University of California
Santa Barbara

Domain-Specific Machine Learning - A Human Learning Perspective

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

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Chuanhe (Jay) Shan

Committee in charge:

Professor Li-C. Wang, Chair
Professor Forrest Brewer
Professor Luke Theogarajan
Professor Zheng Zhang

March 2022

The Dissertation of Chuanhe (Jay) Shan is approved.

Professor Forrest Brewer

Professor Luke Theogarajan

Professor Zheng Zhang

Professor Li-C. Wang, Committee Chair

March 2022

Domain-Specific Machine Learning

- A Human Learning Perspective

by

Chuanhe (Jay) Shan

Dedicated to my mom, dad, grandpa, and all my family and friends who nurtured me and supported me along the way.

Acknowledgements

Special thanks to Dr. Greg Creech at GLC Technologies, Inc. for his generous support and inspiration towards our work. Also special thanks to Sergio Mier from Qualcomm Inc. for his generous support and invaluable input on our work.

We would like to thank National Science Foundation (NSF) for their support of our research. This work was supported in part by NSF Grant No. 2006739 and NSF Grant No. 1618118. We would also like to thank Nimbis Services, Inc./AFRL for their support of our research. Our research would not be possible without these kind supports.

Thanks to my lab-mates for their invaluable friendships and technical feedback: Jenny Zeng, Matthew Nero, Min Jian Yang, Sebastian Siatkowski, Nik Sumikawa, Ahmed Wahba, KK Hsieh, and everyone else whom I was lucky enough to interact with along the way.

Finally, special thanks to our adviser who guided our technical development, helped prepare this thesis, and inspired philosophical thinking behind the technical work. Years ago, our adviser had told us this: "A PhD student learns the techniques. A good PhD student learns the methodology. A great PhD student learns the philosophy. A PhD learns about themselves." And when I was closing the book of my PhD study, I felt that finally, I began to understand what it had meant.

Curriculum Vitæ

Chuanhe (Jay) Shan

Education

2017 - 2022	Ph.D. in Computer Engineering, University of California, Santa Barbara.
2014 - 2017	M.S. in Computer Engineering, University of California, Santa Barbara.
2010 - 2014	B.S. in Computer Engineering, University of California, Santa Barbara.

Publications

1. Y. J. Zeng, L. -C. Wang and **C. J. Shan**, "MINiature Interactive Offset Networks (MINIONs) for Wafer Map Classification," *2021 IEEE International Test Conference (ITC)*, 2021, pp. 190-199, doi: 10.1109/ITC50571.2021.00027.
2. Y. J. Zeng, L. -C. Wang, **C. J. Shan** and N. Sumikawa, "Learning A Wafer Feature With One Training Sample," *2020 IEEE International Test Conference (ITC)*, 2020, pp. 1-10, doi: 10.1109/ITC44778.2020.9325254.
3. **C. Shan**, A. Wahba, L. -C. Wang and N. Sumikawa, "Deploying A Machine Learning Solution As A Surrogate," *2019 IEEE International Test Conference (ITC)*, 2019, pp. 1-10, doi: 10.1109/ITC44170.2019.9000109.
4. A. Wahba, **C. Shan**, L. -C. Wang and N. Sumikawa, "Wafer Plot Classification Using Neural Networks and Tensor Methods," *2019 IEEE International Test Conference in Asia (ITC-Asia)*, 2019, pp. 79-84, doi: 10.1109/ITC-Asia.2019.00027.
5. A. Wahba, **C. J. Shan**, L. -C. Wang and N. Sumikawa, "Primitive Concept Identification In A Given Set Of Wafer Maps," *2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2019, pp. 1-4, doi: 10.1109/VLSI-DAT.2019.8741856.
6. L. -C. Wang, **C. J. Shan** and A. Wahba, "Facilitating Deployment Of A Wafer-Based Analytic Software Using Tensor Methods: Invited Paper," *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1-8, doi: 10.1109/ICCAD45719.2019.8942043.
7. M. Nero, **C. Shan**, L. -C. Wang and N. Sumikawa, "Concept Recognition in Production Yield Data Analytics," *2018 IEEE International Test Conference (ITC)*, 2018, pp. 1-10, doi: 10.1109/TEST.2018.8624714.
8. **C. Shan**, P. Babighian, Y. Pan, J. Carulli and L. -C. Wang, "Systematic defect detection methodology for volume diagnosis: A data mining perspective," *2017 IEEE International Test Conference (ITC)*, 2017, pp. 1-10, doi: 10.1109/TEST.2017.8242050.

9. 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," *2017 IEEE International Test Conference (ITC)*, 2017, pp. 1-10, doi: 10.1109/TEST.2017.8242047.
10. S. Siatkowski, **C. J. Shan**, L. -C. Wang, N. Sumikawat, W. R. Daasch and J. M. Carulli, "Consistency in wafer based outlier screening," *2016 IEEE 34th VLSI Test Symposium (VTS)*, 2016, pp. 1-6, doi: 10.1109/VTS.2016.7477267.

Best Paper Awards

1. "Learning A Wafer Feature With One Training Sample", *IEEE International Test Conference (ITC)*, 2020
2. "Primitive Concept Identification In A Given Set Of Wafer Maps", *VLSI Design Automation and Test Symposium (VLSI-DAT)*, 2019
3. "Consistency in Wafer Based Outlier Screening", *IEEE VLSI Test Symposium (VTS)*, 2016

Honorable Mention Paper Award

1. "Deploying A Machine Learning Solution As A Surrogate", *IEEE International Test Conference (ITC)*, 2019
2. "Concept Recognition in Production Yield Data Analytic", *IEEE International Test Conference (ITC)*, 2018

Distinguished Paper Awards

1. "Wafer Plot Classification Using Neural Networks and Tensor Method", *IEEE International Test Conference Asia (ITC Asia)*, 2019
2. "Some Considerations on Choosing An Outlier Method for Automotive Product Lines", *IEEE International Test Conference (ITC)*, 2017

Other Awards

1. **IEEE International Test Conference 2019 Gerald W. Gordon Student Award**, The Gerald W. Gordon Award recipient must be a student in good standing at an accredited university or college. Award recipients must have done volunteer work for one or more IEEE conferences, symposia, workshops and/or organizations dedicated to the development of electronics design and testing fields. Consideration shall be given to the amount of volunteer service given; the breadth of volunteer service given; the impact of the volunteer service give; and worthiness of the candidate.
2. **ECE Department - Outstanding TA Award 05/2018**

Abstract

Domain-Specific Machine Learning

- A Human Learning Perspective

by

Chuanhe (Jay) Shan

This dissertation focuses on Domain-Specific Machine Learning (DSML) with applications in the semiconductor industry. We first illustrate characteristics of DSML in view of semiconductor design and test processes, where a task is usually carried out in an iterative fashion. We introduce a new concept called Local No-Free-Lunch (Local NFL) to characterize the learning problem in an iteration, where one learns from the data in the current iteration and tries to predict the outcome in the next iteration. A consequence of Local NFL is that optimization of a learning model is no longer as critical in DSML as that in traditional Machine Learning.

To study how to build a practical DSML solution, we take Wafer Map Pattern Recognition (WMPR) as the application context. Historically, WMPR is an important problem to solve for semiconductor manufacturers, where wafer failing patterns are used to guide their yield improvement effort. In recent years, fabless companies became interested in the problem because understanding wafer patterns can help them communicate more effectively with the manufacturers. In the past, prior works treated WMPR as solving a multi-class classification problem. However, in view of DSML, we argue that a multi-class classifier is not what is needed for building a practical DSML solution.

We then develop the requirements and constraints for building a practical DSML solution. From there, we reach a list of tool capabilities to be supported by the solution. To enable these capabilities, we develop novel learning techniques for performing various

analyses on wafer maps. Based on the enabled capabilities, a novel methodology is introduced. Given a sequence of wafer maps, this methodology enables a user to quickly see what potential patterns are in the sequence and based on their own perspective, to make a final decision to classify them.

This methodology enables building a practical DSML solution that helps a user answer two main questions in each iteration: (1) Is there a wafer pattern indicating a potential yield issue? 2) If yes, has this pattern occurred previously? An important aspect of the solution is that it enables a user to quickly grasp what crucial information is contained in the data and facilitates them to make a final classification decision. The solution does not make a final decision for them. From this angle, the focus of a DSML solution is on understanding of the data, rather than on optimization of a learning model.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"The only good is knowledge and the only evil is ignorance."

— Socrates

The tacit success of machine learning (ML) is evident by its seamlessly integration into our daily lives. When an user visits a social media website, their profile and activities data are used to suggest more pages and/or people of interest [1]. Similarly, when a shopper visits an e-commerce website, their profile and activities, such as shopping history and product reviews, are fed into a product recommendation system which aims to enhance the shopper's experience [2].

When someone takes a photo with a smartphone, people, pets, common objects and locations, are automatically identified and tagged [3]. The tagged photos enable searching using keywords, such as "beach". Additionally, the photos are automatically grouped, such as a collection of the same pet. For a more rigorous application, bio-metrics authentication with facial recognition is used abundantly across smartphones [4]. The authentication can still be accurate when the user has different accessories and facial hair.

With the advent of smart-home devices, a user can speak English directly to the device to inquire about today's news or check their order status [5]. When an user travels abroad and sees a road sign in another language, they can translate the sign with their smartphone camera in real-time [6]. When working on a coding project, an user can query a desired prompt in natural language (e.g."Return list with elements incremented by 1.") and get back generated programming code [7].

From these aforementioned examples, we can see that ML has revolutionized our lives, specifically excelling in regression, image classification, and natural language processing tasks. All of us probably have interacted with at least one of the above examples. With ML's evident success, one must wonder "*Why* is Machine Learning so successful?"; followed by "How do we bring ML's success into areas outside of the currently established ones?" We explore the first question for the remainder of this chapter, and aim to answer the second question with the remainder of this thesis.

1.1 Machine Learning & Notable Successes

Although ML encompasses a large field of algorithms and methods to automatically learn from the data, all recent examples above involve some form of *deep learning*. In fact, as of 2020, deep learning has become the dominant application and research in the field of ML [8]. In view of ML's success, we can briefly trace back to its early developments from the 1950s to the 2000s, and then examine the recent inception and dominance of deep learning. These more recent milestones provide valuable insights on how deep learning is being applied to the extent of its current success.

1.1.1 Foundational developments

Starting with the *perceptron* [9] in 1958, the algorithm was invented based on a simplified model of a biological neuron. The perceptron algorithm can perform supervised learning of binary classification, where the input are represented by a numerical vector and the output specifies some class. The perceptron is a linear classifier, where it is only capable of learning linearly separable patterns. For a learning dataset that is not linearly separable (e.g. XOR function), the algorithm does not terminate, as it cannot linearly separate all the input vectors. This is unless an error threshold is specified and met or a preset iteration limit is met. To distinguish it from future developments, the perceptron was also prepended with an identifier and became the *single-layer* perceptron. The single-layer perceptron is considered the simplest form of feedforward *neural network (NN)*, where it served as a building block for future works.

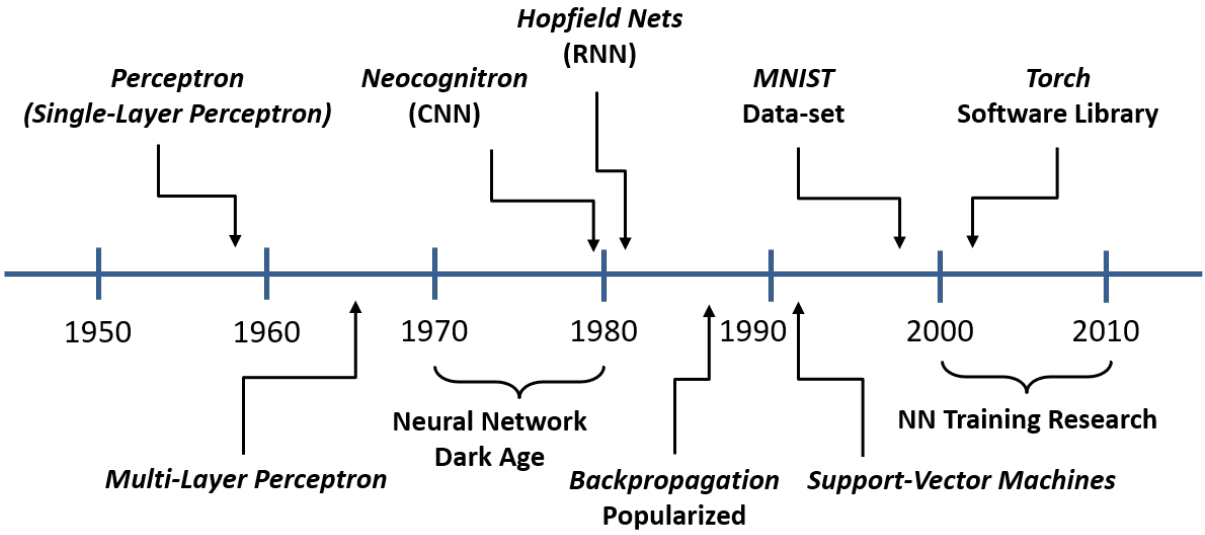

Figure 1.1: Brief history of machine learning foundational developments

Even though the invention of the perceptron generated much excitement at the time in the AI community, it was later proven in 1969 that the algorithm had severe limitations due to only having linear separability [10]. There was also further sentiment that the

extension to *multi-layer* perceptron, invented 4 years earlier in 1965, was not going to solve this issue [10]. In retrospect, this was certainly a large miscalculation. As a result, the NN research and funding areas experienced a stagnation of roughly 10 years. It was later around the 1980s, a resurgence of NN research transpired and many multi-layer perceptron models were being discovered.

During the resurgence around the 1980s, many notable events took place. These include Neocognitron [11] in 1979, which will later inspire *convolutional neural networks (CNNs)*. Soon after, in 1982, the Hopfield network [12] was popularized, which is a form of *recurrent neural networks (RNNs)*. Next in 1986, the widely used *backpropagation* [13] training algorithm was popularized. Backpropagation enabled efficient gradient-based methods for learning a NN and is a core technology still widely used today [14][15].

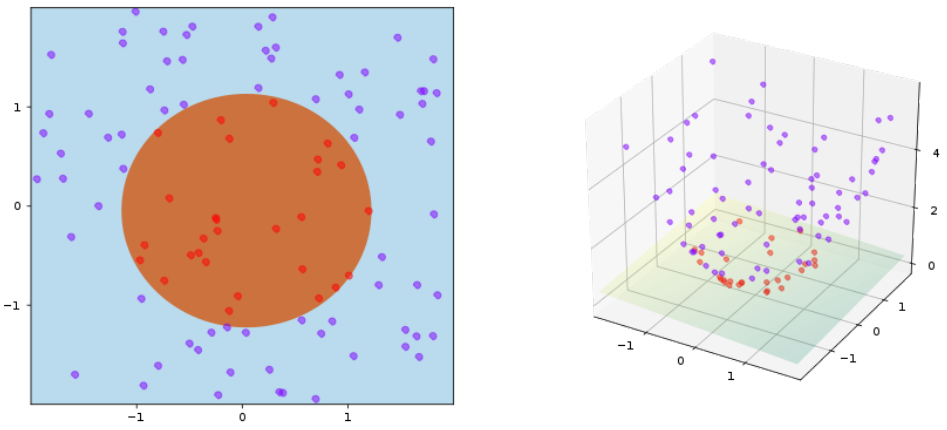

Figure 1.2: The data points are mapped using SVM's kernel from a 2-dimensional space (left) to a 3-dimensional space (right), where a separating hyperplane can be easily found. Illustration from [16].

In 1992, the initial work for *Support-Vector Machines (SVM)* [17] were published. Although SVM is not directly a part of deep learning, it still had significant impacts for classification and regression tasks. SVM was accompanied by strong mathematics and theoretical framework [18], where it could efficiently perform non-linear classification of

samples with the "kernel trick" [19] (Figure 1.2) . This led to SVM quickly becoming the popular classification method around this time, especially on the handwriting recognition problem, courtesy of the NIST database [20]. In addition, SVM's popularity also kicked off a series of subsequent work which focused on kernel-methods [21], all of which employed some form of "kernel trick".

With kernel-methods in favor, in 1998, the *Modified NIST (MNIST)* database [22] was released. This dataset comprises 60,000 training images and 10,000 testing images of curated and labeled handwritten digits from NIST, where it became the standard benchmark for evaluating handwriting recognition classifiers. Although there were neural networks at the time that could offer competitive accuracy [22], the NN approaches would result in much larger models and took a significant amount of computational resources to train. In general, it was widely believed that very large networks are required to capture a detailed learner, but it was impractical to train such classifiers (e.g. vanishing gradient problem [23]).

During the 2000s, kernel-methods remained popular and unsupervised ML become widespread. Notably, the unsupervised learning extension to SVM [24] was published in 2001. Unsupervised ML involves algorithms and methods which does not require user-provided labels in the training dataset for a learning task. There was sentiment that the existing kernel-methods can already appropriately handling their supervised tasks, thus the expansion into unsupervised ML felt like a natural direction to advance ML research. Even with kernel-methods' popularity around this time, there were still notable work developed for neural networks, especially with emphasis on how to efficiently train neural networks with a large amount of data [25][26].

