.551	0.676	0.679	0.656	**0.764**	**0.825**	**0.788**
Ratio	0.795	0.727	0.700	0.885	0.823	0.832	**1.000**	**1.000**	**1.000**

[1] Wu et al., "Wafer map failure pattern recognition and similarity ranking for large-scale data sets," TSM, 2015.

[2] Alawieh et al., "Wafer map defect patterns classification using deep selective learning," DAC, 2020.

Comparison with state of the arts (ii)

- The heat maps of normalized confusion matrixes of three algorithms

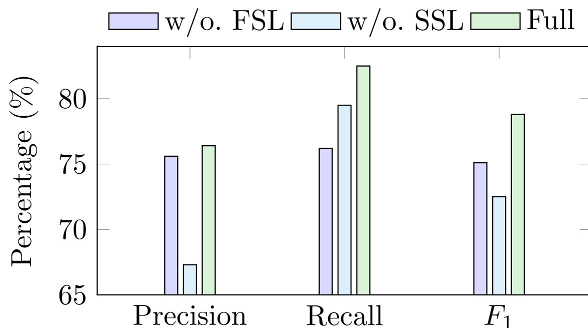
(a) TSM’15

(b) DAC’20

(c) Ours

The Ablation Study

- ‘w/o. FSL”: the flow trained with a typical cross-entropy loss as a replacement to the few-shot learning loss
- “w/o. SSL”: the flow trained without self-supervised learning loss
- “Full": the proposed flow



Conclusion

- An end-to-end, CNN-based wafer failure pattern classification framework
- Two-branch design
- Alleviate the imbalanced distribution issue and Make full use of unlabeled wafer data

THANK YOU!