University of California
Santa Barbara

# Tensor Decomposition for Concept Recognition

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Ahmed Wahba

Committee in charge:

Professor Li-C. Wang, Chair
Professor Forrest Brewer
Professor Malgorzata Marek-Sadowska
Professor Zheng Zhang

September 2019

The Dissertation of Ahmed Wahba is approved.

Professor Forrest Brewer

Professor Malgorzata Marek-Sadowska

Professor Zheng Zhang

Professor Li-C. Wang, Committee Chair

September 2019

Tensor Decomposition for Concept Recognition

by

Ahmed Wahba

This thesis is dedicated to my parents Adel and Salwa, brother Khaled, sister Ayah, and soon to be born niece.

## Acknowledgements

First, I would like to express my gratitude to my adviser Professor Li-C. Wang for his guidance and mentorship over the past five years. His advices were one of the main reasons that helped me mature not only on an academic and professional level, but on a personal level as well. When I got the chance to work closely with him, I learned what it means to understand complex research problems, how to write academic papers. In general Professor Wang helped me understand myself deeper than ever before and work on many of the shortcomings I used to have.

I would like to thank Professor Zheng Zhang for his mentorship over the Spring and Summer of 2018. His class was the seed that grew into a big part of this work. I would also like to thank all the professors that taught me over the past five years, especially, Professor Margaret Marek-Sadowska, Professor Forrest Brewer for being on my PhD defense committee.

I would also like to acknowledgeme the mentorship and friendship Farhan Rahman and Justin Hohnerlein from AMD. I would like to express my appreciation to Dr. Magdy Abadir for his mentorship during my stay in Austin in 2017, and giving me the opportunity to be the part of the organizing committee of MTV 2017.

I graciously thank my labmates, and Professor Wang's graduate students, Jay Shan, KuoKai (KK) Hsieh, Sebastian, and Matthew Nero for their help and support during the past five years. Having such brilliant minds around me, and engaging in research discussions was critical to the success of my research.

I would like to thank my friends, Victor Zakhary, Belal Salama, Jigar Patel, Ahmed Elshafie, Rashad Eletreby, Ahmed Hussein for being there when I needed to take breaks away from research. Special thanks to Mohammed Abdelghany for helping me understand many fundamental concepts.

Most importantly, I would like to thank my amazing family for always being there to provide the much needed support. My mother Salwa, father Adel, brother Khaled, sister Ayah, my grandmother Kawthar. And to my grandfather Abdelrahman, who passed away just a few months after I started my five year journey, Rest In Peace.

# Curriculum Vitæ

Ahmed Wahba

## Education

| | |
|---|---|
| 2014 - 2019 | Ph.D. in Electrical and Computer Engineering,<br>University of California, Santa Barbara. |
| 2011 - 2014 | M.S. in Computer Engineering,<br>Cairo University |
| 2006 - 2011 | B.S. in Electronics and Electrical Communications Engineering,<br>Cairo University |

## Professional Experience

1. Processor Cores Functional Verification Co-op Engineer, AMD, Austin, TX, 2016-2018.
2. Analog/RF Designer and physical layout engineer, Newport Media Inc., Cairo, Egypt, 2012-2014.

## Publications

1. L.-C. Wang, C. Shan, **A. Wahba**, “Facilitating Deployment Of A Wafer-Based Analytic Software Using Tensor Methods (Invited Paper),” in *2019 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Westminster, CO, USA, November 4-7, pp. 1-8, 2019.
2. C. Shan, **A. Wahba**, L.-C. Wang, and N. Sumikawa, “Deploying a machine learning solution as a service,” in *2019 International Test Conference, ITC*, Washington D.C, USA, November 12-14, pp. 1-10, 2019.
3. **A. Wahba**, L.-C. Wang, Z. Zhang, and N. Sumikawa, “Wafer pattern recognition using tucker decomposition,” in *VLSI Test Symposium, VTS, Monterrey, CA, USA, April 23-25, 2019*, pp. 1-6, 2019.
4. **A. Wahba**, C. Shan, L.-C. Wang, and N. Sumikawa, “Wafer plot classification using neural networks and tensor methods,” in *2019 IEEE International Test Conference in Asia, (ITC-Asia)* , Tokyo, Japan, September 3-5, pp. 1-6, 2019.
5. **A. Wahba**, C. Shan, L.-C. Wang, and N. Sumikawa, “Primitive concept identification in a given set of wafer maps,” in *2019 International Symposium on VLSI Design, Automation and Test, VLSI-DAT* , Hsinchu, Taiwan, pp. 1-4, 2019.
6. **A. Wahba**, J. Hohnerlein, F. Rahman, and L.-C. Wang. “Dynamic Exerciser Template Weighting in x86 Processor Verification,” in *2017 18th International Workshop on Microprocessor and SOC Test and Verification (MTV)*, pp. 26-31, 2017.

7. **A. Wahba**, and H. Fahmy, “Area efficient and fast combined binary/decimal floating point fused multiply add unit,” in *IEEE Transactions on Computers 66, no. 2*, pp. 226-239, 2016.

## Abstract

Tensor Decomposition for Concept Recognition

by

Ahmed Wahba

The yield optimization data analytic process is an iterative search process. Each iteration comprises of two main steps: data preparation, and result evaluation. In this thesis, we focus on the result evaluation step, where the analyst evaluates the result from running a certain analytic task looking for some specific *concept*. For example this concept can be in the form a pattern formed by the failing dies on a wafer.

A wafer pattern may hint a particular issue in the production by itself or guide the analysis into a certain direction. In this thesis, we show how Generative Adversarial Networks (GANs) can be used to build concept recognizers, we present the architecture chosen for the convolutional neural networks, we show how the network is trained, and we show how the discriminator network can be used for concept recognition.

However, the main focus of this work is on rules of Tensor decomposition in the automated concept recognition flow. We introduce a novel concept recognition approach based on Tucker decomposition. Tensor-based concept recognizers are combined with GANs-based recognizers to prevent escapes, also known as adversarial examples.

In this work, we are concerned with two main aspects: (1) automation of the concept recognition process, and (2) ensuring robustness. Two tensor methods are introduced to address both aspects.

The first tensor method is based on clustering and is used to automatically extract *concepts* that might be of interest to the analyst as well as choose the set of wafers to be used in training for each concept.

Our second tensor method is to ensure robustness of the GANs-based recognizers by introducing the *containment check* to continuously check for GANs performance and report to the expert if a problem occurs.

We also address other challenges, such as automating the training process for GANs. We also introduce a collaboration view where the machine learning expert is assumed to be a separate entity. Hence he is not allowed to have access to protected information such as yield. we present a tensor-based transformation for training wafers such that the protected information is hidden.

Our automated and robust software is applied to two high reliability automotive products, with 8300 and 7052 wafers respectively.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Machine Learning (ML) based data analytic methods have been widely explored for design automation and test applications in recent years [1]. While many promises have been shown, in practice it can still be a challenge for deploying a ML solution software into a production line.

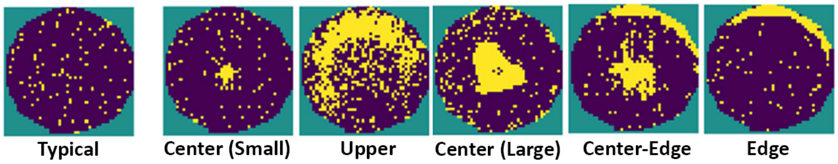


Figure 1.1: Examples of wafers with a special pattern

In this thesis, we use wafer image classification as an application example. In practice, an engineer reviews wafer plots looking for a particular pattern of interest. For example, Figure 1.1 shows six examples. Yellow dots denote the locations of failing dies and purple dots denote the locations of passing dies. Except for the first plot, the other five each has a special pattern which might indicate a certain issue in the manufacturing process, e.g. an issue with a certain tool.

Fig. 1.2 illustrates the application context for this work. A set of *wafer maps* are

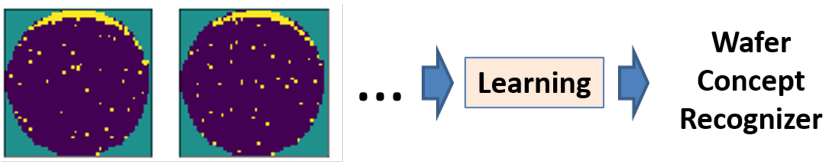


Figure 1.2: Application context for this work

given as examples to represent a *concept* to be learned. The concept shown in the figure can be called *Edge* concept. The goal is to learn a *wafer concept recognizer* (or simply a recognizer). In application, this recognizer scans a large set of wafer maps and captures those wafer maps showing the same concept.

In production, wafers come in batches. When a batch of wafer plots are given, a desired classification system puts them into baskets where each contains wafers with a particular pattern, or determines that a wafer image contains no special pattern. To implement such a classifier, an approach was developed in [2]. The classification is achieved through a set of *recognizers*, each to recognize a particular pattern type (called a *concept* [2]). Each concept recognizer is trained individually as a neural network model. The training is based on Generative Adversarial Networks (GANs) [3][4][5] where the discriminator network is used as the recognizer [2].

Because it is impractical to anticipate all possible pattern types that can appear in a production, training a recognizer has to be done online. This means that when a batch of wafer images come in, the solution needs to do three tasks [6]:

1. decide if an image can be recognized with any of the recognizers already in place.
2. for those unrecognized images, decide if there is a new concept to be learned.
3. if there is a need to build a new recognizer, extract the training and validation samples required to train the GANs-based recognizer.

Our first tensor method is employed to help in the second and the third tasks, and the underlying approach is based on clustering. The roles of the two tensor-based methods are illustrated in Figure 1.3.

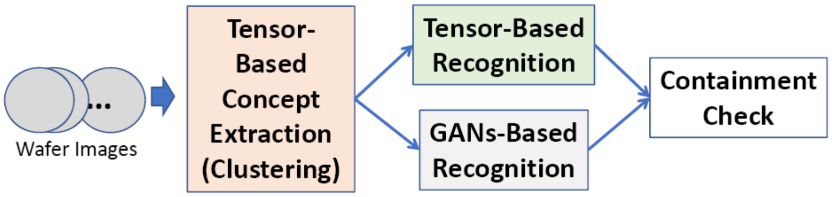


Figure 1.3: The two Tesnor-based methods and their roles

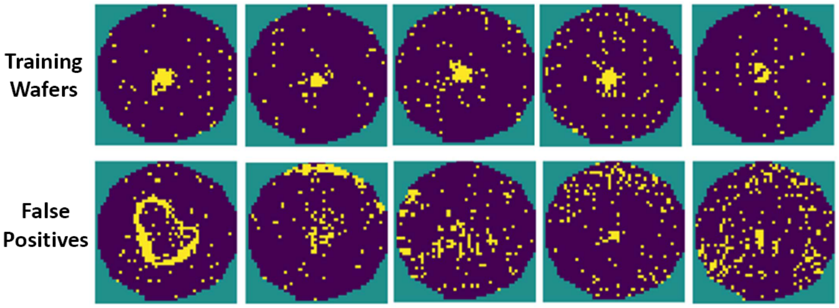


Figure 1.4: Training samples and false positives for the small-center concept

Although GANs is a popular approach, training GANs remains to be tricky [4][5]. As a result, it is difficult to ensure robustness for a recognizer. For example, Figure 1.4 shows that there can be false positives classified by a GANs-based recognizer. In this example, the recognizer is trained to recognize a "small-center" pattern, but ends up also recognizing those not containing the pattern. If the result is utilized by a person, those mistakes could reduce the person's trust on using the tool. If the result is utilized by another analytic software script, mistakes may cause misleading analytic outcome.

To address the robustness concern, a ML solution developer can have two choices. The first is to promise that the solution is robust at the time of its deployment. However, this can be difficult to achieve without some sort of guarantee on the robustness of the underlying ML models in use. The alternative is to deliver a solution where its robustness can be checked during the application. This implies that after the deployment there will be a ramp-up process where the robustness of the solution is continuously checked. Whenever an issue is detected, the developer will be called to fix the problem. This alternative enables the developer to mitigate the robustness issue over time in the actual application setting.

To realize the idea, what we need is a method that can check the recognition result from a neural network model. This is where our second tensor method comes in. For each pattern type to be recognized, the method builds a separate recognition model. Suppose for the same pattern type the set of wafers $S_{GANs}$ are recognized by the neural network model and the set of wafers $S_{Tensor}$ are recognized by the tensor-based model. The robustness is checked by examining the *containment* property of the two sets, $S_{Tensor} \subseteq S_{GANs}$ [6].

Figure 1.5 depicts a deployment setting proposed in this work. There are two sides: the ML expert side (call it Jay) and the deployment side (call it Nik). The assumption is that Jay does not have access to what happens on Nik's side. Jay's overall objective for the software is to minimize the chance that Nik perceives the software as producing "unreasonable" results, while having minimal exposure to sensitive information such as yield.

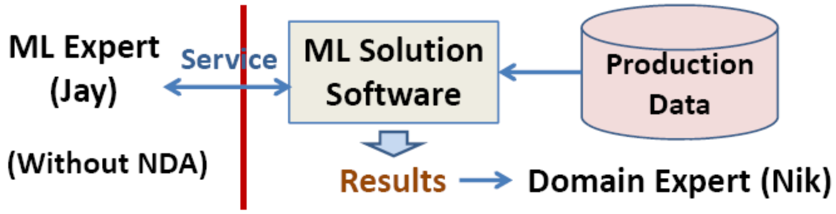


Figure 1.5: Deploying a ML solution as a service

In the next Chapter, basic Tensor notations are introduced, as well as some Tensor Decompositions and Tensor-Matrix operations that is used in this work. In Chapter 3, we give an overview on GANs and how they can be used for concept recognition. Chapter

4 discusses Tensor-based concept recognizers. Our first Tensor method is introduced in Chapter 5. Chapter 5 also discusses the implementation of our automated wafer pattern recognizer software. In Chapter 6 we introduce our second tensor method, and present more details on how the proposed deployment setting can work. Finally, Chapter 7 concludes the work.

# Chapter 2

# Tensor Notations, Decompositions, and Applications in Machine Learning

## 2.1 Introduction

In this chapter, we are going to introduce basic tensor notations, mathematical operations, and some important tensor decompositions and their applications in machine learning related applications.

Tensors are the high-dimensional generalization of vectors and matrices. While vectors are one dimensional, matrices have two dimensions, tensors can have more than two dimensions. Figure 2.1 shows an example of a three dimensional tensor and its Tucker decomposition, to be explained later in this Chapter. In general, an N-way tensor, has N dimensions or ”modes”.

In this thesis, we will follow the notations used in [7]. A tensor is represented by a boldface Euler script letter, for example $\boldsymbol{\mathcal{A}}$. A matrix is represented by a boldface

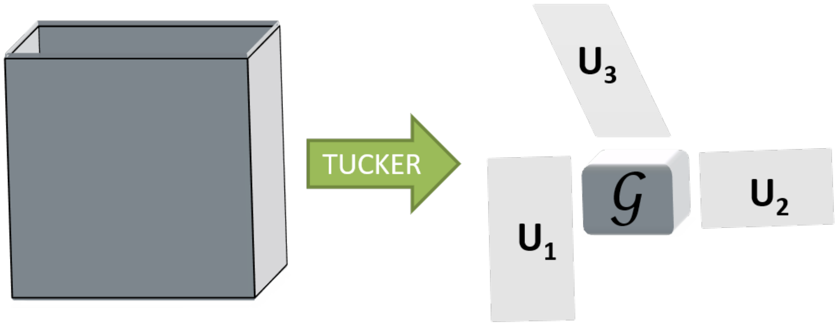


Figure 2.1: Tucker Decomposition for a Three-Dimensional Tensor

uppercase letter, for example **A**. And a vector is represented by a boldface lowercase letter, for example **a**.

In this chapter, we are going to introduce some tensor operations, tensor decompositions and briefly mention their applications in the field of machine learning.

## 2.2 Basic Tensor Operations

In this section, two of the tensor operations that are used in this work are introduced.

### 2.2.1 Mode Unfolding

Mode unfolding along mode n, which is also referred to as tensor matricization, is the operation of unfolding a tensor $\boldsymbol{\mathcal{A}}$ into a matrix, where the size of one dimension of the matrix is $\boldsymbol{R_i}$, and the size of the other dimension is given by $\prod\limits_{\substack{\boldsymbol{i=1}\\ \boldsymbol{i\neq n}}}^{\boldsymbol{M}} \boldsymbol{R_i}$. Where $\boldsymbol{R_i}$ is the size of mode-n of the tensor, M is the number of modes. For example, unfolding a three dimensional tensor of size $\mathbf{7 \times 5 \times 3}$ along the second mode results in a matrix of size $\mathbf{5 \times 21}$

For example, unfolding the three dimensional tensor $\boldsymbol{\mathcal{A}}_{\mathbf{2\times 2\times 3}}$ with the frontal slices of:

$$\boldsymbol{\mathcal{A}}_1 = \begin{bmatrix} \mathbf{1} & \mathbf{3} \\ \mathbf{2} & \mathbf{4} \end{bmatrix}, \boldsymbol{\mathcal{A}}_2 = \begin{bmatrix} \mathbf{5} & \mathbf{7} \\ \mathbf{6} & \mathbf{8} \end{bmatrix}, \boldsymbol{\mathcal{A}}_3 = \begin{bmatrix} \mathbf{9} & \mathbf{11} \\ \mathbf{10} & \mathbf{12} \end{bmatrix}$$

along each of the three modes gives the following matrices:

$$\boldsymbol{A}_{(1)} = \begin{bmatrix} \mathbf{1} & \mathbf{3} & \mathbf{5} & \mathbf{7} & \mathbf{9} & \mathbf{11} \\ \mathbf{2} & \mathbf{4} & \mathbf{6} & \mathbf{8} & \mathbf{10} & \mathbf{12} \end{bmatrix}$$

$$\boldsymbol{A}_{(2)} = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{5} & \mathbf{6} & \mathbf{9} & \mathbf{10} \\ \mathbf{3} & \mathbf{4} & \mathbf{7} & \mathbf{8} & \mathbf{11} & \mathbf{12} \end{bmatrix}$$

$$\boldsymbol{A}_{(3)} = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} \\ \mathbf{9} & \mathbf{10} & \mathbf{11} & \mathbf{12} \end{bmatrix}$$

### 2.2.2 Mode Multiplication

Mode-i Multiplication is the operation of multiplying a tensor and a matrix along a mode-i, and is denoted by the operation $\times_{\boldsymbol{i}}$. The second dimension of the matrix has to agree with the size of mode-i of the tensor. The result is another tensor with the size of the $\boldsymbol{i^{th}}$ mode replaced by the size of the matrix first dimension.

For example, $\boldsymbol{\mathcal{A}}_{\mathbf{7\times5\times3}} \times_{\mathbf{3}} \mathbf{B}_{\mathbf{6\times3}} = \boldsymbol{\mathcal{C}}_{\mathbf{7\times5\times6}}$.

For example, multiplying the tensor $\boldsymbol{\mathcal{A}}_{\mathbf{2\times2\times3}}$ with the matrix $\boldsymbol{B}_{\mathbf{2\times3}}$, where the frontal slices of $\boldsymbol{\mathcal{A}}$ and the matrix $\boldsymbol{B}$ are as following :

$$\boldsymbol{\mathcal{A}}_1 = \begin{bmatrix} \mathbf{1} & \mathbf{3} \\ \mathbf{2} & \mathbf{4} \end{bmatrix}, \boldsymbol{\mathcal{A}}_2 = \begin{bmatrix} \mathbf{5} & \mathbf{7} \\ \mathbf{6} & \mathbf{8} \end{bmatrix}, \boldsymbol{\mathcal{A}}_3 = \begin{bmatrix} \mathbf{9} & \mathbf{11} \\ \mathbf{10} & \mathbf{12} \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{4} & \mathbf{5} & \mathbf{6} \end{bmatrix}$$

will result in the tensor $\boldsymbol{\mathcal{C}}$ with the following frontal slices:

$$\boldsymbol{\mathcal{C}}_1 = \begin{bmatrix} \mathbf{38} & \mathbf{50} \\ \mathbf{44} & \mathbf{56} \end{bmatrix}, \boldsymbol{\mathcal{C}}_2 = \begin{bmatrix} \mathbf{83} & \mathbf{113} \\ \mathbf{98} & \mathbf{128} \end{bmatrix}$$

## 2.3 Tensor Decompositions

Similar to Matrices, there are many ways to decompose a Tensor. CP and Tucker decompositions are the most famous two, and they are explained below. There are many other decompositions, notably the relatively recent Tensor Train Decomposition [8] which is used in achieving a 200000:1 compression ratio in the dense layers of neural networks [9].

### 2.3.1 CANDECOMP/PARAFAC (CP) decomposition

The idea of the polyadic form, i.e. decomposing a tensor into a finite sum of rank-1 tensors, is first proposed in [10][11]. It did not become popular until the introduction of the canonical decomposition CANDECOMP [12] and PARAFAC, or parallel factors [13]. That is where it gets its name (CP). The works in [14] and [15] have also independently discovered CP decomposition and called it the topographic components model.

Figure 2.2 shows the CP decomposition of a three dimensional tensor.

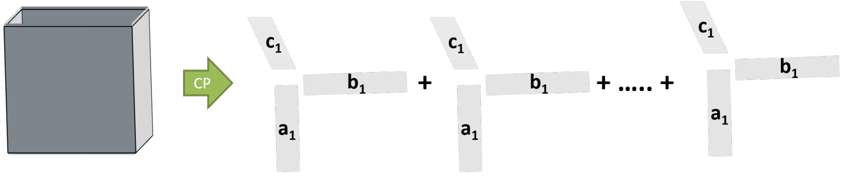


Figure 2.2: CP Decomposition for a Three-Dimensional Tensor

CP decomposition decomposes the tensor into the sum of a finite number of rank-1 tensors. A mode-n rank-1 tensor can be represented as an outer product of n vectors as shown in figure 2.2. For example, a mode-3 tensor $\boldsymbol{\mathcal{X}}$ can be written as:

$$\boldsymbol{\mathcal{X}} = \sum_{\boldsymbol{r=1}}^{\boldsymbol{R}} \boldsymbol{a_r} \circ \boldsymbol{b_r} \circ \boldsymbol{c_r} \tag{2.1}$$

where R is a positive integer and $\circ$ denotes the vector outer product.

Tensor rank, also called CP rank, is defined as the smallest value of R that satisfies equation 2.1. In other words the rank of a tensor $\boldsymbol{\mathcal{X}}$ is the smallest number of rank-1 tensors that generate $\boldsymbol{\mathcal{X}}$ as their sum [10][16] determining the rank of a tensor is NP-Hard [17].

Note that the definition of a tensor rank is analogue to the definition of matrix rank, but the properties of both are very different. One major difference is that the rank of the matrix can not be larger than its smallest dimension. I.e, a matrix $\boldsymbol{M_{n\times m}}$ can not have a rank larger than $\boldsymbol{min(n,m)}$. However, no such upper limit exists for tensors. For example, the work in [18] shows a particular $\boldsymbol{9\times 9\times 9}$ tensor has a rank between 18 and 23.

For a third order tensor $\boldsymbol{\mathcal{X}_{I\times J\times K}}$ only the following weak upper bound exists [18]: $\boldsymbol{Rank(\mathcal{X}) \leq min(IJ, IK, JK)}$

There is no finite algorithm to calculate the exact CP decomposition, the reason is determining the rank of a tensor is NP-Hard. Hence, the first problem with calculating the CP decomposition is choosing the number of rank-1 components. Most algorithms fit different CP decompositions with different number of components until one is good enough. Ideally, we can do that for R = 1,2,3,... until we get a 100% fit.

With a pre-determined number of components, there are many algorithms to compute the CP decomposition. The most used today is the alternating least squares (ALS) proposed in [12][13].

CP decomposition can be used to find low rank approximation of high dimensional and large tensors, hence compression is one of its most used applications. For example in [19] CP is applied for image compression, in [20] it is applied to build a texture database.

CP decomposition was first applied in data mining in [21][22], where the authors apploed different tensor decomposition techniques, including CP, to the problem of discussion detanglement in online chat rooms. While in [23] CP is used for automatic conversation detection in emails.

### 2.3.2 Tucker decomposition

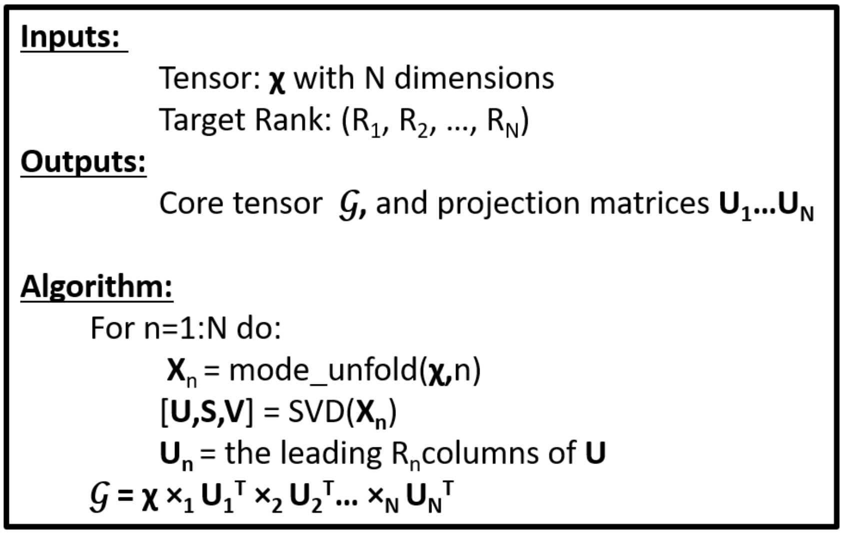
Tucker decomposition was first introduced in [24] and was later refined in [25][26]. Tucker decomposition is a generalization of the matrix Singular Value Decomposition (SVD) in higher dimensions, hence sometimes it can be called *higher order SVD* [27], N-mode SVD [28], or N-mode PCA [29]. Tucker decomposition for a three dimensional tensor is shown in figure 2.1. It decomposes a tensor into a core tensor multiplied (or transformed) by a matrix along each mode. In other words, Tucker decomposition of a d-dimensional tensor $\boldsymbol{\mathcal{X}}$ is shown in equation 2.2 below. Where $\boldsymbol{\mathcal{X}}$ is the core matrix, $\boldsymbol{U_i}$ is the projection or transformation matrix along mode i, and $\times_{\boldsymbol{i}}$ denoted $\boldsymbol{mode_i}$

multiplication. Mode multiplication is explained in Section 2.2.2.

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \prod_{i=1}^{d} \times_i \boldsymbol{U_i} \tag{2.2}$$

Tucker decomposition provides a definition of another type of rank called the n-rank or the vector rank [18][27][11]. The n-rank of an N-dimensional tensor $\boldsymbol{\mathcal{X}}$ is a vector of length N. $\boldsymbol{rank_n(\mathcal{X}) = (R_1, R_2, ..., R_N)}$ Where $\boldsymbol{R_i = Rank(X_n)}$, where $\boldsymbol{X_n}$ is the mode unfolding of tensor $\boldsymbol{\mathcal{X}}$ along mode-n.

The n-rank should not be confused by the tensor rank, i.e. minimum number of rank-1 components that sum up to the tensor, which was introduced in Section 2.3.1.



```
Inputs:
        Tensor: $\boldsymbol{\chi}$ with N dimensions
        Target Rank: $(R_1, R_2, ..., R_N)$
Outputs:
        Core tensor $\mathcal{G}$, and projection matrices $\mathbf{U_1 ... U_N}$

Algorithm:
    For n=1:N do:
        $\mathbf{X}_n$ = mode_unfold($\boldsymbol{\chi}$,n)
        [$\mathbf{U,S,V}$] = SVD($\mathbf{X}_n$)
        $\mathbf{U}_n$ = the leading $R_n$columns of $\mathbf{U}$
    $\mathcal{G} = \boldsymbol{\chi} \times_1 \mathbf{U}_1^T \times_2 \mathbf{U}_2^T ... \times_N \mathbf{U}_N^T$
```

Figure 2.3: Higher Order Singular Value Decomposition (HOSVD) Algorithm

Calculating Tucker decomposition can be done using the Higher Order Singular Value Decomposition (HOSVD) approach shown in figure 2.3.

HOSVD, however, does not provide the bets fit if using truncated core tensor, Which is usually the case if compression is required, in that case the result of HOSVD is fed into the iterative Higher Order Orthogonal Iteration (HOOI) [30].

In this work, we always keep the full ranks, hence HOSVD provides the best fit, and no need to use HOOI.

Applications of Tucker decomposition in signal processing is presented in [31]. The tensor faces algorithm in [28] presented the first use of Tucker decomposition in Machine Learning, which provides a big improvement in face recognition over the standard Principle Component Analysis (PCA) [28]. There are many other Machine Learning applications for Tucker decomposition, for example the work in [32][33] to model facial expressions. Tucker decomposition was also applied to the problem of handwritten digit recognition [34][35].

The novel work presented in this thesis is also primarily based on Tucker decomposition [36][37][38][6][39].

## 2.4 Tensor Applications

Tensor Analysis has become popular in recent years in many applications including machine learning [40][9][41][42][43]. Tensor analysis and decomposition has been also used in many applications which involve dealing with large amounts of data. In [7], the authors discuss the main tensor decompositions and their applications. The work in [9] uses tensors to represent a deep neural network and shows huge memory savings while maintaining the same prediction accuracy. The work in [44] represents the convolution layers of a deep neural network and show significant speedup in the training time while maintaining less than 1% loss in accuracy. Tensor analysis is also used for dimensionality reduction [45] and low rank approximation [13][26].

One of the applications that inspired this work is the tensor outlier analysis methods presented in [46].

### 2.4.1 Tensor Outlier Analysis

Outlier analysis using tensor-based methods is first introduced in [46], where tensor-based methods are used to detect unusual network traffic activity. Traffic on the network is sampled at each time step and encoded as a three-way tensor where the three axes are (source, destination, and port). Three methods were introduced in [46], Offline Tensor Analysis (OTA), Dynamic Tensor Analysis (DTA), and Streaming Tensor Analysis (STA). DTA and STA focus on the case where you have a stream of input tensors, while OTA is used when all the input tensors are available at once.

Offline Tensor Analysis (OTA) is used to build a model that recognizes normal network traffic, hence detecting when an unusual activity occurs.

The goal of OTA is to learn three projection matrices $\boldsymbol{(U_1,\ U_2,\ and\, U_3)}$, one for each of the three tensor axes, that minimize the error shown in equation 2.3. Where d is the number of modes, or axes, of the input tensor which is three in the case of unusual network traffic detection, n is the total number of input tensors, and $\boldsymbol{\mathcal{X}_t}$ is the input tensor at time step $\boldsymbol{t}$.

$$\boldsymbol{e = \sum_{t=1}^{n} \left\| \mathcal{X}_t - \mathcal{X}_t \prod_{i=1}^{d} \times_i (U_i U_i^T) \right\|^2} \tag{2.3}$$

OTA works well to detect unusual traffic data. If OTA can be used to detect ”unusual” wafer patterns, then can use it to find ”unusual” wafers, or wafers that are different from the majority of typical wafers. Those are the wafers with different patterns, i.e. concepts. An example of typical wafers as well as examples from wafers with ”unusual” patterns are shown in Figure 1.1.

In order to test this hypothesis, OTA is applied to 8300 wafer patterns from an automotive product line. In the case of wafer patterns, each input is a matrix, or a two dimensional tensor. The resulting error values are sorted and shown in Figure 2.4.

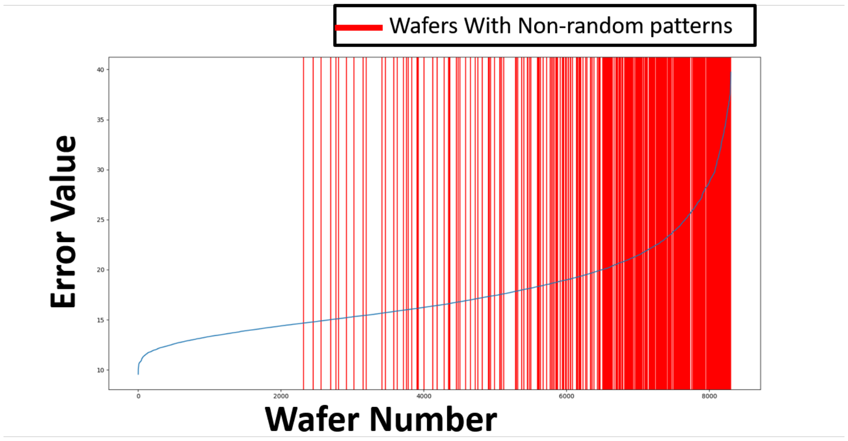


Figure 2.4: Sorted Error Values for all 8300 Wafers Using OTA

The vertical red lines indicate wafers with non-random patterns, i.e. ”unusual” patterns, according to human evaluation. Note that there is a large number of interesting wafers with high errors, but there are many wafers with interesting patterns that had a small error as well. Also, there are many wafers with random failures that had relatively higher error. Figure 2.5 shows the 5 wafers with interesting patterns that had the lowest errors. From these results we can conclude that using OTA directly is not suitable for outlier detection in wafer patterns.

This experiment, however, inspired us to instead of using Tensor analysis for outlier detection, we can use it for inlier detection or *Concept Recognition* as explained in the next chapters of this thesis.