1.1.2 Deep learning in the recent years

In 2009, considered by many as the catalyst of deep learning, ImageNet [27], was released. ImageNet is a database of 14 million labeled full-resolution images, where semantic hierarchies exist between appropriate labels. Before that, WordNet [28], a lexical database of the English language which linked words into semantic (e.g. synonyms) relations, was released in the mid-1990s. At the time of ImageNet's release, WordNet had roughly 80,000 groups of synonyms, called synsets. From the original ImageNet publication: "ImageNet aims to populate the majority of the 80,000 synsets of WordNet with an average of 500-1000 clean and full resolution images." In Figure 1.3, we have

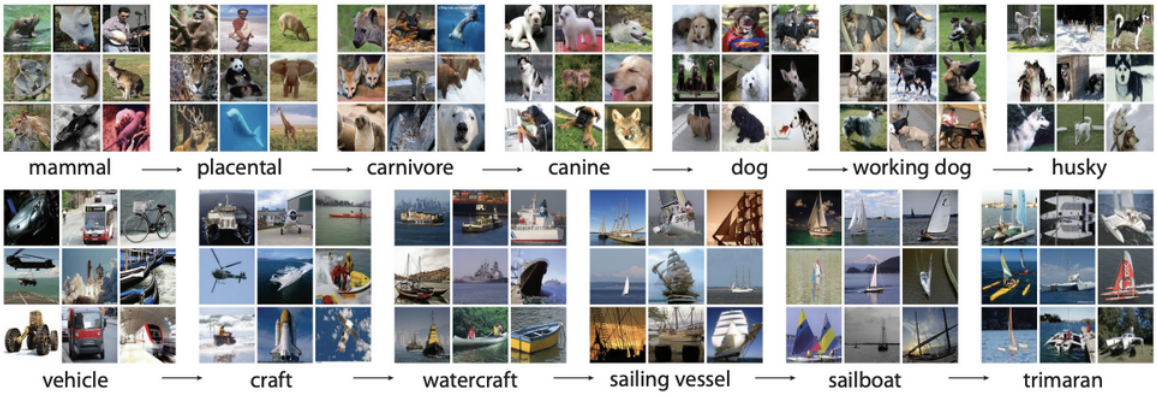

Figure 1.3: ImageNet examples of root-to-leaf branches from [27], "mammal" to "husky" (top) and "vehicle" to "trimaran" (bottom).

two example root-to-node branches of the ImageNet data. On the top, we have a root category "Mammal", where it keeps branching to more hyponymys, until "Husky" on the very right. Similarly, the bottom branch starts from "vehicle" and branches to "trimaran". Each node/category of the branch shows 9 sample images from the dataset.

Accompanying ImageNet's large data collection and annotation effort, the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* [29] is also posted annually. The challenge involves using software to correctly classify and detect objects on a disjoint subset of the ImageNet data, around 1000 classes. This contest kicked off the develop-

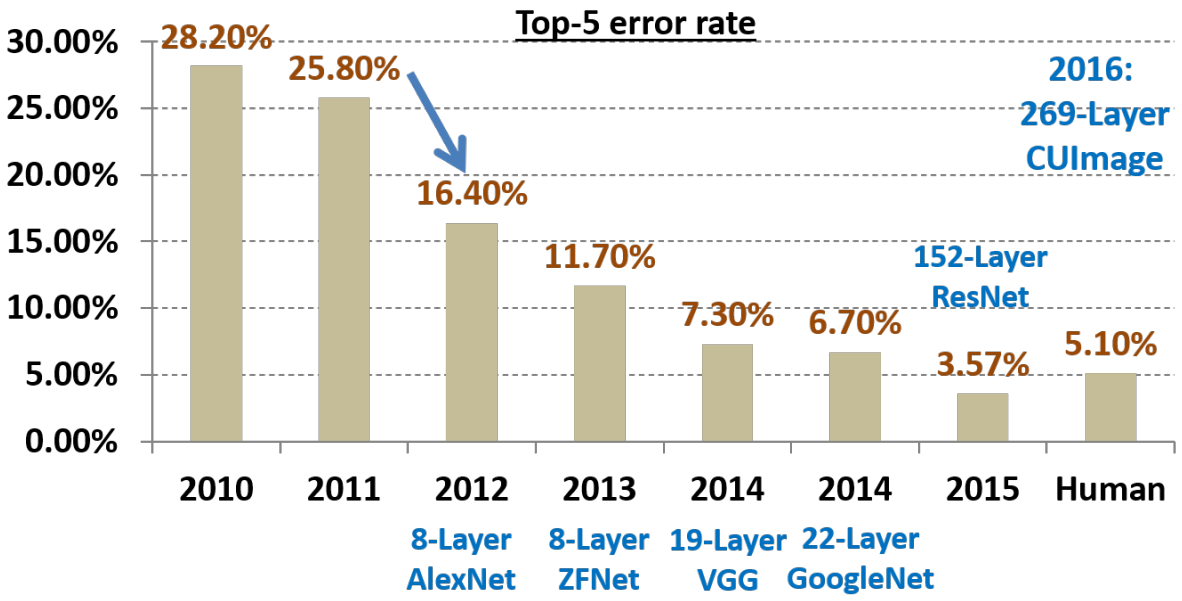

Figure 1.4: The ImageNet challenge drove the development of more accurate classification networks, where these annual winners continued to exhibit deeper architectures.

ment of a series of advanced ML technologies, driven by the goal of achieving a lower error-rate. Figure 1.4 provides a quick summary of the annual winners of ILSVRC and their respective error-rates. Regarded by many as the inception point of deep learning, a notable event took place in 2012, indicated by the blue arrow in Figure 1.4. This event marked a large decrease in error-rate accomplished by AlexNet [30] when compared to the previous 2011 winner. This reduction in error-rate was made feasible by employing a 8-layer network, where the now-affordable Graphic Processing Units (GPUs) are utilized during training, ultimately kicking off the trend of utilizing GPUs and other specialized hardware for training subsequent deeper networks. Aside from the trend of deeper networks leading to higher accuracy, it is interesting to note, by 2015, the winner networks were already outperforming sampled human classifiers.

Ever since the success story of AlexNet in 2012, advancements in neural network architecture had been in full-throttle. Many notable architectures were published in the

following years, including Generative Adversial Networks (GANs) [31] and Variational Autoencoders (VAEs) [32], both of which are generative models [33] and can handle unsupervised learning tasks. These two NN architectures will be covered in Chapters 6 & 7 when we detail our main body of work. Just to give context to what "full-throttle" means, in a little over four years of the original GAN publication, there were roughly 500 publications on variations and modifications of GANs [34].

In additional to the image/object classification problem that catalyzed deep learning, there were also massive deep learning advancements in the domain of natural language processing (NLP), specially language understanding. To put the advancement of models into perspective, as of 2021, the state-of-the-art language model is *Generative Pre-trained Transformer 3 (GPT-3)* [35] created by OpenAI. As its name implies, GPT-3 is the 3rd generation of their language model. Sitting at 175 billion parameters, this generation's capacity is over two orders of magnitude larger than its previous generation [35]. In fact, the next largest model, *Turing Natural Language Generation (T-NLG)* [36] published by Microsoft, sits at 17 billion parameters. For a baseline juxtaposition, *Bidirectional Encoder Representations from Transformers (BERT)* [37] is a language model published in 2018 and achieved state-of-the-art performance on language understanding tasks at that time. BERT boasted 340 million parameters, roughly 500 times smaller than GPT-3.

It is important to note, as their names indicate, both BERT and GPT-n architectures involve *transformers* [38], which is a NN model also designed to handle sequential data, but allows for more parallelization compared to traditional RNNs. This additional parallelization enabled training on larger datasets than previously feasible. As a result, accompanying the enormous capacity of GPT-3 is an enormous amount of training data. Table 1.1 is from the GPT-3 publication [35] and details the amount of training tokens per dataset and their corresponding dataset weighting. During training, the authors used weightings presented in Table 1.1 to draw samples from each dataset. These weights are

Dataset	Quantity(tokens)	Weight in training mix
Common Crawl	410 billion	60%
WebText2	19 billion	22%
Books1	12 billion	8%
Books2	55 billion	8%
Wikipedia	3 billion	3%

Table 1.1: Dataset used to train GPT-3.

intentionally disproportional to the size of each dataset, where higher weightings were assigned to datasets of perceived higher quality. Given the scale of 175 billion parameters and hundreds of billions of training tokens, the training was performed on high-bandwidth clusters equipped with V100 GPUs, which took $3.64E+03$ petaflop-days of training.

Now that we have some context to the massiveness of GPT-3, we can examine its capabilities. Since GPT-3 was trained on data from the open Internet, which includes a large variety of NLP interactions, it does not require further training for distinct NLP tasks. This is unlike most other NLP services which are designed for specific tasks, such as sentiment analysis or language translation. GPT-3's usage revolves around an input prompt, where it will attempt to match whatever context or pattern is present in the prompt and generate a text completion. The following are some examples provided by GPT-3's documentation. User input prompt is highlighted in brown and GPT-3 output is highlighted in blue.

Example 1

Write a tagline for an ice cream shop

We serve up smiles with every scoop!

Example 2

The following is a conversation with an AI assistant. The assistant is helpful, creative, clever, and very friendly.

Human: Hello, who are you?

AI: I am an AI created by OpenAI. How can I help you today?

Human: I'd like to cancel my subscription.

AI: Sure, I can help you cancel your subscription. Just let me know what type of subscription you have.

Example 3

Postgres SQL tables, with their properties:

#

Employee(id, name, department_id)

Department(id, name, address)

Salary_Payments(id, employee_id, amount, date)

#

A query to list the names of the departments which employed more than 10 employees in the last 3 months

SELECT

DISTINCT department.name

FROM department

JOIN employee ON department.id = employee.department_id

JOIN salary_payments ON employee.id = salary_payments.employee_id

WHERE salary_payments.date > (CURRENT_DATE – INTERVAL '3 months')

GROUP BY department.name

HAVING COUNT(employee.id) > 10;

The examples above are just a small snippet of all the official examples on how to leverage GPT-3. In fact, at the time of this writing, there were 49 official examples on how to create prompts for GPT-3, each example aimed at a different NLP task [39]. If we examine GPT-3's output from these examples, we can see that the quality of the generated text is very high. In addition, like many other NN models, GPT-3 also allows for fine-tuning. This is especially helpful for the smaller GPT-3 model variants [35]. When the user encounters NLP tasks and contexts which might not be abundantly available in the GPT-3 training dataset, the user can fine-tune GPT-3 with some additional examples of their specific data, where it can lead to an overall performance improvement [35].

1.2 Machine Learning in Semiconductor D & T

In the previous section, we have briefly covered the developments of ML, to the inception of deep learning, and an example state-of-the-art deep learning model. In this section, we aim to review related works in the domain of **semiconductor design and test (D & T)**. This review is largely based on the summary paper reported in [40] and recent tutorials presented at the IEEE International Test Conference [41].

In semiconductor D & T, tasks can be divided into three separate stages: pre-silicon, post-silicon, and in-field. Figure 1.5 depicts some of the tasks in these three stages.

From left to right, the first is functional verification which concerns the correctness of the simulation model, e.g. the RTL model. Layout verification concerns manufacturability of the layout model, e.g. detecting hot spots. Design-silicon timing verification concerns the convergence between the predicted timing in the pre-silicon stage and the observed timing in the post-silicon stage. Various data learning techniques had found application in these verification tasks in the past [40].

Moving into the post-silicon stage, speed test concerns the quality and cost of delay

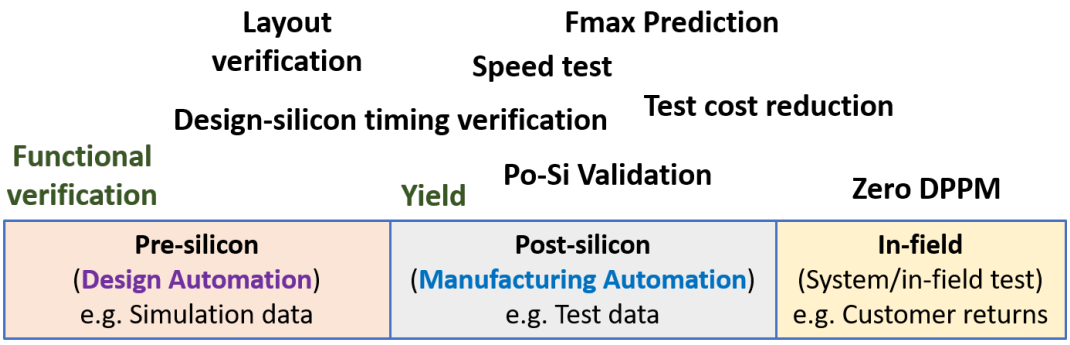

Figure 1.5: Some examples of tasks across semiconductor D & T where ML had found an application [40][41].

testing. Fmax prediction concerns sorting chips into performance bins. Post-silicon (Po-Si) validation concerns validating certain performance metrics that are hard to predict accurately in the pre-silicon stage, such as bit error rate of a high-speed serial link. Test cost reduction concerns optimizing the test efficiency, constrained by a test quality metric. Tasks in this stages rely on measurement data from actual silicon chips, in contrast to those pre-silicon tasks where simulation data are involved.

In the in-field stage, the major concern is to make sure there is almost zero defective chips at the customer site. For example, zero DPPM (defective parts per million) can be a quality metric promised to the customer, which can mean that no single part will be found defective among a million parts. A common approach to attain this promise is to have an effective customer-return resolution methodology in place. This methodology allows the chip supplier to quickly respond to any customer-found defective part and resolve the issue so that no future parts will have a similar problem. Usually, the root cause and the resolution need to be communicated back to the customer. Certain evidence need to be presented for the resolution to be accepted by the customer.

1.2.1 Lacking the "care" samples

One common problem faced when applying ML in a task from D & T is the lack of samples that we really care about. To elaborate, this does not mean we do not have a large amounts of data. It means that, even within a large amount of data, we might not have a large number of samples relevant to the task we are trying to perform, i.e. "care". Let us consider functional verification as an example.

Functional verification starts with a verification plan, specifying the aspects of the design to verify [42], where ultimately the verification quality is measured by some coverage metrics. In addition to manually-written direct tests, tests can be generated by constrained random test generation, which is guided by constraints and biases specified in a test bench (or test template). In functional verification, the main objective is to improve the coverage. Figure 1.6 illustrates the problem setup.

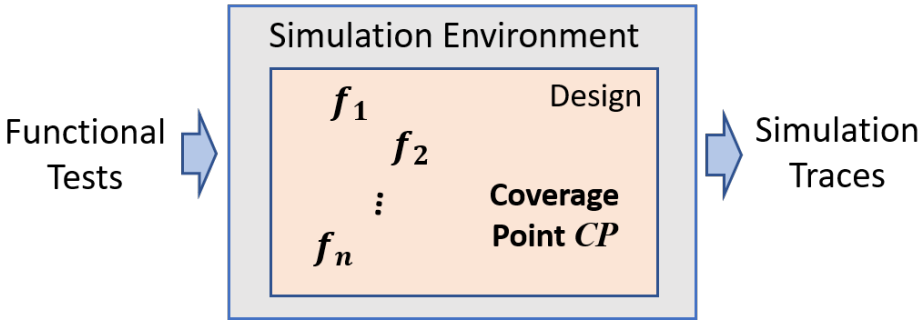

Figure 1.6: Functional verification for improving coverage

Suppose a coverage point CP in the design is not *hit* (or "covered") or only hit a few times (once or twice) by the simulation so far. The task is to attain a functional test that can hit the coverage point CP in the next simulation. Suppose we know how to prepare tests to control a certain set of state variables, $f_1, \ldots, f_n$. In essence, we can say that the underlying question is "what combination of values should be set for those state variables in order to hit CP?" Although, the problem is not as simple because there is

the timing cycle aspect involved in the simulation [43].

We can view the essence of the learning problem as having a dataset similar to the illustration in Figure 1.7. One can think that each sample is an observation extracted from the simulation traces. Each x_{ij} is the value of state variable f_j on observation vector x_i. If CP is hit once, without loss of generality, we have $y_1 = 1$ and all other y's are zero. If CP is never hit, we have $y_i = 0, \forall i$. For a x_i with $y_i = 1$, it is called a *positive sample*. Otherwise, it is called a *negative sample*. Given such a dataset, the learning goal is to understand how to assign values to the state variables to achieve $y_i = 1$.

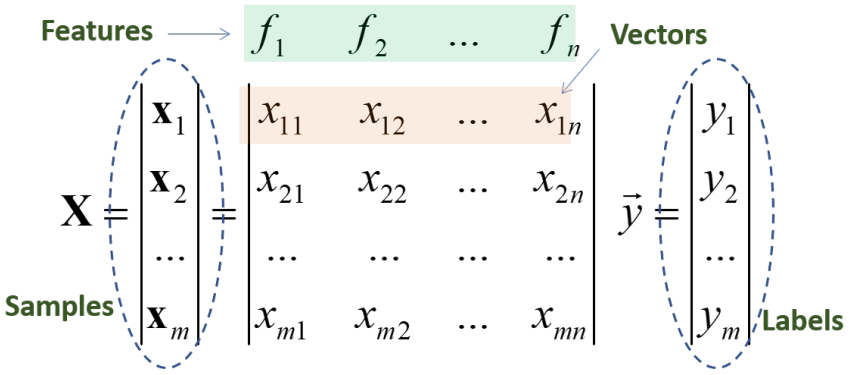

Figure 1.7: An illustration of dataset

A similar example in terms of lacking the care samples is the task of design-silicon timing verification. In this verification task, from the dataset view, a sample x_i is defined as a path in the design. Features $f_1, \ldots, f_n$ are some design features affecting the timing of a path. The label y_i is 1 if the path is deemed to be critical (explained below). Otherwise, y_i is 0. The goal is to use the features to explain the criticality of the path.

For example, Figure 1.8 shows critical paths collected from a silicon experiment [44] and their predicted timing slacks from a Static Timing Analysis (STA) tool. For a path, its actual delays are measured on a set of process cores. The measurement is carried out by *frequency stepping* where step 1 has the lowest frequency. Each dot shown in the plot represents a path. For instance, at step 1 there are four silicon-critical paths.

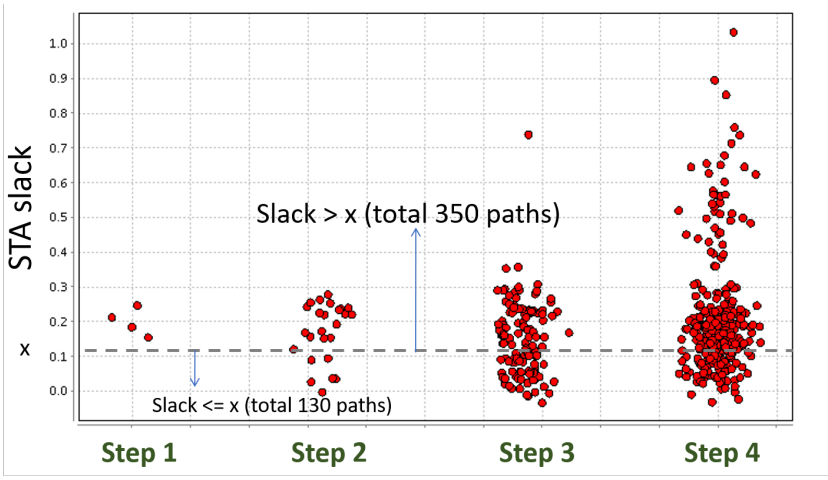

Figure 1.8: Silicon critical paths found in 4 frequency steps [44]

For this particular design, the assumption is that if the timing slack is less than a selected value (denoted as x), then the path is reported as a STA-critical path. In the plot, the (normalized) value of x is shown. It can be observed that 350 out of the 480 paths shown in the plot are not reported as STA-critical paths. In fact, based on the given x value, STA reports 21,589 STA-critical paths and only 130 of those paths show up in this plot as a silicon-critical path [44].

To understand why a silicon-critical path is not a STA-critical path, one can start the analysis with the first four silicon-critical paths shown on the left of the plot. However, a more reasonable way is to analyze them one by one because each might be due to a different reason. In this case, the positive sample is one path. The negative samples could be the 21,459 ($= 21,589 - 130$) STA-critical paths which do not show up as a silicon-critical path. To enable the analysis, a set of design-related features need to be developed for describing a path [44].

The learning problem illustrated above can be stated as the following: Given a dataset with mostly negative samples, the learning goal is to extract a *rule* that explains the dataset [43][44]. This rule is supposed to guide us to find a positive sample.

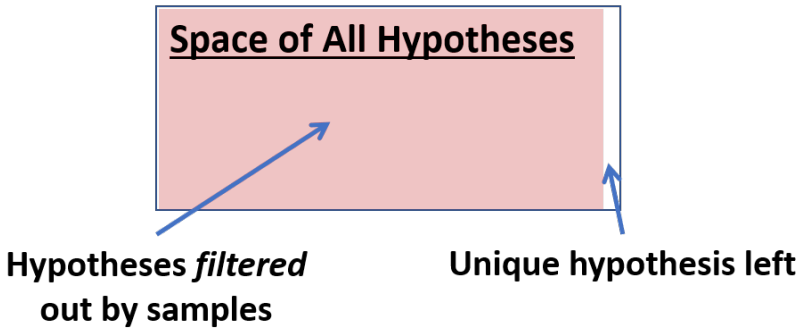

Figure 1.9: Learning as a filtering process [45].

The work in [45] proposes an approach to tackle this learning problem. Figure 1.9 illustrates the thinking behind the approach. Essentially, the approach views the learning problem as a *filtering* process. At the start, a hypothesis space is formed based on the features provided. Then, samples are used to filter out hypotheses that do not fit the dataset. In this process, a positive sample has much more filtering power than a negative sample [45]. However, it is important to note that the filtering power of a negative sample is not zero. The answer, i.e. the rule, is found when the hypothesis space has only one hypothesis left. As a simple example, this answer can be a combination of two features such as "$f_i = 1 \wedge f_j = 1$". In other words, after the filtering, there is an unique answer. The work in [45] develops a tool to check for this uniqueness property. In essence, the tool is evaluating a coverage of the samples on the hypothesis space.

It should be noted that solving the above learning problem with few positive samples, in essence, is the same as with no positive samples. The reason is that given a hypothesis space H, we can use the few positive samples to filter out hypotheses in H to obtain a new and reduced hypothesis space H'. Then, we are left with the same learning problem with H' where there is no positive sample.

Also note that for a given hypothesis space, it is possible that all hypotheses are filtered out by the samples. In this case, the hypothesis space contains no answer and

the "learning" fails. Such a failure can tell the user that a modification of the hypothesis space is required. For example, a user picks an initial set of features to perform the analysis. The tool tells the user no rule can be found. Then, the user can pick a different set of features to search again. In such a search process, having a tool that can decide there is no rule found, can facilitate the search. In contrast, a traditional learning tool would always give its so-called "best" answer and the user has to verify manually if the answer is actually meaningful in view of the task.

The above two examples illustrate a learning problem setup commonly encountered in a task from D & T: Given a dataset with a large number of negative samples and given a hypothesis space, determine if the hypothesis space contains the *cause* for a positive sample. The problem is essentially a "space-pruning" problem. Because the goal is to look for a cause, the problem is more of a diagnosis or debug problem (an induction) than a traditional machine learning problem (a transdunction, see e.g., [46]).

1.2.2 Being able to answer "No Models Found"

One of the most successful results reported from our lab was the yield optimization work documented in [47]. The work started when we were given the data to resolve a production yield issue for a high-volume automotive SoC chip product line. The production had been in progress for more than a year. A significant yield fluctuation was observed, which is illustrated by the left plot in Figure 1.10. This plot is based on approximately 2000 wafers. The yield distribution is shown as a density plot where the box shows the 25%-75% quantile spread with its mean noted by a red line. Prior to our engagement, both the product team and the design team had worked on the problem for many months. One design revision, multiple test revisions, and several adjustments to the manufacturing process had been implemented to improve the yield. None of these

attempts succeeded. We were tasked by the company to investigate if "advanced machine learning" techniques could help discover a new solution.

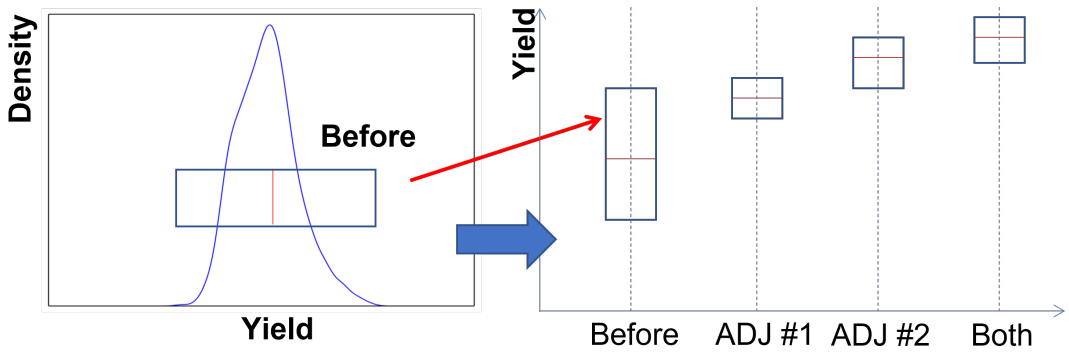

Figure 1.10: Yield improvement result documented in [47].

After working on the problem for weeks, we discovered enough evidence to recommend 5 process parameters for tuning. After several meetings, the contracted foundry accepted and implemented them as two recipe adjustments (ADJ#1 and ADJ#2). Wafers were produced based on applying each adjustment individually, and both together. The right plot in Figure 1.10 shows the resulting yield improvement. In each case, the average yield increased and the fluctuation reduced, both significantly.

There were two aspects with our analytics effort [47]. The first was to search for the process parameters to be tuned. On one end, we had a set of process parameters whose values were measured across wafers. On the other end, we had a set of fail types recorded across wafers. The search was about finding a correlation between a parameter and a fail type. Note that this correlation might exist on a subset of the wafers but not all. While the problem looked like a search problem, the solution also required us to find a way to justify there is no correlation. Suppose we found a correlation model to show that a process parameter p was correlated to a fail type t. In order to implement the process change by adjusting p, we needed to show that p had no correlation to any other fail types. This was to prevent causing more issues by fixing one issue. To ensure

this, we needed a way to say "no correlation" existed [47]. From the high-level, this is similar to the problem discussed above: Given a hypothesis space and the data, we want to determine the space contains no answer with respect to the data.

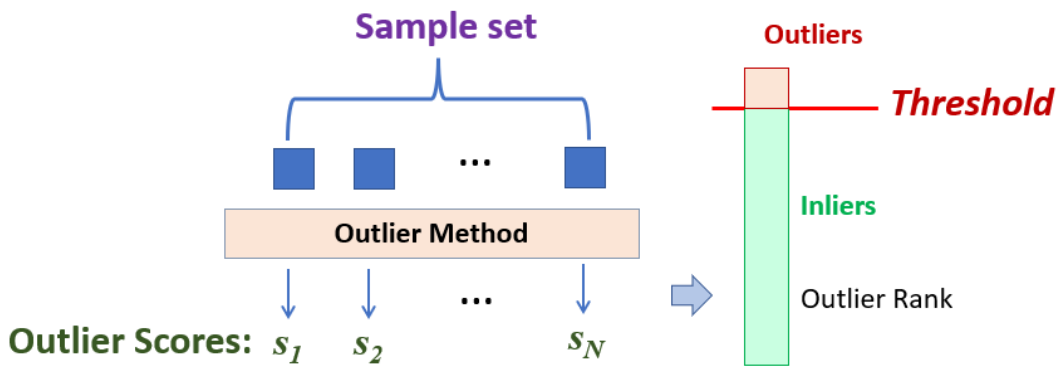

Figure 1.11: Using an outlier method, input samples are assigned outlier scores, where a threshold can be applied to decide the outliers.

The "no model found" decision problem is also observed in other applications in the post-silicon stage. For example, in production testing, one of the most popular screening is based on outlier analysis. Figure 1.11 depicts the outlier analysis setup.

First, a set of samples (chips) are defined. For example, this can be the chips from the same wafer, the same wafer lot (25 wafers), or lots produced in the same week, etc. Then, an outlier method is chosen. This method calculates an outlier score for every sample, based on one or more test values measured on the sample. The scores allow ranking of the samples. Finally, a threshold is chosen to separate outliers from inliers. When developing an outlier model, these three aspects all need to be considered. These decisions are usually based on a collection of chips where some of them are known to be failing chips. Then, the goal is to find a model to screen all known failing chips while not screening too many known good chips.

In some applications, the number of known failing chips is small, e.g. one. For example, in customer return analysis [48], one customer return is given and we are asked

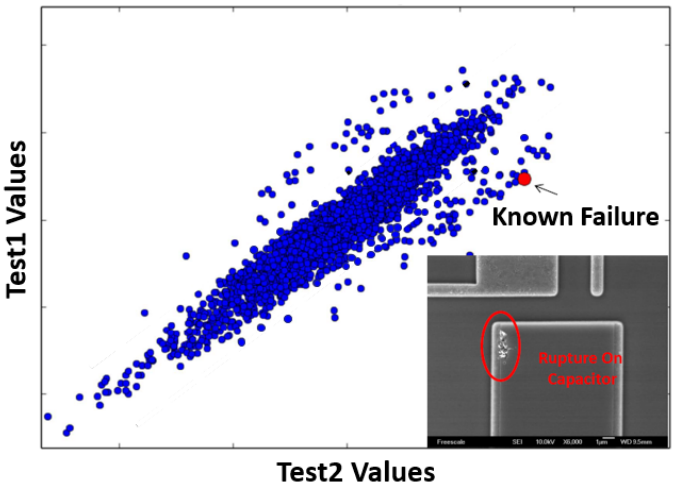

Figure 1.12: Projecting a known failing part as an outlier

to find a model that projects the known failing chip as an outlier. Figure 1.12 depicts such an example. The bottom-right picture shows the actual cause for the failure, a picture taken by the Failure Analysis (FA) team. The plot is based on two test values and every dot represents a chip. The known failing chip seems to be an "outlier" in this space. Similarly, in burn-in cost reduction, one tries to learn an outlier model to screen those chips that are likely to fail during burn-in testing [49].

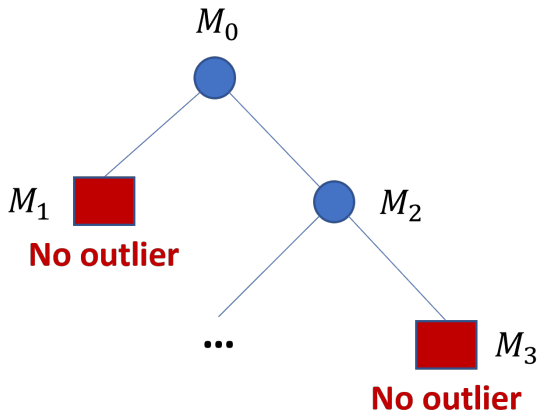

Figure 1.13: Searching for an outlier model for a chip, where each branch is a possible choice of test space defined by one or more tests.

In those applications, one is searching for an outlier model for a given chip. In this

search, what we need is a way to say no outlier model exists that can project the chip as an outlier. This is illustrated in Figure 1.13. Each branch in the search tree can correspond to a choice of a test space, e.g. based on a test or a combination of tests. Then, for example, no outlier model can mean that no outlier method with a reasonable threshold can project the chip as an outlier in that test space. This is in essence the method proposed in [50], which defines a notion called *consistent outlier* to enable making a decision to say there is no outlier.

In the two application contexts described above, the solutions proposed include the capability to provide a "No" answer to the search question. In the first context, given the data, one searches for a correlation model. The solution demands the capability to say there is no correlation model found. In the second context, one searches for an outlier model. The solution demands the capability to say there is no outlier model found.

1.2.3 Applicability check

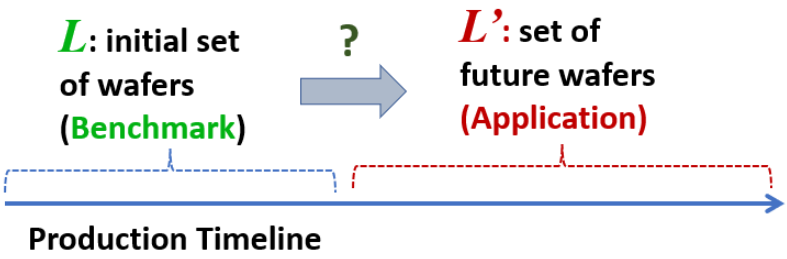

Figure 1.14: Initial set of wafers may not be representative for future set of wafers

As mentioned above, in production testing, one develops an outlier screening model based on the data at hand, for example an initial set of wafers in the early production. However, this initial set may not be representative for future sets of wafers produced. This raises the question of how applicable a model is on the future wafers, if the model is developed based on the early wafers. In production, many things can change over time.

For example, the process might be improved. The test set might be optimized. These changes can affect the characteristics seen in the test data. The work in [51] studies this issue and proposes an *applicability check* method to mitigate the issue. The idea is that, before applying a model to a dataset, applicability check would first check the dataset to see if it is suitable for the model or not.

In a typical learning setup, we are given a dataset and asked to find the best model based on the dataset. In view of applicability check, we are given a model and asked if the dataset is suitable for the model. In [51], the applicability is measured by evaluating against the assumption of the outlier method that built the model. In other words, every learning method starts with an assumption of the underlying hypothesis space, and applicability check should evaluate this assumption against the data.

1.3 Summary

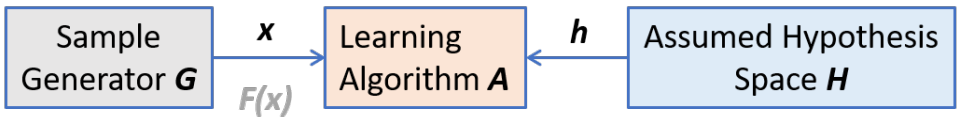

Figure 1.15: Three important components in a ML setup

In a typical machine learning setup, there are three important components to consider, as illustrated in Figure 1.15. There is a data generator G that provides the data. There is an assumed hypothesis space H that contains all possible answers to fit the data. Then, there is a learning algorithm A that finds the best answer in H. If the learning is supervised, for every sample x, a label $F(x)$ is also included. If we view applying ML as a process to find the best model for the given data under a given hypothesis space (e.g. best in terms of an accuracy measure), then it is not surprising that ML largely focuses on the algorithm component.

For all the D & T tasks reviewed above, it seems that all the solutions require attention on issues related to the hypothesis space H and the data generator G. For example, for the two applications reviewed in Section 1.2.1, the solution is required to evaluate a coverage on H. For the two applications reviewed in Section 1.2.2, in order to say no model is found, the solution is required to exhaust the search on H. Finally, in Section 1.2.3, the solution needs to deal with the issue where a model h found based on the current data may not be applicable on future data. This can be seen as G being changed after h is found. In all these tasks, it seems that the real concern is on the other two components but the learning algorithm A.

Chapter 2

Domain-Specific Machine Learning

"Ignorance is preferable to error, and he is less remote from the truth who believes nothing than he who believes what is wrong."