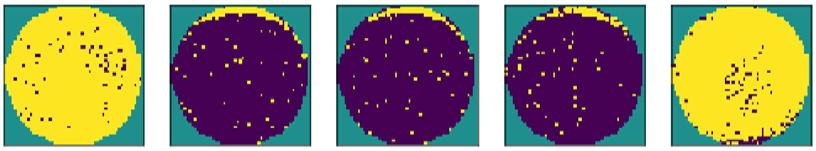


Figure 2.5: The Top Five False Negatives Using OTA

# Chapter 3

# Concept Recognition using Generative Adversarial Networks (GANs)

## 3.1 Introduction

Machine learning has been widely applied in design automation and test in recent years. As motivated in Chapter 1, the goal of this thesis is to provide a robust automated software for concept recognition. The work in [2] provides the basis for the work presented in this thesis. In [2] the authors introduce a novel approach to using Generative Adversarial Networks (GANs) as *concept recognizers*.

In this chapter, we are going to introduce GANs and how we use them for concept recognition. We are also going to present how they can be trained and what are the challenges one might face during training as well as shortcomings in using GANs as concept recognizers.

Taking the wafer image classification as an example, a concept recognizer learns the pattern, also referred to as *concept*, represented by the training wafers. Developing such a concept recognizer can be treated as an unsupervised learning problem. The training data comprise only one particular class of plots. GANs [3] provide a good underlying technology to implement such a recognizer.

Generative Adversarial Networks (GANs) [3] are methods to learn a generative model. Given a dataset, a generative model is a model that can synthesize new samples similar to the training samples. A GANs architecture consists of two neural networks. The generator network **G** is trained to produce the samples. The discriminator network **D** is trained to differentiate the training samples from the samples produced by the generator. Figure 3.1 illustrates the design of GANs. While the main goal of GANs is to learn the generator, after the training, the discriminator can be used as a recognizer for future samples similar to the training samples. Hence, our interest is on training a discriminator to be a recognizer for a class of plots.

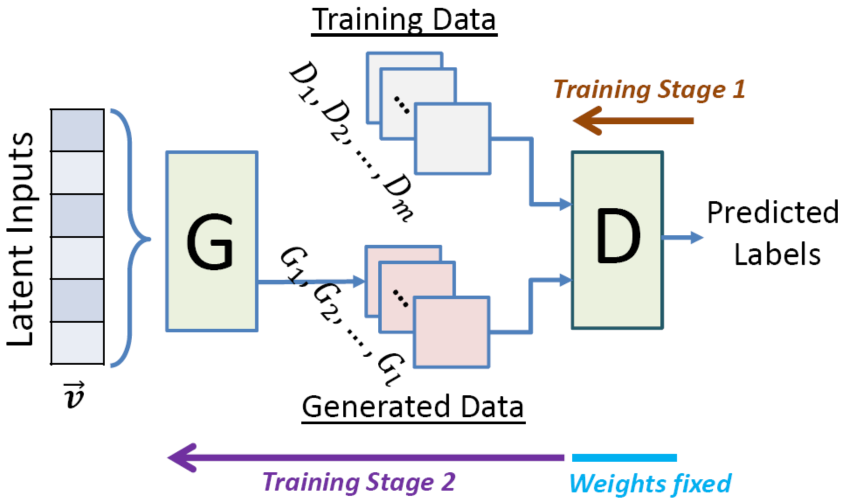


Figure 3.1: Illustration of GANs and their training

## 3.2 GANs-based recognizers Training

To train a recognizer, a number of similar wafer plots are used. Suppose there are $\boldsymbol{m}$ plots and denoted as $\boldsymbol{D_1, \ldots, D_m}$. These are our training data. For training, the generator produces some $\boldsymbol{l}$ images, denoted as $\boldsymbol{G_1, \ldots, G_l}$. Each generated image is produced according to a random vector $\vec{\boldsymbol{v}}$. Each variable of $\vec{\boldsymbol{v}}$ can be thought of as a *latent input*. These variables define a *latent space* where each vector in this space represents an image produced by the generator.

The training process is iterative. Each iteration has two stages and each stage of training can use the common stochastic gradient descent (SGD) approach. In each iteration, two classes of samples $\boldsymbol{D_1, \ldots, D_m}$ and $\boldsymbol{G_1, \ldots, G_l}$ are used. From iteration to iteration, the samples $\boldsymbol{D_1, \ldots, D_m}$ remain the same, but $\boldsymbol{G_1, \ldots, G_l}$ are re-produced by the generator for each iteration based on the weights learned in the previous iteration.

In the first stage of training, the goal is to learn the weights in the **D** network in order to separate $\boldsymbol{D_1, \ldots, D_m}$ from $\boldsymbol{G_1, \ldots, G_l}$ as much as possible. During back propagation, the gradients are computed backward from the output of **D** to its inputs. In the second stage, weights in **D** are fixed. SGD is applied to learning the weights in **G**. The gradients calculated on inputs of **D** are further back propagated to the inputs of **G**. The optimization objective is to have **G** adjust the generated samples such that their output labels by **D** are as close as possible to the output labels of the training samples $\boldsymbol{D_1, \ldots, D_m}$.

The idea of training **D** and **G** can be thought of as playing a game [3] where the **D** network learns to beat the **G** network by discriminating the samples generated by **G** from the training samples $\boldsymbol{D_1, \ldots, D_m}$. On the other hand, the **G** network learns to generate samples to fool the discriminator **D** as much as possible. Over iterations, the generated samples become more like the training samples and it becomes harder for **D** to separate them.

### 3.2.1 The CNN architectures

For computer vision applications, convolutional neural networks (CNNs) have shown remarkable performance in the context of supervised learning in recent years. Using CNNs for unsupervised learning had received less attention until the GANs approach was proposed. In this work, our implementation of GANs is based on two deep CNNs, following the ideas proposed in [5] which suggests a set of constraints on the architectural topology of convolutional GANs to make them stable to train.

Figure 3.2 shows our CNN architecture for the discriminator. The leftmost block shows our input assumption. Each input is an image with 48-by-48 pixels. Each pixel has three values: -1, 0, and +1. These three values indicate failing dies, no die, and passing dies, respectively. Before a plot can be used as an input sample to this CNN, preprocessing is required to convert the plot into this representation.

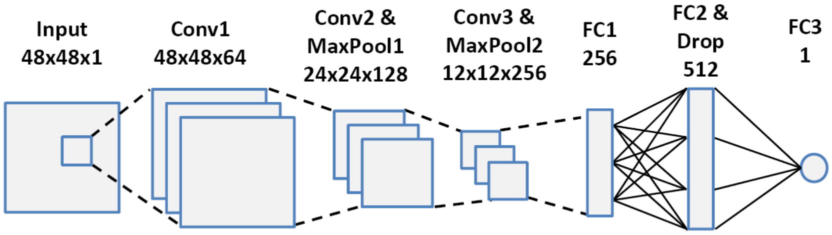


Figure 3.2: CNN architecture for the discriminator

The CNN has three convolutional layers (Conv1 to Conv3) where after Conv2 and Conv3, there is a Max pooling layer denoted as MaxPool1 and MaxPool2, respectively. After the MaxPool2 layer, there are three fully-connected layers (FC1 to FC3). The size and number of channels after each layer are denoted in the figure. For example, after Conv1 the image is transformed from 1 channel of 48×48 to 64 channels of 48×48, using

64 2×2 filters (In our CNNs, the filter size is always 2×2). After Conv2/MaxPool1, the image size is reduced to 24×24 with 128 channels.

The fully-connected layer FC1 has 256 perceptrons (artificial neurons) each receiving inputs from all the 12×12×256 perceptrons in the previous layer. The FC2 has 512 perceptrons. The last layer FC3 has one perceptron which outputs a classification probability. As suggested in [5], Leaky ReLU is used as the activation function for all perceptrons in the CNN. Each perceptron also includes a bias parameter. The total number of parameters (weights) in the CNN is 9,734,592.

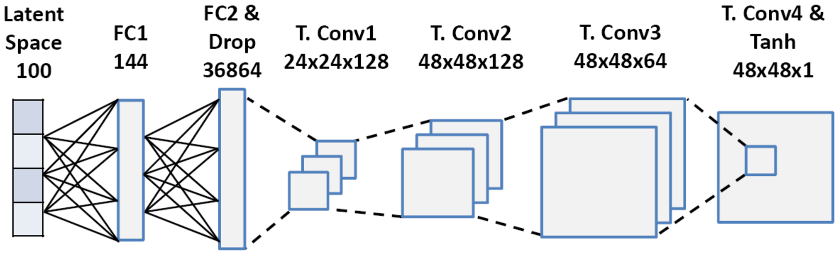


Figure 3.3: CNN architecture for the generator

Figure 3.3 shows the CNN architecture for the generator. There are two fully-connected layers, FC1 and FC2, and four transposed convolutional layers, T.Conv1 to T.Conv4. Like the discriminator CNN, Leaky RuLU and bias parameter are used for all perceptrons. The generator CNN can be thought as the reverse of the discriminator CNN. For the generator CNN, the number of parameters (weights) is 24,333,009. Together, the total number of parameters to be trained in the GANs is 34,067,601.

## 3.3 An example recognizer - edge failing

To train our GANs, we need a dataset divided into a training dataset and a validation dataset. Because it is an unsupervised learning, the validation dataset alone cannot fully determine the stopping point. The validation dataset is used to ensure the discriminator does not over-fit the samples in the training dataset, by ensuring that all samples in the validation set are also classified correctly. In our experiments, the stopping point in the training is assisted by inspecting the samples generated by the generator. If these samples show features similar to the training samples, then we stop. If not, the training is resumed for more iterations.

Because our focus is on the discriminator (used as a recognizer), we concern more about the quality of the discriminator than the quality of the generator. If the latter is our concern, we might need to train with more iterations until the generator is capable of producing plots close to the training samples. This in turn might require additional techniques in the implementation to ensure convergence.

What we found is that the GANs usually do not require a large number of samples to train if those samples share some common features. To illustrate this, Figure 3.4 shows five training samples used for training a recognizer for this concept, i.e."edging failing." The yellow dots represents the failing dies. To enhance the training dataset, each sample is incrementally rotated to produce 12 samples in total. Then, overall we have 60 samples for the training.

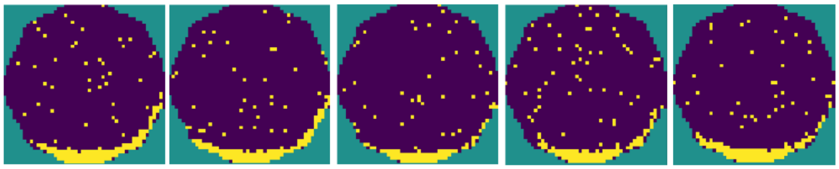


Figure 3.4: Five training samples

Figure 3.5 shows the five samples used for validation. Similarly, each is rotated to produce 12 samples with a total of 60 validation samples. Note that these samples look alike because these are wafers from the same lot.

The training took about 2 hours with a total of 3650 iterations. After the training, the discriminator (treated as the recognizer) is used to recognize the concept on 8300 other

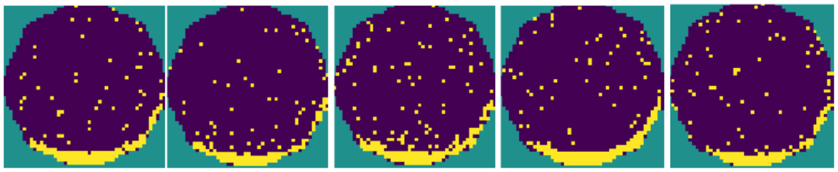


Figure 3.5: Validation samples

wafer maps. The recognizer recognizes 25 wafer maps and some are shown in Figure 3.6. On these samples, we see that they all have an edge failing pattern.

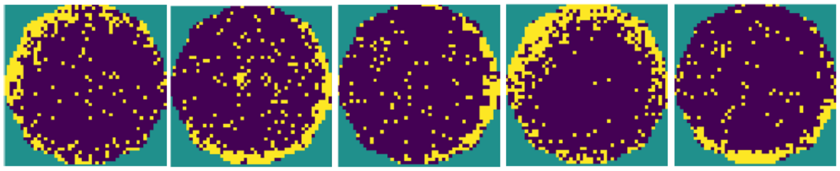


Figure 3.6: Five wafers among the 25 recognized wafers

Because the samples generated by the generator are inspected to determine the stopping point, it would be interesting to show the wafer plots produced by the generator. Figure 3.7 shows five such wafer plots (by giving the generator five random inputs). It can be seen that the generated plots do not look the same as the original plots shown in Figure 3.4. However, the concept of edge failing is clearly present in these generated plots.

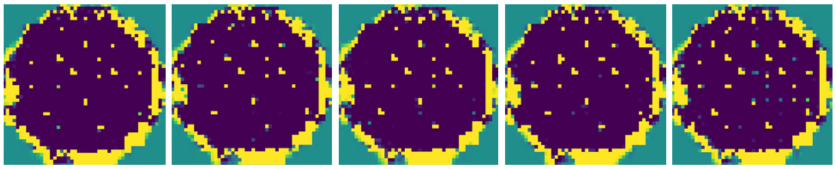


Figure 3.7: Wafer plots produced by the generator

### 3.3.1 Generality of the concept recognizer

The wafers used in the above experiment each has about 2100 dies. Recall from Figure 3.2 that input images are in size of 48×48 pixels. One interesting question to ask would be: how the recognizer performs on those wafer plots from another product line, where each plot is based on more or less number of dies?

Figure 3.8 shows the result of applying the concept recognizer on a 2nd product line. The recognizer was applied to scan 2011 wafer plots and found only 1 recognized plot as shown in the figure. Each wafer for this product line has about 4500 dies, more than twice as many as that in the first product line used for the experiment above. It is interesting to note that after manual inspection, it was confirmed that indeed there was only one wafer plot containing the edge failing concept, out of the total 2011 plots.

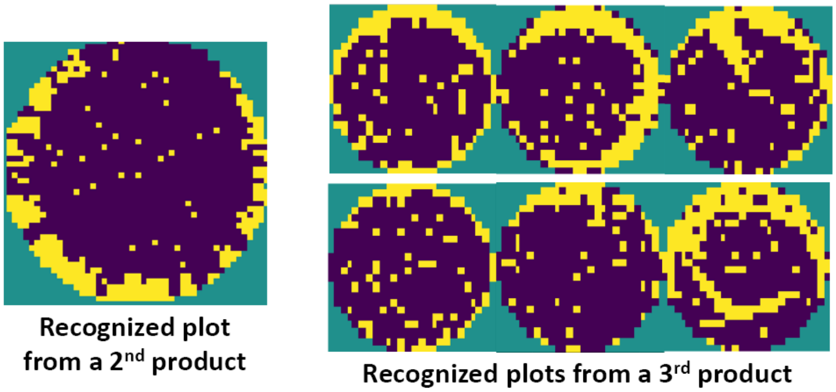


Figure 3.8: Recognized plots from the 2nd and 3rd product lines

Then, the recognizer was applied to a 3rd product line to scan 7052 wafer plots and found 24 recognized wafer plots (also manually confirmed). Some examples are also shown in Figure 3.8. For this product line, each wafer has about 440 dies, much less than that in the first product line. Each recognized plots also show a clear edge failing pattern.

From the results shown above across three product lines, it is interesting to see that the recognizer, trained with a small set of rather similar edge failing patterns as shown in Figure 3.4, is able to generalize the learning and recognize other edge failing patterns even with some noise in the pattern (Figure 3.8). These results show that one might not need to re-train a recognizer for every product line even though their numbers of dies per wafer are different. These results also indicate that one can train a recognizer with a higher-level concept. For example, in the above, the concept is to capture an "edge failing pattern."

## 3.4 Challenges and shortcomings of GANs-based recognizers

As we shown in this chapter, GANs provide a very powerful tool for concept recognition. The discriminator is able to learn the concept represented by the training wafers as well as generalize to consider rotation and shift in the pattern. However, there are some concerns regarding the robustness and automation of the GANs-based recognizers.

1. It's hard to achieve training stability, i.e. the generator and discriminator reaching a balanced state where both can improve without overpowering the other CNN. It happens that the generator can learn faster and get too powerful in tricking the discriminator which in turn over fits to the training data. Using simpler CNN architecture and skewing the training to be in favor of the discriminator can help overcoming this problem.

2. Choosing good training images is shown in 5 to be a key element in achieving convergence and the quality of the resulting discriminator model.

3. The discriminator is a neural network, which vulnerable to adversarial attacks.

4. The training process requires a lot of human interaction, investigating the generated images and deciding when to stop and extract the discriminator model.

In the rest of this work, we are going to address all the above issues. Tensor-based methods are used to help capturing adversarial examples, choose the best training wafers images, and create w quantifiable stopping criteria for training. Tensor-based methods are also used to actively check the performance of the extracted GANs-based model and reporting to the expert whenever a catastrophic failure happens.

## 3.5 Conclusion

In this chapter, we introduced the Generative Adversarial Networks (GANs), and how are they used for concept recognition. The method is applied to learn an ”edge failing pattern” from an automotive product line and the extracted discriminator is successfully applied to recognize wafers with edge patterns from three different product lines. Shortcomings and challenges of building a robust automated GANs-based concept recognition software are also discussed.

# Chapter 4

# Using Tucker Decomposition for Concept Recognition

## 4.1 Introduction

Tensors are the generalized form of matrices. While matrices have two dimensions, tensors can have more than two dimensions. The two tensor-based methods used in our software are derived from the Tucker Decomposition approach.

Tucker decomposition [26][7] is the high dimensional generalization of matrix Singular Value Decomposition (SVD) [47]. In SVD, a given matrix $\boldsymbol{X_{m\times n}}$ is decomposed into $\boldsymbol{U_{m\times r}\Sigma_{r\times r}(V^T)_{r\times n}}$ where $\boldsymbol{\Sigma_{r\times r}}$ is a diagonal matrix. Similarly in Tucker decomposition a 3-dimensional tensor $\boldsymbol{\mathcal{A}}$ of size $\boldsymbol{R_1 \times R_2 \times R_3}$ can be decomposed into $\boldsymbol{3+1}$ components: a core tensor $\boldsymbol{\mathcal{G}}$ of size $\boldsymbol{r_1 \times r_2 \times r_3}$, and 3 orthogonal projection matrices $\boldsymbol{U_1, U_2, U_3}$ each of size $\boldsymbol{R_i \times r_i}$ for $\boldsymbol{i = 1, 2, 3}$, respectively. The goal is to achieve $\boldsymbol{\mathcal{A} \approx \mathcal{G} \prod_{i=1}^{3} \times_i U_i}$, where $\times_i$ represents mode-i multiplication. The choice of the ranks $\boldsymbol{r_1, r_2, r_3}$ determines how accurate the decomposition is for representing the tensor $\boldsymbol{\mathcal{A}}$. if $\boldsymbol{r_1, r_2, r_3 = R_1, R_2, R_3}$

then the decomposition becomes exact, i.e. $\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{G}} \prod_{i=1}^{3} \times_i \boldsymbol{U_i}$. In this work, we chose to set $\boldsymbol{r_1, r_2, r_3 = R_1, R_2, R_3}$ unless mentioned otherwise.

In this chapter, we are going to show how tucker decomposition is used to build our first Tensor method. Recall that our first tensor method is building a Tensor-based concept recognizers. We start by an overview to the matrix Singular Value Decomposition (SVD), pointing out what each component of the SVD represents in the view of image representation. Then we show how SVD can be generalized to higher order tensors using Tucker decomposition. Finally, we show how we build our tensor-based concept recognizers.

## 4.2 Matrix Singular Value Decomposition (SVD)

Matrix Singular Value Decomposition is one of the important matrix decompositions in the field of image processing. SVD decomposes the matrix $\boldsymbol{X}$ into three components: $\boldsymbol{U},\ \boldsymbol{\Sigma},\ \boldsymbol{and\, V^T}$ as shown in Figure 4.1, where $\boldsymbol{X} = \boldsymbol{U\Sigma V^T}$

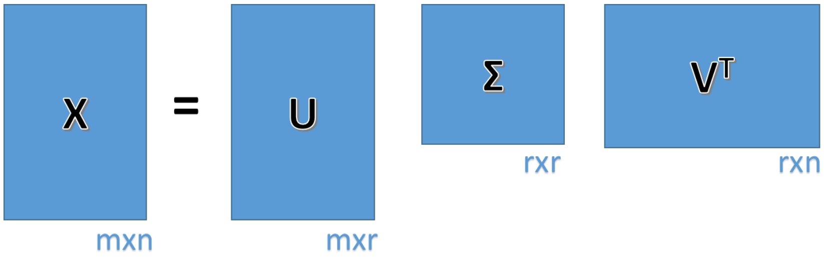


Figure 4.1: Matrix Singular Value Decomposition

1. **The left singular vectors** $\boldsymbol{U}$ is an orthonormal matrix, where each column is a left singular vector of the matrix $\boldsymbol{X}$. Left Singular Vectors of $\boldsymbol{X}$ are also Eigen-Vectors of $(\boldsymbol{XX^T})$.

2. **The Singular Values matrix $\boldsymbol{\Sigma}$** is a diagonal matrix, where diagonal elements are called the singular values of $\boldsymbol{X}$. Singular values are sorted with the largest singular value on the top left and the smallest on the bottom right diagonal element. Number of non-zero singular values is called the rank of the matrix $\boldsymbol{X}$.

3. **The right singular vectors $\boldsymbol{V^T}$** is an orthonormal matrix, where each column is a right singular vector of the matrix $\boldsymbol{X}$. Right Singular Vectors of $\boldsymbol{X}$ are also Eigen-Vectors of $\boldsymbol{(X^TX)}$.

A popular application for SVD is for image compression. For example, suppose an image matrix $\boldsymbol{X_{240\times 200}}$ is given. In SVD, the maximum choice for $\boldsymbol{r}$ is $\boldsymbol{200 = \min(240, 200)}$. For compression, a smaller $\boldsymbol{r}$ can be chosen to store the image as three matrices $\boldsymbol{U_{240\times r}\Sigma_{r\times r}(V^T)_{r\times 200}}$. For example, consider the image shown in Figure 4.2. We start by forcing the rank of the matrix representing the image to $\boldsymbol{r}$, which can be achieved by only keeping the largest $\boldsymbol{r}$ singular values and forcing the rest of the $\boldsymbol{n-r}$ singular values to zero, where n is the rank of the matrix representing the original picture (200 in this example). This means that we kept the information represented by the top $\boldsymbol{r}$ singular vectors and discarded the information in the least important $\boldsymbol{n-r}$ singular vectors.

Figure 4.2 shows how the resulting image looks for each value of $\boldsymbol{r}$. We notice that the top few singular vectors contain the vast majority of the information needed to represent the image, and the more singular vectors we include the more details the resulting image has. At rank $\boldsymbol{r = 25}$ the resulting image becomes indistinguishable from the original image.

Figure 4.3 then shows similar effect on compressing a wafer image. The image is 48×48 and hence the maximum rank is 48. As seen, with $\boldsymbol{r = 10}$, the edge pattern can be mostly restored and with $\boldsymbol{r = 25}$, the image can be mostly restored.

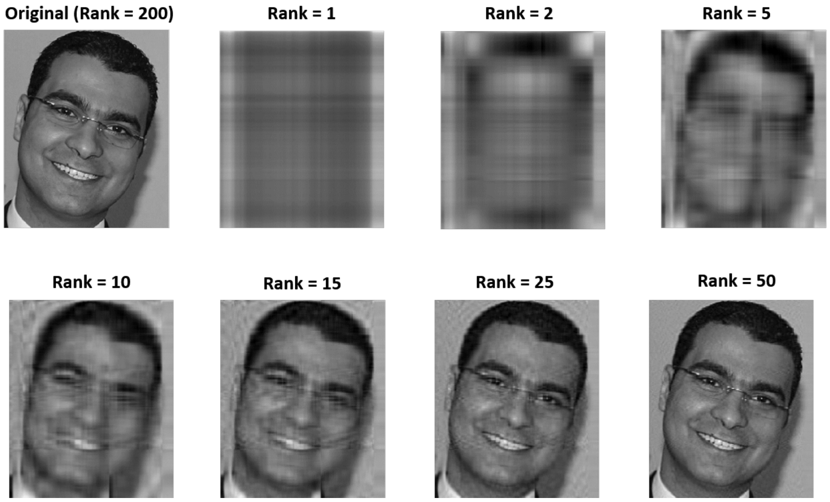


Figure 4.2: Image compression by lowering the rank

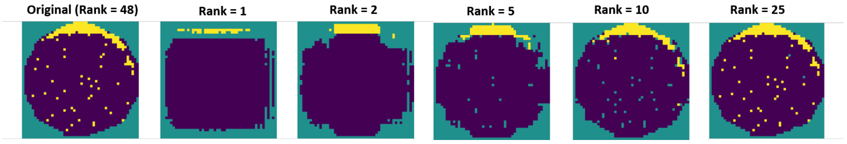


Figure 4.3: Wafer image compression by lowering the rank

In a decomposition, a projection matrix along a dimension can be thought of as the set of basis vectors along that dimension. For example, in SVD where $\boldsymbol{X} = \boldsymbol{U\Sigma V^T}$, $\boldsymbol{U}$ comprises the eigenvectors of $\boldsymbol{XX^T}$ and $\boldsymbol{V}$ comprises the eigenvectors of $\boldsymbol{X^TX}$. In other words, $\boldsymbol{U}$ comprises the basis vectors along the vertical dimension and $\boldsymbol{V}$ comprises the basis vectors along the horizontal dimension. Recall that $\boldsymbol{\Sigma}$ is a diagonal matrix.

To see the intuition, consider the example shown in Figure 4.4. Suppose SVD is applied to the two wafer images $\boldsymbol{W_a, W_b}$. We obtain $\boldsymbol{W_a} = \boldsymbol{U_a\Sigma_a V_a^T}$ and $\boldsymbol{W_b} = \boldsymbol{U_b\Sigma_b V_b^T}$, i.e. the calculations $\boldsymbol{U_a\Sigma_a V_a^T}$ and $\boldsymbol{U_b\Sigma_b V_b^T}$ restore the original images.

The image of $\boldsymbol{U_b\Sigma_a V_a^T}$ uses the left projection matrix $\boldsymbol{U_b}$ from image $\boldsymbol{W_b}$ and applies the projection to image $\boldsymbol{W_a}$. If we view $\boldsymbol{W_b}$ by scanning vertically the rows of pixels from the top to bottom, we see yellow color mostly appearing in the middle rows. This feature is effectively maintained in the image of $\boldsymbol{U_b\Sigma_a V_a^T}$. Similarly, on the image of $\boldsymbol{U_a\Sigma_a V_b^T}$,

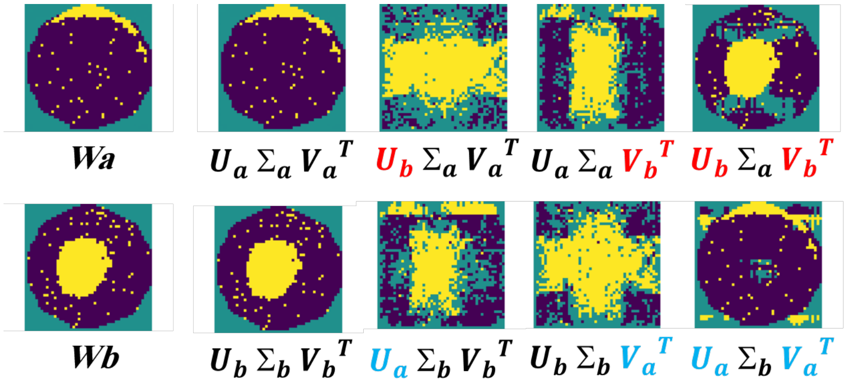


Figure 4.4: Example to illustrate the effects from projection matrices on wafer images

we see horizontally yellow color appears mostly in the middle. Then, $\boldsymbol{U_b \Sigma_a V_b^T}$ makes the image look similar to $\boldsymbol{W_b}$.

Similarly, on the image of $\boldsymbol{U_a \Sigma_b V_b^T}$, the edge feature on the top is maintained. On the image of $\boldsymbol{U_b \Sigma_b V_a^T}$, the yellow spreads out because if we view $\boldsymbol{W_a}$ by scanning horizontally the columns of pixels, we see a spread of the yellow too. Finally, $\boldsymbol{U_a \Sigma_b V_a^T}$ makes the image look similar to $\boldsymbol{W_a}$.

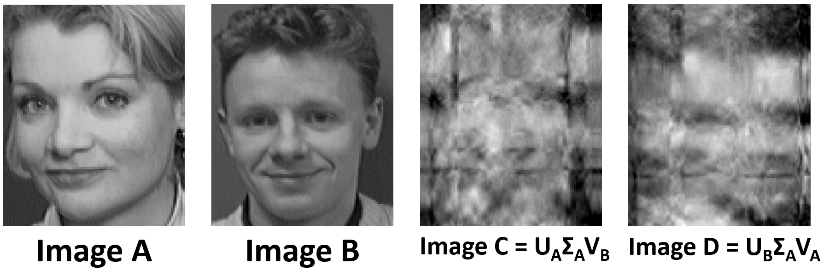


Figure 4.5: Example to illustrate the effects from projection matrices on faces

Another example is shown in Figure 4.5. Decomposing both pictures using SVD gives us $\boldsymbol{(U_A,\ \Sigma_A,\ and\ V_A^T)}$, $\boldsymbol{(U_B,\ \Sigma_B,\ and\ V_B^T)}$ for image A, and B respectively.

Figure 4.5 also shows images C and D. In image C, the right singular vectors of image B are used along side with the singular values and left singular vectors of image A. The resulting image (image C) maintained the vertical features of image A (the woman's mouth can be clearly seen). But those vertical features are scrambled because of the right singular vectors of image B. The same property can be observed about image D, where the vertical features of image B (especially the man's mouth and hair) are maintained but scrambled according to the right singular vectors of image A.

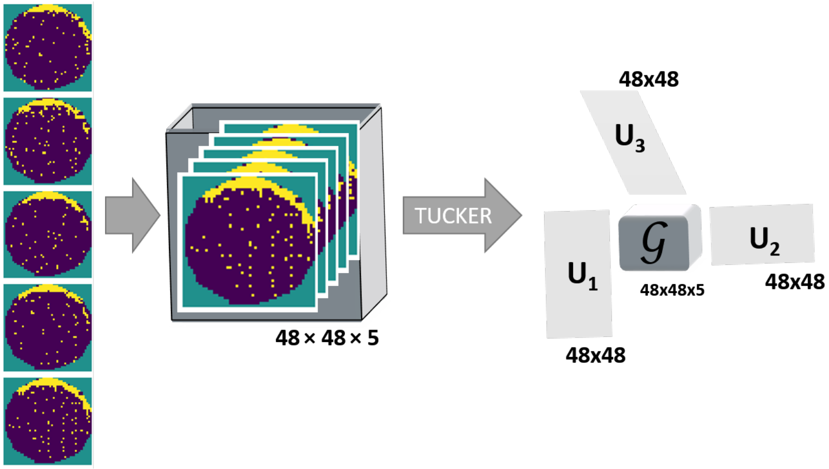


Figure 4.6: Stacking multiple wafers to use Tucker decomposition

Tucker decomposition, instead of SVD, is used when there are more than two dimensions. To apply Tucker decomposition in our context, multiple wafer images (i.e. training samples) are stacked to create the third dimension as illustrated in Figure 4.6. For wafer images of size $\mathbf{48} \times \mathbf{48}$, the size of the third dimension is the number of wafers. The resulting model from Tucker decomposition has four components: the three projection matrices $\boldsymbol{U_1}, \boldsymbol{U_2}, \boldsymbol{U_3}$ and the core matrix $\boldsymbol{\mathcal{G}}$. Following the discussion earlier, $\boldsymbol{U_1}, \boldsymbol{U_2}$ capture the features along the first two dimensions. The matrix $\boldsymbol{U_3}$ captures the features along the third, which can be thought of as capturing the features from wafer-to-wafer variations.

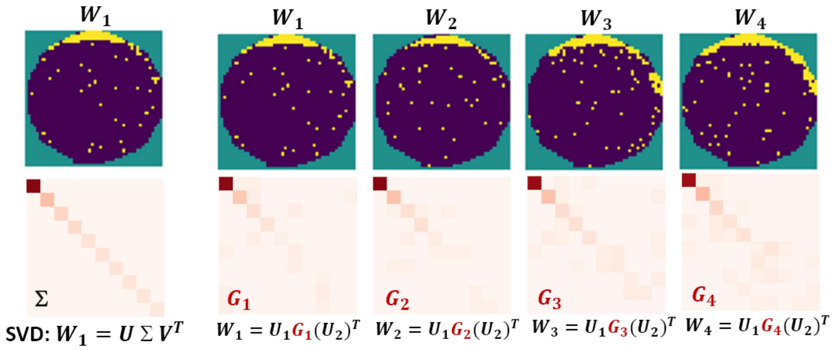


Figure 4.7: Similar wafer images and the projected core matrices

## 4.3 Modeling wafers with similar patterns

Suppose the training wafer images used in applying Tucker decomposition are all similar, i.e. there is no significant wafer-to-wafer variation in terms of their wafer patterns, then $\boldsymbol{U_3}$ is less interesting to us. For example, Figure 4.7 shows an example using four similar wafer images $\boldsymbol{W_1, W_2, W_3, W_4}$.

On the left, the result from SVD is shown for wafer image $\boldsymbol{W_1}$. The upper-left $\boldsymbol{10 \times 10}$ portion of the diagonal matrix $\boldsymbol{\Sigma}$ is shown below the image $\boldsymbol{W_1}$ where an entry with a darker color means the (absolute) value in that entry is larger. Note that all the non-diagonal entries in $\boldsymbol{\Sigma}$ have zero value.