— Thomas Jefferson

As detailed by the last chapter, it seems applying ML in semiconductor D & T is very different from applying general ML. In this chapter, we aim to articulate these differences. Specifically, we aim to demonstrate unique aspects of **Domain-Specific Machine Learning (DSML)** discussed in Section 1.1.

2.1 Iterative Process

As discussed in Section 1.2, all DSML tasks in the semiconductor D & T domain involve an iterative process. This is understandable because in a company, most of the engineering works are carried out in an iterative fashion. It is in such an iterative process that a design is improved, a production process is optimized, a test content is finalized, an issue is resolved, etc. In such an iterative process, one of the most valuable tools an

engineer desires is the one that can help them achieve a *closure* faster. In a high-level sense, a closure is determined if the observed data has stabilized. We use Figure 2.1 to illustrate this point.

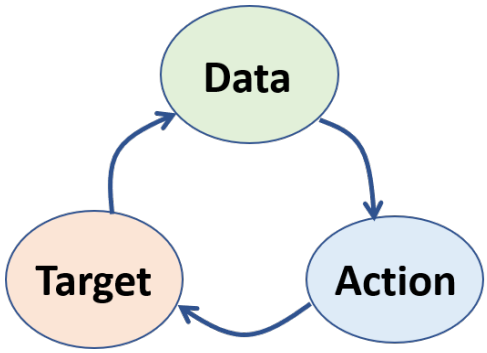

Figure 2.1: Three major components involved in an iterative process

Conceptually, the iterative process is involved to improve on a *target* which can be the design, the production, the test set, and so on. Some characteristics of the current target can be observed in the data collected. For example, if the target is a design, the means for this observation can be a simulation. If the target is a production, the means for this observation can be through some sensors on the equipment. If the target is a test set, the observation is through the test data. An engineer analyzes the data to learn about the characteristics. This learning leads them to take an action to improve the target. The improved target leads to new data. The data leads to new action, and so on. In this iterative process, when the data *stabilized*, i.e. from the engineer's perspective, nothing new can be learned, then a closure is reached.

Recall, in functional verification, an expert is modifying a set of state variables per iteration, trying to hit a specific or more cover points compared to previous iterations. Similarly for yield optimization, the expert is performing an iterative search for the process parameters to be tuned. Likewise, for outlier analysis, the expert is searching for a part's outlier model and adjusting the comprised tests at every iteration. Generally

speaking, these DSML tasks are trying to iteratively debug and improve the current design and production process. These iterative improvements are across all aspects of functional, performance, and/or cost. To state this DSML iterative improvement process in another way, we can refer to Postulate 1 from my colleague's thesis [52], which also indepthly focused on DSML problems.

> **Postulate 1:** For a domain-specific machine learning problem involving a data generator G, the goal of learning based on a data-set D given through G is to modify G to obtain a new data-set D', where D' contains new information that is previously unknown.

In [52], Postulate 1 views an expert's iterative improvement process as a process which generates novel samples (i.e. "surprising" information), which is inherently *unsupervised*. Interpreting this view, it is natural that an expert desires to see new results from their debugging/improvement process, as this is indicative of their further understanding towards their data and the underlying problem to be solved.

To elaborate, during this iterative improvement process, an enormous amount of data are produced. Specifically, the data is accumulated over iterations. In view of applying ML, the goal is to predict results in iteration $i+1$ from the accumulated data at the current iteration i. However, in view of the domain expert's need, they are trying to understand what happened in the data during the current iteration i, to guide them to new data points (i.e. new information) in the next iteration $i+1$. In a sense, the expert is performing a search in this "improvement" search space, where they aim to explore a new region of the search space during every iteration. Consequently, in every iteration, they desire to establish the next search region for their $i+1$ iteration.

Following the essence of Postulate 1, we can refer back to the three main components in a ML setup depicted by Figure 1.15. In a general ML setup, it is presumed the

sample generator G and hypothesis space H are fixed, where the focus is on the learning algorithm A finding a hypothesis h. However, as discussed in Section 1.3, when applying DSML, during every iteration, the expert desires to actively modify G and H (i.e. current search region) based on the results, to obtain potentially new G' and a new H' (i.e. new search region) for establishing their next iteration. In this sense, DSML is trying to achieve a goal that is fundamentally different from general ML. In DSML, the value is in attaining the new G' and/or new H'. In general ML, the value is in accurately modeling G with a given H.

2.2 Embracing *Local No-Free-Lunch* in DSML

This DSML phenomenon of being unpredictable is described as a new notion called *Local No-Free-Lunch* from the related thesis [52], where "local" is referring to the scope of iteration i to $i+1$. To give more context, the No-Free-Lunch theorem [53] "state[s] that any two optimization algorithms are equivalent when their performance is averaged across all possible problems" [54]. Essentially, this theorem implies that given a learning algorithm A, when considering all instances of problems, A does not perform better than random. In view of Figure 1.15, when applying general ML, if G and H are fixed, then we do not know if No-Free-Lunch will strictly apply for learning algorithm A. However, in an iterative DSML context, when G and H can change per iteration, we approach the definition of No-Free-Lunch between iterations, thus the name *"Local No-Free-Lunch"*.

In view of this *Local No-Free-Lunch* phenomenon in DSML, if we look back at the works discussed in Section 1.2, it is natural that they focused on developing capabilities to facilitate the domain experts' iterative search, as opposed to focusing on developing better models to predict future data with current samples.

In other words, an implication of *Local No-Free-Lunch* is that, for DSML contexts, the

learning algorithm A should not make final decisions for the experts. As these decisions could be anti-synergistic towards the experts ultimate goal, to better understand how to modify both G and H. Instead, especially with the enormous amount of data involved, DSML tools should focus on how to summarize the data from the current iteration i, to assist the expert's decision making towards establishing $i + 1$. In fact, Section 1.2 had already discussed some capabilities required by such a DSML data summarization tool. Below are the list of these requirements, where the newly introduced fourth capability will be discussed extensively in later chapters.

1. The capability to evaluate coverage of the samples on the hypothesis space
2. The capability to perform an exhaustive search of a user-defined hypothesis space to get a "no models found".
3. The capability to evaluate a given model's applicability for the current data.
4. The capability to perform an unsupervised sample-based analysis.

In essence, when developing DSML tools, we need to consider the data generator G and assumed hypothesis space H, where they both can be dynamic. The first three requirements above resonate with this theme. In contrast, general ML is more focused on optimizing the learning algorithms. In this sense, when developing a DSML solution, we need to focus on aspects which are complementary to general ML, or simply **Co-ML**. The four capabilities listed above are examples for the problems considered in Co-ML. However, even though the focus is different from a traditional ML algorithm, existing ML techniques might still be useful to implement a Co-ML approach.

In this thesis, our application of focus is the *Wafer Map Pattern Recognition (WMPR)* problem. It is in this application context we observe the need for the fourth capability listed above. To attain this capability, ML techniques are utilized. The capability calls

for the ability to analyze data at a sample-to-sample unsupervised fashion (explained in later chapters). In the later chapters, we will introduce and detail the WMPR problem. Additionally, using the WMPR problem as a DSML application context, we will discuss intuitions and implementations of our DSML data summarization tools.

2.3 Existing Workflow

Note that DSML applied to D & T usually aims for an existing task context. In a company, this task is accomplished with an existing workflow or methodology which can be quite manual. However, because the existence of such a workflow, it is important to keep in mind that a DSML solution can only receive wide adoption if it provides a clear added value in view of the current workflow. Therefore, a second postulate for DSML can be included to emphasize this baseline for considering the added value [52].

> **Postulate 2:** For a domain-specific machine learning problem, there exists a non-machine-learning approach for solving the same learning problem. Consequently, failure to solve the problem with a machine learning approach is not catastrophic.

Postulate 2 states that DSML is not necessary for performing the target task. Because DSML is not necessary, its added value then needs to be justified. For example, if DSML produces a piece of information from the data that is already well known to the user, then this result has no value. In contrast, if the information presents a surprise, then it is valuable. In this sense, DSML might be seen as *learning to surprise its user*. This "surpise" aspect needs to be taken into account in our data summarization view of DSML, i.e. data which tend to be more valuable to the user should be presented first.

Postulate 2 can have another implication. As a DSML technology provider, we aim

to provide a DSML software system to be used in performing a target task. However, we cannot assume we would know all the detail regarding the existing workflow or have direct access to all the data sources internal to the company. This is because such detail and data source access might be confidential. In this regard, we can get certain functionality requirements to facilitate the development of the DSML solution, but it would be very hard for a DSML technology provider to invent a new workflow intended to replace the existing one. This is a crucial limitation to consider. The consequence is that, we need to identify places in an existing workflow where DSML can provide added value. If such a place does not exist, then DSML is not suitable for the particular task context.

Chapter 3

The Principle of Parsimony

> *"The first principle is that you must not fool yourself and you are the easiest person to fool."*
>
> — Richard P. Feynman

For all the DSML problems discussed before, we can view the essence of the problem as playing a guessing game. Figure 3.1 depicts this view.

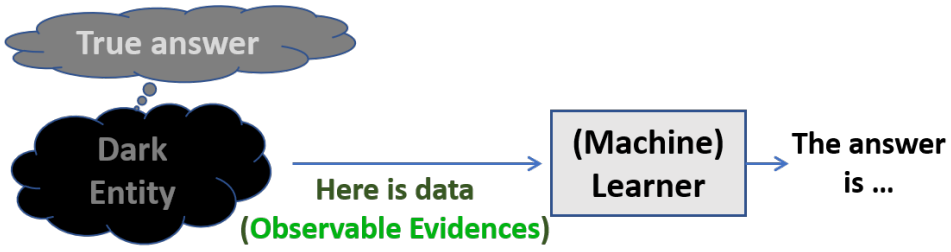

Figure 3.1: DSML problem as a guessing game

In the figure, the "dark entity" represents the target we try to improve on. There is a "true answer" associated with the dark entity. Observable evidences are revealed to the learner about this true answer. The job of the learner is to guess correctly what the answer is.

In this picture, the learner can simply be an engineer. The engineer does possess some "domain knowledge" regarding the dark entity. The domain knowledge helps the engineer in playing the game. If we replace the leaner entirely with a machine, this is one way to see what "applying ML" means. In this chapter, we will discuss the challenges associated with taking this view.

3.1 Occam's Razor in ML

We can look at how the guessing game can be played with a traditional ML approach. To play the game, the learner starts with an assumption of a hypothesis space containing all possible answers. Note that the number of the answers can be infinite or even not enumerable, depending on the assumption. Regardless of what kind of assumed hypothesis space, the important aspect is that, there is such an assumption.

It is also important to emphasize that here we really refer to the *effective hypothesis space* in view of the three components depicted in Figure 2.1 before. The point is that, even a hypothesis space contains an answer h, if the complexity for reaching this answer h is too high for a learning algorithm to pick the answer, then effectively this answer does not exist in the hypothesis space. Hence, an effective hypothesis space depends on the learning algorithm.

With a hypothesis space, we can essentially look at learning as following two steps: (1) The first step is to use the data samples to filter out all inconsistent answers. (2) For all the remaining hypotheses, find the *simplest* one as our answer. This means that a ML algorithm usually tries to minimize the objective with a form like the following:

$$(\text{Fitting Error}) + \alpha(\text{Model Complexity}) \tag{3.1}$$

For example, if our aim is to filter out *all* inconsistent hypotheses, then we require the fitting error to be zero. Under that objective, we also minimize the model complexity in our answer, i.e. picking the simplest one. In an actual implementation, tradeoff between these two objectives is controlled by a user-defined parameter called α.

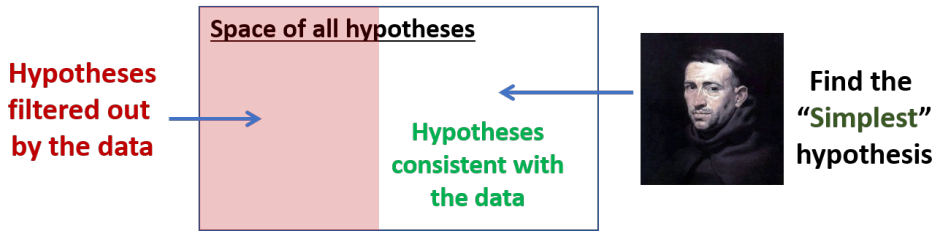

Figure 3.2: Learning as "filtering" + Occam's razor

Figure 3.2 illustrates in essence what the learning intends to do. Of course, in a practical implement when an objective function like equation (3.1) is employed, it is not exactly doing what is shown in the figure. However, conceptually we can still use the picture to illustrate its fundamental issues.

3.2 Occam's Razor in Learning

Occam's razor, in many scientific fields is also known as the Principle of Parsimony. Occam's razor states "Entities should not be multiplied unnecessarily" [55]. From the surface, it seems that the Occam's Razor is saying that "the simplest answer is the best". This idea is certainly behind the thinking depicted in Figure 3.2 above. It should be noted that this thinking essentially treats Occam's Razor as the first principle in learning. When two answers both can fit the data, the principle gives preference to the simpler one without the need of a justification.

However, Occam's Razor does not need to be taken as a first principle. It can be justified in view of Bayesian inference. In this view, we are given two hypothesis space

$\mathcal{H}_1$ and $\mathcal{H}_2$ and are asked to decide which hypothesis space is more likely based on the given data D. Using Bayes' theorem, we can evaluate the probabilities of the two hypotheses by considering their probability ratio as the following:

$$\frac{P(\mathcal{H}_1|D)}{P(\mathcal{H}_2|D)} = \frac{P(\mathcal{H}_1)}{P(\mathcal{H}_2)} \frac{P(D|\mathcal{H}_1)}{P(D|\mathcal{H}_2)} \tag{3.2}$$

Note that $P(\mathcal{H}_1)$ and $P(\mathcal{H}_2)$ are our prior probability assumption toward the two hypothesis spaces. Assuming they are equal, then the comparison is based on the two posterior probabilities, $P(D|\mathcal{H}_1)$ and $P(D|\mathcal{H}_2)$. Assuming that in each of the hypothesis spaces, there is exactly one hypothesis that fits the data. Further assume that $\mathcal{H}_1$ is simpler than $\mathcal{H}_2$, which can be characterized in their "size". For example, we have $|\mathcal{H}_1| = N_1$ and $|\mathcal{H}_2| = N_2$ where $N_1 < N_2$. Then, simply we have $P(D|\mathcal{H}_1) = \frac{1}{N_1}$ and $P(D|\mathcal{H}_2) = \frac{1}{N_2}$, where $\frac{1}{N_1} > \frac{1}{N_2}$. In other words, the answer from simpler hypothesis space is more likely to be the answer.

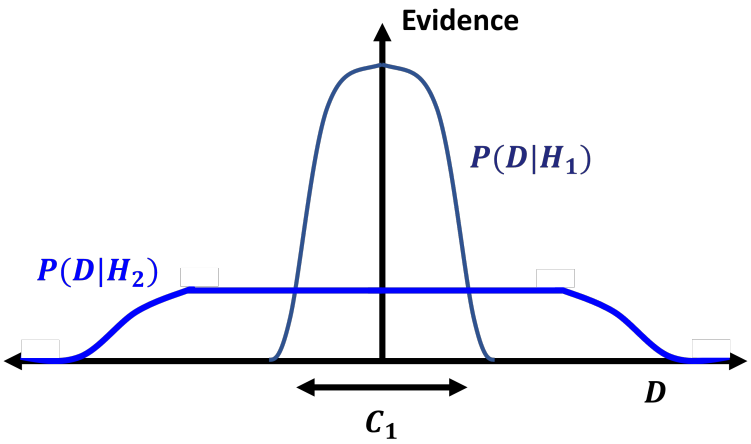

Figure 3.3: Normalized probability distribution of models $\mathcal{H}_1$ & $\mathcal{H}_2$ on dataset D. Figure adopted from [56].

This concept can be illustrated in Figure 3.3, where the normalized probability distributions of predictions on D by $\mathcal{H}_1$ and $\mathcal{H}_2$ are visualized. $\mathcal{H}_2$ is a more complex (i.e. has more free parameters) model than $\mathcal{H}_1$, thus it can make predictions that are spread

out across the space of data samples D. Whereas $\mathcal{H}_1$ is a less powerful model, which results in predictions that are concentrated around the evidence provided by the data. If the dataset to be predicted falls in the range indicated by c_1, then the simpler $\mathcal{H}_1$ will be a more probable model as it makes stronger predictions compared to $\mathcal{H}_2$.

In view of Bayesian inference, Occam's Razor makes sense. However, in this view, Occam's Razor is used to pick a hypothesis space rather than to pick an answer from a fixed hypothesis space. In DSML, it seems that taking the Bayesian inference view to interpret Occam's Razor is more appropriate. This is because, as mentioned before in view of the three components Figure 1.15, DSML is more about concerns on the hypothesis space assumption and the data generation, and less about the learning algorithm. If our focus was on the algorithm (with a fixed hypothesis space assumption to begin with), then taking Occam's Razor as the first principle facilitates implementation of the algorithm, i.e. equation (3.1). However, if our goal is to search for a hypothesis space assumption, then Occan's Razor should not be taken as the first principle. It becomes a principle justified by Bayesian inference.

3.3 Local No-Free-Lunch

As discussed in 2.2, from one iteration to the next, DSML is facing a local No-Free-Lunch problem. In the following, we will use a simple example to illustrate that if Occam's Razor is taken as the first principle, it cannot help solve the problem.