On the right, the four images used to build a 3D tensor for running Tucker decomposition are shown. After the decomposition, $\boldsymbol{U_3}$ is ignored, and $\boldsymbol{U_1, U_2}$ are used to calculate a *projected core matrix* $\boldsymbol{G_i}$ for image $\boldsymbol{W_i}$, respectively. Each $\boldsymbol{G_i}$ is obtain such that $\boldsymbol{W_i = U_1 G_i (U_2)^T}$. It is interesting to observe that each $\boldsymbol{G_i}$ looks similar to $\boldsymbol{\Sigma}$ where the diagonal entries have a darker color and the entry (1,1) is the darkest. Each $\boldsymbol{G_i}$, however, is no longer diagonal, i.e. non-diagonal entries can have a non-zero value.

Figure 4.3 earlier shows how a wafer image is reconstructed by using a lower rank under SVD. The same idea can be applied by using the matrices $\boldsymbol{G_1, G_2, G_3, G_4}$ in Figure 4.7. For example, in Figure 4.8 the reconstruction is based on the upper-left

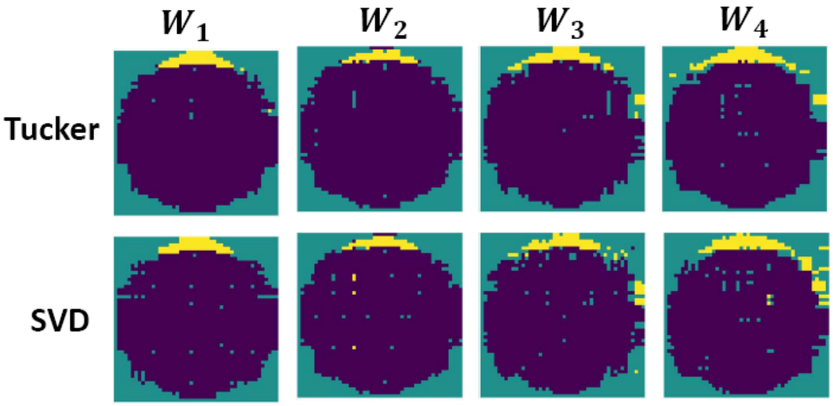


Figure 4.8: Reconstruction using the upper-left $\mathbf{5} \times \mathbf{5}$ entries of $\boldsymbol{G_i}$

$\mathbf{5} \times \mathbf{5}$ entries of each $\boldsymbol{G_i}$, respectively. For comparison, the construction result using SVD with rank 5 (as in Figure 4.3) is also shown. Observe that for all four wafers, the edge pattern can be restored and the differences between using SVD and using $\boldsymbol{G_i}$ with $\boldsymbol{U_1}, \boldsymbol{U_2}$ are not significant.

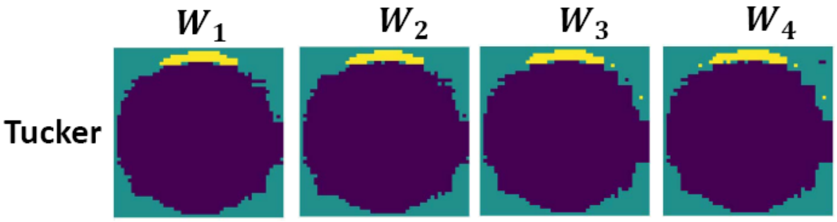


Figure 4.9: Reconstruction using the first 5 diagonal entries of $\boldsymbol{G_i}$

Because each $\boldsymbol{G_i}$ is not a diagonal matrix, it would be interesting to see how the reconstruction works if we use only the diagonal entries from each $\boldsymbol{G_i}$. Figure 4.9 shows the results. Interestingly, the edge pattern becomes even more apparent in each case and all "random noises" seem to be removed.

### 4.3.1 Effect of using wafers with dissimilar patterns

In Figure 4.7, four similar wafer images are used. In Figure 4.10, the wafer images $\boldsymbol{W_1}, \boldsymbol{W_3}$ from Figure 4.7 are reused with two new images $\boldsymbol{W_5}, \boldsymbol{W_6}$ that have a different

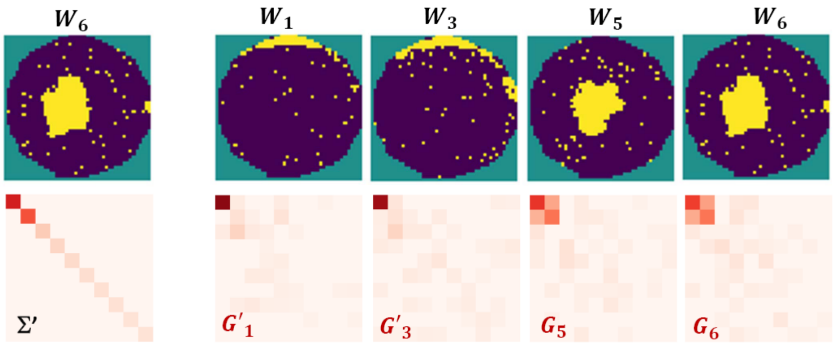


Figure 4.10: Dissimilar wafer images and the projected core matrices

pattern. The left of Figure 4.10 shows the SVD's diagonal matrix for $\boldsymbol{W_6}$. Because Tucker decomposition results in different projection matrices for $\boldsymbol{W_1}, \boldsymbol{W_3}$ (from those used in Figure 4.7), the new projected core matrices $\boldsymbol{G'_1}, \boldsymbol{G'_3}$ are shown. Additionally, $\boldsymbol{G_5}, \boldsymbol{G_6}$ are obtained for $\boldsymbol{W_5}, \boldsymbol{W_6}$.

It is interesting to observe that the diagonal trend seen on the $\boldsymbol{G}$'s matrices in Figure 4.7 is much less apparent on the $\boldsymbol{G}$'s matrices in Figure 4.10. This is especially true for $\boldsymbol{G_5}, \boldsymbol{G_6}$ where the non-diagonal entries have large values.

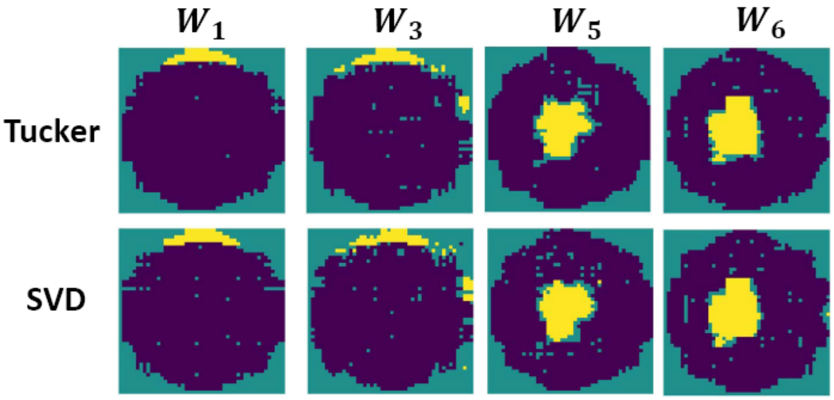


Figure 4.11: Reconstruction using the upper-left $\mathbf{5} \times \mathbf{5}$ entries of $\boldsymbol{G_i}$

Then, Figure 4.11 and Figure 4.12 show similar results as those shown in Figure 4.8 and Figure 4.9 before. As seen in Figure 4.11, in each case the wafer pattern can be mostly restored. However, this is not the case if we use only the diagonal entries as shown in Figure 4.12.

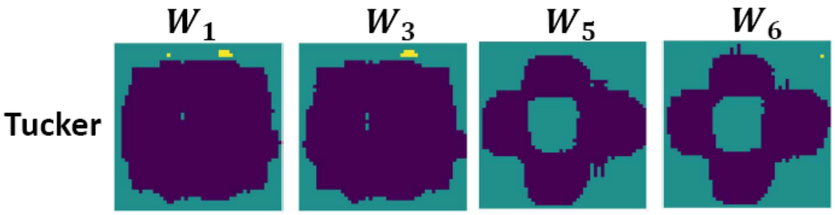


Figure 4.12: Fail to reconstruct by using only the first 5 diagonal entries of $\boldsymbol{G_i}$

Comparing Figure 4.12 to Figure 4.9, we see that if the projection matrices are based on dissimilar wafers, then the non-diagonal entries in the resulting $\boldsymbol{G}$'s matrices become important for restoring the pattern, i.e. the non-diagonal entries have large values that cannot be ignored. This observation suggests that the diagonality might be an important property to indicate the extent of similarity among wafer images.

## 4.4 The first tensor-based method

As explained in Chapter 1, our first tensor-based method is for building a classification model to recognize wafer images with a particular pattern, i.e. given a wafer image deciding either that image is in-class or out-of-class.

Example in Figure 4.4 provides an initial hint as how a classification model may be built. As seen there, using $\boldsymbol{U_a}, \boldsymbol{V_a}$ from $\boldsymbol{W_a}$, the result of $\boldsymbol{U_a}\boldsymbol{\Sigma_a}\boldsymbol{V_a^T}$ is to restore the image $\boldsymbol{W_a}$ while the result of $\boldsymbol{U_a}\boldsymbol{\Sigma_b}\boldsymbol{V_a^T}$ transforms $\boldsymbol{W_b}$ into into an image with an edge pattern which is quite different from $\boldsymbol{W_b}$. The same effect can also be seen on the result of $\boldsymbol{U_b}\boldsymbol{\Sigma_a}\boldsymbol{V_b^T}$.

In other words, given a wafer image $\boldsymbol{W_j}$ to be classified it seems that we can use the reconstruction result to decide how similar $\boldsymbol{W_j}$ is to the training image, e.g. to $\boldsymbol{W_a}$ by using $\boldsymbol{U_a}, \boldsymbol{V_a}$. Suppose the reconstructed image is $\boldsymbol{W_j'}$. If $\boldsymbol{W_j}$ is similar to the training image, we expect the difference between $\boldsymbol{W_j}$ and $\boldsymbol{W_j'}$ to be small. Otherwise, we expect the difference to be large.

We can extend this idea to use multiple images. Given a set of training wafer images $\boldsymbol{W_1, \ldots, W_n}$, we first apply Tucker decomposition to obtain the two projection matrices $\boldsymbol{U_1, U_2}$. Given an image $\boldsymbol{W_j}$ to be classified, we first apply SVD on $\boldsymbol{W_j}$ to obtain $\boldsymbol{\Sigma_j}$. Then, we can use $\boldsymbol{U_1 \Sigma_j U_2^T}$ to obtain $\boldsymbol{W_j'}$. The difference between $\boldsymbol{W_j}$ and $\boldsymbol{W_j'}$ can then be used to judge how similar $\boldsymbol{W_j}$ is to the training images.

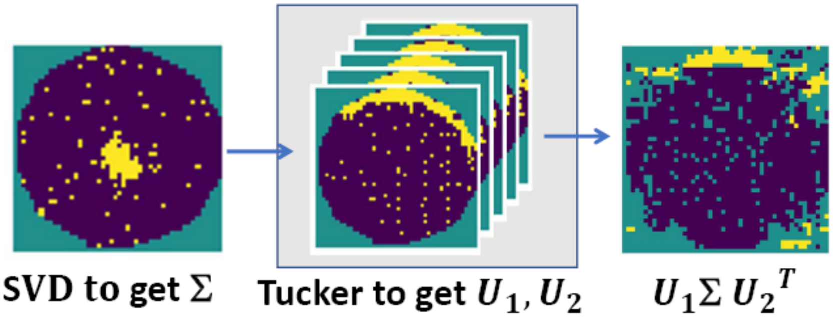


Figure 4.13: An edge model transforms a center image into an edge image

Figure 4.13 provides an example to support this initial idea. The five training images are all similar, each containing an edge pattern. The input image contains a small-center pattern, different from the training images. The reconstruction produces an image that exhibits an edge pattern.

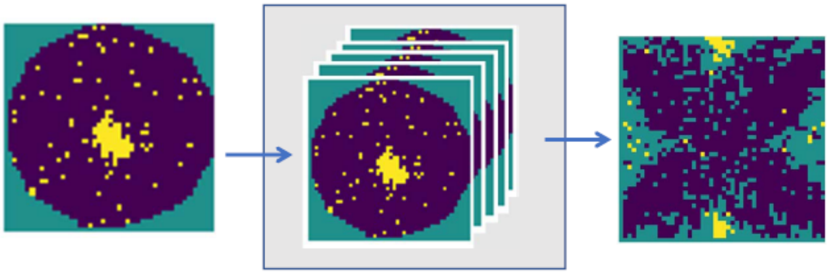


Figure 4.14: A center model fails to reconstruct a training image

However, Figure 4.14 shows that the idea might not work. In this case, the five training images are similar to the input image to be classified. The reconstructed image looks quite differently from the input image. The reason is that in order to restore the image using the two projection matrices $\boldsymbol{U_1}, \boldsymbol{U_2}$, several diagonal entries in the projected core matrix would have negative values. On the other hand, the SVD's entries are all non-negative. As a result, the image is not restored. Hence, to implement the idea we cannot simply use the diagonal matrix from SVD in the reconstruction.

### 4.4.1 Using the projected core matrix

For a given image $\boldsymbol{W_j}$, the alternative is to use the projected core matrix $\boldsymbol{G_j}$ calculated based on $\boldsymbol{U_1}, \boldsymbol{U_2}$. However, we cannot use the entirety of $\boldsymbol{G_j}$ and otherwise, we always get back to the original image $\boldsymbol{W_j}$, since $\boldsymbol{U_1}, \boldsymbol{U_2}$ are orthonormal matrices.

There can be two methods here to overcome the issue: (1) We can use only an upper-left portion of $\boldsymbol{G_j}$, similar to what we did in those examples presented in Figure 4.8 and Figure 4.11 before. (2) We can use only the diagonal entries of $\boldsymbol{G_j}$ and ignore all non-diagonal values.

In the first method, we calculate $\boldsymbol{G_j}$ where $\boldsymbol{W_j} = \boldsymbol{U_1 G_j (U_2)^T}$. Then, we use only the upper-left $\boldsymbol{r} \times \boldsymbol{r}$ entries to obtain $\boldsymbol{G_j^r}$. We calculate $\boldsymbol{W_j'} = \boldsymbol{U_1 G_j^r (U_2)^T}$ and then use the difference between $\boldsymbol{W_j}$ and $\boldsymbol{W_j'}$ to decide if $\boldsymbol{W_j}$ is similar to the training samples. Note that this alternative is essentially the idea that would have been suggested by the work in [48] if the approach was applied to our context.

### 4.4.2 Using $\boldsymbol{G_j^r}$ might not work

Suppose a model $\boldsymbol{M}$ is built based on a set of training wafer images. The $\boldsymbol{M}$ basically comprises the two projection matrices $\boldsymbol{U_1}, \boldsymbol{U_2}$. Given an image $\boldsymbol{W_j}$, we obtain $\boldsymbol{G_j^r}$ for a chosen $\boldsymbol{r}$ and consequently obtain the reconstructed image $\boldsymbol{W_j'}$. Suppose we define a difference measure $\boldsymbol{DIFF(W_j, W_j')}$. For example, the difference measure can be based on summing up the squares of the values in $\boldsymbol{W_j - W_j'}$. Then, we can let the recognition error measure of the model $\boldsymbol{M}$ (for a chosen $\boldsymbol{r}$) be $\boldsymbol{Err_{M,r}(W_j) = DIFF(W_j, W_j')}$.

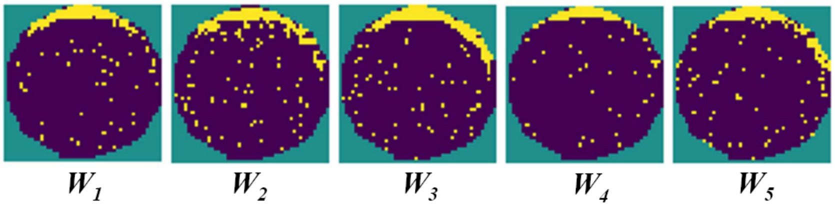


Figure 4.15: The five training samples

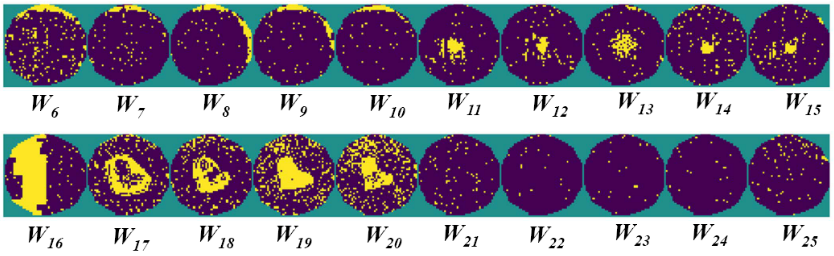


Figure 4.16: The 20 wafer images to be recognized

To illustrate how the error measure might be used, Figure 4.15 shows five images (labeled $\boldsymbol{W_1}$ to $\boldsymbol{W_5}$) for building a model $\boldsymbol{M}$. Figure 4.16 then shows four groups of images, each with five images for a total of 20 images (labeled $\boldsymbol{W_6}$ to $\boldsymbol{W_{25}}$).

If we use $\boldsymbol{M}$ and the error measure to recognize the 20 images $\boldsymbol{W_6}$ to $\boldsymbol{W_{25}}$, we expect $\boldsymbol{W_6}$ to $\boldsymbol{W_{10}}$ to be recognized as in-class and $\boldsymbol{W_{11}}$ to $\boldsymbol{W_{25}}$ as out-of-class. This is because $\boldsymbol{W_6}$ to $\boldsymbol{W_{10}}$ all exhibit an edge pattern as that in images $\boldsymbol{W_1}$ to $\boldsymbol{W_5}$ used to build the model.

For each $\boldsymbol{W_j}$ we calculate $\boldsymbol{Err_{M,r}(W_j)}$ for $\boldsymbol{j = 1, \ldots, 25}$. Figure 4.17 plots the $\boldsymbol{Err_{M,r}}$ values. For every wafer, three values are plotted by using $\boldsymbol{r = 2, 5, 25}$. Among the three values, the smallest value (in black color) is based on $\boldsymbol{r = 25}$ and the largest (in blue color) is based on $\boldsymbol{r = 2}$. The middle (in red color) is based on $\boldsymbol{r = 5}$.

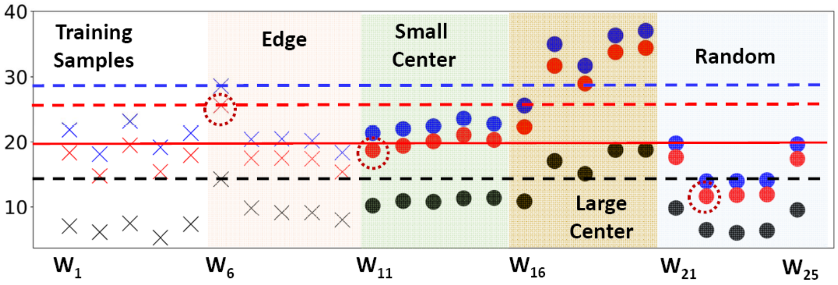


Figure 4.17: Difficulty to set a threshold to achieve correct classification

For $\boldsymbol{W_6}$ to $\boldsymbol{W_{10}}$, they should be recognized as in-class and hence we need to set a threshold such that all error values of $\boldsymbol{W_6}$ to $\boldsymbol{W_{10}}$ are below or on the threshold. Note that we expect the error value from a wafer image similar to the training samples to be small (including $\boldsymbol{W_1}$ to $\boldsymbol{W_5}$ themselves).

In Figure 4.17, observe that it is impossible to set a threshold to include $\boldsymbol{W_6}$ as in-class while leave $\boldsymbol{W_{11}}$ to $\boldsymbol{W_{25}}$ to be out-of-class. For example, consider using the error values with $\boldsymbol{r} = \mathbf{5}$ (all shown in red color). The horizontal red dash line shows the minimum threshold to classify $\boldsymbol{W_6}$ as in-class. But with this threshold, all the "small-center" images $\boldsymbol{W_{11}}$ to $\boldsymbol{W_{15}}$, $\boldsymbol{W_{16}}$, and all the "random" images $\boldsymbol{W_{21}}$ to $\boldsymbol{W_{25}}$ would also be classified as in-class, i.e. below the red dash line.

Similarly, the black and blue dash lines are the minimum thresholds to classify $\boldsymbol{W_6}$ as in-class for $\boldsymbol{r} = \mathbf{2}$ and $\boldsymbol{r} = \mathbf{25}$, respectively. As seen, with those thresholds many of the images from $\boldsymbol{W_{11}}$ to $\boldsymbol{W_{25}}$ are also classified as in-class.

Suppose we are willing to accept misclassification of $\boldsymbol{W_6}$ as out-of-class in order to lower the threshold. This lower threshold for $\boldsymbol{r} = \mathbf{5}$ is shown as the solid red horizontal line. Even with this threshold, we see that $\boldsymbol{W_{11}, W_{12}}$ and all the random images $\boldsymbol{W_{21}}$ to $\boldsymbol{W_{25}}$ are still below the line.

In Figure 4.18, $\boldsymbol{W_6, W_{11}, W_{22}}$ are selected (in Figure 4.17 they are marked with a red dash circle) to show the original image and the reconstructed image (for $\boldsymbol{r} = \mathbf{5}$).

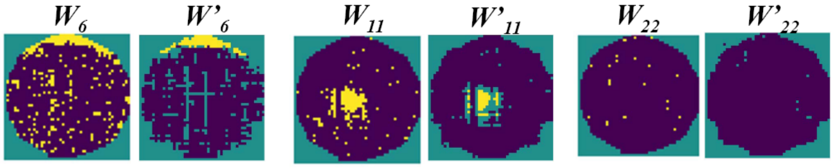


Figure 4.18: The difference between $\boldsymbol{W_j}$ and $\boldsymbol{W_j'}$ for $\boldsymbol{j = 6, 11, 22}$

Observe that through the reconstruction both the edge pattern on $\boldsymbol{W_6}$ and the small-center pattern on $\boldsymbol{W_{11}}$ are somewhat kept. On the other hand, most of the random yellow dots have disappeared. Sometimes, a yellow dot is replaced with a green dot (which represents a no die).

If the effect of the reconstruction is basically to somewhat keep the pattern (regardless it is an edge pattern or not) and replace most of the yellow dots with green or purple dots, then there is no clear reason why the $\boldsymbol{Err_{M,r}}$ has the ability to differentiate in-class images (with an edge pattern) from out-of-class images (without an edge pattern). The example in Figure 4.17 shows that it would be difficult to make it work by using $\boldsymbol{G_j^r}$ in the reconstruction.

### 4.4.3 Using only the diagonal entries in $\boldsymbol{G_j}$

Earlier in Section 4.4.1, two methods are mentioned: one is using $\boldsymbol{G_j^r}$ and the other is using only the diagonal entries in $\boldsymbol{G_j}$. Next, we let $\boldsymbol{Err_M(W_j) = DIFF(W_j, W_j')}$ where $\boldsymbol{W_j'}$ is reconstructed by using only the diagonal entries in $\boldsymbol{G_j}$.

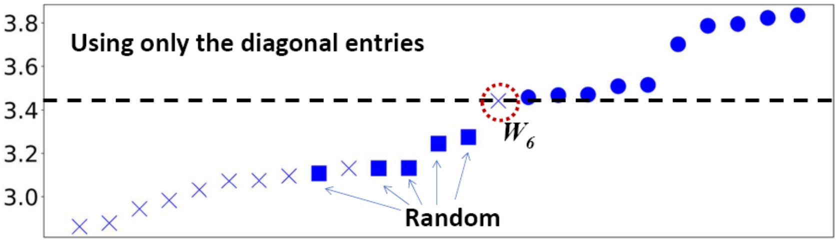


Figure 4.19: Errors calculated based on only the diagonal entries

Figure 4.19 shows the error values based on $\boldsymbol{Err_M(W_j)}$ for all 25 wafer images. In this plot, the error values are sorted. All the "edge" wafer images are shown with a "×" marker. The y-axis is in log scale (in natural log).

Observe in Figure 4.19 that a threshold can be set to classify all the "edge" and all the "random" wafer images as in-class. This can be seen as an improvement from the earlier result in Figure 4.17, because now we can treat "random" wafer image as a special case and try to find a way to exclude them.

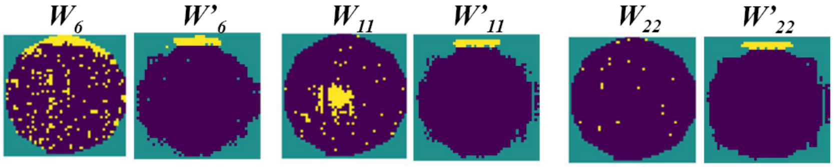


Figure 4.20: The difference between $\boldsymbol{W_j}$ and $\boldsymbol{W'_j}$ for $\boldsymbol{j = 6, 11, 22}$

Similar to Figure 4.18, for $\boldsymbol{W_6, W_{11}, W_{22}}$ Figure 4.20 shows the original image, and the reconstructed images using only the diagonal entries. It is interesting to see that an edge pattern appears in all the reconstructed images. This effect is similar to what is shown in Figure 4.4 earlier. For example, in Figure 4.20 the original $\boldsymbol{W_{11}}$ has a center pattern and the reconstructed $\boldsymbol{W'_{11}}$ has an edge pattern. Hence, the difference function $\boldsymbol{DIFF}$ would be measuring the difference between the two patterns, which is more intuitive in terms of what we desire the difference measure should have done.

### 4.4.4 Using a diagonality measure on $\boldsymbol{G_j}$

Using only the diagonal entries in $\boldsymbol{G_j}$ hints that the values in the diagonal entries are more important than the values in the non-diagonal entries. This is somewhat indicated by the example in Figure 4.7 before, where if the model is built on a set of "edge" wafer images, for a given "edge" wafer image the projected core matrix $\boldsymbol{G_j}$ would have large values in the diagonal entries and small values in the non-diagonal entries.

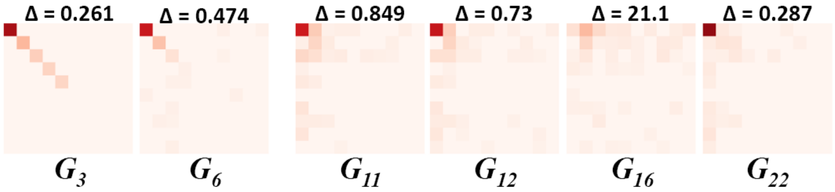


Figure 4.21: The upper-left $\mathbf{10} \times \mathbf{10}$ entries of various $\boldsymbol{G_j}$

Figure 4.21 plots the projected core matrices for $\boldsymbol{W_3}$, $\boldsymbol{W_6}$, $\boldsymbol{W_{11}}$, $\boldsymbol{W_{12}}$, $\boldsymbol{W_{16}}$, and $\boldsymbol{W_{22}}$ in a similar fashion as that used to plot Figure 4.7. Observe that $\boldsymbol{W_3, W_6}$ are "edge" wafer images and their projected core matrices exhibit the *diagonality* property. In contrast, $\boldsymbol{W_{11}, W_{12}, W_{16}, W_{22}}$ do not have an edge pattern. Their projected core matrices do not exhibit the *diagonality* property.

If the diagonality property is the underlying effect to achieve the result shown in Figure 4.19, instead of reconstructing an image and checking its difference to the original image, a better alternative can be to measure the diagonality directly, and use such a measure as the distance measure between a given wafer image and the set of the training images used to build the model $\boldsymbol{M}$.

Given a model $\boldsymbol{M}$, a wafer image $\boldsymbol{W}$, and its projected core matrix $\boldsymbol{G}$, the measure of diagonality can be defined as:

$$\boldsymbol{\Delta_{M \to W}} = \frac{\sum_{\forall i \neq j} \boldsymbol{G[i,j]^2}}{\sum_{\forall i} \boldsymbol{G[i,i]^2}} \tag{4.1}$$

In Figure 4.21, this $\boldsymbol{\Delta}$ value is also shown for each of the six wafer images. It is interesting to note that $\boldsymbol{W_6}$ and $\boldsymbol{W_{11}}$ are two of the images causing difficulty in Figure 4.17 before. Now the $\boldsymbol{W_{11}}$ has a $\boldsymbol{\Delta}$ value 0.849 which is much bigger than the $\boldsymbol{\Delta}$ value of $\boldsymbol{W_6}$ which is 0.474.

Then, the top plot in Figure 4.22 shows the $\boldsymbol{\Delta}$ values for all $\boldsymbol{W_1}$ to $\boldsymbol{W_{25}}$, which are sorted (again, y-axis in natural log scale). The result is similar to that shown in

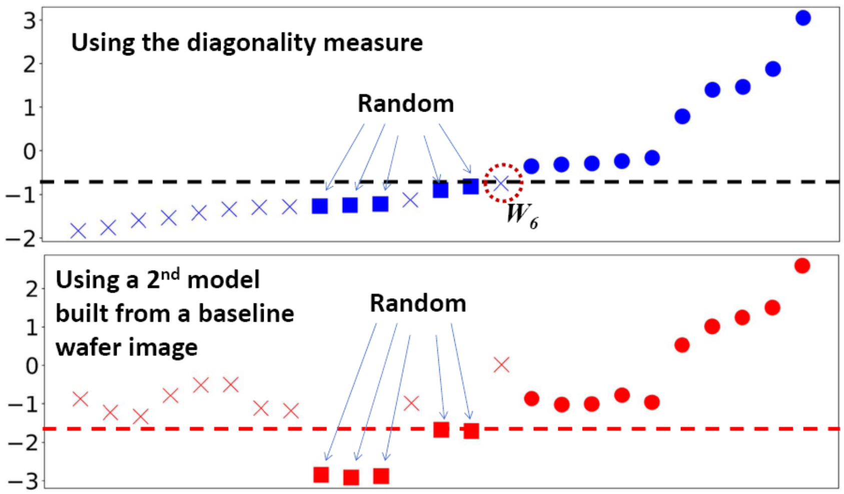


Figure 4.22: Errors calculated based on the diagonality measure

Figure 4.19, and is slightly better in the sense that the in-class region is smaller than the out-of-class region.

However, all the "random" images are still classified as in-class. In order to filter them out, we use a 2nd model and call it a *baseline* model, $\boldsymbol{M_{base}}$. In this experiment, $\boldsymbol{M_{base}}$ is simply based on a wafer with no fail, i.e. no yellow dot at all. The bottom plot in Figure 4.22 then shows the $\boldsymbol{\Delta}$ values for $\boldsymbol{W_1}$ to $\boldsymbol{W_{25}}$ based on $\boldsymbol{M_{base}}$. It is interesting to observe that in this plot, a threshold can be set to classify all "random" wafer images as in-class and leave all others as out-of-class.

### 4.4.5 Using the diagonality measure on $\boldsymbol{G_j}$ without entry (1,1)

Observe in Figure 4.21 that the (1,1) entry in most of the projected core matrices have the largest values. In equation (4.1), this large entry effectively makes the $\boldsymbol{\Delta}$ value smaller. However, if our goal is to differentiate, say $\boldsymbol{W_3, W_6}$ from $\boldsymbol{W_{11}, W_{12}, W_{22}}$, it seems that using the large values in the (1,1) entry of $\boldsymbol{G_{11}, G_{12}, G_{22}}$ is not desirable. This motivates us to try a 2nd distance measure by removing the entry (1,1) from the

$\boldsymbol{G_j}$ matrix in the $\boldsymbol{\Delta}$ calculation:

$$\boldsymbol{\Delta'_{M \to W}} = \frac{\sum_{\forall i \neq j} \boldsymbol{G[i,j]^2}}{\sum_{\forall i, i \neq 1} \boldsymbol{G[i,i]^2}} \tag{4.2}$$

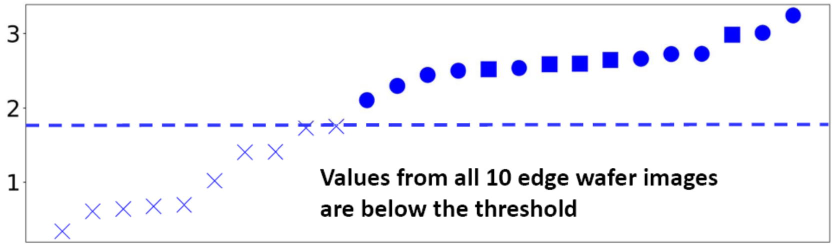


Figure 4.23: Errors calculated based by removing entry (1,1)

Figure 4.23 then shows the $\boldsymbol{\Delta'}$ values for $\boldsymbol{W_1}$ to $\boldsymbol{W_{25}}$, sorted. It is interesting to observe that now a threshold can be set to classify all the "edge" wafer images below and all the other images above. This is the most desirable outcome among all the classification results discussed so far.

The discussion so far points to two distance measures: the $\boldsymbol{\Delta}$ calculation in equation (4.1) and the $\boldsymbol{\Delta'}$ calculation in equation (4.2). Comparing the result in Figure 4.23 to Figure 4.22, it seems that the $\boldsymbol{\Delta'}$ calculation is better. While this is true for the particular example with the 25 wafer images, this might not be true in general. The caveat can be observed in Figure 4.21.

Observe that in $\boldsymbol{G_{16}}$, the entry (1,1) does not have a large value. If we exclude entry (1,1), the distance values for say, $\boldsymbol{G_3, G_6}$ increase (which can be quite significant) but the distance value for $\boldsymbol{G_{16}}$ does not change much because it does not have a large value in (1,1) to begin with. As a result, removing the entry (1,1) effectively makes the distance value of $\boldsymbol{G_{16}}$ closer to the distance values of $\boldsymbol{G_3}$ and $\boldsymbol{G_6}$ and hence, makes $\boldsymbol{W_{16}}$ less differentiable from the in-class wafer images. While this effect does not cause a problem in this particular example, it can be a concern in general. Hence our choice of distance (or error) measures is the $\boldsymbol{\Delta}$ calculation in equation (4.1).