For the two learning contexts discussed in Section 1.2.1, we can consider the learning as solving a Monomial Learning problem. In essence, we are given a set of feature $f_1, \ldots, f_n$ which are Boolean variables. From the data we are interested in learning what the underlying cause is, represented as a monomial based on the feature variables, i.e. a combination of features.

As a simple example, suppose we are given a dataset with three feature f_1, f_2, and f_3 which looks like the following:

f_1	f_2	f_3	$label$
1	1	1	1
1	0	1	1

Table 3.1: An example dataset for learning a monomial

The dataset implies three possible answers: f_1, f_3, and $f_1 f_3$. Without further data, we would not know which one is the cause. For example, if we get to see one future sample as depicted in the left in Table 3.2, then we know that the answer must include f_1. However, at this point we still do not know if the answer is f_1 or $f_1 f_3$.

Answer includes f_1

f_1	f_2	f_3	$label$
1	1	1	1
1	0	1	1
0	0	1	0

Answer is $f_1 f_3$

f_1	f_2	f_3	$label$
1	1	1	1
1	0	1	1
0	0	1	0
1	0	0	0

Table 3.2: Future samples reveal the true answer

The true answer is revealed only we see an additional sample as depicted in the right in Table 3.2. At this point, we know the answer is $f_1 f_3$.

The local No-Free-Lunch situation takes place when we are in Table 3.1 and in the left of Table 3.2. In the first case, we are left with three possible answers. In this case, even applying Occam's Razor as the first principle would not save us because we still would not know how to choose between f_1 and f_3. In the second case, applying Occam's Razor as the first principle would provide us with a clear choice. We would choose f_1 as the answer because it is the simpler one between the two choices. However, there is no justification for the answer other than we believe in the Occam's Razor principle. In fact, when we see the last sample in the right of Table 3.2, we would realize that this belief

was misleading. The simple example illustrates the risk of treating Occam's Razor as the first principle in DSML. One may argue that providing f_1 before seeing the true answer can still offer "half" the value to the user. However, as a DSML technology provider we would not really know this, because how the user perceives and utilizes the answer depends on the task's context and user's domain knowledge.

3.4 Occam's Razor in DSML

The above discussion explains the risk of taking Occam's Razor as the first principle in DSML. The implication is that, in DSML we would not apply Occam's Razor to pick an answer for the user when multiple answers (from a hypothesis space) are all suitable for explaining the data. In other words, we would reserve from make a decision for the user when such a decision is not completely justifiable from the data.

On the other hand, Occam's Razor can still be a guided principle to select in between hypothesis space assumptions, where the selection is justifiable in view of Bayesian inference. Practically though, this is hard to implement. To realize such a scheme, we require the user, as a starting point, to provide a set of hypothesis space assumptions where they can be somehow ranked according to some complexity measure. This can put too much burden on the user. However, with this thinking, it seems that the underlying problem is actually in obtaining a manageable set of initial assumptions. As a DSML software provider, it seems that we would like to provide a software system to facilitate attaining a manageable set of assumptions for the user. Then, the user can pick an hypothesis space assumption that is more suitable in view of their perspective driven by their domain knowledge. In the rest of the thesis, we will present a sequence of works leading to the realization of such a software system.

Chapter 4

Law of the Instrument

> *"Do not try bending the spoon. That is not possible. Only try to realize the truth: There is no spoon. Then you will see. It is not the spoon that is bending. It's only yourself."*
>
> — The Matrix, 1999

To recap, the previous chapters have laid down key considerations for developing a DSML solution. It is important to recognize that in view of Local No-Free-Lunch and the lack of sufficient domain knowledge from the perspective of a DSML technology provider, we should not try to apply ML aiming to build a model that makes decisions for the user. Instead, the solution should aim to facilitate the user to make those decisions themselves. In other words, the solution should play the role of an assistant to the user rather than a decision maker (to replace them).

In this chapter, we will be providing a concrete example to illustrate the key considerations in DSML. For the remainder of this thesis, we will be using the **Wafer Map Pattern Recognition (WMPR)** problem as our DSML application context.

Before we dive into the WMPR problem, as the title of this chapter suggests, there

is an interesting phenomenon to be discussed, known as the *Law of the Instrument*. To quote the American psychologist Abraham Maslow in 1966, "I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail." [57]. Simply put, *Law of the Instrument* is a cognitive bias where when we discover a new tool or acquire a new skill, we tend to see opportunities to use it as much as we can. Sometimes, we could be biased by our desire to utilize the new tool/skill and overlook if it is actually suitable for a given scenario. In a sense, this is not surprising, as everyone would want to formulate their problems in ways that leverage their well-developed techniques.

With the success of general ML techniques, this phenomenon is also prevalent in the WMPR problem. Although we have already defined many aspects of DSML which do not align with general ML, courtesy of *Law of the Instrument*, it is not surprising that many works had tried to apply general ML techniques in the context of WMPR. In the following sections, we will introduce the WMPR problem and review these works.

4.1 Wafer Map Pattern Recognition

In the semiconductor testing process, each die gets tested across multiple stages, under various conditions, such as temperature. A wafer map is a composite image of these tested dies, usually visualizing the test results of dies from the same wafer. Each die location on the image is marked with a value, such as a binary value indicating pass/fail, or a numerical value indicating a parametric measurement.

Analysis of wafer maps is commonly practiced for investigating potential yield issues, as visual patterns can shed probable insights to problematic manufacturing equipment. Specifically, when presented with the visual failure patterns, experienced process engineers can identify the cause of such failures. Due to the extremely large scale production and the increasingly more complex technology nodes, it is desirable to automate the anal-

ysis of wafer maps. It should be noted that the need to automate WMPR originally came from semiconductor manufacturers. In the past, yield was an issue largely concerned by the manufacturers. If there was a yield issue, the responsibility largely fell on the manufacturers to fix it. A fabless company only needed to report the issue, which could be simply based on observing the yield. In recent years, fabless companies became more interested in WMPR because they would like to pursue more effective communications with the manufacturers. In other words, fabless companies can help improve the yield more effectively if they can feedback more insightful yield information.

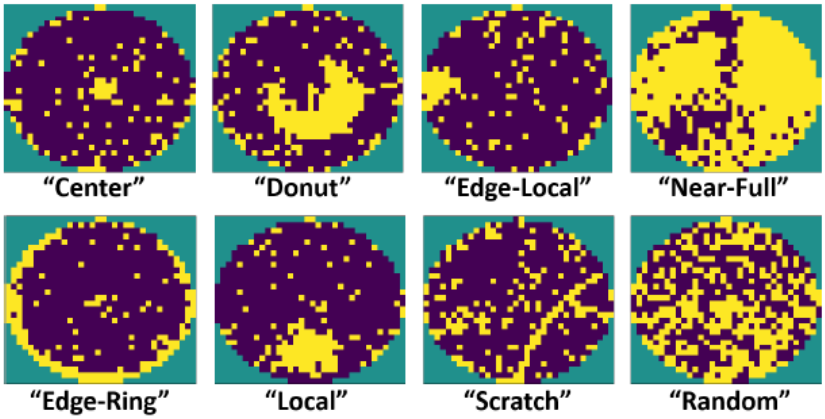

Figure 4.1: Wafer map examples from the 8 labeled classes in the WM-811K dataset. Yellow dots indicate failed dies.

With the goal to automate WMPR, in 2015, the authors in [58] developed and published a comprehensive wafer map dataset called WM-811K. As its name suggests, this dataset contains 811,457 binary pass/fail wafer maps fabricated by TSMC, where 172,950 of these were labeled into 9 classes. Figure 4.1 illustrates the 8 pattern classes (where yellow dots indicate failed dies). In addition, the 9^{th} class is the "None" class, used to denote wafer maps containing no patterns.

Following the publication of this dataset, many ML focused works emerged, where they could be categorized into two common ML paradigms. As shown in Figure 4.2, in the Traditional ML paradigm, feature extraction is performed by the expert, where

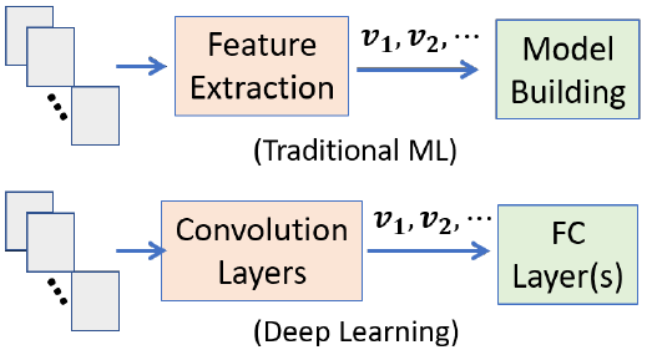

Figure 4.2: Traditional ML approach versus Deep Learning approach for wafer map pattern recognition.

the ML portion stresses on the model building aspect. In the Deep Learning paradigm, neural networks, e.g. CNNs, are used to perform automatic feature extraction, as part of the end-to-end classification model.

4.2 Traditional Machine Learning

Accompanying the release of their WM-811K dataset, the authors in [58] developed a methodology for wafer map pattern recognition and similarity ranking. The work employed several types of expertly-crafted wafer map features, including features based on Radon transform [59] and geometric properties of failure patterns (e.g. failing die counts, region labeling, line detection, etc.). Then, these expertly-crafted features were fed into SVM [17] for the model building step. This work reported an overall 94.63% accuracy on the labeled dataset, compared to the deep learning approach at the time which achieved 89.64% accuracy.

Similarly, the work in [60] followed the same methodology approach as above, but with a slightly different feature set, aimed at improving multi-pattern detection accuracy. Another work in [61] advocated using more discriminant features based on Linear Discriminant Analysis (LDA) to simplify the model building step. With Radon transform

based features, the work in [62] proposed a special Decision Tree based ensemble learning method, as Decision Tree models are generally more interpretable than SVM models.

4.3 Deep Learning

The work in [63] transitions into the deep learning view, where they proposed using an Autoencoder (AE) [64] as a pre-training step to learn features before leaning a classifier. However, the authors found that expertly-crafted features were still needed, thus these feature vectors were fed into the AE, as opposed to the wafer map images themselves. Later in work [65], the authors proposed a 2-stage classification process: first to classify between having a pattern and having no pattern (i.e. "None" in the WM-811K dataset). If there is a pattern, then proceed with the classification and output one of the 8 pattern classes. Both authors of [63] and [65] acknowledged that the wafer map images were noisy and a de-noising step was required to make the learning work.

With the aim of addressing the extreme imbalanced between classes in the WM-811K dataset, the work in [66] & [67] proposed special data augmentation methods based on Generative Adversarial Networks (GANs) [31] and Autoencoders respectively. Moreover, the authors in [67] found that augmenting samples with rotation could help the data enhancement effort. In contrast, the authors in [68] did not rely on NNs to augment data samples, instead they used pre-determined methods and applied a deeper CNN for training the classifier.

The authors in [69] approached the WM-811K dataset from a different angle. They tried to address the concern that a wafer map to be predicted might contain a new pattern not seen in the training dataset, which includes multi-pattern failures. As a result, the classifier's prediction might not be reliable. The work proposed using Selective Learning to estimate confidence of a prediction. Then, the deep learning model could include the

choice to abstain from making a prediction.

4.4 Issues

The WM-811K dataset is valuable from an academic research point of view. It enables many studies of various modern ML techniques and their applicability to the dataset. However, as discussed previously, the core issues in DSML are not in the learning algorithm, but in other components in a learning setup. In this section, we illustrates what those issues are. Consider how WMPR is solved in all the prior works. They essentially follow a flow like the following:

1. Define a set of pattern classes.
2. Collect a set of training samples and a set of test samples.
3. Search for the best multi-class classifier.
4. Determine if the accuracy is acceptable for deploying the classifier.

First of all, step 1 takes the position that solving the WMPR problem should start with a definition of the pattern classes, e.g. there are 8 classes of patterns instead of 12 classes. This is a strong assumption to begin with, assuming that someone has the full knowledge about what is happening in the manufacturing process and what will happen in the future. This someone therefore has the knowledge to provide a good definition of the classes.

Then, step 2 takes the position that enough samples can be collected to enable training a sufficiently good multi-class classifier. This means perhaps that they are not aiming at applications in the early stages of a production. Although, in practice, it is in the early stages where the yield is the top concern.

If one accepts the first two positions, then the next step is to search for the best classifier. The assumption here is that the test set can provide a reliable assessment of the goodness of a classifier, in view of its actual application in a production. This raises the question on the quality of the test set.

The last step assumes that a domain expert would know how to judge a classifier's accuracy value in terms of its acceptance for deployment. For example, such an expert understands what 97% accuracy means and would consider this number enough for deployment.

As seen, treating WMPR as solving a multi-class classification problem can raise many concerns to its practicality. In fact, in the WM-811K dataset, we can demonstrate such a concern, where there are samples whose labels are quite questionable. This is illustrated in Figure 4.3. The label ambiguity is exhibited in three pairs of samples where each pair of samples have a similar pattern but are labeled differently.

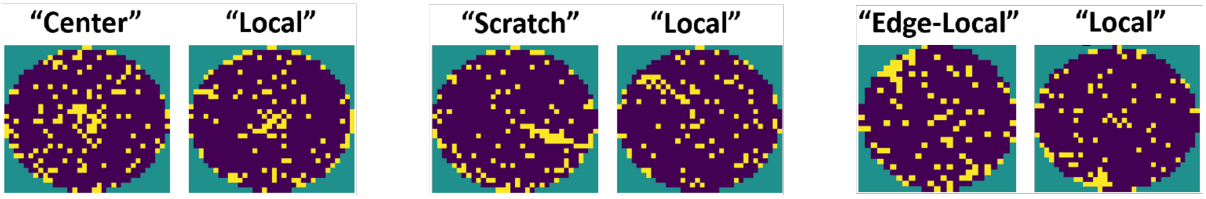

Figure 4.3: 3 example pairs of ambiguously labeled samples from the WM-811K dataset.

Figure 4.4 illustrates another issue with the labeling. Each of the three wafer maps has a pattern that is somewhat unique, comparing to other wafer maps in the same class. Hence, if they are not included in the training set but in the test test, a classifier is likely to misclassify them. These examples show that one can assign the same label to two wafer maps, but that does not mean these two wafer maps must have a similar pattern. The assignment is inherently subjective. For example, two engineers can disagree on the label for a given wafer.

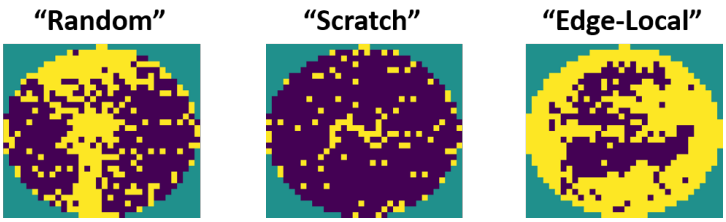

Figure 4.4: 3 examples of unique pattern

4.5 A DSML View

If we consider a scenario where WMPR is utilized in the early phase of a production, then we cannot assume that (1) there is enough data to train a classifier, and (2) we have seen all possible patterns. If we take a step back, then determining a classification of the patterns is part of the problem. In other words, given a sequence of wafer maps $w_1, \ldots, w_n$, our first goal should be to determine how many classes of patterns are in this sequence of wafer maps.

It is important to also realize that there might not be a golden answer to this definition of classes. Different engineers might consider different definitions in their favor. Even for the same engineer, the perception to this golden definition might change over time, after learning more about the production process.

Instead of a traditional flow to see the WMPR problem as shown in the previous section, in view of DSML, we can list a set of different requirements as the following:

1. Defining a set of pattern classes is part of the WMPR problem.
2. There is no golden answer to the definition.
3. If two sets of classes are presented to a user as two options, the user can judge which is the preferred one.

The first requirement essentially treats WMPR as an unsupervised learning problem. The second requirement says the problem cannot be seen as an optimization problem

— there is no well-defined optimization objective. The third requirement puts the final decision of a classification definition at the hand of the user. In other words, a DSML solution can present different options for the user to choose. If these options are presented in an easy way for the user to see (e.g. in a single screen), then it should not be too hard for the user to make a decision to choose the best definition.

Without a well-defined optimization objective, we need to state an end goal for the DSML solution to satisfy. We can assume two such goals in view of an iterative learning process. In each iteration, a batch of wafers are given. An engineer aims to attain the answers for two questions as their end goals:

1. Is there a pattern that represents a potential yield issue?

2. If yes, is the pattern new or has it occurred previously?

It is important to note that, a DSML solution cannot provide a definite answer to the first question if the pattern is new. This is because determining whether a new pattern represents a potential yield issue or not, can require domain knowledge beyond what is learnable from the data. Hence, only the user can make the final call to the question. Although, a DSML solution can provide recommendations.

Regarding the second question, it is essentially a similar-pattern search problem. However, the challenge here can be that we are given only one wafer map to search on, and in the previous iterations, the pattern appeared only on one wafer map. In this worst-case scenario, we are required to enable the search based on only one given sample.

4.6 The No-Pattern Class

Suppose we have a way to build a recognizer R for a given pattern P. There is a subtlety in what we are asking R to do. In one case, R would take a wafer map w, where

we know w contains a pattern, and determine if w's pattern is of type R. In the other case, R would take a wafer map w and determine whether w has a pattern. Then, if w has a pattern, whether the pattern is of type R. Building a recognizer in the second case is obviously harder.

Indeed, in an actual production, there can be many wafer maps without a pattern of interest. They should be put into a "no pattern" class. This class is special. In view of image classification, this is almost saying that we should have a class where the image is not classifiable (e.g. because there is no interesting object in the image). This means that WMPR includes not only the traditional classification aspect but also the aspect of "no classification needed". This is analogous to the application contexts discussed above where DSML is required to include the capability to say "no model found". For example, it would be ideal when a DSML solution is asked to learn a recognizer for a given wafer map that has no pattern, the DSML solution can decide that there is no pattern to be learned on and output "no model found".

In fact, the no-pattern class is well represented in the WM-811K dataset. For example, Table 4.1 shows the sample count and percentages of each labeled class for these 23,802 wafers from the largest 2 wafer sizes in the dataset. As seen, more than 95% of the wafer maps are labeled as no-pattern, i.e. the "None" class.

Center	Donut	Edge-L	Edge-R	Loc	N-Full	Random	Scratch	None
81	10	402	9	345	16	28	76	22115
0.351%	0.043%	1.742%	0.039%	1.495%	0.069%	0.121%	0.329%	95.811%

Table 4.1: Labeled wafer maps distribution from the WM-811K dataset (2 largest wafer map sizes).

Prior works reviewed above treated the no-pattern class effectively as another class. With this view, they took the problem as a dataset imbalance problem. Then, the motivation to employ data augmentation in some prior works reviewed in Section 4.3

becomes understandable. It is important, though, to recognize that doing so assumes that classification into no-pattern is the same problem as classification into various patterns. This assumption is not justified in any of the prior works.

4.7 Summary

In this chapter, we use WMPR as an example application context to illustrate the differences between taking a ML view to the problem and taking a DSML view to the problem. We review those prior works following a traditional ML approach as well as a deep learning approach. We bring up *Law of the Instrument* in the review, not to devalue the prior works, but to emphasize other practical considerations overlooked in the past research. Those considerations are important for developing a useful DSML solution. We emphasize the risk of being biased by *Law of the Instrument* because in the industry, it is often tempting to consider DSML as simply "applying existing ML techniques to analyze domain-specific data". This simple view can often lead to quick result, but rarely to a practical DSML solution. This is because a practical DSML solution needs to address issues other than the learning algorithm component in Figure 1.15. As it will be shown later, after considering all the requirements listed previously in Section 4.5, we eventually reached a conclusion that no existing ML technique is suitable to meet our needs, thus we had to develop a novel learning technique to enable a practical DSML solution.

Chapter 5

Structural Risk Minimization

"In so far as a scientific statement speaks about reality, it must be falsifiable; and in so far as it is not falsifiable, it does not speak about reality."

— Karl Popper

In Chapter 3, we discuss Occam's Razor which is commonly taken as the first principle in an ML algorithm. In DSML, the concern is less about the ML algorithm, but more about searching for a hypothesis space assumption in view of a data generator that can change from one iteration to the next in an iterative process. When our focus is on the hypothesis space assumption, Occam's Razor can be a principle justified by Bayesian inference. However, while conceptually Occam's Razor still makes sense, practically it can be challenging to implement this principle for the search. This is because the starting point asks for a set of hypothesis space assumptions. In DSML, these assumptions need to be made explicit so that the evaluation result can be understandable by a user in order for them to take an action (see Figure 2.1).

In this chapter, we will use WMPR as the application example again to illustrate the problem of searching for a hypothesis space assumption. In particular, we will exam-

ple the idea of Structural Risk Minimization (SRM) [17] which is a theoretical scheme developed to optimize the process of learning a model. SRM was introduced in a 1974 paper by Vladimir Vapnik and Alexey Chervonenkis (VC). Associated with the scheme is the concept called VC Dimension which is used as a measure for the complexity of a hypothesis space assumption. The SRM scheme can be interpreted as following the Occam's razor principle under certain assumptions.

First, SRM states that there exists a "structure" of smallest (i.e. simplest) to largest (i.e. most complex) hypothesis space for a class of fitting functions (e.g. polynomial functions). This statement required a measure of modeling complexity for a function, which was also developed by the same authors, known as VC Dimension. Second, SRM states that there exists a way to homogeneously grow from a smaller hypothesis space to the largest one (e.g. increasing the degree of polynomial functions), where each smaller space is a subset of a bigger space. In other words, the subset hypothesis space has a smaller VC Dimension compared to its immediate larger space. Under these assumptions, a linearly search can be performed to find a first-fit (i.e. simplest & fitting) hypothesis starting from the smallest hypothesis space, extending the search to increasing more complex ones as needed.

Figure 5.1 demonstrates the learning setup proposed by SRM. In the figure, S_1 is the simplest hypothesis space, where it can produce a hypothesis h_1 such that the empirical risk (i.e. error on the training data) is high, thus is considered "underfitting". In contrast, S_N is a more complex hypothesis space, where it can produce a hypothesis h_n such that the empirical risk is very low but the complexity of h_n is very high, thus it is considered "overfitting". The ideal "fitting" hypothesis space, S_T, can produce a hypothesis h_t such that the generalization error is the lowest. By following SRM's hypothesis spaces structure, this ensures the arrival of an effective hypothesis space, S_T, the first time when enough samples are seen. In summary, the SRM principle follows Occam's razor for a

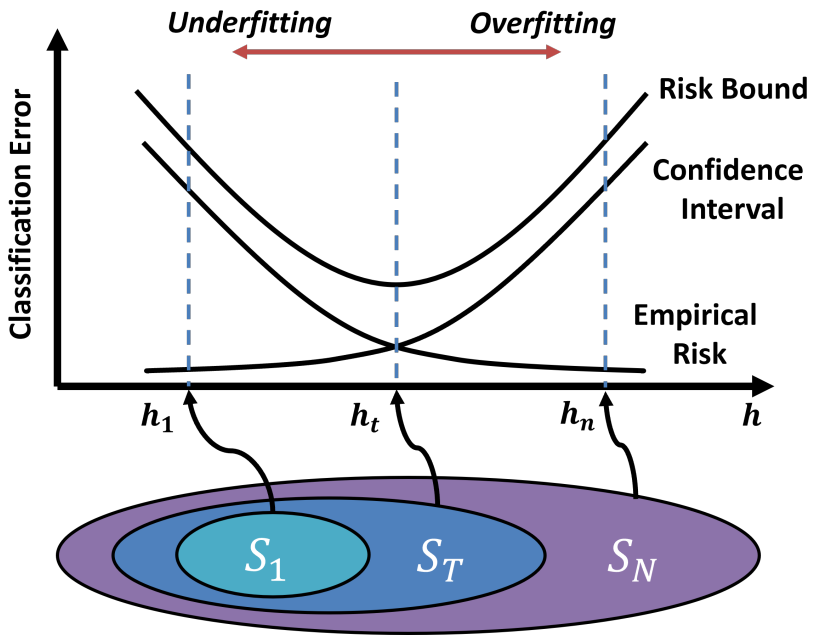

Figure 5.1: A visualization of the homogeneously growing hypothesis space proposed by SRM. Illustration adopted from [70].

class of fitting functions (i.e. a definition of a set of hypothesis spaces) by stressing the importance of selecting models from a structure of increasing complexity.

5.1 Applying SRM in ML

Although SRM provides an outline of how to performing model selection for better generalization, it is difficult to implement strictly in practice. For most practical methods using nonlinear approximating functions, such as various types of neural networks, calculating the VC Dimension analytically or estimating the effective VC Dimension are both difficult. Thus, it can be difficult to explicitly define and search the structure of increasingly complex hypothesis spaces. Instead, for practical model selection, a mixture of two objectives, the fitting error and the model complexity, are minimized together, as previously illustrated in equation (3.1). For example, given free model parameters $p_1, \ldots, p_n$, the complexity measure of the model can be formulated as the L_2 norm $\sum_{i=1}^{n}(p_i)^2$.

The strategy is commonly employed in supervised training, such as training a neural network (NN). First, it is assumed that the NN's capacity (e.g. number of trainable parameters) is proportional to the complexity of the hypothesis space. To ensure a large enough formulation of hypotheses, a NN is usually specified with the highest capacity you can afford in terms of training. Then using large amounts of training samples, gradient-based optimizers adjust the NN's parameters to assign a fitting model. As a co-objective, a complexity measure (e.g. L_2 Norm of weights) for the current NN is used to guide (i.e. *regularize*) the model selection process to encourage smaller "effective capacity" (e.g. non-zero weights) models. In this example, the fitting error captures the differences between training labels and predictions, the complexity measure is the estimated complexity of the NN model based on its current trainable parameters, and α in equation (3.1) is the coefficient to control the extent of regularization.

5.2 Unsupervised Learning in WMPR

In general, the training objective in equation (3.1) can be used for all learning tasks when there is a suitable way to define and calculate the fitting error. When the definition of fitting error is not certain, this uncertainty presents another level of challenge for implementing SRM.

For the WMPR problem, the learning objectives are inherently unsupervised, as explained in Section 4 where several requirements are stated for solving the problem in view of DSML. The first requirement for a DSML solution is to have the ability that helps a user discover a set of pattern classes in a given batch of wafer maps.

Consider a scenario where N wafer maps are given. A naive way to think about the complexity of a hypothesis space is to use the number of classes to define the complexity. For example, without doing anything, every wafer map is in its own class. Hence, we

have N classes. Similarly, without doing everything, we can say all wafer maps are in the same class, leading to only one class. A tool can be designed to ask a user to supply a pre-defined number of classes k and the algorithm finds what it thinks is the best partitioning of wafer maps to fit the k-class assumption. This is the idea behind the popular k-means clustering. This clustering is largely criticised by practitioners because it is not easy to choose the right k. Often, a practitioner tries a few selections of k and find none of the results as desirable. Then, the practitioner gets stuck.

Other more sophisticated algorithms try to improve k-means by removing the need to ask the user to choose k in advance (by replacing it with some other parameter such as choosing the minimal number of samples in a cluster, etc. which might be more intuitive for a user to decide). In view of DSML, this does not solve the problem. The issue is that, if a clustering result is not desirable to a user, when facing a more complicated algorithm, it would be harder for the user to know how to fix it.

5.3 Lower-Bound and Upper-Bound Clustering

Ideally, we would like to adopt SRM in our clustering approach to meet the first requirement in our DSML solution for WMPR. Because we have to ensure the clustering result makes sense to the user, we have to make the clustering process *transparent* to the user. Because whether the result makes sense to the user or not depends on the user's subjective perspective, we have to find a way to take that perspective into account. While accomplishing all these goals, we also must have a solution that is easy to use and provides added value to the user. With all these considerations from a DSML perspective in mind, we then need to articulate the requirements for the underlying learning algorithm.

Given N wafer maps, as mentioned above, the two extreme partitions are treating them as N classes and treating them as one class. We can call the first as a *lower-bound*

clustering and the second as a *upper-bound* clustering. It is trivial to say the user desires a clustering that is between these two. And if our DSML solution does not do anything further (other than just presenting the wafer maps to the user), we are basically asking the user to find the clusters themselves. While this thinking is trivial, it can be taken as our starting point to further think about what we want our DSML solution to do.

Suppose we have an algorithm L to determine that given two wafer maps, they must belong to the same pattern class and it is very unlikely for a user to argue about this decision. Then, we can apply L to improve our lower-bound clustering. Similarly, suppose have an algorithm U to determine that given two wafer maps, they cannot be in the same pattern class and it is very unlikely for a user to argue about that decision. Then, we can apply U to improve our upper-bound clustering. Finally, suppose we have an algorithm F that can identify wafer maps and determine that they have no pattern. Similarly, it is very unlikely for a user to argue about those decisions. Then, we can simplify the clustering problem up front. With these three algorithms, we can have a DSML solution that dramatically reduces the complexity of the search for an acceptable pattern classification. Once the search space is largely reduced, the solution can present the result and make recommendation in a summary window to a user. The user then can decide what the final pattern classification should be.

5.4 In View Of SRM

The strategy to find a better lower-bound and a better upper-bound clustering can be viewed from the SRM perspective. In view of SRM, there is a target hypothesis space whose complexity is the smallest that can perfectly fit the data. As mentioned before, in WMPR we do not know what this "perfectly fit" means for a learning algorithm. We only know that it means "perfectly fit from a user's perspective". Hence, we cannot use

an algorithm to find the target hypothesis space. This target hypothesis space remains in the user's mind, and our goal for the DSML solution is to help the user to *articulate it*. In this articulation process, the solution can help to make it easier for the user, i.e. by reducing the search space. In a sense, the solution enables the user to approximate an SRM-driven learning by themselves. Figure 5.1 illustrates this "Approximate-SRM" thinking by showing 3 hypothesis spaces, the lower-bound hypothesis space S_L, the target hypothesis space S_T, and the upper-bound hypothesis space S_U.

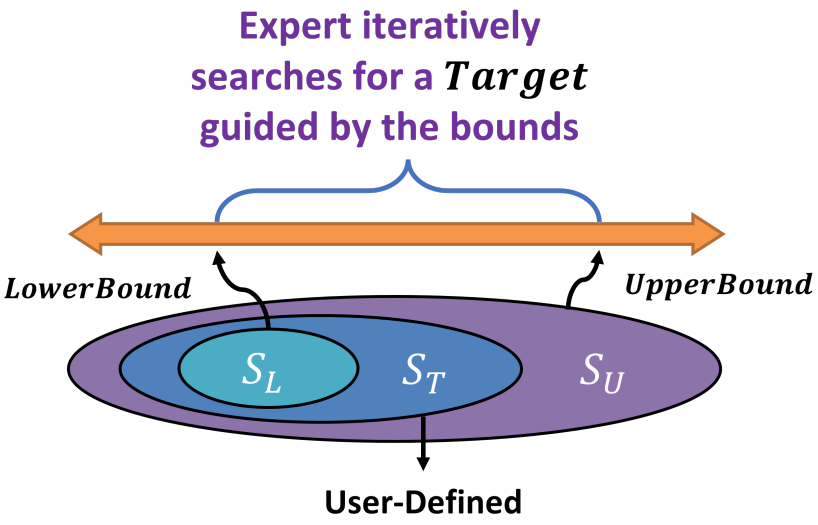

Figure 5.2: Approximating SRM learning by defining a lower-bound and upper-bound hypothesis space, facilitating an expert's decision of a target hypothesis space.

Behaving similarly to the example hypothesis space S_1 shown in Figure 5.1, our goal is to provide a lower-bound S_L. S_L is considered as a lower-bound because no two wafer maps that have been put into the same pattern class will be separated in future analysis by the DSML solution. In this lower-bound space, the pattern classes on many wafer maps can still be undecided. That is why it is a hypothesis space with many possible answers. Itself is not an answer.

Similarly, S_U is considered as a upper-bound because no two wafer maps that have not been put into the same pattern class shall be in the same class in future analysis by

the DSML solution. In this upper-bound space, the pattern classes on many wafer maps can still be changed, i.e. a wafer map can be removed from a class. That is why it is a hypothesis space with many possible answers. Itself is not an answer.

The target S_T stays in between S_L and S_U. Unlike SRM which relies a learning algorithm to decide S_T, in DSML, this decision should ultimately be left to the user. In this search, the role of DSML is to *assist* the user and not to make a final decision for them.

As summarized by Figure 5.2, by following the specified 3 hypothesis spaces structure, we can aim to enable SRM-like learning for a user. The structure above enables a user to reach a pattern class definition that is understandable and acceptable to them. To implement such a search scheme, we need novel methods to attain four capabilities in our DSML solution:

1. The capability to obtain a lower-bound model
2. The capability to obtain an upper-bound model
3. The capability to facilitate the search for the target model
4. The capability to filter out wafer maps with no pattern

Note that these four capabilities are only for meeting the first DSML requirement discussed in Section 4.5. There are other requirements to be considered. In the following chapters, we will discuss various methods developed in our research in view of these capabilities and requirements.

Chapter 6

Lower-Bound Model as a Checker

"Science without religion is lame, religion without science is blind."

— Albert Einstein

In this chapter, we focus on two capabilities raised in Section 5.4 for our WMPR DSML solution. The first capability is to derive a lower-bound model from a WMPR dataset, and the second is to filter out wafer maps with no patterns. In the following sections, we will discuss implementations of these two capabilities. With these two capabilities fulfilled, we attempt to deploy an initial DSML software solution to observe their effects. Specifically, we show that our lower-bound model can work as a checker for our deployed WMPR software.

6.1 Lower-Bound Model: Tensor-Based Methods

With the goal of attaining a lower-bound clustering model, we developed recognition methods based on properties of tensor decomposition (i.e. tensor-based), specifically Tucker Decomposition [71][72].

6.1.1 Tucker Decomposition

In essence, Tucker Decomposition is the high-dimensional generalization of matrix Singular Value Decomposition (SVD) [73]. In SVD, a given matrix $X_{n\times m}$ is decomposed into $U_{n\times r}\Sigma_{r\times r}(V^T)_{r\times m}$, where $\Sigma_{r\times r}$ is a diagonal matrix. Similarly in Tucker decomposition, a 3-dimensional tensor $\mathcal{A}$ of size $R_1 \times R_2 \times R_3$ can be decomposed into $3+1$ components: a core tensor $\mathcal{G}$ of size $r_1 \times r_2 \times r_3$, and 3 orthogonal projection matrices U_1, U_2, U_3 each of size $R_i \times r_i$ for $i = 1, 2, 3$, respectively. The goal is to achieve $\mathcal{A} \approx \mathcal{G} \prod_{i=1}^{3} \times_i U_i$. The choice of the ranks r_1, r_2, r_3 determines how accurately the decomposition can represent the tensor $\mathcal{A}$.

Figure 6.1: Image compression by lowering the SVD rank.