In the experiments above, a classification attempt is based on setting a threshold to separate the in-class images from the out-of-class images. In practice, using a threshold is not desirable. A threshold can be a source for non-robustness because a threshold is set using some benchmarks and it can be difficult to ensure that all future unseen data are represented by the benchmarks. Hence, in the actual classification software we need to find a way to avoid using a threshold.

The idea adopted in the software implementation reported in [6] and [36] is to use the distance measure in a relative way. Given two models $\boldsymbol{M_1, M_2}$ and a wafer image $\boldsymbol{W}$, we compare $\boldsymbol{\Delta_{M_1 \to W}}$ and $\boldsymbol{\Delta_{M_2 \to W}}$ to see which distance is smaller (i.e. closer). If $\boldsymbol{W}$ is closer to $\boldsymbol{M_1}$, it is considered recognized by $\boldsymbol{M_1}$ and vice versa. The same applies if any of the other distance measures is used.

To implement the idea, the trick is to use a set of *baseline* models $\boldsymbol{M_{base_1}, \ldots, M_{base_m}}$ as the basis for comparison. Hence, a wafer would be recognized by a model $\boldsymbol{M}$ only if it is closer to $\boldsymbol{M}$ than to any of the baseline models, i.e. having the smallest distance value among all.

To generalize this, if we have a set of recognition models $\boldsymbol{M_1, \ldots, M_n}$ and a set of baseline models, a wafer would be considered recognized by the model $\boldsymbol{M_i}$ if the wafer has the smallest distance to $\boldsymbol{M_i}$.

The discussion above points to using the diagonality measure $\boldsymbol{\Delta}$ as our distance measure. The following experiment strengthens these results. In the following experiments, we are going to compare the following error measurements (all introduced above) while also using a *baseline* model on a larger set of wafers. 1243 wafers from the first 50 lots of an automotive product line are used.

1. using the $\boldsymbol{\Delta}$ value.

2. using the $\boldsymbol{\Delta'}$ value

3. Using only the diagonal elements from $\boldsymbol{G_j}$.

4. Using the Truncated $\boldsymbol{G_j^r}$.

| Metric | $\boldsymbol{\Delta}$ | $\boldsymbol{\Delta'}$ | $\boldsymbol{G_j}$ | $\boldsymbol{G_j^r}$ |
|---|---|---|---|---|
| Correct Recognition | 29 | 29 | 29 | 25 |
| False Negatives | 0 | 0 | 0 | 4 |
| False Positives | 19 | 24 | 19 | 361 |

Table 4.1: Recognition Summary When Using Each of The Three Error Metrics

The results shown in Table 4.1 shows that using $\boldsymbol{\Delta}$ and using $\boldsymbol{G_j}$ are achieving a similar goal through two different methods, hence only using $\boldsymbol{\Delta}$ is considered. However, they suffer from an undesirably large number of false positives. But, 18 of these 19 false positives are shown in Figure 4.24. These 18 wafers are all part of a different concept called the *Upper* concept and it is understandable that it gets confused with an "Edge" concept since they have similar characteristics. The large number of false positives can be easily reduced by adding an *Upper* concept to recognize them. The other false positive is shown in Figure 4.25, this wafer has only random failures, the reason it was not captured by the baseline model as we would expect is that it has a much higher density of failures than the wafers used to build the *baseline* model. Using more than one baseline model with different densities should be able to remove this false positive.

An *Upper* concept recognizer as well as two other *baseline* models are built using the wafers shown in Figure 4.26. After adding these two new concepts, zero false positives were found while using $\boldsymbol{\Delta}$, $\boldsymbol{\Delta'}$, and $\boldsymbol{G_j}$.

However, using $\boldsymbol{G_j^r}$ is far worse. It resulted in a large number of false positives, which shows that $\boldsymbol{G_j^r}$ is just not as accurate as we need and hence excluded.

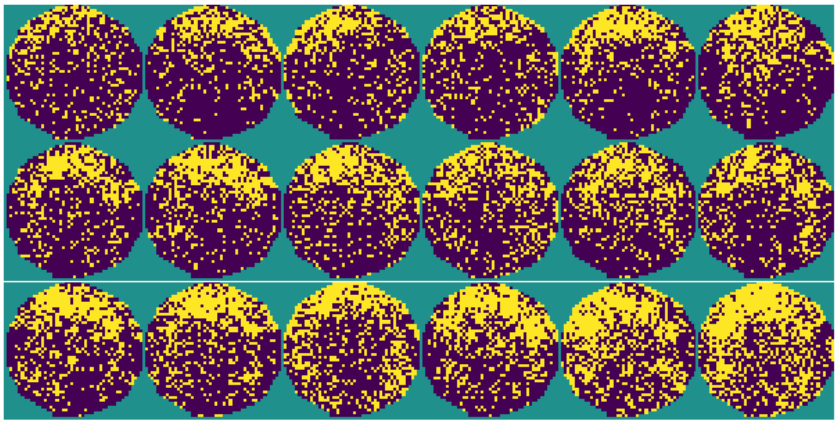


Figure 4.24: False Positive Wafers - Upper Concept

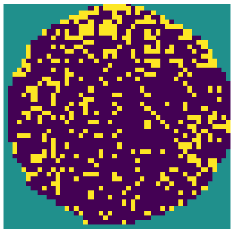


Figure 4.25: False Positive Wafers - High Density Random

These results strengthen our conclusion from the first experiment. And since $\boldsymbol{\Delta}$ and the diagonal elements of $\boldsymbol{G_j}$ gives very close performance, our choice is using $\boldsymbol{\Delta}$, since it requires less computation.

The idea of using a *baseline* can be extended to more than just one model. Multiple models can be built using multiple sets of training wafers representing different concepts, for example $\boldsymbol{M_{edge}}$, $\boldsymbol{M_{center}}$, $\boldsymbol{M_{ring}}$, $\boldsymbol{etc}$, as well as a set of *baseline* recognizers $\boldsymbol{M_{base}}$. The calculated $\boldsymbol{\Delta}$ value from all concepts are then compared, and the model reporting the smallest $\boldsymbol{\Delta}$ value recognizes the wafer. Unless one of the *baseline* models reports the smallest $\boldsymbol{\Delta}$ value, in that case the wafer is said to be “unrecognized”. The final recognition algorithm is shown in Figure 4.27.

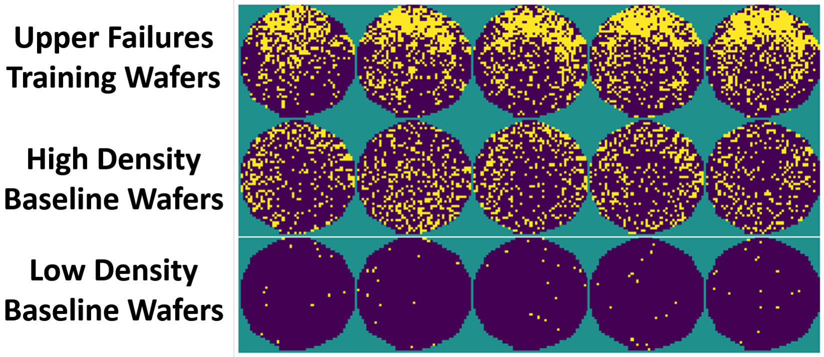
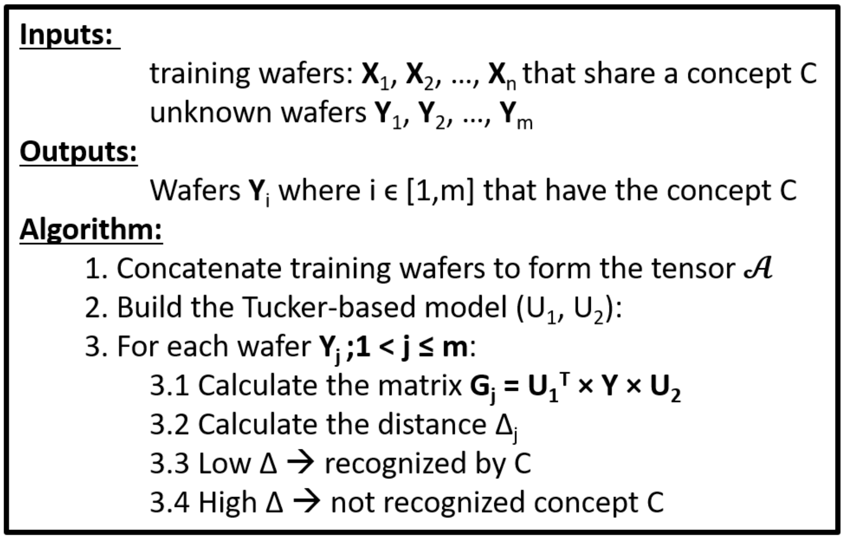


Figure 4.26: Training Wafers for New Concepts



```
Inputs:
        training wafers: $X_1$, $X_2$, ..., $X_n$ that share a concept C
        unknown wafers $Y_1$, $Y_2$, ..., $Y_m$
Outputs:
        Wafers $Y_i$ where i ∈ [1,m] that have the concept C
Algorithm:
    1. Concatenate training wafers to form the tensor $\mathcal{A}$
    2. Build the Tucker-based model ($U_1$, $U_2$):
    3. For each wafer $Y_j$ ;$1 < j \leq m$:
        3.1 Calculate the matrix $G_j = U_1^T \times Y \times U_2$
        3.2 Calculate the distance $\Delta_j$
        3.3 Low $\Delta$ → recognized by C
        3.4 High $\Delta$ → not recognized concept C
```

Figure 4.27: Concept Recognition Algorithm

## 4.5 Multi-Concept Recognizer

In the previous section, we presented how to build a single tensor-based concept recognizer, we introduced the *baseline* model as a non-subjective replacement to setting a threshold and showed why we chose the diagonality measure $\boldsymbol{\Delta}$ as a distance metric for best recognition results. In this section, we introduce the Multi-Concept Recognizer, present results to show its performance for recognizing concepts in wafer patterns from an automotive product.

The multi-concept recognizer is an extension to the single concept recognizer idea presented earlier. Instead of having one concept and one baseline concept, we can have multiple concept recognizers and more than one baseline concept. For each unknown wafer, we calculate the distance from all concepts as well as from all baseline concepts. The wafer is said to be recognized by the concept reporting the smallest error, if that concept was a baseline concept this wafer is not recognized by any of the concept recognizers.

8300 wafers from an automotive product line is used to test our multi-concept recognizer. For simplicity, the 8300 wafers are divided into three groups, low failure density, medium failure density, and high failure density. We manually defined the concepts in each of the three wafer groups and chose the training wafers for each concept.

1. **Low failure density wafers:** there are 6000 wafers with less than 5% failure density. These 6000 wafers do not have any interesting patterns, hence are skipped.

2. **Medium failure density wafers** are the wafers with larger than 5% failure density and less than 22% failure density. There are 2000 wafers in this group. We found three concepts in this group: *Edge, Grid*, and *Center*. A sample of each of these three concepts is shown in Figure 4.28

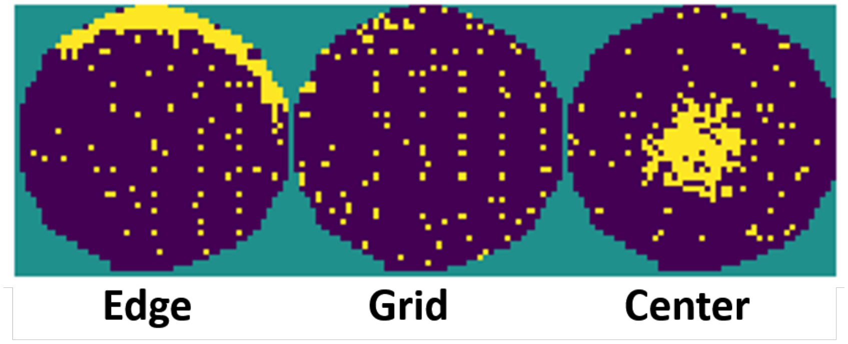


Figure 4.28: Medium Density Concepts

Table 4.2 shows the results from the medium density wafers. We notice a very few false positives. However, there is a large number of false negatives (i.e. wafers with a certain concept that are non recognized). The reason for this is the less-optimal training wafers. One approach to tackle this problem is by automatic extraction of concepts and training wafers for each concept. A brief idea on training wafer extraction is discussed later in this Chapter, while our second Tensor method, which will be explained in the Chapter 5 aims to automate concept extraction.

| Concept | Edge Failures | Grid Failures | Center Failures |
|---|---|---|---|
| Correct Recognition | 28 | 57 | 47 |
| False Negatives | 12 | 100+ | 47 |
| False Positives | 3 | 0 | 0 |

Table 4.2: Summary of The Multi-Concept Recognizer for Medium Density Wafers

3. **High failure density wafers** are the wafers with larger than 22% failure density. There are 300 wafers in this group, among which six concepts were found: *Edge, Center, Middle Ring, Massive, Upper*, and *Outer Ring.* An example from each of the six concepts is shown in Figure 4.29. Recognition results summary is shown in Table 4.3.

| Concept | Edge | Center | Middle Ring | Massive | Upper | Outer Ring |
|---|---|---|---|---|---|---|
| Correct Recognition | 15 | 26 | 5 | 13 | 63 | 8 |
| False Negatives | 0 | 0 | 0 | 0 | 6 | 3 |
| False Positives | 0 | 2 | 6 | 0 | 0 | 3 |

Table 4.3: Summary of The Multi-Concept Recognizer for High Density Wafers

Three baseline concepts were also used **Empty Wafers, Low Density Random Failures, and High Density Random Failures**. Training wafers for these concepts are shown in Figure 4.30.

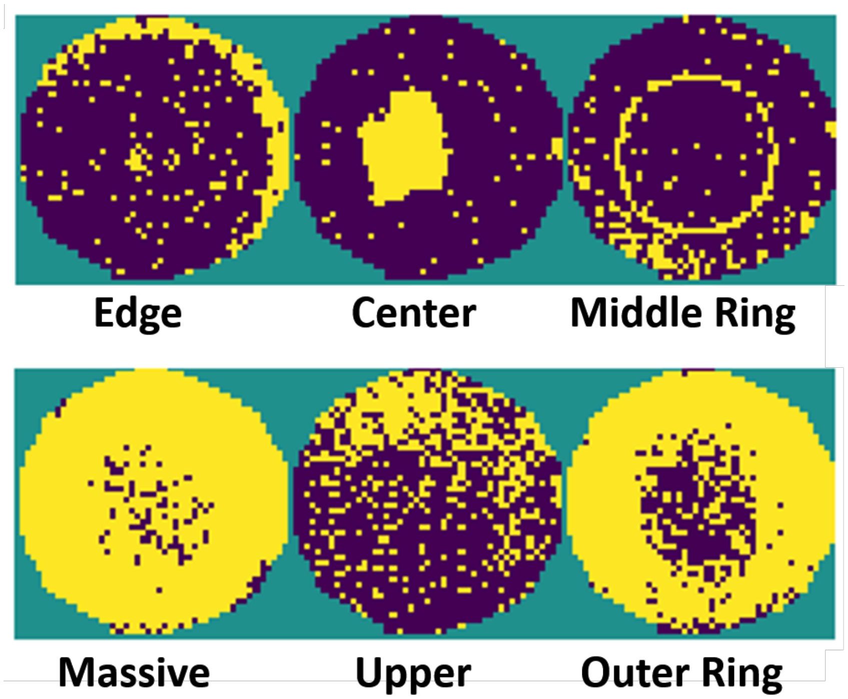


Figure 4.29: High Density Concepts

## 4.6 Learnability Measure

One of the challenges in practice for building a concept recognizer is choosing the training samples. This is because concept recognition is based on unsupervised learning. When a set of training samples are given, it is unknown if the training samples should be treated as a single concept class or multiple concept classes. It would be desirable to have a way to assess that question.

The diaginality measure $\boldsymbol{\Delta}$ provides a convenient way to develop a method for that purpose. The quantity $\boldsymbol{LB} = \frac{\mathbf{1}}{\boldsymbol{\Delta}}$ can be thought of as how good the projection matrices from the Tucker decomposition can be used to represent a wafer map, i.e. similar to the notion of model *fitting* in machine learning. For example if the Tucker model is built from a set of identical wafer maps, we would have $\boldsymbol{\Delta} = \mathbf{0}$ i.e. $\boldsymbol{LB} = \boldsymbol{\infty}$ for every wafer. Intuitively, we can use $\boldsymbol{LB}$ to indicate how well a tucker model fits a wafer map.

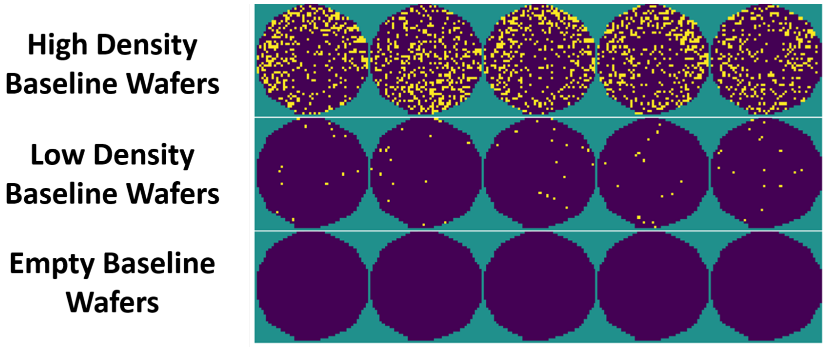


Figure 4.30: Baseline Concepts

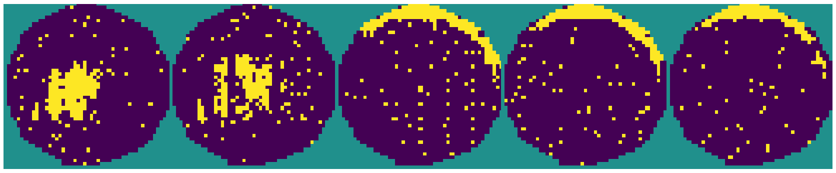


Figure 4.31: A Training Set With Samples From Two Concepts

To illustrate the point, Figure 4.31 shows five wafer maps from the *Edge* and *Center* concepts in the medium failure density group. Their $\boldsymbol{LB}$ values are [1.31, 1.23, 2.4, 2.63, 3.03] following the same order as shown in the figure. Observe that the $\boldsymbol{LB}$ values for the two center failures wafer maps are noticeably lower than the other three.

Recall that for the concepts in the medium-loss group, five samples are used in training. Table 4.4 shows their $\boldsymbol{LB}$ values after the training. Observe that the average $\boldsymbol{LB}$ value in the Center Failures case is much lower than others while the last two samples have noticeably lower $\boldsymbol{LB}$ values. This indicates that for this concept, there might be room for improvement in terms of choosing a better training set.

| Training Wafers | Learnability Vector | Average |
|---|---|---|
| Edge Failures | [4.17, 4.35, 5.26, 4.35, 3.85] | 4.39 |
| Systematic Failures | [4.35, 4.35, 3.85, 4.0, 4.17] | 4.14 |
| Center Failures | [2.56 , 2.38 , 2.56 , 1.89 , 1.54] | 2.19 |

Table 4.4: $\boldsymbol{LB}$ values from the medium-loss group

An application to average learnability of training wafers $\boldsymbol{LB}$ is to choose the best training wafer set. Suppose we are given a set $\boldsymbol{S}$ of $\boldsymbol{n}$ samples. The goal is to choose a subset of samples as our training set. Suppose we have also a test set $\boldsymbol{T}$ of $\boldsymbol{m}$ samples. Samples in $\boldsymbol{S}$ and $\boldsymbol{T}$ are manually selected and visually determined to be in the same concept. The goal is to choose the best set of samples from $\boldsymbol{S}$ to build a Tucker model. Note that from this perspective, it might be more intuitive to think the method is for filtering out "noisy" samples rather than for "choosing" samples.

The idea is simple. Suppose samples in $\boldsymbol{S}$ are ordered by their $\boldsymbol{LB}$ values as $\boldsymbol{s_1, \ldots, s_n}$. Let $\boldsymbol{S_i = \{s_1, \ldots, s_i\}}$. The average $\boldsymbol{LB_i}$ is calculated by applying the Tucker model from the set $\boldsymbol{S_i}$ to $\boldsymbol{T}$. For example, let $\boldsymbol{n = 10}$ and $\boldsymbol{m = 15}$. Figure 4.32 shows the average $\boldsymbol{LB}$ results for the three concepts from the medium-loss group.

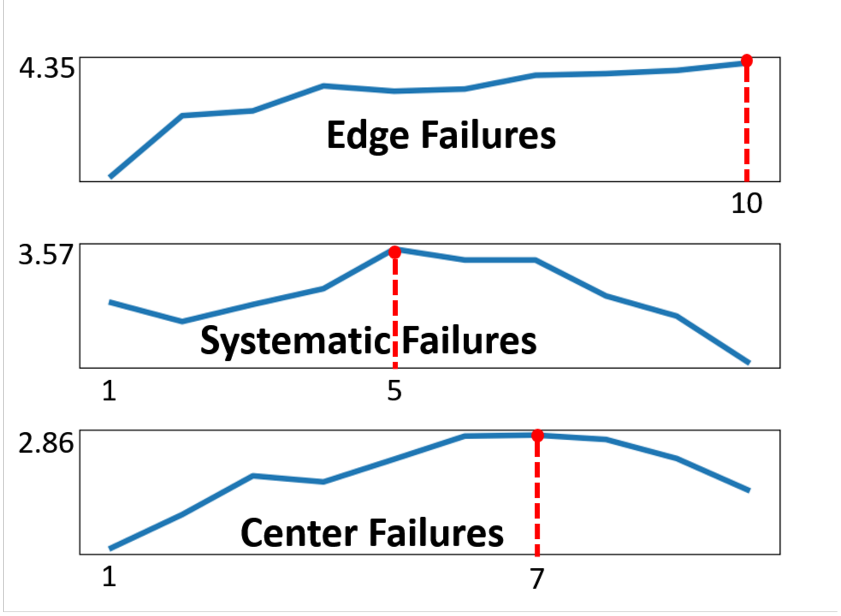


Figure 4.32: Deciding the Best Training Set Using Average $\boldsymbol{LB}$

As shown in the figure (x-axis is the number of wafer maps and y-axis is the average $\boldsymbol{LB}$), for Edge Failures, it reaches the best result when all 10 training wafers are used. For others, the best result happens with fewer samples. The models were re-built using the new training sets. Table 4.5 shows the recognition summary with the improved training wafers. The number of correctly recognized wafers certainly improved. However, the number of escapes (false negatives) is still high for both Center failures and Grid failures. The reason for the large number of escapes, as will be shown in Chapter 5 is that we are missing a few concepts. For example, when automatically extracting concepts, four different grid patterns were found. For humans all four looks similar, however, for a Tensor-based model they are very different. Moreover, many of the center failures false negatives have much less failure concentration than the training wafers we chose. In fact, automated concept extraction found two center failure concepts, where the one with the less failure concept ration captured most of the false negatives.

| Concept | Edge Failures | Grid Failures | Center Failures |
|---|---|---|---|
| Correct Recognition | 39 | 61 | 56 |
| False Negatives | 1 | 100+ | 38 |
| False Positives | 0 | 0 | 0 |

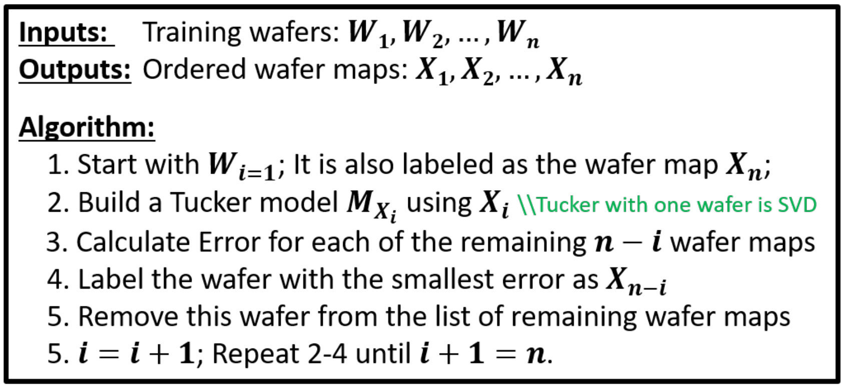
Table 4.5: Summary of The Multi-Concept Recognizer for Medium Density Wafers With Improved Training Wafers

## 4.7 Primitive Concept Identification

As discussed in Chapter 3, one of the issues in the GANs-based approach for concept recognition is in the challenge for achieving a *converging point* in training where the *discriminator* at that point is used as a recognizer for the concept. It is well known that training GANs require several dedicated techniques to ensure *training stability* [4], where both the *generator* and the *discriminator* continue to improve their accuracy while the discriminator is not forced to over-fit the training samples. Maintaining learning stability in order to achieve a converging point can be difficult when the underlying concept to be learned is *complex*.

To overcome this issue, it is desirable to develop a method that can take a given set of wafer maps and decide if the entire set is suitable for training a recognizer. In this method, wafer maps in a given set are partitioned into groups where each group represents a more primitive concept. Then, one can decide if GANs training should be applied to each primitive concept individually or some of them combined, depending on how difficult to converge the training. This method is called *primitive concept identification* to differentiate it from concept recognition.

The primitive concept identification is implemented with two algorithms as presented in Figure 4.33 and in Figure 4.34, respectively. Given a set of wafer maps $\mathbf{W_1, \dots, W_n}$, Algorithm 1 first reorders these wafer maps such that similar wafer maps stay closer to each other. The reordered wafer maps are then processed by Algorithm 2 to identify a primitive concept and its associated wafer maps.

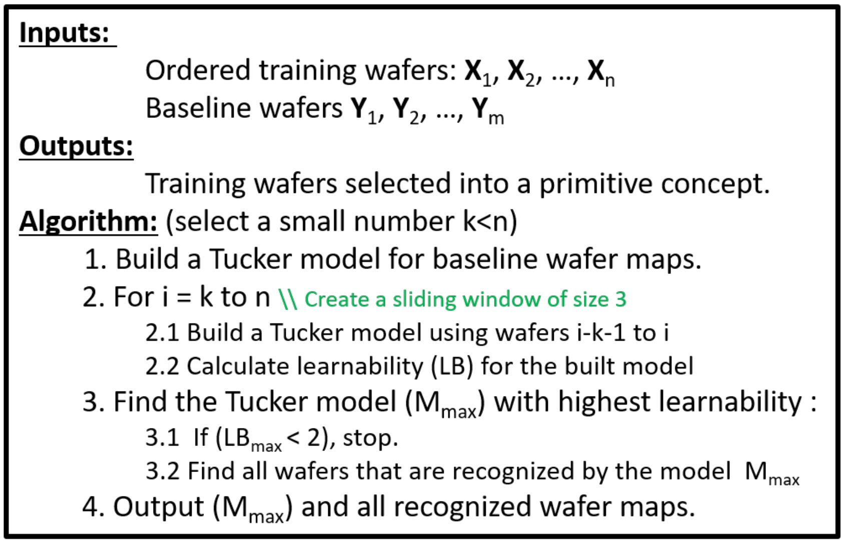


```
Inputs:   Training wafers: $W_1, W_2, \dots, W_n$
Outputs:  Ordered wafer maps: $X_1, X_2, \dots, X_n$

Algorithm:
    1. Start with $W_{i=1}$; It is also labeled as the wafer map $X_n$;
    2. Build a Tucker model $M_{X_i}$ using $X_i$ \\Tucker with one wafer is SVD
    3. Calculate Error for each of the remaining $n - i$ wafer maps
    4. Label the wafer with the smallest error as $X_{n-i}$
    5. Remove this wafer from the list of remaining wafer maps
    5. $i = i + 1$; Repeat 2-4 until $i + 1 = n$.
```

Figure 4.33: Algorithm 1: Wafer Reordering

The key idea in Algorithm 1 is based on using the distance measure $\mathbf{\Delta}$ as a similarity measure between two wafer maps. Given a wafer map $\mathbf{W}_i$, A Tucker-based model is build using this wafer as explained in 4. This is essentially performing a Singular Value Decomposition (SVD), since only one training wafer map is used. The model is then applied to the each wafer $\mathbf{W}_j$ of remaining wafer maps and the $\mathbf{\Delta}$ value is calculated for each wafer map. The wafer with the smallest $\mathbf{\Delta}_{i\to j}$ is put next to $\mathbf{W}_i$ and then, in the next step this wafer map $\mathbf{W}_j$ is used to build the Tensor-based model.



```
Inputs:
        Ordered training wafers: $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$
        Baseline wafers $\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_m$
Outputs:
        Training wafers selected into a primitive concept.
Algorithm: (select a small number k<n)
    1. Build a Tucker model for baseline wafer maps.
    2. For i = k to n \\ Create a sliding window of size 3
        2.1 Build a Tucker model using wafers i-k-1 to i
        2.2 Calculate learnability (LB) for the built model
    3. Find the Tucker model ($M_{max}$) with highest learnability :
        3.1 If ($LB_{max}$ < 2), stop.
        3.2 Find all wafers that are recognized by the model $M_{max}$
    4. Output ($M_{max}$) and all recognized wafer maps.
```

Figure 4.34: Algorithm 2: Wafer Concept Identification

Given the wafer maps ordered based on their similarity, in Algorithm 2 a Tensor-based model is built based on a sequence of $\boldsymbol{k}$ consecutive wafer maps. This $\boldsymbol{k}$ can be a small number such as 2 or 3. Consequently, for $\boldsymbol{n}$ given wafer maps, there are $\boldsymbol{n-k+1}$ Tensor-based recognizer models built. Each model is based on $\boldsymbol{k}$ wafer maps. Then the average $\boldsymbol{LB}$ across these $\boldsymbol{k}$ wafer maps can be calculated. This average $\boldsymbol{LB}$ is treated as the learnability for the subset. The model $\mathrm{M}_{\boldsymbol{max}}$ with the highest learnability is selected. This model is applied to every wafer map in the set to calculate its $\boldsymbol{\Delta}$ value.

To decide if a wafer map is recognized by the model $\mathrm{M}_{\boldsymbol{max}}$ or not, a baseline model is used. This baseline model is based on a selected set of wafer maps representing “Random Failures” with no specific pattern. The model $\mathrm{M}_{\boldsymbol{base}}$ is built and the $\boldsymbol{\Delta}$ value for each wafer map based on this model is also calculated. Then, a wafer map is classified as recognized by $\mathrm{M}_{\boldsymbol{max}}$ if its $\boldsymbol{\Delta}$ from $\mathrm{M}_{\boldsymbol{max}}$ is smaller than its $\boldsymbol{\Delta}$ from $\mathrm{M}_{\boldsymbol{base}}$. In other words, the classification is done in a relative way, not absolutely based on a given error threshold as discussed in Chapter 4.

An application of Algorithm 2 onto a list of ordered wafer maps can give a primitive concept and all wafers recognized by the primitive concept recognizer $\mathrm{M}_{\boldsymbol{max}}$. Algorithm 2 can then be repeatedly applied to the remaining wafers to extract a second primitive concept, and so on.

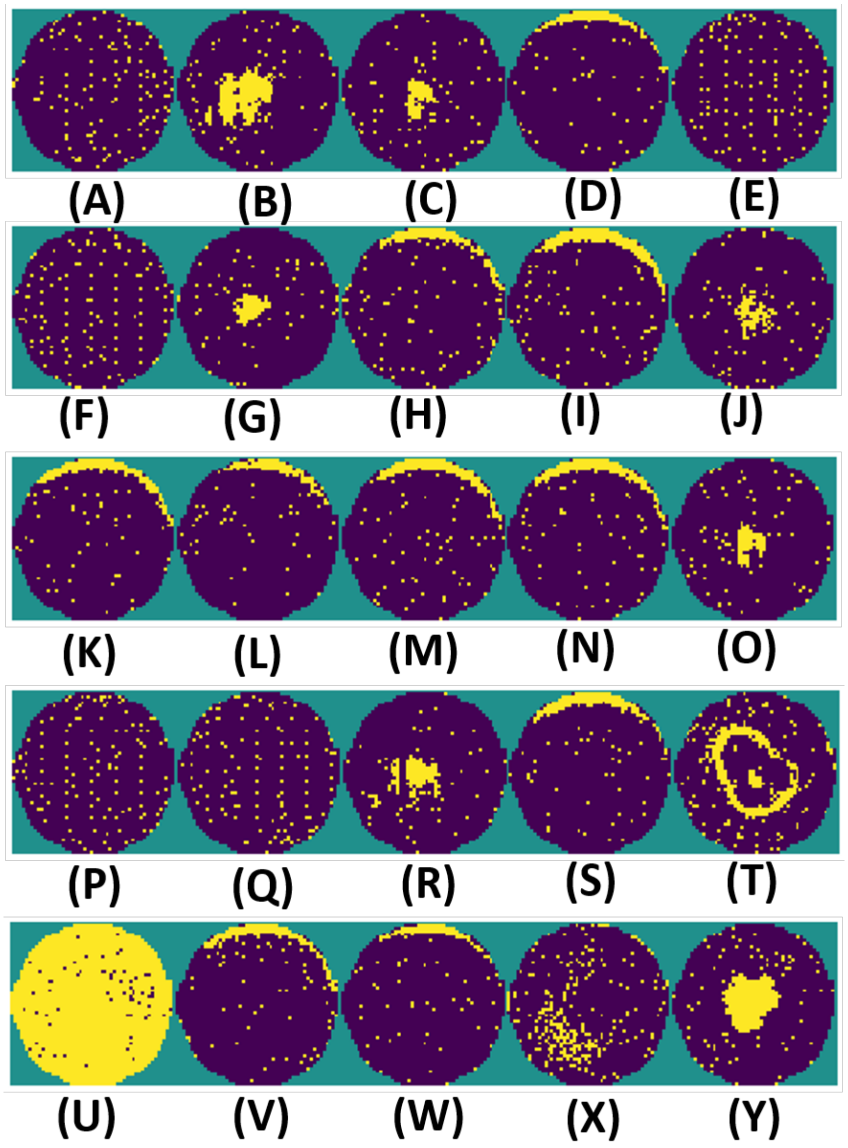


Figure 4.35: An Illustrative Example With 25 Wafer Maps

### 4.7.1 An illustrative example

To illustrate the operations of Algorithm 1 and Algorithm 2, Figure 4.35 shows 25 wafer maps as the input to Algorithm 1. After running Algorithm 1 on this list, the ordered wafer maps is given as "U, S, M, N, K, D, V, W, I, L, H, T, O, C, R, J, G, Y, B, X, E, F, Q, P, A". Note that the wafer map A is processed first and hence, in the list it appears last, i.e. the list is in reverse ordering as they are selected.

Observe that wafer maps S to H are similar. Wafer maps O to B are similar and wafer maps E to A are similar. Wafer maps U, T, and X are not similar to any other. Ideally, in the next step the primitive concept recognition should identify the three groups of wafer maps as three primitive concepts.

### 4.7.2 Primitive concepts

In Algorithm 2, let $\boldsymbol{k} = \mathbf{3}$ and hence there are 23 Tucker models built, each based on consecutive 3 wafer maps in the ordered list. Figure 4.36 shows the learnability values (i.e. average $\boldsymbol{LB}$ across all 3 wafer maps) for the 23 models. The subset [E,F,Q] has the highest learnability value 7.69. Hence, its model $\mathrm{M}_{\boldsymbol{EFQ}}$ is selected.

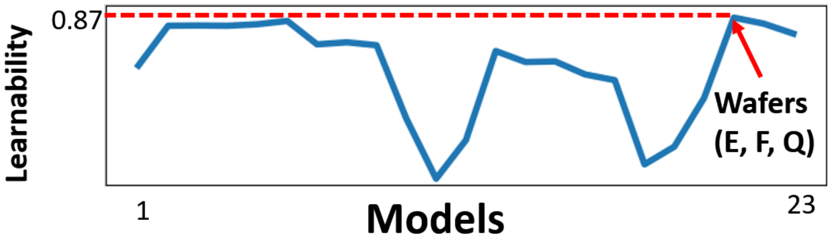


Figure 4.36: Learnability Values Across All Models With $\boldsymbol{k} = \mathbf{3}$

After applying $\mathrm{M}_{\boldsymbol{EFQ}}$ and $\mathrm{M}_{\boldsymbol{base}}$ to the remaining 22 wafer maps, A, P have $\boldsymbol{\Delta}$ values from $\mathrm{M}_{\boldsymbol{EFQ}}$ than from $\mathrm{M}_{\boldsymbol{base}}$: $\mathrm{M}_{\boldsymbol{EFQ}}(\mathrm{A}) = \mathbf{0.216} < \mathrm{M}_{\boldsymbol{base}}(\mathrm{A}) = \mathbf{0.311}$, $\mathrm{M}_{\boldsymbol{EFQ}}(\mathrm{P}) = \mathbf{0.196} < \mathrm{M}_{\boldsymbol{base}}(\mathrm{P}) = \mathbf{0.319}$.

Hence, A,P are recognized as part of the same primitive concept as E,F,Q. For the rest, their $\boldsymbol{\Delta}$ values from $\mathrm{M}_{\boldsymbol{base}}$ are smaller than from $\mathrm{M}_{\boldsymbol{EFQ}}$.

For example, $\mathrm{M}_{\boldsymbol{EFQ}}(\mathrm{X}) = \mathbf{0.84} > \mathrm{M}_{\boldsymbol{base}}(\mathrm{X}) = \mathbf{0.75}$, and $\mathrm{M}_{\boldsymbol{EFQ}}(\mathrm{T}) = \mathbf{1.463} > \mathrm{M}_{\boldsymbol{base}}(\mathrm{T}) = \mathbf{1.242}$, and so on.

After [E,F,Q,P,A] are recognized as one primitive concept, there are 20 wafer maps remaining. Algorithm 2 is applied again. The maximum learnability value is 6.67 for the model $\mathrm{M}_{\boldsymbol{VWI}}$ based on V,W,I. Then, this model recognizes [S,M,N,K,D,L,H] as the same primitive concept.

Algorithm 2 is applied to the remaining 10 wafer maps. The maximum learnability value is 4.35 for the model $\mathrm{M}_{\boldsymbol{OCR}}$ based on O,C,R, which recognizes [J,G,Y,B]. For the remaining 3 wafer maps, U,T,X, the learnability is 1.73 and hence, there is no acceptable model.

### 4.7.3 Wafer maps with edge failures

As discussed in Chapter 4, a Tucker model can be location sensitive and is not rotation invariant. For example, given a wafer map with an edge failure pattern in one direction if the map is rotated by 90 degrees, it would be considered as a different primitive concept. In contrast, the GANs-based approach in [2] does not have this restriction.

Figure 4.37 shows a list of 18 wafer maps all appearing to have some sort of edge failures. After running Algorithm 1 and Algorithm 2 as before, $\mathrm{M}_{\boldsymbol{nop}}$ based on n,o,p has the learnability 5.88 and recognizes [d,e,k,l,m,q,r]. Then, $\mathrm{M}_{\boldsymbol{hij}}$ based on h,i,j has the learnability 2.7 and recognizes [a,b,c,g]. At the end, wafer map f stands alone.

This indicates that two Tensor-based models should be built one based on wafers [n,o,p,d,e,k,l,m,q,r], and another model based on wafers [h,i,j,a,b,c,g]. For GANs-based recognizers, the analyst can decide that GANs can learn the complex concept based on all

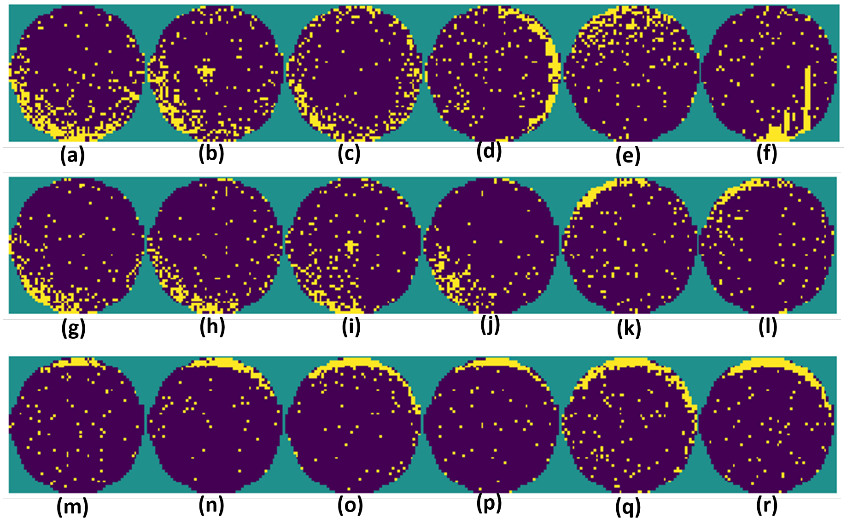


Figure 4.37: Edge Training Wafers

wafers 17 wafers and hence train only one recognizer. However, the following experiment shows that it is easier to train a model based on each primitive concept.

## 4.8 Helping The GANs-Based Approach

Suppose the set of 13 wafer maps in Figure 4.38 are given for training a GANs-based recognizer. First, Algorithm 1 and Algorithm 2 are applied to identify primitive concepts in this set. Two primitive concepts are found: Concept-1 based on [2,9,10,11,13] and Concept-2 based on [3,4,5,7,8]. Wafer maps 1,6,12 stand alone.

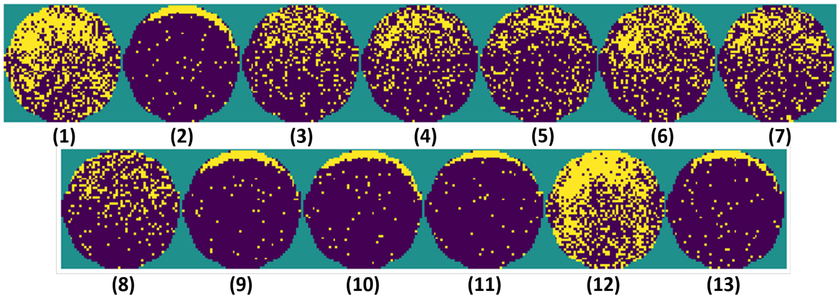


Figure 4.38: Training Set Given To GANs Training

Figure 4.8 shows how the GANs training behaves when only wafer maps from Concept-1 are used, when wafer maps from Concept-1 and Concept-2 are combined, and when all maps are used. The x-axis shows a range of 10K training iterations. The y-axis shows the *separability* between in-class samples and out-of-class samples in the validation dataset - a larger value shows that the recognizer is doing better [2].

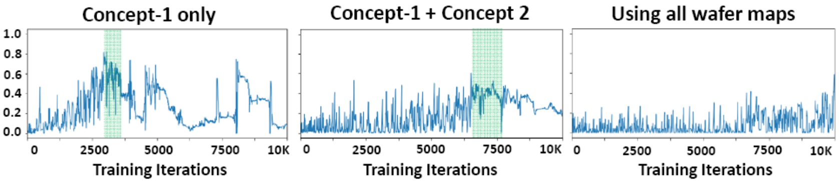


Figure 4.39: Converging Issue in GANs Training

In the first case, observe the recognizer model continues to improve and peak around 3500 iterations. The training can stop there and extract the model. In the second case, the improvement is slow and peaks around 6500 iterations. But the separability value is much lower. In the third case, there is no clear converging point to pick a model. Figure 4.39 shows that the proposed primitive concept identification approach can be used to help pick the best subset of wafer maps to train a GANs-based recognizer. In this example, two recognizers might be trained for Concept-1 and Concept-2 separably.

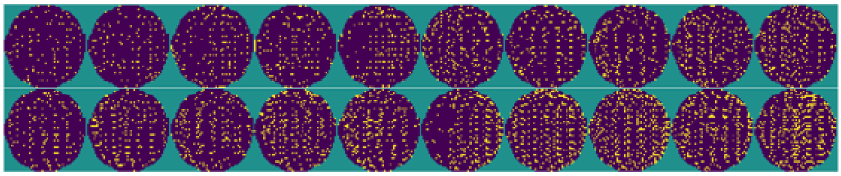


Figure 4.40: Grid Concept Training Wafers

Another example is shown Figure 4.40. The figure shows 20 wafers that have some sort of a grid pattern. To the human eye, it looks like all the 20 wafers belong to the same concept. However, the wafers can be further split into four different primitive concepts. When *primitive concept identification* is applied to the 20 wafers, four primitive concepts were found, these four primitive concepts are shown in figure 4.41.

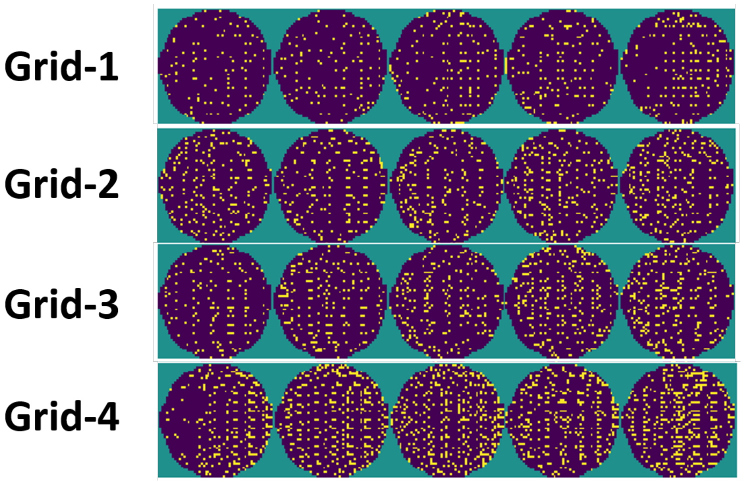


Figure 4.41: Four Primitive Grid Patterns

Although it does not seem intuitive to the human eye, training four recognizers one for each grid pattern makes training the GANs a lot quicker.

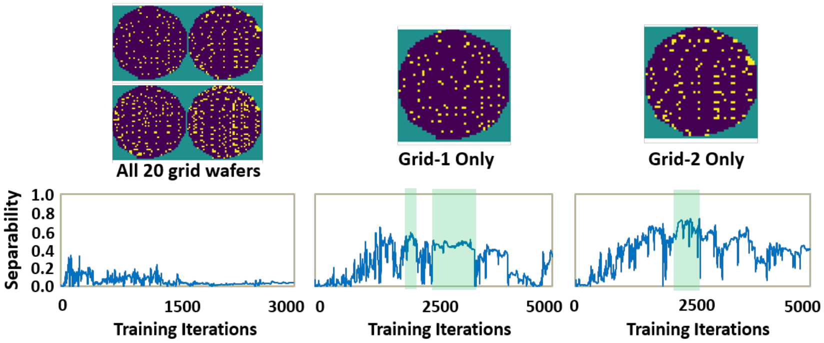


Figure 4.42: Training with All Grid Wafers vs Splitting Into Primitive Concepts

Figure 4.42 shows average separability during GANs training. The figure shows the imporvement in separability when training with a single primitive concept versus when combining primitive concepts, even if the wafers look somewhat similar to the human eye.

## 4.9 Adversarial Examples in GANs-based Recognizers

Another issue with the GANs-based concept recognizers is that the discriminator is a convolutional neural network which can have *adversarial examples* [49]. In our context, an adversarial example is a slightly perturbed wafer map that can cause the concept recognizer to miss it. It is observed in [2] that if the wafer maps in a given training set include more diverse patterns, then it is more likely for adversarial wafer maps to occur.

Wafer maps [2,9,10,11,13] in Figure 4.37 are given to the GANs approach [2] to train a recognizer. Based on map 2, three pixels are randomly perturbed each time to produce 3000 maps. Among them, 1 adversarial example is found, which is not recognized by the GANs-based recognizer. This adversarial example is shown in Figure 4.9.

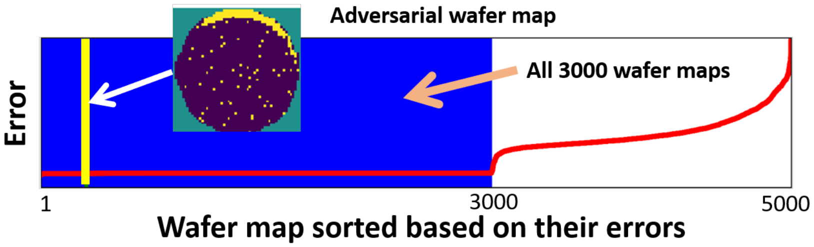


Figure 4.43: An Adversarial Example

In contrast, if we build the Tucker model using [2,9,10,11,13] and apply this model to the 3000 sampled maps, together with 2000 additional wafer maps from the automotive product line, their error values are sorted and shown in Figure 4.43. As seen, all 3000 sample maps have very low error values. The error of the adversarial example is very low. This result shows that a Tucker model can be used in a post-recognition step after applying a GANs model to avoid missing an adversarial example.

## 4.10 Final Remarks

One of the problems that happens because of our choice of the distance metric $\boldsymbol{\Delta}$, is that the model can confuse two "inverted" wafers. For example, a center failure concept can recognize a ring wafer, and a low density random failure concept can recognize a massive failure wafer.

This happens because we represent the wafer map as a +1/-1 value for passing/failing dies, hence two "inverted" wafers would be similar if we flip the sign of each element in the matrix representing the wafer map. Hence, the resulting diagonal elements in the core matrix will be exactly the same, but their signs will be flipped. Also, the diagonality measure $\boldsymbol{\Delta}$ squares all the elements in the core matrix $\boldsymbol{G}$, hence ignores this sign difference. Our solution to this problem is to add a sanity check on the sign of the top diagonal element, if it is not the same sign as one of the training wafers top diagonal element, it will not get recognized by this concept.

## 4.11 Conclusion

In this chapter, we introduced the details of implementing our first Tensor method. We introduced how to use Tucker decomposition to build the concept recognizer model. The key idea is to define a distance measure between a set of training wafer images and a given wafer image to be classified. Four distance measures are extensively studied, and experiments showed using the measure $\boldsymbol{\Delta}$ in Equation 4.1 is our best choice, however, $\boldsymbol{\Delta'}$ can also be used. Finally a learnability measure is introduced as a way to quantify the quality of the training wafers in representing a concept. An algorithm for selecting the best training wafers to represent each concept is also introduced.

# Chapter 5

# Our Automated Concept Recognition Flow

## 5.1 Introduction

A classification flow takes a collection of wafer plots as input. The output is a classification of these wafers such that (1) plots showing similar patterns of interest are put into the same group and assigned with a unique label of interest; (2) plots that cannot be classified with a label of interest are classified as "unrecognized". At the high level, this can be seen as an unsupervised learning problem where wafers enter the flow without a label and leave the flow with one.

The purpose of this chapter is to develop an automated flow for the wafer plot classification. Because the underlying problem is unsupervised, it is intuitive to think that the problem can be solved with a learning algorithm like clustering. For example, we can develop a way to measure the *similarity* between two given wafers $\boldsymbol{W_i}, \boldsymbol{W_j}$ as $\boldsymbol{k(W_i, W_j)}$. The function $\boldsymbol{k()}$ essentially serves as a *kernel function* in traditional kernel-based learning. Then, one can apply a clustering algorithm using $\boldsymbol{k()}$ as its "distance" measure.

There are two major challenges with a clustering approach. First, we need to define a $\boldsymbol{k}()$ function such that the result of classification would be acceptable to human perception. In other words, we would like the classification to follow human perception such that if two wafers are clustered together, they would be likely accepted by the engineer that the two plots show similar patterns of interest.

The second challenge is more fundamental, that clustering is known to be very non-robust. For example, Fig. 5.1 illustrates one aspect in its non-robustness. Suppose we apply clustering to some earlier samples. On the left, it would be easy to see there are three natural clusters. The main issue is where to draw the boundary for each cluster.

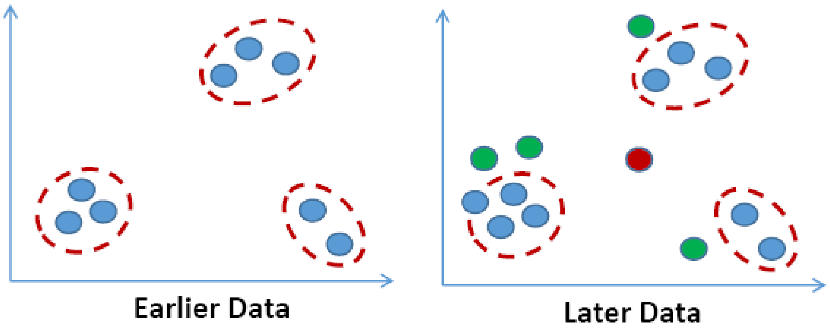


Figure 5.1: Illustration of an issue with clustering

On one hand, we do not want the boundary to be too tight, or we may classify a similar wafer plot as an outlier. On the other hand, we do not want the boundary to be too loose, or we may classify an outlier as one of the interesting patterns. This is an inherent issue by using a clustering approach that it lacks the information to tell what the best way is to draw a clustering boundary.

In Fig. 5.1, suppose later more samples become available. As seen in the illustration, those boundaries drawn earlier now become less optimal. For example, it would make more sense to include those green samples into those corresponding clusters. The red sample, on the other hand, should be kept as an outlier.

Instead of naively solving it as a clustering problem, in this chapter we present a more sophisticated classification flow. The focus is on making the flow as robust as possible, with two objectives in mind: (1) we would like our classification results to be acceptable based on human perception; (2) we would like our classification results to be *consistent* across production time line where wafers are provided over time on a batch-by-batch basis.

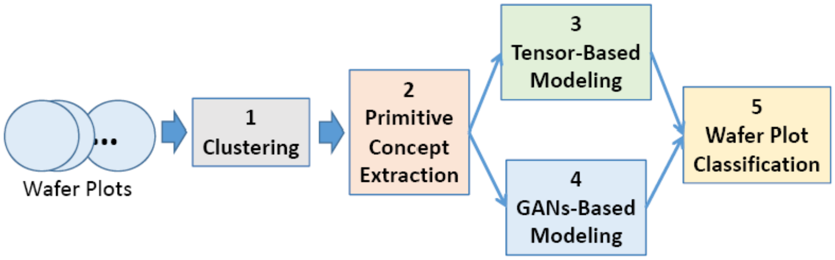


Figure 5.2: Steps in our classification flow

Fig. 5.2 shows the steps involved in our classification flow:

1. A clustering scheme is applied to group similar wafer plots. The main concerns here are (1) to define a distance function $\boldsymbol{k}()$, and (2) to choose the clustering algorithm that works best with the distance function.

2. The goal of this step is to pick the five wafer plots that are the best to capture a *primitive concept* (i.e. a class of patterns of interest) shown in each cluster of wafer plots. The reason why it is five will be explained later in the section explaining this step.

3. For each primitive concept, the five wafer plots are used to build a Tensor-based *recognizer* for the concept as explained in Chapter 4

4. For each primitive concept, the same five wafer plots are used to build a *recognizer* for the concept based on GANs (Generative Adversarial Neural Networks) as

explained in Chapter 3.

5. For each primitive concept, we now have two recognizers. Both recognizers are used to classify a wafer plot. In this step, a classification rule is implemented in order to improve the robustness of the overall classification result. This rule will be discussed in detail in the corresponding section.

This Chpater puts together what we have learned in the previous Chapters and develops a classification flow that can be deployed into a production line. Complete automation of the flow is the focus. To achieve this, there are two major challenges as the following:

- In practice, wafer plots do not come at once. Collections of wafer plots become available over time. Our classification flow can be equipped with some built-in concept recognizers as a start. However, as more wafer plots are examined, it might become necessary to learn a new concept recognizer on the fly.

  Therefore, automation means that (1) we need to automatically detect there is a need to learn a new concept recognizer; (2) the GANs-based learning needs to be automated; and (3) the Tensor-based learning needs to be automated. One may wonder why the last two are of concern. This is because in learning a model, usually some user-specified parameters are required. For example, in GANs-based training one needs to decide a stopping criterion [2].

- As batches of wafer plots come in over time, it is important to maintain *consistency* in the classification. On one hand, we desire that the capability of our classification flow can be improved as more wafer plots are seen over time. On the other hand, we do not want the label of a previously-classified wafer plot to be changed at a later time.

## 5.2 Step 1: Clustering

A wafer plot is encoded as a matrix where a $+1$ corresponds to passing, $-1$ corresponds to failing, and $0$ means no die on the location. Each matrix is re-sized into a $48 \times 48$ matrix internally used by the flow. This re-sizing is done with the image re-sizing tool available from the Python machine learning library Scikit-learn [50].

To implement a clustering scheme, we need a distance function $\boldsymbol{k}()$ to measure similarity between any pair of two matrices. Given a wafer matrix $\boldsymbol{W_a}$, we can build a Tensor-based model $\boldsymbol{T_a}$ based on only $\boldsymbol{W_a}$ to recognize wafer plots similar to $\boldsymbol{W_a}$. This model building is based on the approach proposed in [36] and explained in detail in Chapter 4. When $\boldsymbol{T_a}$ is applied to a wafer $\boldsymbol{W_b}$, the result is a transformed core matrix $\boldsymbol{G_b}$. What the particular Tensor technique achieves is that a resulting $\boldsymbol{G}$ matrix has this special property as illustrated in Fig. 5.3.

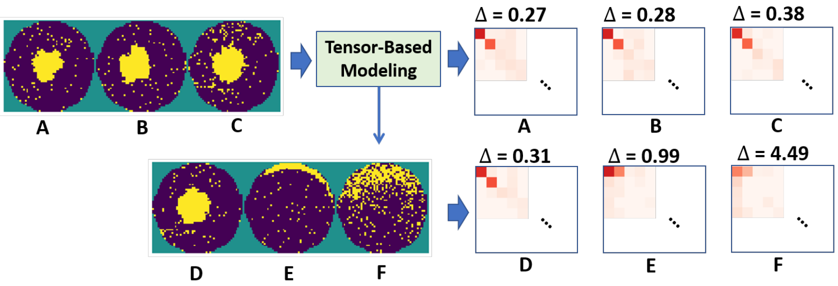


Figure 5.3: Effect of Tensor-based modeling - Examples

If $\boldsymbol{W_b}$ is similar to $\boldsymbol{W_a}$, $\boldsymbol{G_b}$ should have relatively large absolute values in diagonal entries and close-to-zero values in non-diagonal entries. In contrast, if $\boldsymbol{W_b}$ is not similar to $\boldsymbol{W_a}$, $\boldsymbol{G_b}$ would have large values in non-diagonal entries.

The Tensor model can be built based on one or more wafer plots together. In Fig. 5.3, three wafer plots A, B, and C are used. As seen, after the model is built and applying the model to the three plots A, B, and C, their transformed matrices have large absolute values in the diagonal entries (darker) and close-to-zero values in non-diagonal entries (lighter). For those remaining entries left uncolored for simplicity reason, note that their values are all very close to zero.

When we apply the model to plots D, E, and F, the transformed matrix from plot D shows the same property, but the matrices from plots E and F do not. In particular, the transformed core matrices from plots E and F have large values in the non-diagonal entries (e.g. dark entry at 1st row 2nd column).

Let $\boldsymbol{\sigma_{ij}}$ denote the value of entry $\boldsymbol{(i,j)}$ in a transformed matrix. We utilize this "diagonal-ness" property the distance measure $\boldsymbol{\Delta}$ as the following equation:

$$\boldsymbol{\Delta} = \frac{\sum_{\forall i \neq j} \boldsymbol{\sigma_{ij}^2}}{\sum_{\forall i} \boldsymbol{\sigma_{ii}^2}} \tag{5.1}$$

A smaller $\boldsymbol{\Delta}$ means the wafer plot is closer to those used to build the model. A larger $\boldsymbol{\Delta}$ means the reverse.

Now given two wafer plots $\boldsymbol{W_a}$ and $\boldsymbol{W_b}$, let $\boldsymbol{T_a}$ and $\boldsymbol{T_b}$ be the respective two Tensor models built using $\boldsymbol{W_a}$ and $\boldsymbol{W_b}$ individually. Let $\boldsymbol{\Delta_{a\to b}}$ denote the $\boldsymbol{\Delta}$ by applying $\boldsymbol{T_a}$ to $\boldsymbol{W_b}$. Let $\boldsymbol{\Delta_{b\to a}}$ be the reverse. In general, $\boldsymbol{\Delta_{a\to b}} \neq \boldsymbol{\Delta_{b\to a}}$.

A naive thought could consider the average $\frac{\boldsymbol{\Delta_{a\to b}}+\boldsymbol{\Delta_{b\to a}}}{\boldsymbol{2}}$ as a measure of the distance between the two wafer plots. However, this measure has an issue to be used in clustering as illustrated in Fig. 5.4.

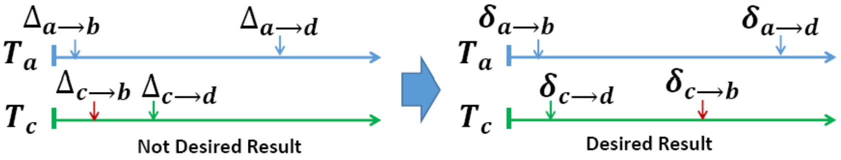


Figure 5.4: $\boldsymbol{W_b}$ tends to have a smaller $\boldsymbol{\Delta}$ value

Suppose we have two models $\boldsymbol{T_a}$ and $\boldsymbol{T_c}$. Suppose $\boldsymbol{W_b}$ is similar to $\boldsymbol{W_a}$ and not similar to $\boldsymbol{W_c}$, and $\boldsymbol{W_d}$ is similar to $\boldsymbol{W_c}$ and not similar to $\boldsymbol{W_a}$. Ideally, we want a small $\boldsymbol{\Delta_{a\to b}}$, a large $\boldsymbol{\Delta_{a\to d}}$, a small $\boldsymbol{\Delta_{c\to d}}$, and a large $\boldsymbol{\Delta_{c\to b}}$.

However, in our experiment we observe that there can be some wafer $\boldsymbol{W_b}$ such that it tends to result in a smaller $\boldsymbol{\Delta}$ value across different models even though these models are built from plots that look quite differently. For example, a wafer plot with low density random fails but still exposing a "Grid" trend (examples shown later) could have this property. Then, instead of having $\boldsymbol{\Delta_{c\to b}}$ to be large, we would end up in a situation where $\boldsymbol{\Delta_{c\to b}}$ is even smaller than $\boldsymbol{\Delta_{c\to d}}$. This problem is illustrated in Fig. 5.4.

To show that using $\boldsymbol{\frac{\Delta_{a\to b}+\Delta_{b\to a}}{2}}$ as a measure of the distance between two wafers does not work, we can run the HDBSCAN clustering on a small set of wafers that we know how they should be clustered, i.e. a benchmark for clustering. Figure 5.5 shows the benchmark set of 25 wafers. As can be seen in the Figure, the 25 wafers can be clustered into 5 clusters corresponding to *Grid, Small Center, Edge, Upper, and Random* concepts. The result of running the clustering algorithm on the benchmark wafers is that the clustering thinks that none of the 25 wafers belongs to any cluster.

Our solution to the problem is to use a baseline wafer plot such as a wafer without a failing die. This baseline plot serves as a reference point for all wafer plots. Let its model be $\boldsymbol{T_{base}}$. Then, we define a new directional distance measure between two wafer plots as:

$$\boldsymbol{\delta_{a\to b}} = \frac{\boldsymbol{\Delta_{a\to b}}}{\boldsymbol{\Delta_{a\to b}} + \boldsymbol{\Delta_{base\to b}}} \tag{5.2}$$

Note that this distance measure has the advantage of being normalized between 0 and 1, which makes the clustering more stable. This is why using $\boldsymbol{\delta'_{a\to b}} = \frac{\boldsymbol{\Delta_{a\to b}}}{\boldsymbol{\Delta_{base\to b}}}$ does not work as good. In fact running the clustring algorithm on the benchmark wafers shown in Figure 5.5, the clustering algorithm found only three clusters, putting the *Grid, Small Center, and Random* concepts into the same cluster as shown in Figure 5.6.

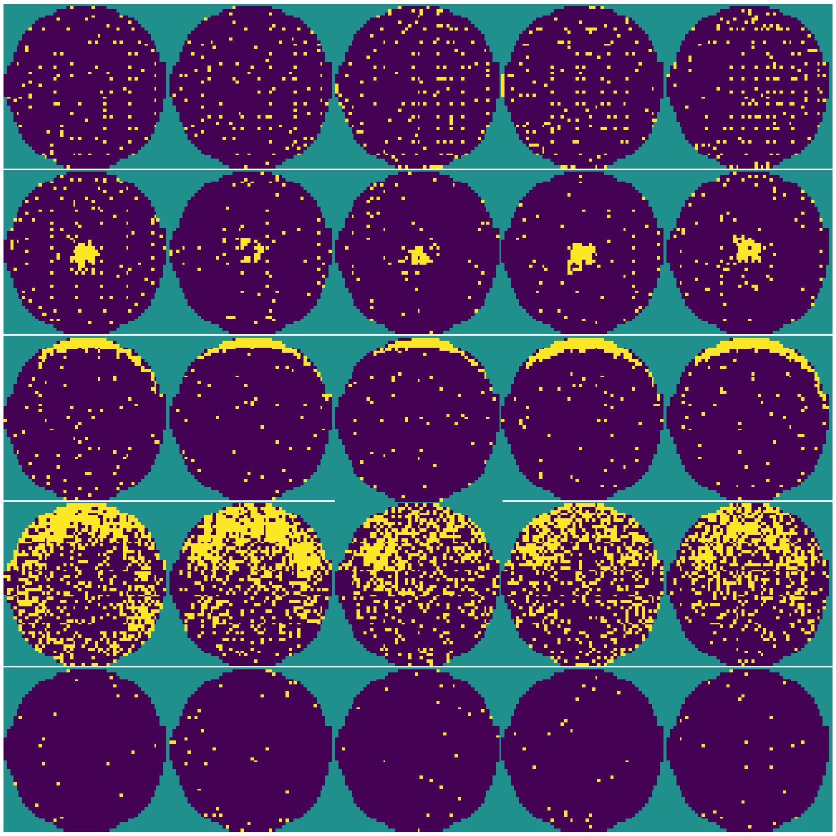


Figure 5.5: Benchmark Wafers

Notice that each $\boldsymbol{\Delta_{base\rightarrow *}}$ is used to re-scale the value $\boldsymbol{\Delta_{a\rightarrow *}}$. For most of the wafer plots exposing an interesting pattern, we found that $\boldsymbol{\Delta_{base\rightarrow *}}$ is rather large and consequently, $\boldsymbol{\delta_{a\rightarrow b}} \approx \frac{\boldsymbol{\Delta_{a\rightarrow b}}}{\boldsymbol{\Delta_{base\rightarrow b}}}$.