A popular application to employ tensor decomposition is image compression. Suppose we have an image matrix $X_{240\times200}$ to be compressed. Using SVD, the maximum choice for the rank r during decomposition is $200 = min(200, 240)$. To compression this image, a smaller r can be chosen to store the image as three matrices $U_{240\times r}\Sigma_{r\times r}(V^T)_{r\times200}$. Figure 6.1 illustrates the effects of compressing a 240×200-pixel image by varying the rank. As we can see from the figure, when we select $r = 50$, the compressed image is visually indistinguishable from the original. The original grey-scale image requires $240 \times 200 = 48,000$ values to store, where as the compressed version requires $(240 \times 50) + (50 \times 50) + (50 \times 200) = 24,500$ values to store, roughly 50% less.

Likewise, for WMPR, Figure 6.2 shows similar effects of compressing a wafer image. A wafer image is encoded as a matrix where a +1 entry (yellow) corresponds to failing,

−1 entry (purple) corresponds to passing, and a 0 entry (green) means that location has no die. The wafer image is 48×48 and hence the maximum rank is 48. As seen, with $rank = 10$, the edge pattern can be mostly restored, and when $rank = 25$, the wafer image itself can be mostly restored.

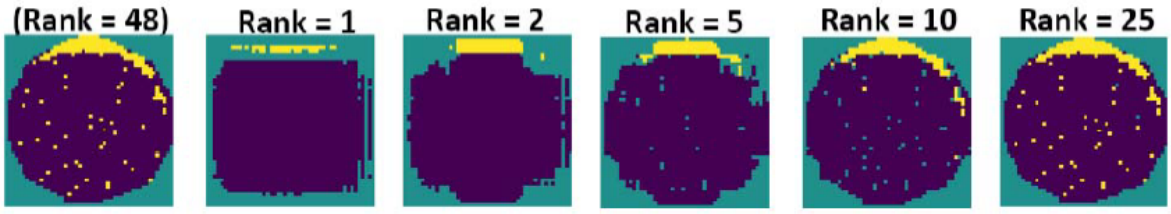

Figure 6.2: Wafer image compression by lowering the SVD rank.

6.1.2 Projection matrices

During decomposition, a projection matrix along a dimension can be interpreted as the set of basis vectors along that dimension. For example, in SVD, the original matrix X can be decomposed into $X = U\Sigma V^T$, where U comprises the eigenvectors of XX^T, V comprises the eigenvectors of X^TX, and Σ is a diagonal matrix. This is to say, U comprises the basis vectors along the vertical dimension and V comprises the basis vectors along the horizontal dimension. To see the intuition, consider the example shown in Figure 6.3.

Suppose SVD is applied to the two wafer images W_a and W_b. We obtain $W_a = U_a\Sigma_a V_a^T$ and $W_b = U_b\Sigma_b V_b^T$, i.e. the calculations which can restore their original images. The image $U_b\Sigma_a V_a^T$ is obtained by using the left projection matrix U_b from image W_b and applying this projection to image W_a. If we view W_b by scanning vertically the rows of pixels from the top to bottom, we see that the yellow dots appear mostly in the middle rows. This feature is effectively maintained in the image of $U_b\Sigma_a V_a^T$. Similarly, for $U_a\Sigma_a V_b^T$, we see yellow dots appear mostly in the middle, but horizontally. Then, $U_b\Sigma_a V_b^T$ makes the

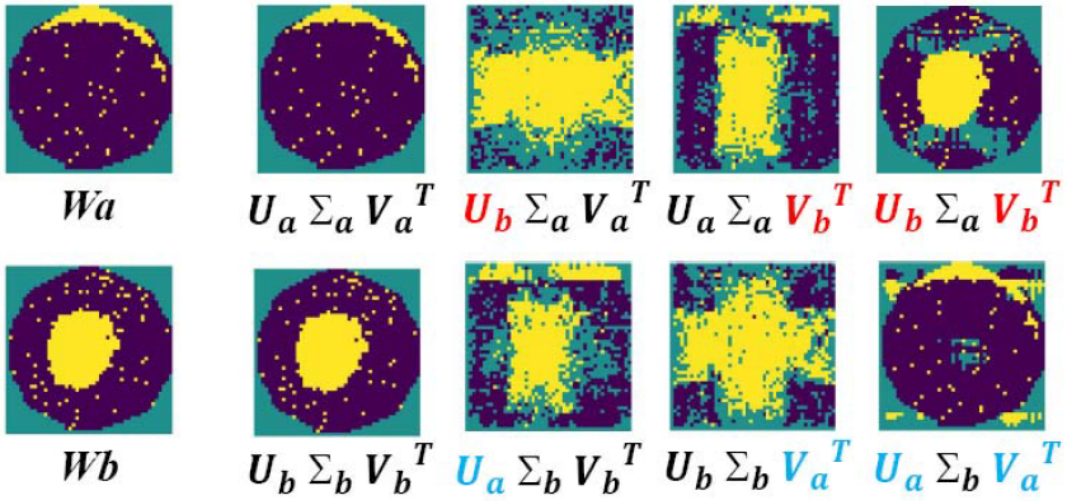

Figure 6.3: Example to illustrate the effects from projection matrices.

image look similar to W_b. Likewise, in $U_a\Sigma_b V_b^T$, the edge feature on the top is maintained. In $U_b\Sigma_b V_a^T$, the yellow dots are spread out because if we scan W_a horizontally by viewing the columns of pixels, we can see a spread of yellow too. Finally, $U_a\Sigma_b V_a^T$ makes the image look similar to W_a.

6.1.3 Adding the third dimension

Tucker Decomposition, instead of SVD, is used when there are more than two dimensions. To apply Tucker Decomposition in the WMC context, multiple wafer images (i.e. training samples) are stacked to create the third dimension as illustrated in Figure 6.4. For wafer images of size 48×48, the size of the third dimension n is the number of wafers. The resulting model from Tucker decomposition has four components: the three projection matrices U_1, U_2, U_3 and the core matrix $\mathcal{G}$. In this example, we use a single rank r and let $r = r_1 = r_2$. Following the discussion from Section 6.1.2, U_1 and U_2 capture the features along the first two dimensions. The matrix U_3 captures the features along the third, which can be interpreted as capturing features from wafer-to-wafer variations.

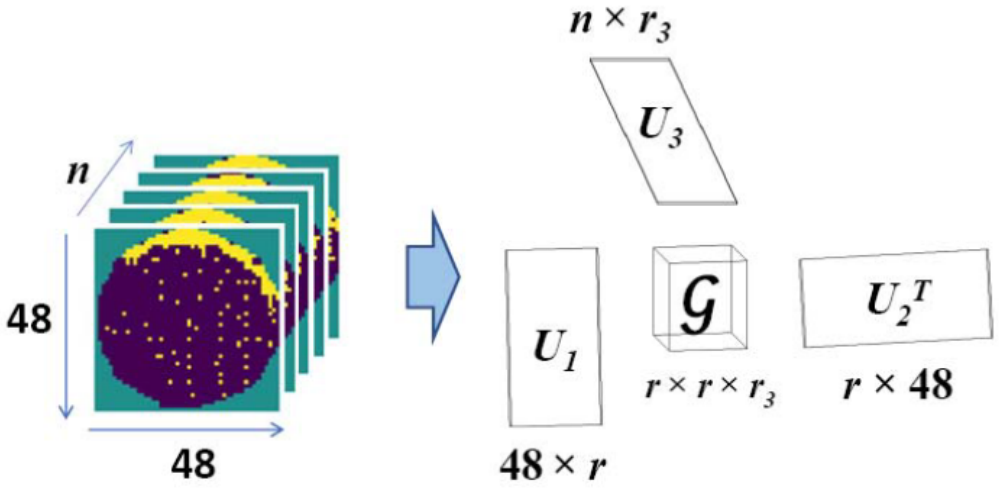

Figure 6.4: Stacking multiple wafers to use Tucker decomposition.

6.1.4 Reconstruction with similar wafer patterns

Suppose the training wafer images used in applying Tucker Decomposition are all similar. Specifically, there is no significant wafer-to-wafer variation in terms of their wafer patterns, where U_3 is less interesting to us. Figure 6.5 shows an example of using

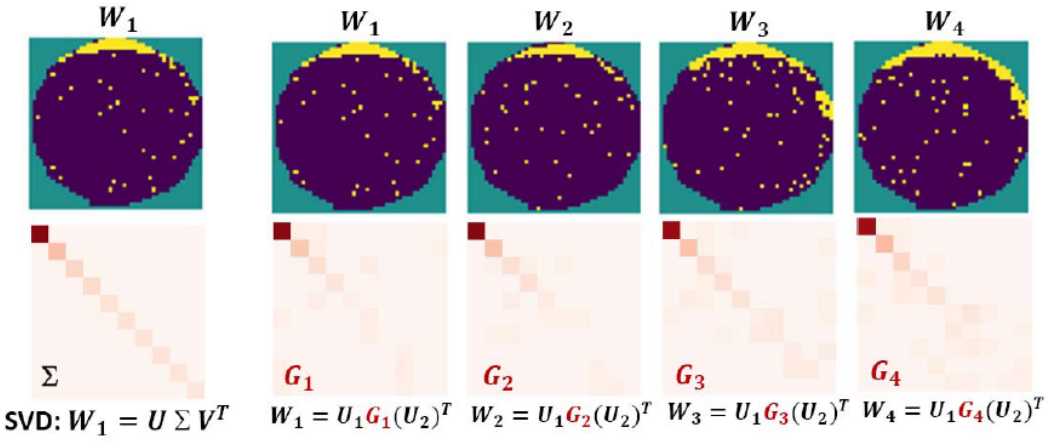

Figure 6.5: Similar wafer images and the projected core matrices.

four similar wafer images W_1, W_2, W_3, W_4. On the very left, SVD is performed on wafer W_1 and its resulting Σ matrix is shown. For ease of visualization, only the upper-left

10×10 portion of the Σ diagonal matrix is shown below the original image, where an entry with a darker color means the (absolute) value in that entry is larger. Note that all the non-diagonal entries in Σ have zero value.

On the right, Tucker Decomposition is performed on a 3D tensor composed by the four wafer images, like previously demonstrated in Figure 6.4. After decomposition, U_3 is ignored, and U_1, U_2 are used to calculate a projected core matrix G_i for image W_i, respectively. Each G_i is obtain such that $W_i = U_1 G_i (U_2)^T$. It is interesting to observe that each G_i looks similar to Σ, where the diagonal entries have a darker color and the entry $[0, 0]$ is the darkest. However, each G_i is no longer strictly diagonal (i.e. non-diagonal entries can have a non-zero value).

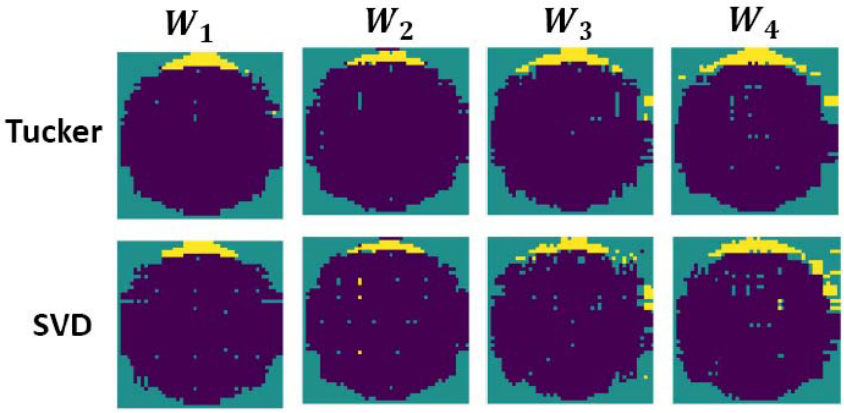

Figure 6.6: Reconstruction using the upper-left 5×5 entries of G_i.

Recall Figure 6.2 demonstrates how a wafer image can be reconstructed by using lower ranks under SVD. The same idea can be applied by using the G_i matrices from Figure 6.5. To illustrate, in Figure 6.6, the top row of "Tucker" images are reconstructed based on the upper-left 5×5 entries of each G_i, respectively. For comparison, the bottom row of reconstructed images are obtained using "SVD" with rank $r = 5$. Observe that for all four wafers, the edge pattern can be restored and the differences between using SVD and using G_i with U_1, U_2 are not significant. Because each G_i is not a diagonal matrix,

it would be interesting to see how the reconstruction works if we only use the diagonal entries from each G_i. Figure 6.7 shows these results, where the edge pattern becomes even more apparent in each case and all "random noise" from each wafer seem to be removed.

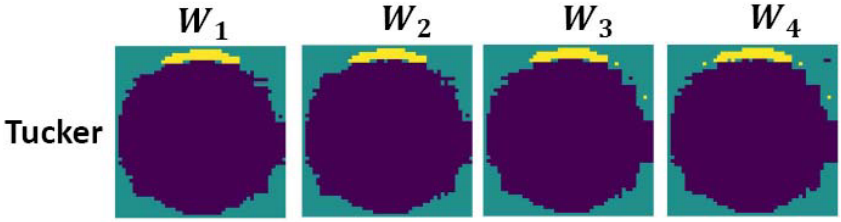

Figure 6.7: Reconstruction using the first 5 diagonal entries of G_i.

6.1.5 Reconstruction with dissimilar wafer patterns

In Figure 6.5, decomposition results from four visually similar wafer images are presented. In Figure 6.8, we continue the exploration by reusing wafer images W_1, W_3 from Figure 6.5, in addition to two new images W_5, W_6 which contain a different pattern. The very left of Figure 6.8 shows SVD's diagonal matrix for W_6. Due to Tucker Decomposition producing different projection matrices for W_1, W_3 (from those used in Figure 6.5), the newly produced core matrices G'_1, G'_3 are shown. Additionally, G_5, G_6 are obtained for W_5, W_6. It is interesting to observe that the diagonal trend seen on the **G** matrices in Figure 6.5 is much less apparent on the **G** matrices in Figure 6.8. This is especially true for G_5 and G_6 where the non-diagonal entries have large values.

Wafer images in Figure 6.9 and Figure 6.10 are obtained in the same way as those shown previously in Figure 6.6 and Figure 6.7. As seen in Figure 6.9, each of the wafer patterns can be mostly restored. However, this is not the case if we only use the diagonal entries as shown in Figure 6.10. Comparing Figure 6.10 to Figure 6.7, we see that if the projection matrices are based on dissimilar wafers, then the non-diagonal entries in

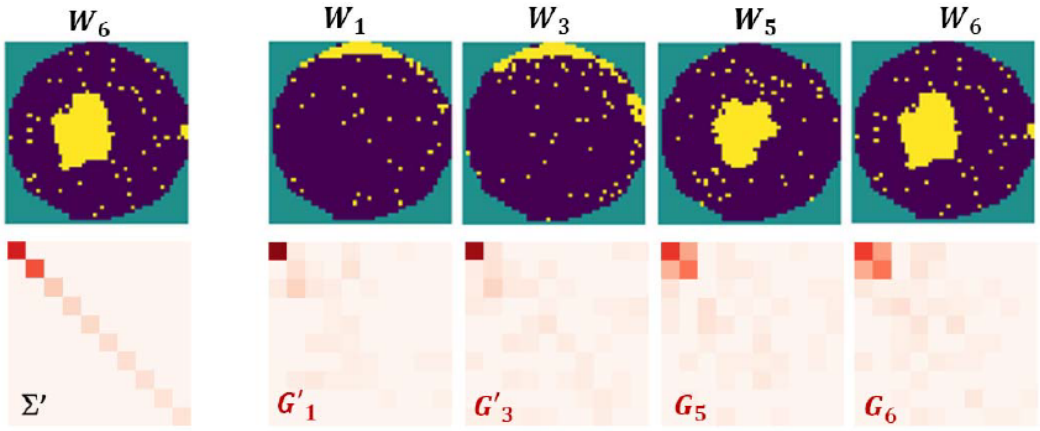

Figure 6.8: Dissimilar wafer images and the projected core matrices.

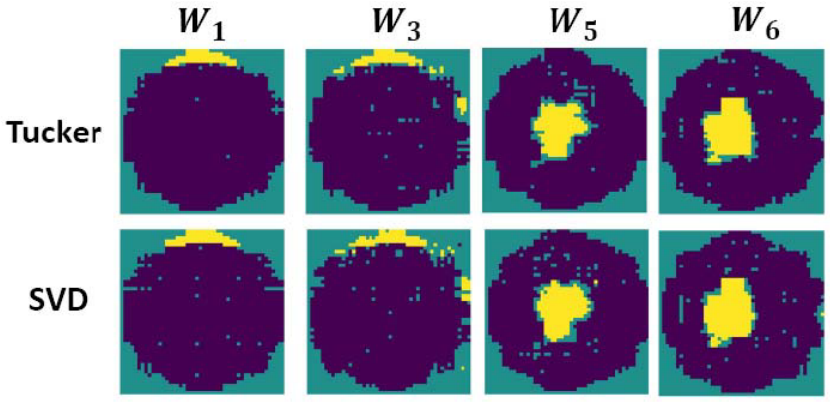

Figure 6.9: Reconstruction using the upper-left 5×5 entries of G_i.

the resulting **G** matrices become important for restoring the pattern. In other words, the non-diagonal entries which have large values cannot be ignored. This observation suggests that the "diagonality" might be an important property to indicate the extent of similarity among wafer images.