However, for a wafer plot $\boldsymbol{W_b}$ as shown in Fig. 5.4, $\boldsymbol{\Delta_{base\rightarrow b}}$ is also very small. This effectively scale up the $\boldsymbol{\delta_{c\rightarrow b}}$ value. In contrast, $\boldsymbol{\Delta_{base\rightarrow d}}$ remains large, effectively scaling down the $\boldsymbol{\delta_{c\rightarrow d}}$ value. Together, they achieve the desired outcome as shown in the figure.

Finally, when we ran the clustering algorithm on the benchmark wafers shown in Figure 5.5, the clustering algorithm was able to separate them into the desired five clusters as shown in Figure 5.7.

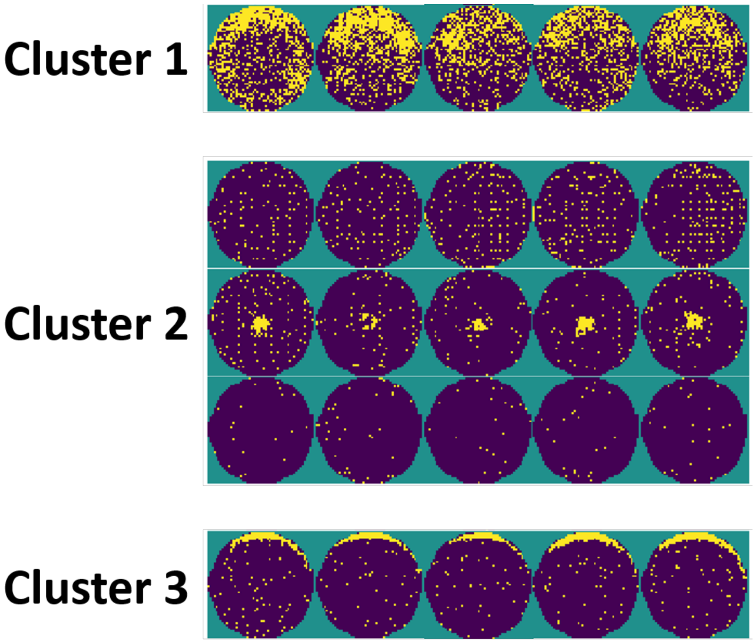


Figure 5.6: Clustring Result Using $\boldsymbol{\delta'}$

### 5.2.1 Clustering Algorithm

With equation (5.2), we can then define our distance function as $\boldsymbol{k(W_a, W_b)} = \boldsymbol{\frac{\delta_{a\rightarrow b}+\delta_{b\rightarrow a}}{2}}$. For every pair of $\boldsymbol{W_a, W_b}$ we calculate their distance. The result is a distance matrix which is passed to a clustering algorithm.

After extensive experiments, our choice of the clustering algorithm is the Hierarchical DBSCAN [51] (HDBSCAN). To run HDBSCAN, the minimum cluster size needs to be specified. The choice for this number is 5. The reason is because our earlier study in Chapters 3, and 4 have shown that 5 wafer plots are sufficient to construct a good GANs and Tensor model.

DBSCAN is a density-based clustering algorithm widely used today [52]. The intuition behind DBSCAN is the notion of density in a neighborhood, marking points in lower-density regions as out-of-cluster. HDBSCAN [53], is the hierarchical clustering extension to DBSCAN. By performing DBSCAN in a hierarchical fashion, the algorithm is

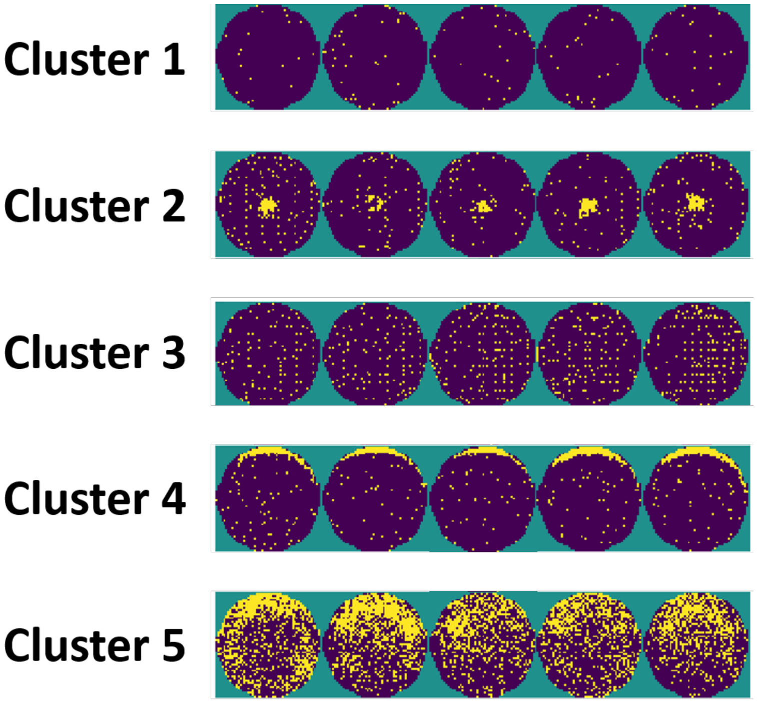


Figure 5.7: Clustring Result Using Equation 5.2

applied over varying cluster size values, where the integrated clustering result is selected based on cluster stability [54]. Some benefits over DBSCAN include the ability to find clusters of varying densities and being more robust. In our flow, the HDBSCAN tool from [51] is integrated.

### 5.2.2 Clustering result - experiment

For the experiment, 1243 wafer plots were taken from a first set of 50 lots. These lots were obtained from a high-reliability product line. We used this first set to build concept recognizers which are then applied to wafer plots from the subsequent lots.

It is important to note that clustering is used in our flow to obtain an initial grouping of wafer plots. Therefore, it does not have to be perfect. Subsequent steps utilize the clusters as a starting point to build the actual concept recognizers.

Before applying the clustering step, a *pre-filtering* step is applied first to filter out wafer plots with random fails, i.e. those would not be considered for showing a special pattern. In the work presented in [2], three GANs-based recognizers are built to recognize random-failing wafer plots based on their failing density. They are used in this pre-filtering step.

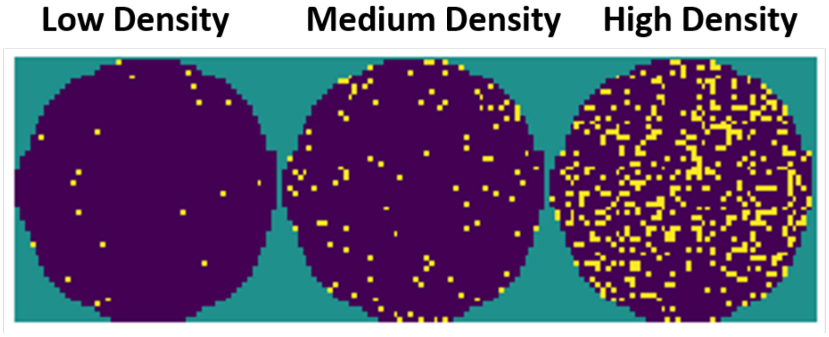


Figure 5.8: Examples of random-failing wafer plots

Fig. 5.8 shows one example for each failing density level. After this pre-filtering step, 290 wafer plots are left. Then, the clustering method described above is applied to these 290 wafers, resulting in 9 clusters (see table below) and 199 wafer plots not included in any cluster.

| Cluster | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Number of wafer plots | 9 | 8 | 11 | 16 | 5 | 21 | 6 | 5 | 10 |
| Max in-cluster distance | .48 | .48 | .46 | .48 | .48 | .41 | .37 | .37 | .36 |
| Min out-cluster distance | .50 | .49 | .49 | .49 | .49 | .47 | .42 | .40 | .40 |

The table above provides a summary of the clustering result. As seen, most of the clusters have a small number of wafer plots. The “Max in-cluster distance” is the maximum distance between any two plots in the respective cluster. The “Min out-cluster distance” is the minimum distance between a plot in the respective cluster and a plot outside. The difference between the Max and the Min can be thought of as a stability measure for the cluster. We see that clusters 6-9 are more stable than clusters 1-5.

Fig. 5.9 shows two example plots from each cluster. Notice that clusters 1-4 look more similar to each other than to clusters 7-9. Cluster 5 also looks more similar to clusters 1-4. These mutual similarities can be a cause for their smaller Max/Min differences seen in the table above.

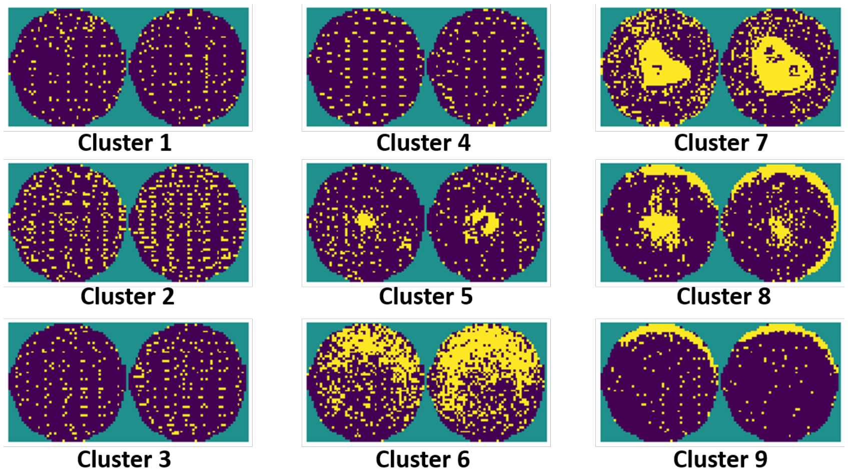

Cluster 1
Cluster 4
Cluster 7
Cluster 2
Cluster 5
Cluster 8
Cluster 3
Cluster 6
Cluster 9

Figure 5.9: Two example plots from each cluster

Fig. 5.10 then shows five examples of wafer plots not included in any cluster. From the left, notice that the 3rd plot looks similar to those in cluster 5 and the 4th looks similar to those in cluster 1. These similarities can also mean smaller in-cluster/out-cluster distance differences for those clusters.

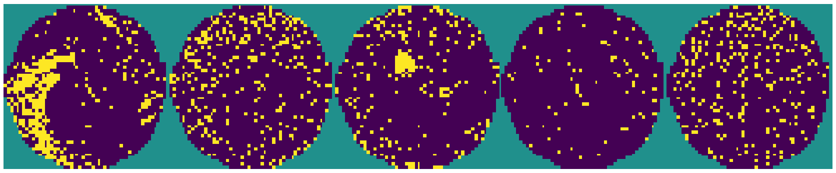


Figure 5.10: Examples of wafer plots not in any cluster

## 5.3 Step2: Primitive Concept Extraction

As shown in Section 4.7, it is easier to train GANs as well as Tensor-based models using primitive concepts rather than supposedly complex ones. The primitive concept identification methodology presented in Section 4.7 is used to divide each cluster into its primitive concepts.

When primitive concept identification is applied to each of the 9 clusters found by our clustering algorithm, only one promitive concept was found in each cluster. This can be thought of as a validation for the clustering result. Nevertheless, the primitive concept identification algorithms is implemented and included in the flow, for robustness reason.

### 5.3.1 Training Wafer Selection

Given a cluster, one can simply take all wafer plots from the cluster as training samples and learn a concept recognizer. This intuitive thought essentially considers the clustering scheme as somewhat ideal. However, as mentioned before, clustering might not be robust. Hence, a methodology to choose the best training wafers is introduced.

The basic idea in this step is based on a so-called *learnability* measure defined in Chapter 4. Given a set of $\boldsymbol{k}$ wafer plots $\boldsymbol{W_1, \ldots, W_k}$, suppose we build a Tensor model $\boldsymbol{T_{1-k}}$ and apply this model back to the $\boldsymbol{k}$ wafer plots. The result by following the equation (5.1) is a set of $\boldsymbol{k}$ $\boldsymbol{\Delta}$'s values, $\boldsymbol{\Delta_1, \ldots, \Delta_k}$. We calculate the learnability of the set $\boldsymbol{S_k = \{W_1, \ldots, W_k\}}$ as:

$$\boldsymbol{LB_{S_k} = \frac{1}{\frac{1}{k}\sum \Delta_i}} \tag{5.3}$$

With the $\boldsymbol{LB}$ calculation, our goal is to search in each cluster for the five wafer plots that give the highest learnability. When the cluster size is small, this search can be exhaustive. Otherwise, the wafers are sorted by their $\mathbf{\Delta}$ value from the model $\boldsymbol{T_{1-k}}$, and the top 5 wafers are chosen. At the end of this search, we end up with a Tensor model $\boldsymbol{T_{best}}$ based on a set of five wafer plots. For example, Fig. 5.11 shows these five plots for cluster 1 and for cluster 9.

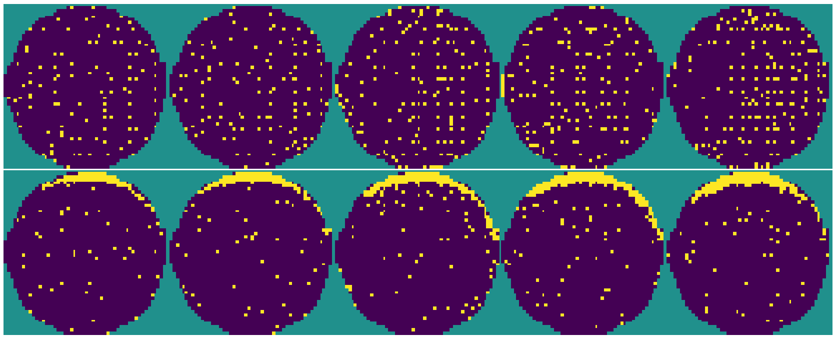


Figure 5.11: The five training wafer plots for concept 1 and for concept 9

## 5.4 Step 3: Tensor-Based Modeling

Now for each of the clusters shown in Fig. 5.9, after Step 2, we end up with 9 Tensor models. Given a wafer plot, to classify the plot we can simply apply these 9 models and see which model gives the smallest $\mathbf{\Delta}$'s value. This approach, however, does not allow a wafer plot to be unrecognized.

To solve this issue, we use a set of baseline Tensor models. There are four of them. Three are similar to the three types of random fails shown in Fig. 5.8 before. The fourth is based on the plot where there is no failing die. Then, we say that in order for a wafer plot to be classified as in one of the nine concepts, its $\mathbf{\Delta}$'s value has to be smaller than all the four $\mathbf{\Delta}$'s values from the four baseline models.

Recall that there are 290 wafer plots after the pre-filtering step. The following table summarizes the result of applying the 9 Tensor models to these 290 plots.

| Concept | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| # Recognized | 13 | 18 | 18 | 9 | 6 | 18 | 6 | 7 | 16 | 114 |
| Cluster size | 9 | 8 | 11 | 16 | 5 | 21 | 6 | 5 | 10 | 91 |

Overall, the total number of classifiable plots is larger than the total number of plots put into a cluster by our clustering scheme. However, for an individual concept the number of classifiable plots can be smaller than its corresponding cluster size. These results show the effects by including Step 2 and Step 3 after the clustering.

## 5.5 Step 4: GANs-Based Modeling

Step 4 also takes the clustering result as a starting point. In addition, Step 4 utilizes the primitive concepts extracted in Step 3 and their corresponding five samples as the training samples. The GANs-based method for building a recognizer is already studied in [2] and explained in Chapter 3. However, there are two fundamental issues for fully automating the training process.

The first is in the in-class sample selection. Training a GANs recognizer requires a set of in-class training samples. In [2], the in-class samples are manually selected. Because GANs training process and its resulting model can be sensitive to this selection, in [2] most of the effort was spent on experimenting with different sets of in-class samples.

Step 2 discussed above provides an automatic way for the in-class sample selection. Each in-class sample set for a concept comprises the 60 samples by taking the five wafer plots from Step 2 and rotating each plot 12 times clockwise.

The second issue is in the availability of out-of-class validation samples. These samples are important for deciding a stopping point in the training as well as for deciding a threshold on the output of the Discriminator neural network. In the past, the stopping point was based on the images generated by the Generator of the GANs, which are usually not very effective causing more manual engineering effort to decide a stopping point and tweaking the threshold.

The clustering result from Step 1 provides a straightforward solution to collect those out-of-class validation samples, i.e. the out-of-class validation samples are those plots in all other clusters. Effectively, we are asking the GANs training to build a model that can separate the five training samples from plots in all other clusters.

For a given sample, the Discriminator outputs a decision value between 0 and 1 as an indicator between out-of-class and in-class. GANs' training is iterative. In each iteration, we calculate the average of Discriminator's outputs for all in-class samples and the average for all out-of-class samples. The difference is treated as a *separability* measure. Then, we say the best model is the one with the highest separability.

The search for the best model is simple. We allow GANs' training to run for a maximum number of iterations and then pick the model with the highest separability. In our implementation this maximum iteration number is set at 5K. This setting is mostly due to the hardware constraint based on the machine we were using. With this number, the training time usually runs for around 2 hours based on our machine equipped with 2 nVidia 980Ti GPUs each with 6Gb VRAM. The implementation of the GANs [2] is based on TensorFlow. The GANs' model size is about 330Mb.

### 5.5.1 Picking the best model

Fig. 5.12 shows how the separability value changes in each training iteration across all 5K iterations, for all 9 primitive concepts. The iteration where the best model is selected is also highlighted with a green bar.

After applying the 9 GANs models to the 290 wafer plots, 104 plots are uniquely recognized by one GANs model. However, 27 plots are recognized by two GANs models.

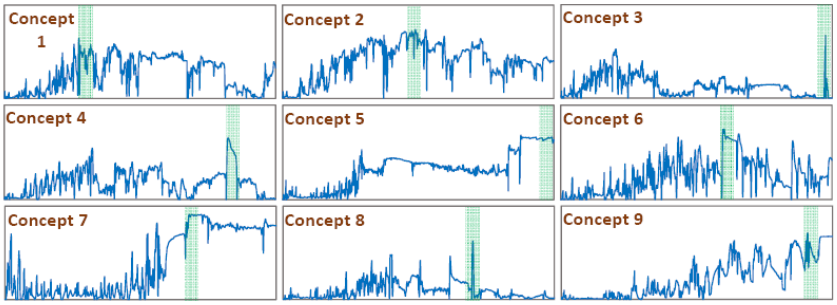


Figure 5.12: Model separability across 5K training iterations - y-axis (the separability measure) is from 0 to 1 and x-axis is from 1 to 5K

In contrast, recall that the sets of wafer plots recognized by the Tensor models are mutually disjoint. The disjoint property is enforced because recognition by a Tensor model is done relatively to other models, not by a fixed threshold.

## 5.6 Step 5: Wafer Plot Classification

Each concept has two recognizers, a Tensor-based model and a GANs-based model. In the actual wafer plot classification, both are used by following the three steps:

1. If a wafer plot is uniquely recognized by a GANs model, then it is recognized in that concept.

2. If a wafer plot is recognized by more than one GANs models, use their respective Tensor models as the tie-breaker. Recall that recognitions by Tensor models are disjoint. However, it is also possible that no Tensor model recognizes the plot. If that happens, the plot is reported as recognized in multiple concepts.

3. For those unrecognized plots after (1) and (2), if it is recognized by a Tensor model, it is considered recognized.

We call this the “Hybrid” rule for concept recognition.

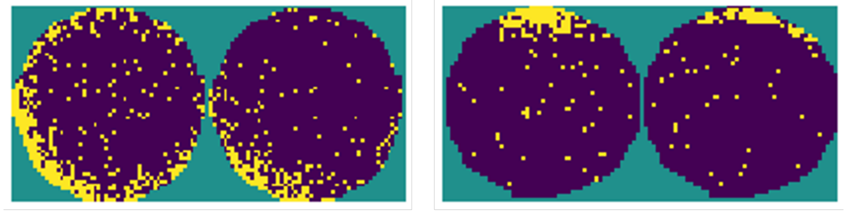
The Hybrid rule is based on the following observations from the past studies [2][36][37]:

1. GANs models are usually more general than the Tensor models and hence, for maximal coverage GANs models are applied first.

2. Because GANs models are more general, a plot can be recognized by multiple models. In this case we need the Tensor models for tie-breaking.

3. GANs models can have adversarial examples [37]. Hence, we need the Tensor models as a backup for recognizing adversarial examples.

For comparison, Fig. 5.13 shows examples recognized in concept 9. As seen, the GANs model can recognize wafer plots more general than those shown for concept 9 in Fig. 5.11. On the other hand, the GANs model can miss wafer plots that look similar to those shown in Fig. 5.11 (i.e. these can be thought of as adversarial examples).



Recognized only by GANs

Recognized only by Tensors

Figure 5.13: Examples uniquely recognized in concept 9

The table below summarizes the result of wafer plot classification for the 290 wafer plots from the first 50 lots. The "Total" is the total number of plots recognized by any recognizer. Notice that for concepts 2, 3, and 8, the Hybrid's numbers are smaller than the Tensor's numbers. This is because a plot can be recognized by a Tensor model for concept $\boldsymbol{i}$ and a GANs model for concept $\boldsymbol{j}$. In this case, it is classified as concept $\boldsymbol{j}$. Also note that even with the Hybrid rule, 6 plots still end up being recognized in two concepts.

Then, the recognizers for the 9 concepts are applied to 7057 wafer plots from the next 287 lots. First, 5903 of them are filtered out by the pre-filtering step. This leaves 1154

| Concept | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| By GANs only | 12 | 16 | 6 | 8 | 61 | 12 | 6 | 5 | 32 | 131* |
| By Tensor only | 13 | 18 | 18 | 9 | 6 | 18 | 6 | 7 | 16 | 114 |
| By Hybrid | 14 | 16 | 12 | 10 | 47 | 18 | 7 | 5 | 33 | 156** |

*keep in mind 27 plots are recognized by 2 GANs models
**6 plots still end up being recognized in 2 concepts

wafer plots to be classified. The table below summarizes the result. Note that for the GANs result, 281 plots are uniquely recognized. 49 plots are recognized by two models and 4 plots are recognized by three models.

| Concept | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| By GANs only | 32 | 47 | 0 | 0 | 179 | 12 | 12 | 3 | 106 | 334* |
| By Tensor only | 41 | 58 | 3 | 0 | 52 | 17 | 28 | 3 | 18 | 220 |
| By Hybrid | 41 | 55 | 1 | 0 | 172 | 19 | 19 | 4 | 93 | 393** |

*keep in mind 53 plots are recognized by $>$ **1** GANs models
**11 plots still end up being recognized in 2 concepts

For the 761 unrecognized plots, Steps 1-4 are applied again to see if there is any new concept that can be extracted and learned. Two additional concepts are found. Fig. 5.14 shows two training examples for each concept.

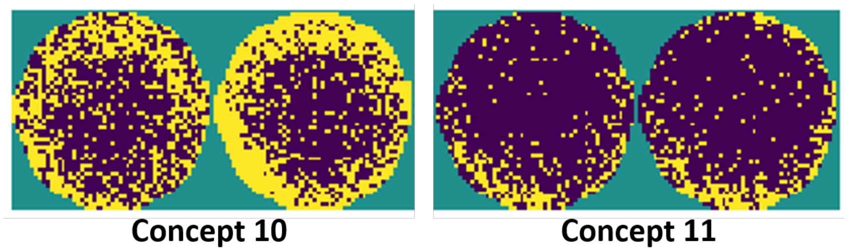


Figure 5.14: Examples from the two newly-extracted concepts

Out of the 761 unrecognized plots, 21 plots are recognized in concept 10, and 48 plots are recognized in concept 11. This results in 692 unrecognized plots.

### 5.6.1 Reporting unrecognized wafer plots

From the first 50 lots, there are $\mathbf{290 - 156 = 134}$ unrecognized plots. With the additional 692 unrecognized plots from the second batch, the total number of unrecognized plots is 826. For reporting, we consider those plots only with a yield loss above a minimum level. This ends up with 712 unrecognized plots to be reported.

For all 826 plots we manually inspect them to see if any can be considered interesting. We found 10 of them which are shown in Fig. 5.15. For reporting, our goal is to report unrecognized plots in a ranking such that those 10 interesting ones can show up, say within the first 100.

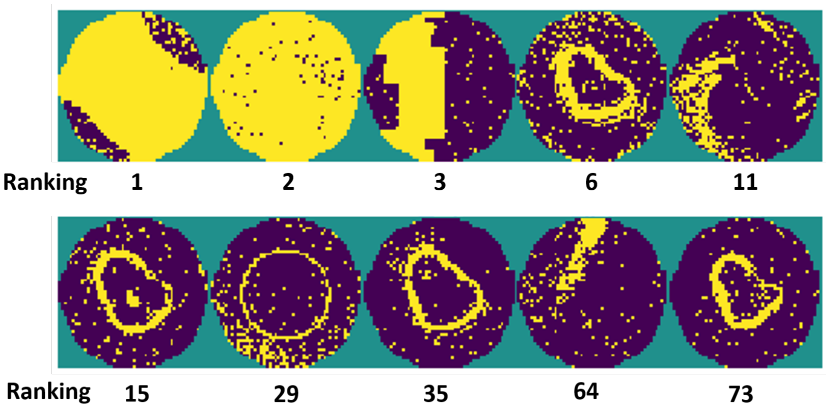


Figure 5.15: Interesting wafer plots and their reported ranking

The simple solution to the report ranking problem is by using a baseline Tensor model, e.g. the low-density random-fail model used before. The $\mathbf{\Delta}$'s values given by this model are used for the ranking, i.e. larger $\mathbf{\Delta}$ value is ranked higher. With this simple method, the ranking for each of the 10 interesting plots are also shown in Fig. 5.15.

In Fig. 5.15, notice that wafer plots ranked 6, 15, 35, and 73 look similar. A concept is not learned for them because there are only four such wafer plots in the data, i.e. it is below our minimum number 5 for learning a model.

## 5.7 Conclusion

In this chapter, we present a software flow that integrates Tensor-based modeling and GANs-based modeling for wafer plot classification. The development focuses on complete automation of the flow and improving the robustness of the classification. The flow is applied to a commercial product line to analyze $\mathbf{1243 + 7057 = 8300}$ wafers. There are 11 concepts automatically learned which together successfully classify 618 wafer plots. The classification results were manually inspected and we found no unacceptable classification. The remaining wafer plots were also manually inspected for their non-interest, except for 10 plots which could be considered as being interesting but were missed by our classification flow. These 10 plots were then reported by the flow as among the top 100 unrecognized plots. This short list facilitates its user to inspect the unrecognized plots.

# Chapter 6

# Deploying the robust automated solution as a surrogate

## 6.1 Introduction

In the previous chapters, we introduced an automated software for concept recognition using the wafer image classification as an application example. Our automated software uses Tensor decomposition for automating concept extraction, and for choosing training wafers as well as helping with the recognition part. However, the main recognition brain in our software is based on Generative Adversarial Networks (GANs). GANs are based on Convolutional Neural Networks, A machine learning algorithm.

The main challenge is that it is very difficult, if not impossible, to guarantee the robustness of a machine learning solution software. For example, the developer might expect that a model built by the solution has an accuracy above 90%, but in a deployment, the software results in a model with only 70% accuracy. This can happen because a solution is developed by using some data as a guide, which might not reveal all the learning software requirements on future unseen data. Hence, a deployment can demand an extensive ramp-up process where the developer is asked to incrementally enhance the capability of the learning software.

The focus of the chapter is on the capability of the GANs-based learning component. Although GANs has become a popular approach for unsupervised learning, training GANs remains to be tricky [4][5]. Much engineering effort can be spent on developing an initial GANs-based learning component but it is difficult for a developer to anticipate all issues possibly encountered from future unseen data.

Note that the concern here is on the capability of the learning component. In a deployment, execution of the learning software automatically builds models online. During the ramp-up, a traditional working model could be that: (1) someone on the deployment site observes that result from a model is "unreasonable" and (2) someone from the deployment site sends some relevant data to the developer for improving the learning software.

Figure 6.1 depicts an alternative deployment setting considered in this chapter. There are two sides: the ML expert side (call it Jay) and the deployment side (call it Nik). The assumption is that Jay does not have access to what happens on Nik's side. Jay's overall objective for the software is to minimize the chance that Nik perceives the software as producing "unreasonable" results.

Of course, this "unreasonable" can be quite subjective. Nevertheless, through the ramp-up process Jay desires to improve the robustness of the software as much as he knows how to.

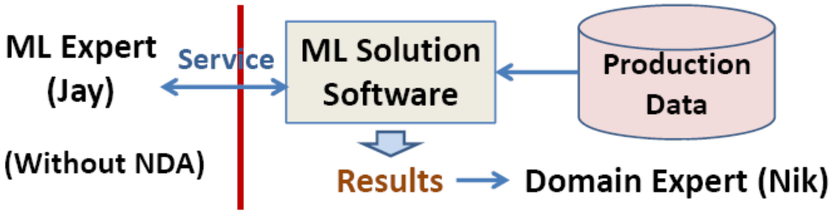


Figure 6.1: Deploying a ML solution as a service

In Figure 6.1, Jay will provide a *service* when a deficiency with the software is detected. A main goal of this work is to enable the software with some *self-checking* capability to detect a deficiency. From Jay's perspective, he desires to hide any deficiency from Nik.

Furthermore, it might be difficult for an NDA agreement to be achieved between the two sides. As a result, when Jay is called for a service, it might be the case that he can only receive very limited information to help improve the software. In this work, we also take this into account and study what minimal information is needed without revealing the confidential information, such as yield.

Overall, Figure 6.1 depicts a collaboration view where the ML software acts as a *surrogate* for Jay. The view enables Jay to stay outside Nik's entity and still be able to provide his service to the entity. Also, if this view can be realized, Jay's service is not limited to just one entity.

There are two novel aspects in this chapter: (1) For each GANs-based recognizer, we introduce a corresponding *check*. Failing this check will result in calling Jay for a service. (2) When Jay is called for a service, we introduce a *transformation* method to hide the original wafer images from Jay. Further, only very few transformed samples are provided so that Jay has no way to infer the yield.

In this work, we utilize the production data from a high-reliability product line for the experiments to articulate the key ideas in our software design. The data comprises 8300 wafers. We first use the earliest 50 lots to run the software. Then, we run it on the remaining 287 lots where a service call is triggered. Then, we will discuss how Jay improves the software. After the software is improved and updated, we run it again on the 287 lots, and continue to run it on a 2nd high-reliability product line. Results from all these runs are used in the presentation of our key ideas for deploying the ML software solution as a service.

## 6.2 The Wafer Classification Software

Figure 6.2 illustrates the workflow employed by the ML software Jay desires to deploy. The first step is going through a Filtering Random box where the task is to filter out wafer images showing sparse random fails. The purpose is to facilitate execution of the subsequent steps. Implementation detail of this box is discussed in [6].

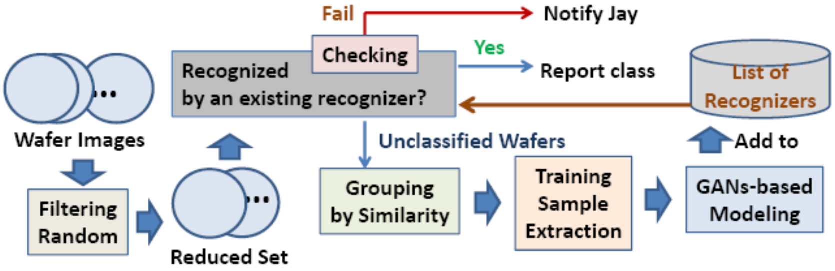


Figure 6.2: Workflow employed by the ML solution software

After the filtering, each remaining wafer goes through a recognition box. Initially, if there is no recognizer available, this step is skipped and all wafers are unclassified. If a wafer image is recognized by a recognizer, then the wafer image is reported in the corresponding class.

The recognition is checked with a Checking box, which is a focus of this chapter. Failing this Checking box results in a notification sent to Jay.

After the recognition box, unclassified wafers are evaluated to see if a new recognizer can be learned. The learning goes through three boxes: (1) Grouping by Similarity, where multiple *primitive concepts* are identified; (2) Training Sample Extraction, where five training samples are extracted for each concept; and (3) GANs-Based Modeling, where a recognizer is learned for each identified concept. For the purpose of making this chapter self-contained, the following provides more explanation for these three boxes first. Then, Section 6.2.4 discusses the application results. More details on the three steps can be found in previous chapters.

### 6.2.1 Grouping by Similarity

This step implements a clustering scheme. Our choice of the clustering algorithm is the Hierarchical DBSCAN [51] (HDBSCAN) with a minimum cluster size specified at 5. This means that if the number of similar wafer images is less than 5, it is deemed insufficient to learn a new class.

The most subtle part in the cluster scheme is the definition of the distance function $\boldsymbol{Dist()}$ that measures a distance between two images. An image is encoded as a matrix where a $\mathbf{+1}$ corresponds to passing, $\mathbf{-1}$ corresponds to failing, and $\mathbf{0}$ means no die on the location.

Given a wafer matrix $\boldsymbol{W_A}$, we first build a Tensor model $\boldsymbol{T_A}$ using the method proposed in Chapter 4: When $\boldsymbol{T_A}$ is applied to another wafer matrix $\boldsymbol{W_B}$, the result is a *progected core matrix* $\boldsymbol{M_{A\rightarrow B}}$. If $\boldsymbol{W_B}$ is similar to $\boldsymbol{W_A}$, $\boldsymbol{M_{A\rightarrow B}}$ would have relatively large absolute values in the diagonal entries and close-to-zero values in non-diagonal entries. Otherwise, $\boldsymbol{M_{A\rightarrow B}}$ would have large values in non-diagonal entries. Hence, the method in Chapter 4 utilize this *diagonality* property to define a *directional* distance measure $\boldsymbol{\Delta}$ from $\boldsymbol{W_A}$ to $\boldsymbol{W_B}$ as:

$$\boldsymbol{\Delta_{A\rightarrow B}} = \frac{\sum_{\forall i \neq j} \boldsymbol{M_{A\rightarrow B}[i,j]^2}}{\sum_{\forall i} \boldsymbol{M_{A\rightarrow B}[i,i]^2}} \tag{6.1}$$

The value $\boldsymbol{\Delta_{A\rightarrow B}}$ is based on $\boldsymbol{W_A}$ and is not directly comparable to another value $\boldsymbol{\Delta_{C\rightarrow B}}$ based on another $\boldsymbol{W_C}$. Our solution to the problem is to use a baseline wafer $\boldsymbol{W_{base}}$ such as a wafer without a failing die. This baseline image serves as a reference point for all wafer plots. Then, $\boldsymbol{\Delta_{A\rightarrow B}}$ is "normalized" as a new directional distance:

$$\boldsymbol{\delta_{A\rightarrow B}} = \frac{\boldsymbol{\Delta_{A\rightarrow B}}}{\boldsymbol{\Delta_{A\rightarrow B}} + \boldsymbol{\Delta_{base\rightarrow B}}} \tag{6.2}$$

The distance between $\boldsymbol{W_A}$ and $\boldsymbol{W_B}$ is then calculated as the average of $\boldsymbol{\delta_{A\to B}}$ and $\boldsymbol{\delta_{B\to A}}$. Figure 6.3 uses a wafer A and its distances to three wafers B,C,D as an example to illustrate the distance calculation.

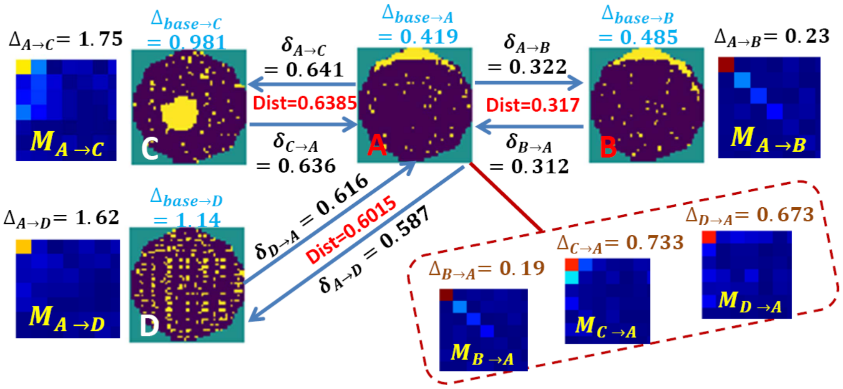

$\Delta_{A\to C}= 1.75$
$\Delta_{base\to C} = 0.981$
$\delta_{A\to C} = 0.641$
Dist=0.6385
$\delta_{C\to A} = 0.636$
$\Delta_{base\to A} = 0.419$
$\delta_{A\to B} = 0.322$
Dist=0.317
$\delta_{B\to A} = 0.312$
$\Delta_{base\to B} = 0.485$
$\Delta_{A\to B}= 0.23$
$M_{A\to C}$
C
A
B
$M_{A\to B}$
$\Delta_{A\to D}= 1.62$
$\Delta_{base\to D} = 1.14$
$M_{A\to D}$
D
$\delta_{D\to A} = 0.616$
Dist=0.6015
$\delta_{A\to D} = 0.587$
$\Delta_{B\to A}= 0.19$
$\Delta_{C\to A}= 0.733$
$\Delta_{D\to A}= 0.673$
$M_{B\to A}$
$M_{C\to A}$
$M_{D\to A}$

Figure 6.3: Illustration of distance calculation used in the clustering scheme (Each $\boldsymbol{M}$ matrix is actually bigger and only the top left portion is shown)

Notice that wafer A and wafer B are similar. The matrix $\boldsymbol{M_{A\to B}}$ has large values on the diagonal entries and almost zero on the non-diagonal entries, thus $\boldsymbol{\Delta_{A\to B}}$ is 0.23. The matrix $\boldsymbol{M_{B\to A}}$ on the bottom shows a similar property with $\boldsymbol{\Delta_{B\to A} = 0.19}$. On the other hand, $\boldsymbol{M_{A\to C}}$ and $\boldsymbol{M_{A\to D}}$ do not have the property because they are not similar to wafer A. Their $\boldsymbol{\Delta}$ values are larger, i.e. 1.75 and 1.62, respectively. The resulting distances are in red. We see that the distance between A and B is 0.317, smaller than the other two distances 0.6385 and 0.6015.

### 6.2.2 GANs-based Modeling

Output from the Group by Similarity box is a set of clusters where each cluster contains at least 5 wafers. Next, in the Training Sample Extraction, for each cluster the best 5 samples are identified for learning a model. This evaluation is based on the *learnability* measure proposed in Chapter 4 where the best 5 samples have the highest learnability value.

For each cluster, the 5 training wafers are then given to the GANs-based Modeling box. In GANs [3], there are two neural networks: the generator **G** and the discriminator **D**. The GANs modeling follows the same approach and training strategy presented in [2] and explained in Chapter 3.

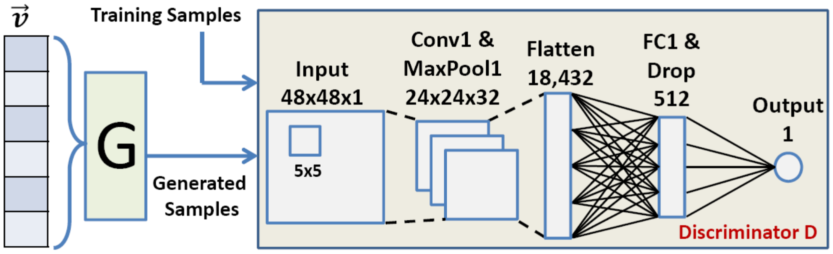


Figure 6.4: Discriminator neural network used in GANs modeling

The generator's input is a randomly generated 100-dimension latent vector $\vec{v}$. The training is iterative. In each iteration, the generator tries to improve its neural network model for generating samples closer to the training samples. The discriminator tries to improve its model for separating the generated samples from the training samples.

In this version of the software, the discriminator is dramatically simplified as shown in Figure 6.4 (as compared to the complex architecture employed in Chapter 3). There are several considerations behind this simplification.

As mentioned before, training GANs can be tricky [4] [5]. For instance, balancing the convergence between **G** and **D** can be quite a challenge, and often **D** can win easily. In general, **D** handles a learning task that is easier than **G**. Therefore, if we design both **D** and **G** with the same capacity, it is likely that **D** would win in the early iterations of the training, making **G** harder to converge. In the past, we already observed that ensuring convergence of both networks could be challenging [2].

The simplification makes the capacity of **D** smaller than the capacity of **G**. This idea is inspired by the regularization ideas proposed in [55][56]. In our context, we desire to reduce the capacity of **D** to make the training more robust while still ensuring sufficient capacity to handle the complexity involved in the recognition task.

The reader might wonder, if GANs training is so tricky, why didn't we choose to solve the problem as a supervised learning problem (i.e. multi-class classification) using a single convolution neural network (CNN). The simple answer is, that would require many more samples in each class to train a good classifier. In practice, we simply do not have enough samples to do that for every class. The main benefit of using GANs is that it needs very few samples (in our case only 5 samples) to obtain a high-quality recognizer [2].

### 6.2.3 Separability measure in GANs training

GANs, however, are mostly for obtaining a good generator to be used in an application. Our usage is different - we are interested in using the discriminator as a recognizer. This objective makes our GANs training even trickier, for instance, training with more iterations does not imply that a better discriminator model can be found.

To evaluate the discriminator for our purpose, we need a model evaluation scheme independent of the GANs training. In our case, the discriminator model in each training iteration is evaluated based on a *separability* measure proposed in [38] and explained in Chapter 5. The training samples are used as the in-class sample set $\boldsymbol{S_{in}}$. To calculate separability, we need an out-of-class sample set $\boldsymbol{S_{out}}$. In our implementation, the out-of-class samples are wafers in other clusters based on the clustering result from the Grouping by Similarity box.

Separability is a value between 0 and 1 [38], telling how well a **D** model separates the samples between the two sets. A larger value means more separation.

There can be two types of separability, *average separability* as used in Chapter 5 and *strict separability* that is used in our implementation. Strict separability follows the same calculation method as average separability, except that it uses the worst-case separability value instead of the average (between in- and out-of-class samples).

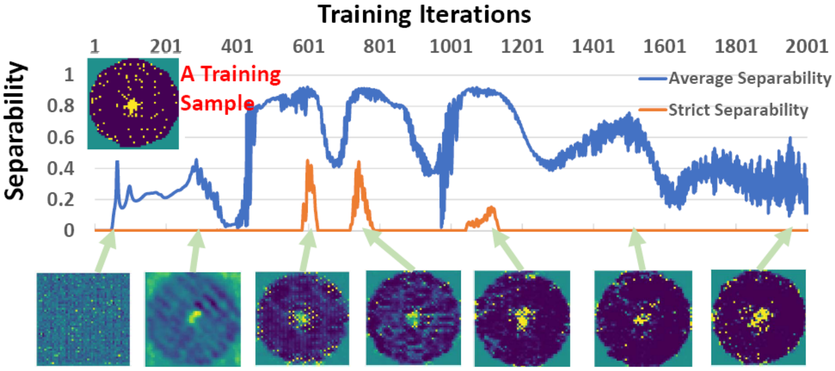


Figure 6.5: Illustration of a GANs training process over 2K iterations

To illustrate the training process, Figure 6.5 shows the results from 2K iterations of training a concept recognizer (took about 5 minutes to run on a machine with 2 nVidia 980Ti GPUs each with 6Gb VRAM). The training samples started with 5 wafer images showing clusters of fails on the center (one training sample is shown on the top left). Then, each is rotated 12 times to obtain in total 60 samples.

Both average and strict separability values are plotted. It can be seen that average separability is always much higher. Most of the strict separability values are zero because as long as the **D** model cannot separate any pair of the in-class and out-of-class samples, this value is zero.

On the bottom, the samples generated from the generator **G** are shown. It is interesting to see that with more iterations, the generated samples become closer to the training samples. In contrast, we see that while the average separability is generally higher in the 400-1200 range and lower after that, the value can fluctuate significantly. The strict separability has three "bumps" and otherwise is zero everywhere. Keep in mind that samples in $\boldsymbol{S_{out}}$ are used only in calculating the separability, and not used in training **D**. Hence, **D** is not optimized for separability at all. The generated samples, on the other hand, have nothing to do with the out-of-class samples in use. This is why more iteration does not mean a better **D** in our context, and we need the separability measure to select the **D** model as our recognizer.

Notice in the figure that the highest separability happens well before the generator reaches the point of generating quality images close to the training samples. This is because once the generator starts to learn well, the discriminator (in order to separate the generated samples from the training samples) might start over-fitting the training samples. Once this over-fitting takes place, the discriminator model essentially loses its ability to separate the training samples from those unseen samples in the out-of-class set $\boldsymbol{S_{out}}$.

### 6.2.4 Result from the first 50 lots

At first, the software is run on the first 50 lots. Figure 6.6 illustrates the result for each box. In this run, 9 recognizers are learned. Table 6.1 then summarizes the recognition results by these recognizers.

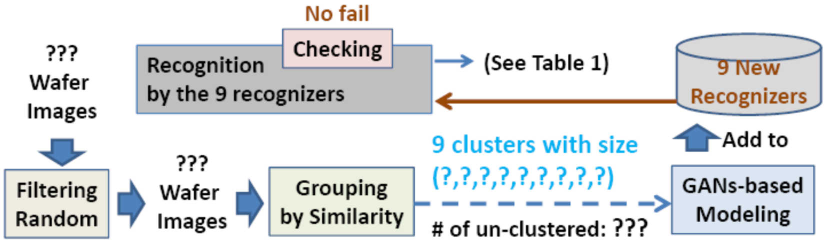


Figure 6.6: Workflow illustration for the first 50 lots

Notice in Table 6.1 the number of wafers recognized in each concept is generally larger than the corresponding cluster size shown in Figure 6.6, except for Concept 5. As explained in Chapter 5, clustering is not very robust and hence, can only be used as a starting point. Clustering result can be sensitive to the distance calculation and the hyper-parameter setting in the clustering algorithm.

Table 6.1: Result From The first 50 lots

| **Concept** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| **Recognized Wafers**** | 16 | 21 | 21 | 14 | 34 | 10 | 12 | 39 | 31 | 178* |
| **False Positives†** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 14 | 11 | 26 |
| **Checking** | √ | √ | √ | √ | √ | √ | √ | √ | √ | - |

√: model passing the Checking box
*Collectively 178 wafers are recognized in total
because a few wafers are recognized in two or more concepts
**After the recognition, there remains 180 unclassified wafers
† known only after manual inspection

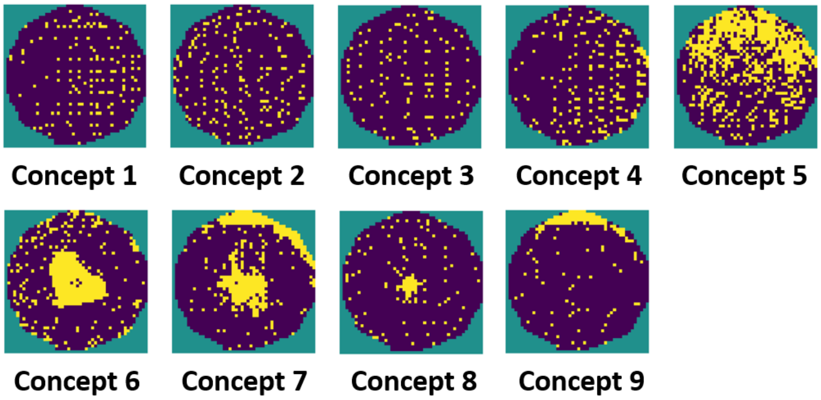


Figure 6.7: Typical example wafer image from each concept

Figure 6.7 provides a typical example from each concept to illustrate what they look like. Notice that Concepts 1-4 all look similar, i.e. containing some "Grid" patterns. They are separated as four by the "Grouping by Similarity" box. Recall that this box implements clustering using distances based on the Tensor method. As discussed in Chapter 5, a Tensor model is more specific and hence, ends up separating Concepts 1-4 as different concepts. Note that Concepts 1-9 are the same ones introduced in Chapter 5, but with their ordering changed.

For each concept we manually inspect each recognized wafer image and identify those possibly false positives. The information is also included in the table. In practice, Jay would not know this information. Nik might not know either if he does not do the specific inspection. Note that the impact of these false positives would depend on how the classification results are utilized in an analytic workflow.

We see that Concept 8 and Concept 9 have significantly more false positives than others. Some false positives from Concepts 5, 8, and 9 are shown in Figure 6.8. For a learning model, achieving 100% accuracy is difficult, if not impossible. Hence, understandably a recognizer can have false positives. However, too many false positives can trigger Nik to perceive the results unreasonable when he or his software script tries to utilize them.

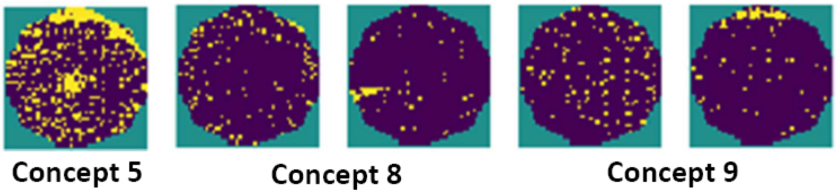


Figure 6.8: False positive examples

In that regard, the numbers of false positives for Concepts 8 and 9 seem high. Figure 6.9 shows 11 false positives from Concept 8, and 5 false positives from Concept 9. These false positives are all recognized by two or more recognizers (recognizer for Concept $\boldsymbol{i}$ is denoted as C$\boldsymbol{i}$).

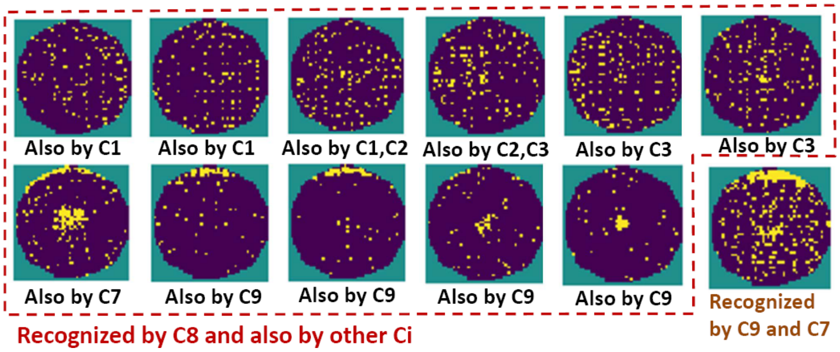


Figure 6.9: Other false positives for Concepts 8 and 9

Observe that 4 wafers are recognized by both C8 and C9. There is one wafer recognized by C1, C2 (and C8) and one wafer by C2, C3 (and C8). Note that these two wafers are not counted as false positives (in C1-C3) because we could not say for certain which one grid class they should belong to and not another.

These results show that not only C8 and C9 have unusually large numbers of false positives, a big part of it (12 in total) can be attributed to wafers recognized in other concepts as well. This shows that there is some potential deficiency in the C8 and C9 recognizers.

For each concept, the recognition goes through the Checking box (discussed in detail later) as shown in Figure 6.6. Table 6.1 shows no recognition failing this Checking and hence, the potential deficiencies in recognizers C8 and C9 are not known to Jay. Nik might not know either if the result is consumed by his software script and not by him.

### 6.2.5 Result from the remaining 287 lots

The software continues its run on the remaining 287 lots. Figure 6.10 illustrates the results through the workflow. The difference between Figure 6.6 and Figure 6.10 is that in Figure 6.10 now we have 9 recognizers already in the Recognition box to begin with.

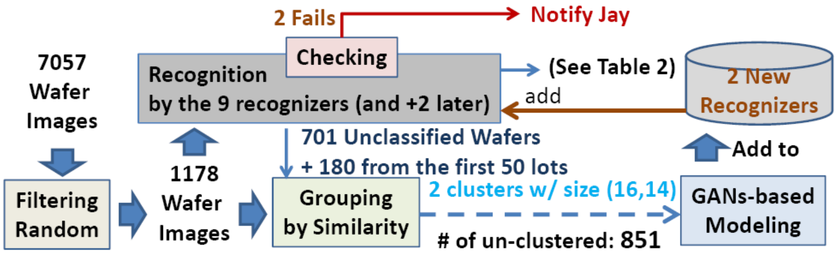


Figure 6.10: Workflow illustration for the remaining 287 lots

Table 6.2 summarizes the results from the Recognition box. This time, recognition for Concept 8 and Concept 9 fails the Checking box. As a result, Jay is called for a service to improve the wafer classification software.

Table 6.2: Result From The remaining 287 lots

| **Concept** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Recognized Wafers*** | 14 | 46 | 29 | 25 | 76 | 70 | 13 | 139 | 118 | 19 | 113 |
| **False Positives**** | 0 | 1 | 0 | 2 | 2 | 0 | 2 | 29 | 39 | 0 | 1 |
| **Checking** | √ | √ | √ | √ | √ | √ | √ | × | × | √ | √ |

×: model failing the Checking box
*Collectively, in total 477 wafers are recognized by C1-C9, which means after C1-C9 recognition, there remains 701 unclassified wafers
**53 of them due to multiple recognition, and among them
35 multiple-recognized wafers are due to C8 and C9

In Table 6.2, Concepts 2, 4, 7, and 11 have false positives, whereas no false positives for these Concepts were found in Table 6.1. Figure 6.11 shows what these new types of false positives look like.

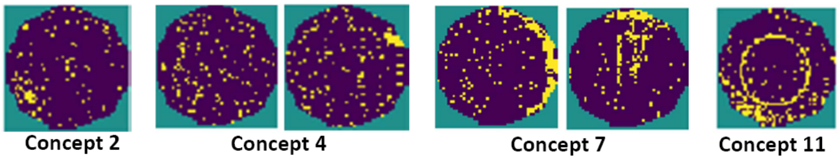


Figure 6.11: Additional false positive examples

Notice that after the recognition, two more concepts are identified with two new recognizers learned. Figure 6.12 shows what they look like.

## 6.3 Implementation Of The Checking Box

Following the Tensor modeling method proposed in [36] and explained in Chapter 4, for each concept class, the five samples used to train a GANs model are also used to build a corresponding Tensor model. As a result, for each Concept $\boldsymbol{i}$, the software maintains a GANs recognizer $\boldsymbol{C_i}$ and a corresponding Tensor model $\boldsymbol{T_{Ci}}$.

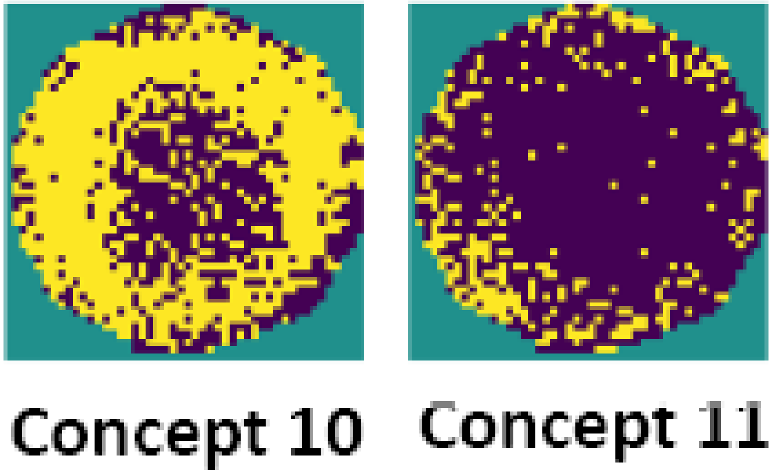


Figure 6.12: Two new Concepts learned

Let $\boldsymbol{S_{C_i}}$ be the set of wafers recognized by $\boldsymbol{C_i}$, and let $\boldsymbol{S_{T_{Ci}}}$ be the set of wafers classified by $\boldsymbol{T_{Ci}}$ as in-class to Concept $\boldsymbol{i}$. The general idea for the Checking box is that it implements a *containment check* to see if $\boldsymbol{S_{T_{Ci}}} \subseteq \boldsymbol{S_{C_i}}$. This containment check is considered failing if more than one wafer that is in $\boldsymbol{S_{T_{Ci}}}$ but not recognized as in $\boldsymbol{S_{C_i}}$. Note that with this check, one escape is acceptable because we cannot guarantee that the containment property is strictly true even for a very good GANs recognizer.

The reasoning behind the containment check is based on the observation that generally a Tensor model is more specific than its corresponding GANs model [38]. However, this property has not been mathematically proven yet. Such a proof would reveal how exactly a neural network model recognizes a pattern, which is still considered more or less a mystery in the machine learning literature.

### 6.3.1 Binning based on Tensor models

Classification by the Tensor models is done based on the $\boldsymbol{\Delta}$ value calculation mentioned in Section 6.2.1. Suppose 9 Tensor models $\boldsymbol{T_{C1}, \ldots, T_{C9}}$ are given for the 9 concepts, each built with the same 5 training wafers used to train the GANs model. The

Tensor-based classification is achieved through a *binning* process as described below.

First, a set of *base* Tensor models are constructed. In the implementation, there are four baseline models representing wafers with no fail, with low-density random fails, with medium-density random fails, and with high-density random fails. Let them be denoted as $\boldsymbol{T_{base1}} \ldots \boldsymbol{T_{base4}}$.

Given a wafer image, thirteen $\boldsymbol{\Delta}$ values are calculated, nine values from the $\boldsymbol{T_{Ci}}$ models and four values from the base models. A wafer $\boldsymbol{W}$ is put into the concept bin $\#\boldsymbol{i}$ only if its $\boldsymbol{\Delta}$ value from $\boldsymbol{T_{Ci}}$ (denoted as $\boldsymbol{\Delta_{Ci \to W}}$) is the smallest. This binning would classify $\boldsymbol{W}$ into either one of the 9 concept bins, or none (if $\boldsymbol{\Delta}$ value from one of the baseline concepts is the smallest).

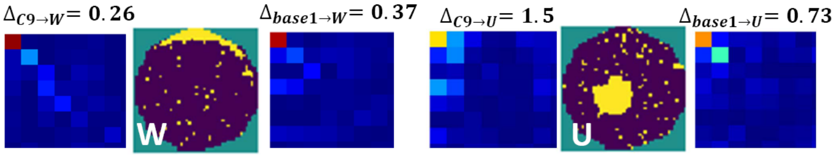


Figure 6.13: Illustration of binning process by Tensor models (Each matrix is actually bigger and only the top left portion is shown)

Figure 6.13 illustrates the $\boldsymbol{\Delta}$ value calculation for two wafers $\boldsymbol{W}$ and $\boldsymbol{U}$ based on $\boldsymbol{T_{C9}}$ for Concept 9 and $\boldsymbol{T_{base1}}$ for the no-failing model. As seen, the $\boldsymbol{W}$ wafer, which does show an edge failing pattern, has $\boldsymbol{\Delta_{C9 \to W} = 0.26}$ which is smaller than $\boldsymbol{\Delta_{base1 \to W} = 0.37}$. On the other hand, the $\boldsymbol{U}$ wafer, which does not show an edge failing pattern, has $\boldsymbol{\Delta_{C9 \to U} = 1.5}$ which is not only much larger, but also larger than $\boldsymbol{\Delta_{base1 \to W} = 0.73}$. As a result, the $\boldsymbol{U}$ wafer would not be in the concept bin #9. The $\boldsymbol{W}$ wafer would be in concept bin #9 because $\boldsymbol{\Delta_{C9 \to W} = 0.26}$ is the smallest among all the thirteen $\boldsymbol{\Delta}$ values (not shown in the figure).

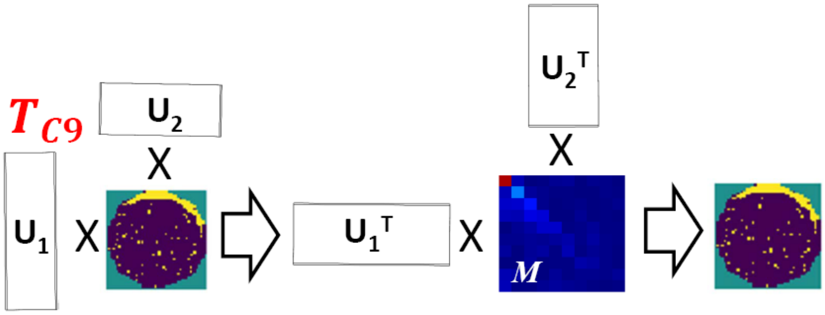


Figure 6.14: Illustration of the restoration effect of a matrix $\boldsymbol{M}$

### 6.3.2 The restoration effect of a core matrix $\boldsymbol{M}$

Effectively, a Tensor model $\boldsymbol{T_{Ci}}$ comprises two orthogonal projection matrices $\boldsymbol{U_1}$ and $\boldsymbol{U_2}$. Given a wafer image matrix $\boldsymbol{W}$, the projected core matrix $\boldsymbol{M}$ (mentioned in Section 6.2.1) for $\boldsymbol{W}$ is obtained by the calculation $\boldsymbol{U_1} \times \boldsymbol{W} \times \boldsymbol{U_2}$. This process is illustrated in Figure 6.14. Also notice that the calculation $\boldsymbol{U_1^T} \times \boldsymbol{W} \times \boldsymbol{U_2^T}$ restores the wafer matrix $\boldsymbol{W}$.

What if we take a core matrix $\boldsymbol{M}$ and make all non-diagonal entries zero before we perform the restoration? Call this the *1st restoration*. Figure 6.15 shows the result by taking the Tensor model $\boldsymbol{T_{C9}}$ with this restoration. For the leftmost wafer (with edge failing), observe that the effect is like removing other random fails not on the edge pattern. The shape of the edge pattern is kept. Recall that $\boldsymbol{T_{C9}}$ is for capturing the concept with an edge-failing pattern.

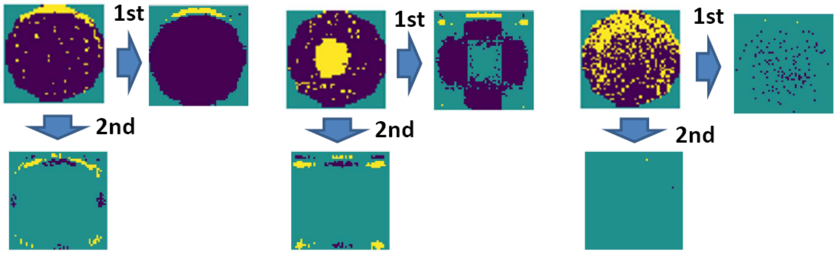


Figure 6.15: Illustration of the 1st and 2nd modeling effects

For the middle wafer, the restored wafer looks differently. There is an edge without the curve shape. The yellow failing dots on the center are all gone. For the rightmost wafer, the failing pattern is basically gone.

From Figure 6.13 above (and Figure 6.3 earlier), observe that for all core matrices, the first entry [1,1] always has the largest value. Recall the $\boldsymbol{\Delta}$ calculation for model $\boldsymbol{T_{Ci}}$ on wafer $\boldsymbol{W}$ is $\boldsymbol{\Delta_{Ci\to W}} = \frac{\sum_{\forall i\neq j} \boldsymbol{M_{Ci\to W}[i,j]^2}}{\sum_{\forall i} \boldsymbol{M_{Ci\to W}[i,i]^2}}$.

The large value in entry [1,1] effectively makes the $\boldsymbol{\Delta}$ value small. Recall the property we are looking for is that all non-diagonal entries should have close-to-zero values. Hence, we can try a modified $\boldsymbol{\Delta}$ calculation by ignoring the entry [1,1]. Call this the $\boldsymbol{\Delta'}$ calculation:

$$\boldsymbol{\Delta'_{Ci\to W}} = \frac{\sum_{\forall i\neq j} \boldsymbol{M_{Ci\to W}[i,j]^2}}{\sum_{\forall i, i\neq 1} \boldsymbol{M_{Ci\to W}[i,i]^2}}.$$

What is the effect of ignoring the entry [1,1] in the restoration process? Call this the *2nd restoration* and Figure 6.15 shows the effect. For the leftmost wafer, the shape of yellow edge is maintained. However, for all wafers the wafer background as well as the random failing yellow dots are gone. These examples show that the 2nd restoration process results in three samples that are still differentiable.

### 6.3.3 Taking intersection as the containment set

The $\boldsymbol{\Delta'}$ calculation provides an alternative to implement the binning process described in Section 6.3.1. Let $\boldsymbol{S}$ and $\boldsymbol{S'}$ be the set of wafers in a concept bin based on $\boldsymbol{\Delta}$ and $\boldsymbol{\Delta'}$, respectively. The set $\boldsymbol{S} \cap \boldsymbol{S'}$ is used by the containment check.

Table 6.3 shows the sizes of each concept bin after the two independent binning processes for wafers from the first 50 lots. Similarly, Table 6.4 shows those sizes for wafers from the remaining 287 lots.

Table 6.3: Containment Set Generation - First 50 lots

| **Concept** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **1st Binning, $\boldsymbol{S}$** | 21 | 19 | 8 | 12 | 33 | 5 | 9 | 13 | 25 |
| **2nd Binning, $\boldsymbol{S'}$** | 17 | 19 | 8 | 12 | 11 | 5 | 10 | 9 | 19 |
| **Intersection Set $\boldsymbol{S \cap S'}$** | 17 | 17 | 8 | 12 | 9 | 5 | 9 | 9 | 15 |

Table 6.4: Containment Set Generation - Remaining 287 lots

| **Concept** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1st Binning, $\boldsymbol{S}$** | 22 | 69 | 4 | 0 | 34 | 22 | 5 | 97 | 25 | 23 | 43 |
| **2nd Binning, $\boldsymbol{S'}$** | 18 | 58 | 4 | 1 | 15 | 41 | 3 | 48 | 22 | 22 | 37 |
| **Intersection Set $\boldsymbol{S \cap S'}$** | 17 | 27 | 3 | 0 | 14 | 21 | 2 | 47 | 10 | 12 | 20 |

Taking the intersection improves the robustness of the containment check. In other words, we desire to minimize the chance for including a wafer in a concept bin when it should not be. For example, Figure 6.16 shows examples binned into a Concept $\boldsymbol{i}$ by the 1st binning but not by the 2nd binning, i.e. for those cases where there is a difference between the 1st binning number and the intersection number. Comparing to Figure 6.7 before, we see that those wafers are really marginal. The effect of taking the intersection removes those marginal wafers and hence, makes the containment check more robust.

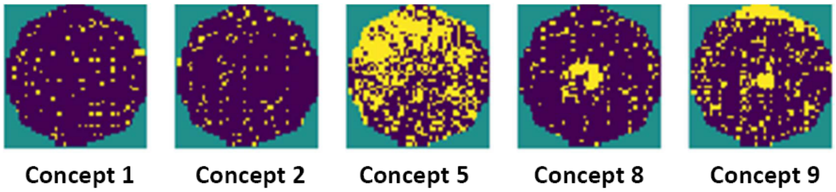


Figure 6.16: Examples in 1st binning set $\boldsymbol{S}$ but not in 2nd binning set $\boldsymbol{S'}$

### 6.3.4 Containment check result

Table 6.2 earlier shows that the containment check fails for Concept 8 and Concept 9. Table 6.5 provides more detail after seeing the result from Table 6.4. The GANs recognizers C8 and C9 fail because they have more than one escapes from the containment check set.

Table 6.5: Containment Check In The remaining 287 lots

| **Concept** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Checking Set Size** | 17 | 27 | 3 | 0 | 14 | 21 | 2 | 47 | 10 | 12 | 20 |
| **# of Escapes** | 1 | 0 | 0 | - | 0 | 1 | 0 | 7 | 6 | 0 | 0 |
| **Checking Result** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | × | ✓ | ✓ |

×: model failing the containment check

Figure 6.17 shows the 7 escapes from C8 during the containment check. Observe that none cannot be said that it should not belong to Concept 8. Hence, there is some deficiency when the software builds the recognizer C8.

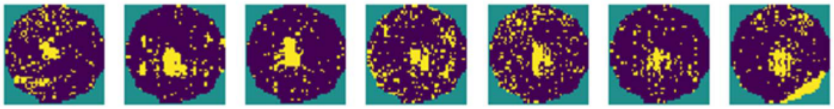


Figure 6.17: The 7 escapes by the GANs recognizer C8 for Concept 8

## 6.4 Service By The ML Expert

Jay is now called for a service. However, Jay knows nothing about the application results described above. From the company side, Nik does not want to reveal too much information about the production, especially the yield. Nevertheless, when the software calls Jay for a service, it still needs to send some data to help Jay improve the GANs-based learning component.

Our approach to resolve this conflict is by sending Jay a few samples that creates a learning problem harder than the current learning problem for the particular concept where its recognizer fails the containment check.

### 6.4.1 Samples sent to Jay

Recall the 1st restoration discussed in Section 6.3.2 where Figure 6.15 illustrates the method with some examples. We use this transformation to create the in-class samples for Jay by transforming each of the five training wafer images. For example, Figure 6.18 shows the five training wafers for Concept 8 and the resulting in-class images. In this transformation, we use the projection matrices of the Tensor model $\boldsymbol{T_{C8}}$ built based on the five training wafers.

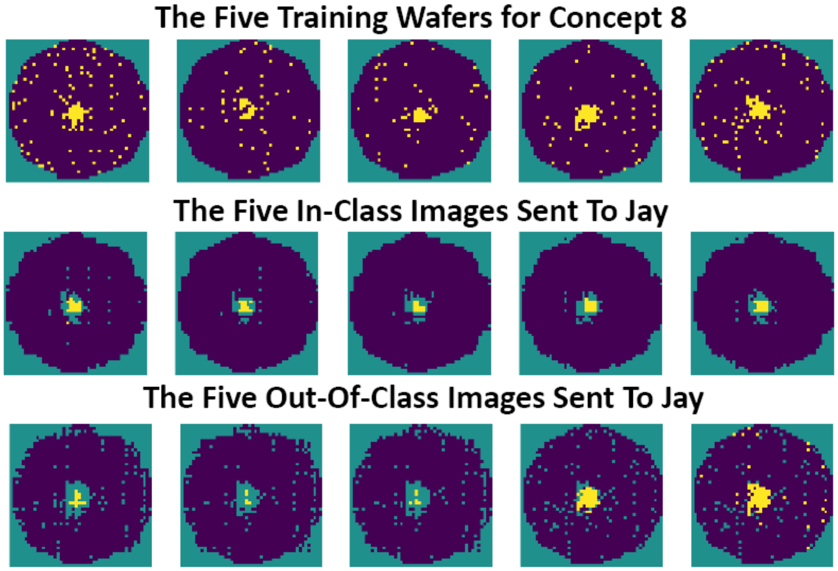

The Five Training Wafers for Concept 8
The Five In-Class Images Sent To Jay
The Five Out-Of-Class Images Sent To Jay

Figure 6.18: The 10 images sent to Jay for Concept 8

To create out-of-class samples for Jay, we use a different Tensor model $\boldsymbol{U_{C8}}$. First, the last two training wafer images shown in Figure 6.18 are stacked to create a new wafer image, i.e. combining all the yellow dots. Let $\boldsymbol{U_{C8}}$ be the Tensor model for this stacked wafer image. Using $\boldsymbol{U_{C8}}$'s projection matrices, we again perform the 1st restoration on each of the five training wafer images and then obtain the five out-of-class samples as shown on the 3rd row in Figure 6.18. Similarly, the lower two rows in Figure 6.19 show the 10 images sent to Jay for Concept 9.

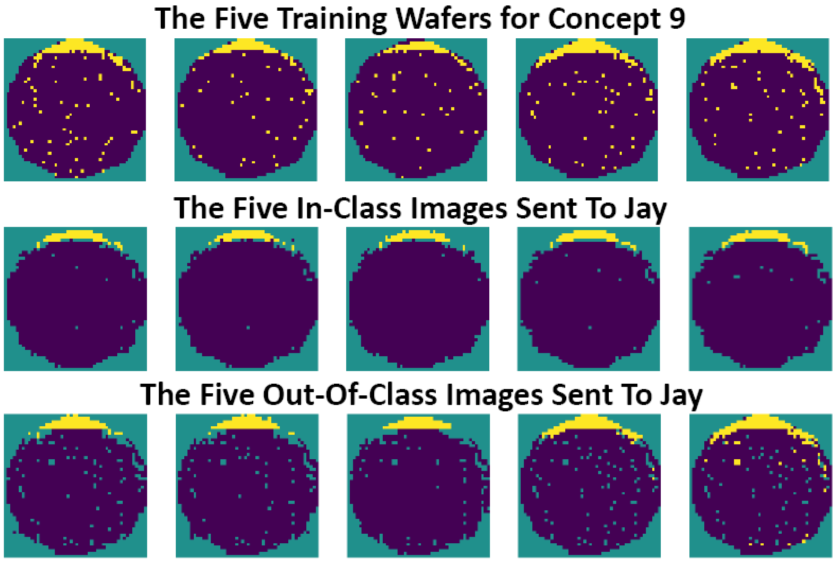

The Five Training Wafers for Concept 9
The Five In-Class Images Sent To Jay
The Five Out-Of-Class Images Sent To Jay

Figure 6.19: The 10 images sent to Jay for Concept 9

Note that on all images sent to Jay, almost all sparse random fails are removed by the transformation (as discussed in Section 6.3.2). Also, only five wafers are used to create the 10 transformed images. Also, notice that the out-of-class images differ only slightly from the in-class images. This effectively creates a *harder* learning problem for Jay. In contrast, on the deployment side the out-of-class images are those from other classes deemed by the clustering (discussed in Section 6.2.2) to be dissimilar to the in-class images.

For each image Jay receives, it is then rotated 12 times to obtain 60 in-class training images and 60 out-of-class validation images (same as that discussed in Section 6.2.3). Jay then relies on these transformed images to improve the GANs training component.

### 6.4.2 Improvements to the GANs modeling box

First, Jay checked the performance of the GANs modeling box on the given images for Concept 8. Figure 6.20 shows the strict separability value at each iteration of the GANs training based on the given images. These are examples from multiple training runs. Because the training process is statistical, each run would be different. As seen, generally the maximum separability value is rather low, revealing that the discriminator is unable to find a model that can separate the in-class samples well from the out-of-class samples.

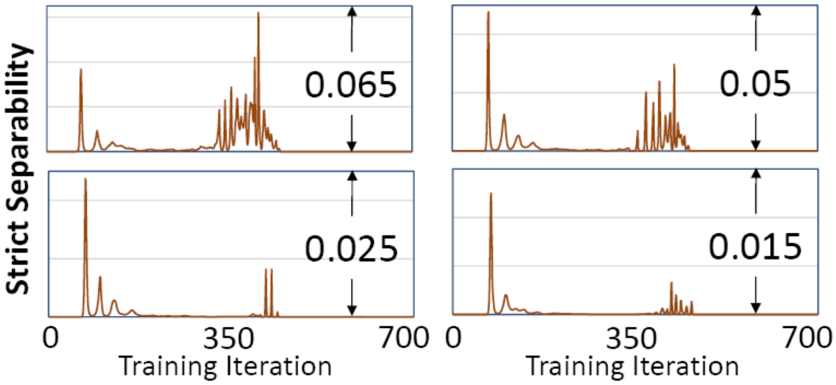


Figure 6.20: Extremely low separability in Concept 8 training

To confirm the issue, Jay also checked the performance of the GANs modeling for Concept 9. Figure 6.21 shows the same issue observed in Figure 6.20.

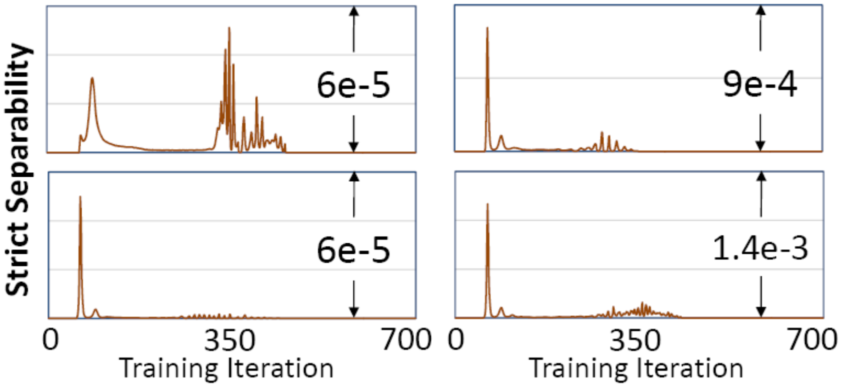


Figure 6.21: Extremely low separability in Concept 9 training

After some experiments, Jay decided to re-design the CNN architecture used by the discriminator to increase its capacity. Figure 6.22 shows the new architecture. With this new architecture, Jay also needed to double check the setting of the various hyper-parameters [2] used in the training and to tune some of the values if needed.

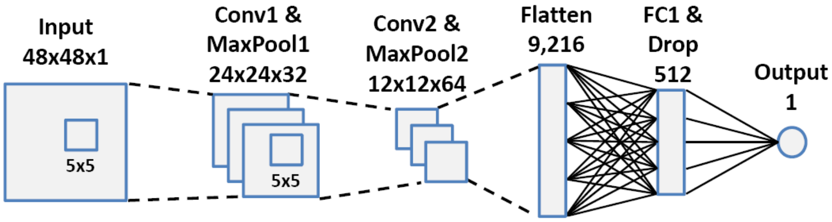


Figure 6.22: The enhanced CNN architecture

Figure 6.23 and Figure 6.24 then show examples of the separability trend inspected by Jay. This time, it can be seen that the maximum strict separability values are much higher than before. These values indicated that with the new CNN architecture, the discriminator would be able to find a model that can separate the given in-class samples well from the out-of-class samples. Note that this observation made by Jay is empirical, which could be viewed as part of the ML expertise the ML expert accumulated through experience.

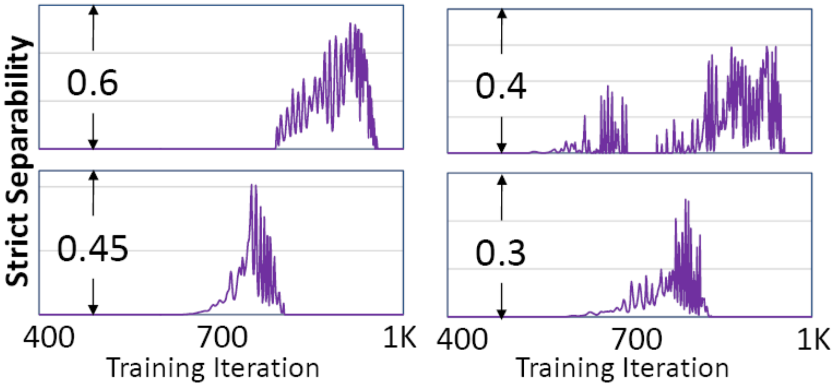


Figure 6.23: Improved separability seen in Concept 8 training

Although the strict separability was much improved, due to the statistical nature of the training process, Jay is also concerned about the stability of the training. The next item to check is how frequently a good separability can be achieved in a large number of training runs.

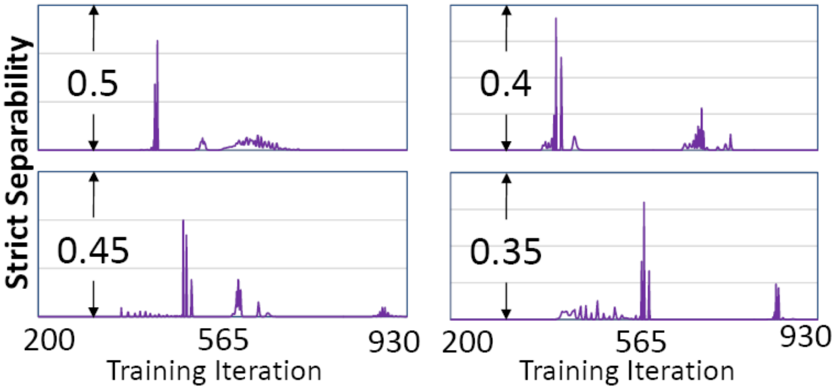


Figure 6.24: Improved separability seen in Concept 9 training

### 6.4.3 Hint for ensemble learning

Figure 6.25 shows a histogram plot of the maximum strict separability over 100 runs of training. Observe that for Concept 8, most of the separability values are greater than 0.5 but it is not the case for Concept 9. This means that the problem for recognizing Concept 9 is harder than the problem for recognizing Concept 8.

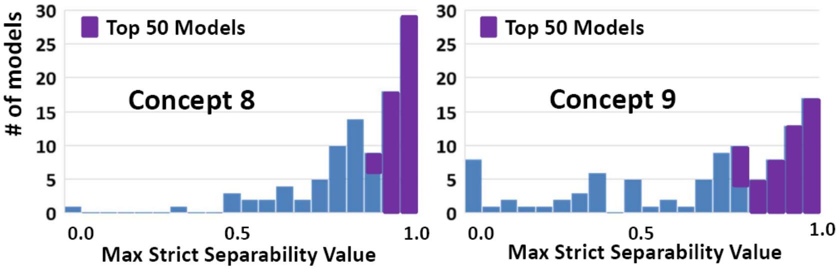


Figure 6.25: Max Separability histogram from 100 training runs

The instability in the training results for Concept 9 shows that one run of training might not be reliable, i.e. we might end up with a model with very low separability value even though that value is the maximum seen in the training iterations. This concern motivated Jay to adopt *ensemble learning* to mitigate the issue.

In the ensemble learning, 100 discriminator models are collected with 100 runs of training. Then, the 50 models with the highest separability are used. The recognition is then decided by the 50 recognizers collectively. The decision rule is that if 50% or more recognizers agree that a wafer is in-class, then the wafer is considered recognized.

Note that this 50% rule is empirical and is based on the thinking that if a containment check fails with this rule, then the issue is related more to the CNN architecture itself than to the use of the ensemble learning. In other words, the 50% is more like a lower bound. If the containment check fails in the future, Jay will be called for a service and the improvement will focus on the CNN architecture again.

## 6.5 Re-Deployment Results

After the software is modified by Jay, it is re-deployed. In this section, we focus on discussing the re-deployment results for Concept 8 and Concept 9 on the 287 lots, which shows failing the containment check before.

For illustration, Figure 6.26 shows the effect of ensemble learning in view of the containment check which passes at 68% point, well before the 50% rule setting (recall that one escape is acceptable). With the passing of the check, the next important question is: how many false positives are with the 50% decision rule.

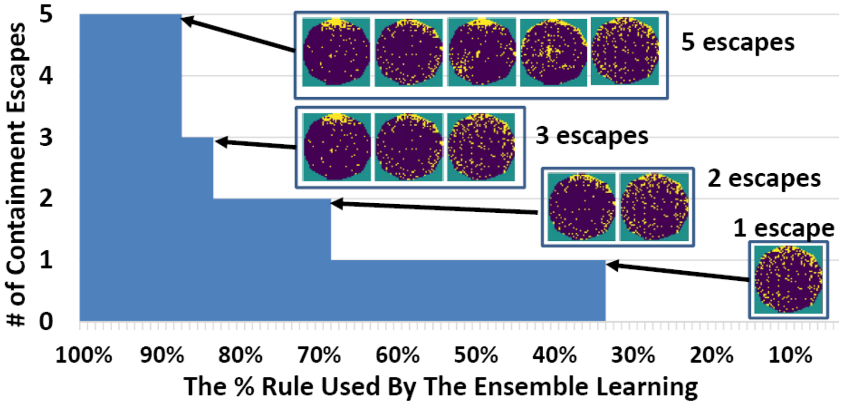


Figure 6.26: The effect of ensemble learning on Concept 9

Table 6.6 summarizes the new results on Concept 8 and Concept 9. With the improved software, the large numbers of false positives shown in Table 6.2 for both concepts are no longer there and also, both pass the containment check.

Table 6.6: Re-deployment results on the 287 Lots

| **Concept** | 8 | 9 |
|---|---|---|
| **Recognized Wafers** | 145 | 77 |
| **False Positive** | 0 | 1 |
| **Checking Wafers** | 47 | 10 |
| **Missing Wafer** | 0 | 1 |
| **Containment Check** | √ | √ |

√: model passing the containment check

For Concept 9, there is still one false positive and one escape from the containment check. They are different wafers and Figure 6.27 shows what they look like. For the false positive, observe that it does have concentrating failures on the top edge, causing confusion to the recognizer looking for an edge failing pattern. For the escape, the edge failing pattern is marginal and hence, is missed by the committee of the fifty Concept 9 recognizers.

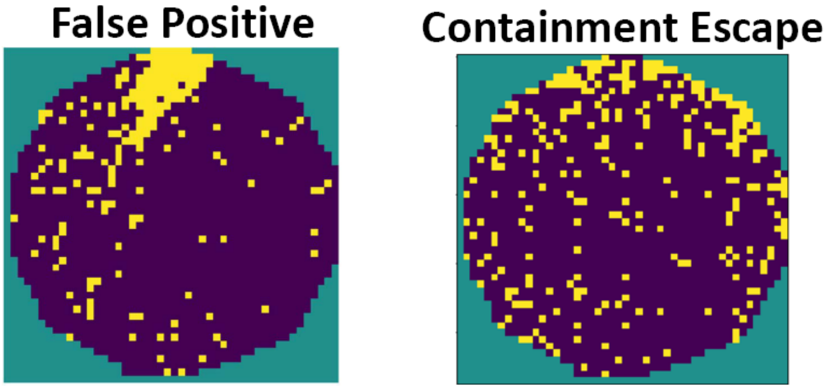


Figure 6.27: The false positive and escape for Concept 9 in Table 6.6

## 6.6 Result From The 2nd Product Lines

The software is also run on a 2nd product line with 7052 wafers coming from 295 lots. Note that due to die size differences, recognizers learned from one product line are not reused in another product line. For example, the die size of the 2nd product is much larger than the die size of the 1st product. Hence, the run begins with a fresh start.

For simplicity, we only report results by giving all wafers at once to the software. After the Filtering Random box, there are 1905 wafers remaining. The clustering step finds 3 clusters with sizes, 27, 64, and 5, respectively. Call them Concepts X, Y, and Z and Figure 6.28 shows two examples of their five training wafers to illustrate what they look like.

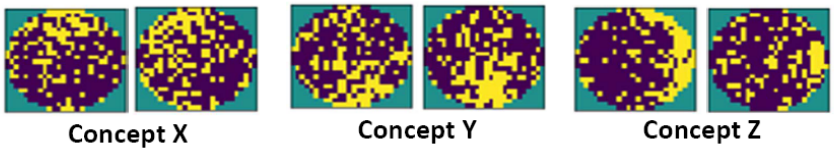


Figure 6.28: Example wafers from Concepts X, Y, and Z

Table 7 summarizes the results from the recognition and containment check. Notice that even though many wafers are not filtered out by the Filtering Random box, much fewer wafers are considered to be non-random after going through the entire workflow. For those unclassified wafers, we did manually inspect them to confirm that they indeed contain no obvious pattern, justifying the report by the software that wafers from this product line are mostly with random fails.

Finally, Figure 6.29 shows what the false positives look like, to confirm there is no systematic surprise in the result.

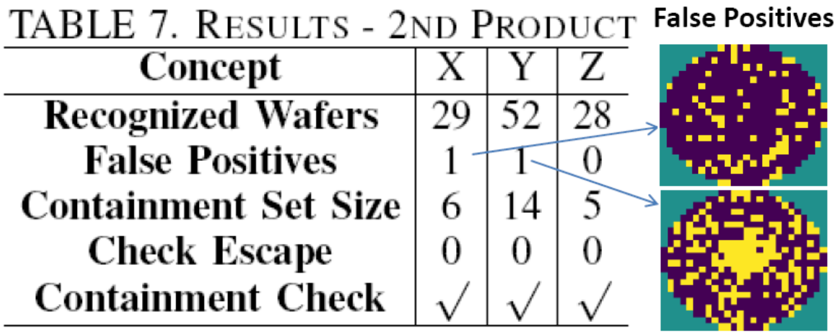


TABLE 7. RESULTS - 2ND PRODUCT

| Concept | X | Y | Z |
|---|---|---|---|
| Recognized Wafers | 29 | 52 | 28 |
| False Positives | 1 | 1 | 0 |
| Containment Set Size | 6 | 14 | 5 |
| Check Escape | 0 | 0 | 0 |
| Containment Check | √ | √ | √ |

False Positives

Figure 6.29: Summary table, the false positive and the escape

## 6.7 Conclusion

In this chapter, we presented the idea of deploying a ML solution as a service using wafer image classification as an application example. A key reason for the idea is that a ML solution cannot guarantee its robustness to begin with. Each new deployment might demand a substantial and long ramp-up period and our software design facilitates the ramp-up.

To realize the idea, the two major requirements are: (1) The software needs to have a way to check its ML modeling capability. If the data in the production cannot be handled with that capability, the software needs to call the ML expert for a service. (2) When a service is called, the samples sent to the ML expert should not give out confidential information, such as yield. This requires taking the original training wafer images and going through some transformations to obtain the in-class as well as the out-of-class samples for the ML expert.

In this chapter, the software is run on two actual product lines. The deployment and re-deployment results are used to explain the design of the software and the techniques involved to meet the two requirements.

On the surface, our software design might seem rather complicated for solving a seemingly common classification problem. First, due to the constraint that some classes might have very few samples available for training a classifier, supervised learning (e.g. multi-class classification) generally would not work. Hence, the classification is treated as a multi-class recognition problem where recognition of each class is treated as an unsupervised learning problem. For solving the multi-class recognition problem, clustering is applicable. However, clustering is known to be not robust and hence, it is used only as the initial step. Other more sophisticated approaches are required to improve the robustness. Finally, the need to enable a service by ML expert further adds to the complication of our software design.

# Chapter 7

# Conclusion and Future Work

In this thesis, we presented a software solution for finding interesting patterns in wafer image classification. Our focus was on two aspects (1) Complete automation, where human input is minimized as much as possible, and (2) Ensuring robustness, that means the results of running our software needs to be as trusted as possible.

Automation is achieved through the following:

1. An automated clustering-based concept extraction algorithm with a novel clustering distance calculation method.

2. An automated algorithm for choosing best training wafers through a novel *learnability* calculation.

3. Automating the GANs training and model extraction, which is achieved by fixing the Discriminator architecture, and introducing a novel strict separability measure.

While the robustness aspect was addressed through introducing the novel Tensor-based *Containment Check* to continuously check results of the GANs, and reporting any problems to the expert.

Moreover, we studied what informationn can be passed to the expert in case of a service request. We perform a Tensor-based transformation on the wafers sent to the expert. This transformation hides secret information such as the yield.

Besides the points mentioned above, the fundemental contribution of this work is introducing the Tensor-based concept recognition which is built on top of the $\boldsymbol{\Delta}$ distance measure presented in Equation 4.1. The automation and robustness of the software is demonstrated by applying it to two automotive product lines with 8300, and 7052 wafers respectively.

## 7.1 Future Work

### 7.1.1 Tensor-Based Oytlier Analysis

The work in [57] provides a robust Tensor low-rank approximation, where a Tensor $\boldsymbol{\mathcal{X}}$ is decomposed according to equation 7.1 to a low-rank component $\boldsymbol{\mathcal{L}}$ which contains the underlying high dimensional pattern, and the sparse, or error, component $\boldsymbol{\mathcal{E}}$.

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{L}} + \boldsymbol{\mathcal{E}} \tag{7.1}$$

This decomposition idea can be applied to outlier analysis in wafer probe testing data. It provides a new outlier score [58][59][60], which contain high-dimensional information.

### 7.1.2 Tensor-Based Classification

The concept recognition idea, can be extended to include multi-class classification. For example the handwritten digit recognition problem [61]. For each class (or digit), primitive concept extraction can be applied to it training data, and a Tucker-based concept recognizer can be build for each of the primitive concepts. Recognized digits for each of primitive concepts extracted from each digit's training data can then combined as one class.

# Bibliography

[1] L.-C. Wang, "Experience of data analytics in EDA and test - principles, promises, and challenges," *IEEE Trans. on CAD*, vol. 36, no. 6, pp. 885–898, 2017.

[2] Removed, "Concept recognition in production yield data analytics," in *IEEE International Test Conferencel*, pp. 1–10, IEEE, 2018.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and J. Bengio, "Generative adversarial networks," *arXiv:1406.2661*, 2014.

[4] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," *arXiv:1606.03498v1*, 2016.

[5] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv:1511.06434v2*, 2016.

[6] C.-H. J. Shan, A. Wahba, L.-C. Wang, and N. Sumikawa, "Deploying a machine learning solution as a surrogate," *IEEE International Test Conference*, pp. 1–10, 2019.

[7] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[8] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[9] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

[10] F. L. Hitchcock, "The expression of a tensor or a polyadic as a sum of products," *Journal of Mathematics and Physics*, vol. 6, no. 1-4, pp. 164–189, 1927.

[11] F. L. Hitchcock, "Multiple invariants and generalized rank of a p-way matrix or tensor," *Journal of Mathematics and Physics*, vol. 7, no. 1-4, pp. 39–79, 1928.

[12] J. D. Carroll and J.-J. Chang, “Analysis of individual differences in multidimensional scaling via an n-way generalization of “eckart-young” decomposition,” *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[13] R. A. Harshman *et al.*, “Foundations of the parafac procedure: Models and conditions for an” explanatory” multimodal factor analysis,” 1970.

[14] H. A. Kiers, “Towards a standardized notation and terminology in multiway analysis,” *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 14, no. 3, pp. 105–122, 2000.

[15] J. Mocks, “Topographic components model for event-related potentials and some biophysical considerations,” *IEEE transactions on biomedical engineering*, vol. 35, no. 6, pp. 482–484, 1988.

[16] J. B. Kruskal, “Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics,” *Linear algebra and its applications*, vol. 18, no. 2, pp. 95–138, 1977.

[17] J. Håstad, “Tensor rank is np-complete,” *Journal of Algorithms*, vol. 11, no. 4, pp. 644–654, 1990.

[18] J. B. Kruskal, “Rank, decomposition, and uniqueness for 3-way and n-way arrays,” *Multiway data analysis*, pp. 7–18, 1989.

[19] A. Shashua and A. Levin, “Linear image coding for regression and classification using the tensor-rank principle,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–I, IEEE, 2001.

[20] R. Furukawa, H. Kawasaki, K. Ikeuchi, and M. Sakauchi, “Appearance based object modeling using texture database: Acquisition compression and rendering.,” in *Rendering Techniques*, pp. 257–266, 2002.

[21] E. Acar, S. A. Çamtepe, M. S. Krishnamoorthy, and B. Yener, “Modeling and multiway analysis of chatroom tensors,” in *International Conference on Intelligence and Security Informatics*, pp. 256–268, Springer, 2005.

[22] E. Acar, S. A. Camtepe, and B. Yener, “Collective sampling and analysis of high order tensors for chatroom communications,” in *International Conference on Intelligence and Security Informatics*, pp. 213–224, Springer, 2006.

[23] B. W. Bader, M. W. Berry, and M. Browne, “Discussion tracking in enron email using parafac,” in *Survey of Text Mining II*, pp. 147–163, Springer, 2008.

[24] L. R. Tucker, “Implications of factor analysis of three-way matrices for measurement of change,” *Problems in measuring change*, vol. 15, pp. 122–137, 1963.

[25] L. R. Tucker *et al.*, “The extension of factor analysis to three-dimensional matrices,” *Contributions to mathematical psychology*, vol. 110119, 1964.

[26] L. R. Tucker, “Some mathematical notes on three-mode factor analysis,” *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[27] L. De Lathauwer, B. De Moor, and J. Vandewalle, “A multilinear singular value decomposition,” *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[28] M. A. O. Vasilescu and D. Terzopoulos, “Multilinear analysis of image ensembles: Tensorfaces,” in *European Conference on Computer Vision*, pp. 447–460, Springer, 2002.

[29] A. Kapteyn, H. Neudecker, and T. Wansbeek, “An approach ton-mode components analysis,” *Psychometrika*, vol. 51, no. 2, pp. 269–275, 1986.

[30] L. De Lathauwer, B. De Moor, and J. Vandewalle, “On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors,” *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1324–1342, 2000.

[31] L. De Lathauwer and J. Vandewalle, “Dimensionality reduction in higher-order signal processing and rank-(r1, r2,. . . , rn) reduction in multilinear algebra,” *Linear Algebra and its Applications*, vol. 391, pp. 31–55, 2004.

[32] H. Wang and N. Ahuja, “Compact representation of multidimensional data using tensor rank-one decomposition,” *vectors*, vol. 1, p. 5, 2004.

[33] D. Vlasic, M. Brand, H. Pfister, and J. Popović, “Face transfer with multilinear models,” in *ACM transactions on graphics (TOG)*, vol. 24, pp. 426–433, ACM, 2005.

[34] B. Savas, “Analyses and tests of handwritten digit recognition algorithms,” *LiTH-MAT-EX-2003-01, Linkˆping University, Department of Mathematics*, 2003.

[35] B. Savas and L. Eldén, “Handwritten digit classification using higher order singular value decomposition,” *Pattern recognition*, vol. 40, no. 3, pp. 993–1003, 2007.

[36] A. Wahba, L.-C. Wang, Z. Zhang, and N. Sumikawa, “Wafer pattern recognition using tucker decomposition,” in *VLSI Test Symposium (VTS), 2019 IEEE 37th*, pp. 1–6, IEEE, 2019.

[37] A. Wahba, C. Shan, L.-C. Wang, and N. Sumikawa, “Measuring the complexity of learning in concept recognition,” in *Int. Symposium on VLSI Design, Automation and Test*, pp. 1–4, IEEE, 2019.

[38] A. Wahba, J. Shan, L.-C. Wang, and N. Sumikawa, "Wafer plot classification using neural networks and tensor methods," in *Submission To ITC-ASIA*, pp. 1–6, IEEE, 2019.

[39] L.-C. Wang, C. Shan, and A. Wahba, "Facilitating deployment of a wafer-based analytic software using tensor methods (invited paper)," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[40] S. Rabanser, O. Shchur, and S. Günnemann, "Introduction to tensor decompositions and their applications in machine learning," *arXiv preprint arXiv:1711.10781*, 2017.

[41] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[42] N. Hao, M. E. Kilmer, K. Braman, and R. C. Hoover, "Facial recognition using tensor-tensor decompositions," *SIAM Journal on Imaging Sciences*, vol. 6, no. 1, pp. 437–463, 2013.

[43] Y. Fu and T. S. Huang, "Image classification using correlation tensor analysis," *IEEE Transactions on Image Processing*, vol. 17, no. 2, pp. 226–234, 2008.

[44] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[45] L. Kuang, F. Hao, L. T. Yang, M. Lin, C. Luo, and G. Min, "A tensor-based approach for big data representation and dimensionality reduction," *IEEE transactions on emerging topics in computing*, vol. 2, no. 3, pp. 280–291, 2014.

[46] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: dynamic tensor analysis," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 374–383, ACM, 2006.

[47] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Transactions on automatic control*, vol. 25, no. 2, pp. 164–176, 1980.

[48] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: dynamic tensor analysis," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 374–383, 2006.

[49] C. Szegedy and et al., "Intriguing properties of neural networks." arXiv:1312.6199v4, 2013.

[50] Pedregosa and et al., "Scikit-learn: Machine learning in python," *JMLR*, pp. 2825–2830, 2011.

[51] L. McInnes, J. Healy, and S. Astels, "hdbscan: Hierarchical density based clustering," *The Journal of Open Source Software*, vol. 2, 2017.

[52] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[53] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 1, p. 5, 2015.

[54] U. Von Luxburg *et al.*, "Clustering stability: an overview," *Foundations and Trends® in Machine Learning*, vol. 2, no. 3, pp. 235–274, 2010.

[55] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv:1701.07875v3*, 2017.

[56] D. Berthelot, T. Schumm, and L. Metz, "Began: Boundary equilibrium generative adversarial networks," *arXiv:1703.10717v4*, 2017.

[57] C. Lu, J. Feng, W. Liu, Z. Lin, S. Yan, *et al.*, "Tensor robust principal component analysis with a new tensor nuclear norm," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[58] L.-C. Wang, S. Siatkowski, C. Shan, M. Nero, N. Sumikawa, and L. Winemberg, "Some considerations on choosing an outlier method for automotive product lines," in *2017 IEEE International Test Conference (ITC)*, pp. 1–10, IEEE, 2017.

[59] S. Siatkowski, C. J. Shan, L.-C. Wang, N. Sumikawat, W. R. Daasch, and J. M. Carulli, "Consistency in wafer based outlier screening," in *2016 IEEE 34th VLSI Test Symposium (VTS)*, pp. 1–6, IEEE, 2016.

[60] S. Siatkowski, C.-L. Chang, L.-C. Wang, N. Sumikawa, L. Winemberg, and W. R. Daasch, "Generalization of an outlier model into a "global" perspective," in *2015 IEEE International Test Conference (ITC)*, pp. 1–10, IEEE, 2015.

[61] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.