6.1.6 Comparing a wafer image with its reconstruction

As seen from Figure 6.3, we can obtain projection matrices U_a and V_a from wafer W_a and Σ_b from wafer W_b, to attempt to reconstruct W_b by $U_a \Sigma_b V_a^T$. In other words, given a

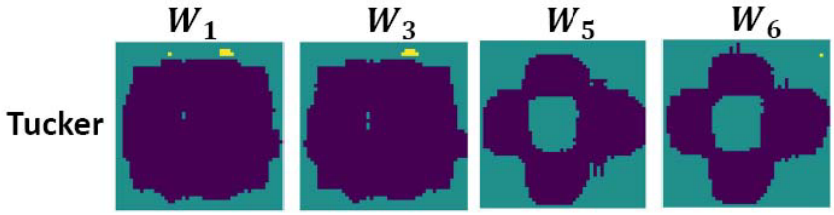

Figure 6.10: Fail to reconstruct by using only the first 5 diagonal entries of G_i.

training image W_a and image W_b to be classified, it seems we can use the reconstruction result W_b' to decide how similar W_b is to the training image W_a. If W_b is similar to the training image, we expect the difference between W_b and its reconstruction W_b' to be small. Otherwise, we expect their difference to be large.

We can extend this idea to use multiple images. Given a set of training wafer images $W_1, ..., W_n$, we first apply Tucker Decomposition to obtain the two projection matrices $U1, U2$. Then given an image W_j to be classified, we can apply SVD on W_j to obtain Σ_j. Combining with the projection matrices from Tucker Decomposition, we can use $U_1 \Sigma_j U_2^T$ to obtain W_j' The difference between W_j and W_j' can then be used to judge how similar W_j is to the training images. Figure 6.11 provides an example to support this

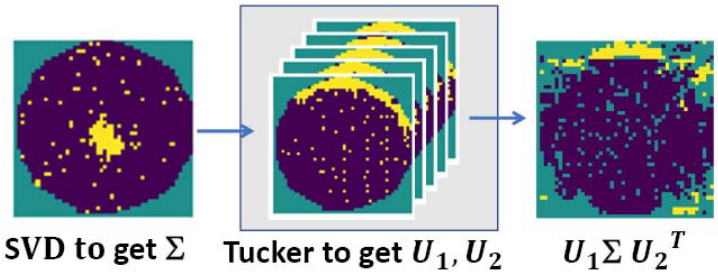

Figure 6.11: An edge model transforms a center image into an edge image.

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.

However, Figure 6.12 shows this idea might not work. In this case, the five training

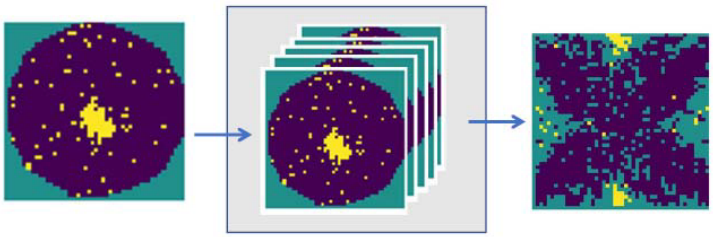

Figure 6.12: A center model fails to reconstruct a training image.

images are similar to the input image to be classified. The reconstructed image looks quite different from the input image. This is because in order to restore the image using the two projection matrices U_1, U_2, several diagonal entries in the projected core matrix would need to have negative values. On the other hand, SVD's entries are all non-negative, resulting in the image not being properly restored. Hence, we cannot simply use the diagonal matrix from SVD in the reconstruction.

6.1.7 Using the projected core matrix

For a given image W_j, the alternative is to use the projected core matrix G_j calculated based on U_1, U_2. However, we cannot use the entirety of G_j, since we always get back to the original image W_j. To overcome this issue: (1) We can use only an upper-left portion of G_j, similar to the examples presented in Figure 6.6 and 6.9. (2) We can use only the diagonal entries of G_j and ignore all non-diagonal values.

6.1.7.1 Method (1)

Suppose a model M is built based on a set of training wafer images. M essentially comprises the two projection matrices U_1, U_2. We calculate G_j where $W_j = U_1 G_j (U_2)^T$. Then, using only the upper-left $r \times r$ entries to obtain G_j^r, we calculate $W_j' = U_1 G_j^r (U_2)^T$. Finally, the difference between W_j and W_j' is used to decide if W_j is similar to the training samples. Note, this alternative is essentially the idea that would have been suggested

by [74] if the approach was applied to our context. Let us explicitly define a difference measure $DIFF(W_j, W'_j)$. For example, $DIFF(W_j, W'_j)$ can be based on summing up the squares of values in $W_j - W'_j$. Then, we can let the recognition error measure of the model M (for a chosen r) be $Err_{M,r}(W_j) = DIFF(W_j, W'_j)$.

To illustrate how the error measure might be used, Figure 6.13 shows five images (labeled W_1 to W_5) for building a model M. Figure 6.14 then shows four groups of images, each with five images for a total of 20 images (labeled W_6 to W_{25}). If we use M and the error measure to recognize the 20 images W_6 to W_{25}, we expect W_6 to W_{10} to be recognized as in-class and W_{11} to W_{25} as out-of-class. This is because W_6 to W_{10} all exhibit an edge pattern similar to images W_1 to W_5 used when building the model.

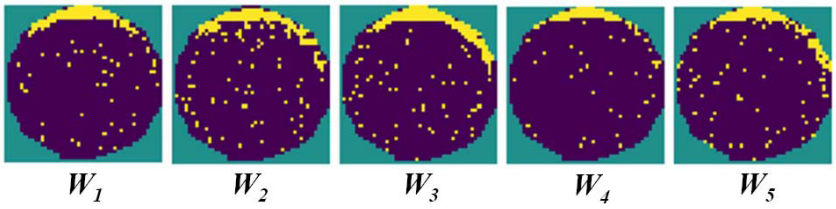

Figure 6.13: The five training wafer images.

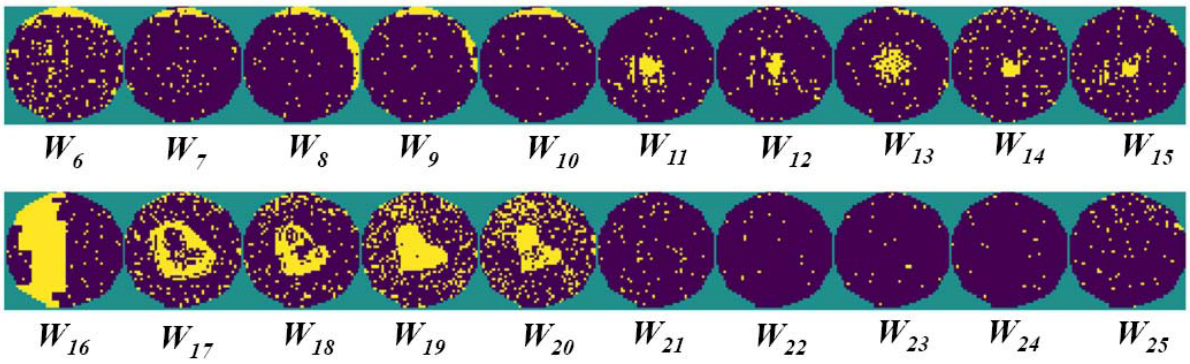

Figure 6.14: The 20 wafer images to be recognized.

For each W_j, we calculate $Err_{M,r}(W_j)$ for $j = 1, ..., 25$. Figure 6.15 plots the $Err_{M,r}$ values. For every wafer, three values are plotted by using $r = 2, 5, 25$. Among the three values, the smallest value (in black color) is based on $r = 25$ and the largest (in blue

color) is based on r = 2. The middle (in red color) is based on r = 5. For W_6 to W_{10},

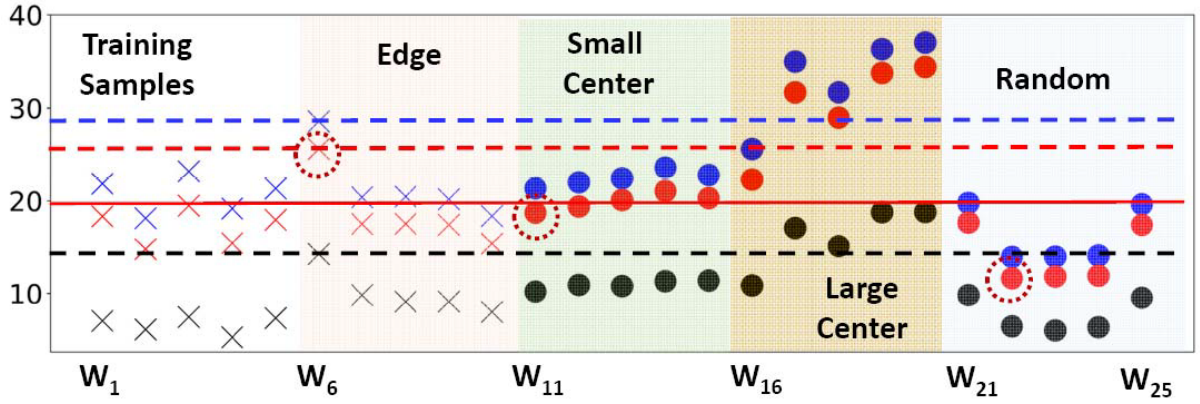

Figure 6.15: Difficulty to set a threshold to achieve correct classification.

they should be recognized as in-class and hence we need to set a threshold such that all error values of W_6 to 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 W_1 to W_5 themselves).

In this figure, we observe that it is impossible to set a threshold to include W_6 as in-class while leaving W_{11} to W_{25} to be out-of-class. For example, consider using the error values with r = 5 (shown in red). The horizontal red dashed line shows the minimum threshold to classify W_6 as in-class. But with this threshold, all the "small-center" images W_{11} to W_{15}, W_{16}, and all the "random" images W_{21} to W_{25} would also be classified as in-class (i.e. below the red dashed line). Similarly, the black and blue dashed lines are the minimum thresholds to classify W_6 as in-class for r = 2 and r = 25, respectively. As seen, with these thresholds, many of the images from W_{11} to W_{25} are also classified as in-class. Suppose we are willing to accept mis-classification of W_6 as out-of-class in order to lower the threshold. This lower threshold for r = 5 is shown as the solid red horizontal line. Even with this threshold, we see that W_{11}, W_{12} and all of the random images W_{21} to W_{25} are still below the line.

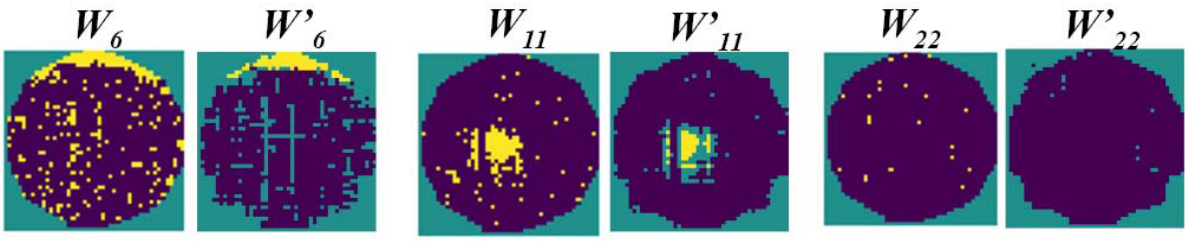

Figure 6.16: Method (1): the difference between W_j and W'_j for $j = 6, 11, 22$.

In Figure 6.16, W_6, W_{11}, W_{22} are selected (marked as red dashed circles in Figure 6.15) to show the original images and their reconstructed images (for $r = 5$). Observe that through the reconstruction, both the edge pattern on W_6 and the small-center pattern on 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. If the effect of the reconstruction is basically to somewhat keep the pattern (regardless of having an edge pattern) and replace most of the yellow dots with green or purple dots, then there is no clear reason why the current $Err_{M,r}$ has the ability to differentiate in-class images (with an edge pattern) from out-of-class images (without an edge pattern). In summary, Figure 6.15 shows that it would be difficult to make the recognition work by using G_j^r in the reconstruction.

6.1.7.2 Method (2)

In this method, we can investigate the effects of only using the diagonal entries of G_j. Specifically, we can define a new error measure, $Err_M(W_j) = DIFF(W_j, W'_j)$ where W'_j is reconstructed by using only the diagonal entries of G_j. Figure 6.17 shows the error values based on $Err_M(W_j)$ 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 natural-log scale. We can observe from this figure, 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

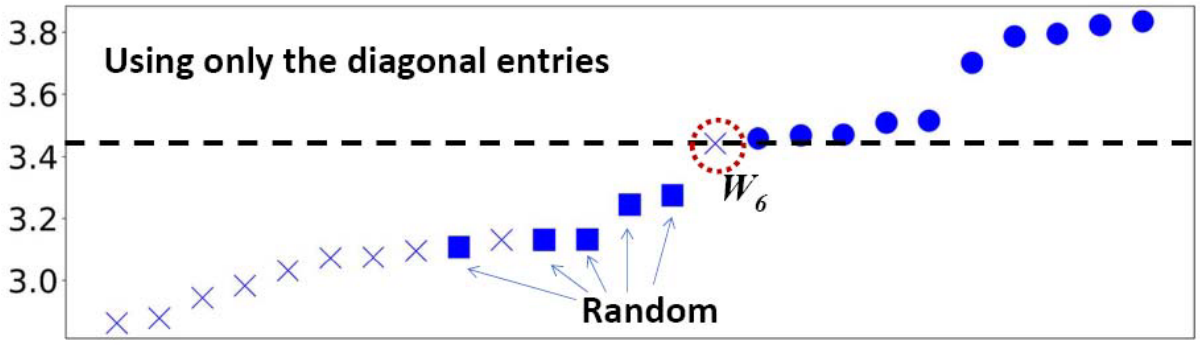

Figure 6.17: Sorted error values calculated based on only the diagonal entries.

improvement from the result in Figure 6.15, because now we can treat "random" wafer image as a special case and try to find a way to exclude them.

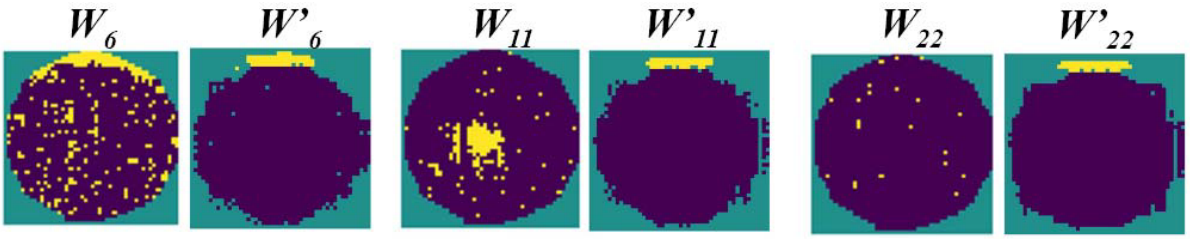

Figure 6.18: Method (2): the difference between W_j and W'_j for $j = 6, 11, 22$.

Similar to Figure 6.16 for W_6, W_{11}, W_{22}, Figure 6.18 shows the original images and their reconstructed images using only the diagonal entries. It is interesting to see that an edge pattern appears in all of the reconstructed images. This effect is similar to what was shown in Figure 6.3 previously. Specifically, the original $W_1 1$ wafer has a center pattern and the reconstructed W'_{11} has an edge pattern. Hence, the difference function $DIFF$ would be measuring the difference between the two patterns, which is more intuitive in terms of what we desire the difference measure to be.

6.1.8 Measuring diagonality on G_j

Using only the diagonal entries in 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 6.5 previously, where if the model is built on a set of "edge" wafer images, for a given "edge" wafer image, the projected core matrix G_j would have large values in the diagonal entries and small values in the non-diagonal entries. Figure 6.19 plots the projected core matrices for $W_3, W_6, W_{11}, W_{12}, W_{16}$, and W_{22}

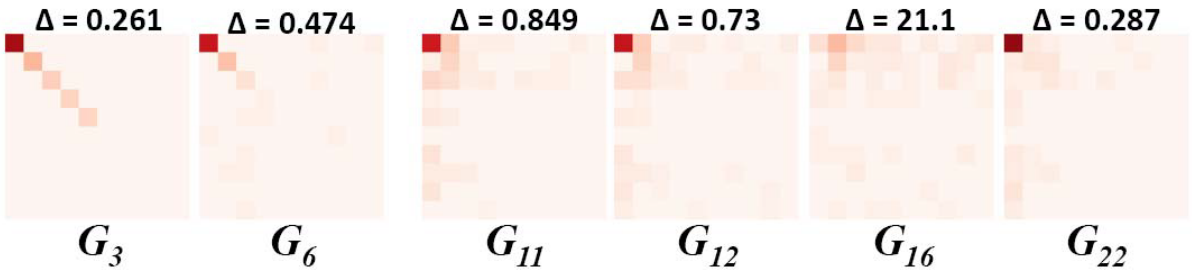

Figure 6.19: The upper-left 10×10 entries of various G_j matrices.

in a similar fashion compared to Figure 6.5. Observe that W_3 and W_6 are "edge" wafer images and their projected core matrices exhibit the diagonality property. In contrast, $W_{11}, W_{12}, W_{16}, W_{22}$ do not have an edge pattern, and their projected core matrices do not exhibit the diagonality property.

If the diagonality property is the underlying effect for results seen in Figure 6.17, instead of measuring on $W_j - W'_j$, a better alternative can be to measure the diagonality property directly. Where this diagonality measure can be used as the distance/similarity measure between a given wafer image and the set of training images used to build the model M. Specifically, given a model M, a wafer image W, and its projected core matrix $\mathbf{G}$, the measure of diagonality Δ can be defined as:

$$\Delta_{M \to W} = \frac{\Sigma_{\forall i \neq j} G[i,j]^2}{\Sigma_{\forall i} G[i,i]^2} \tag{6.1}$$

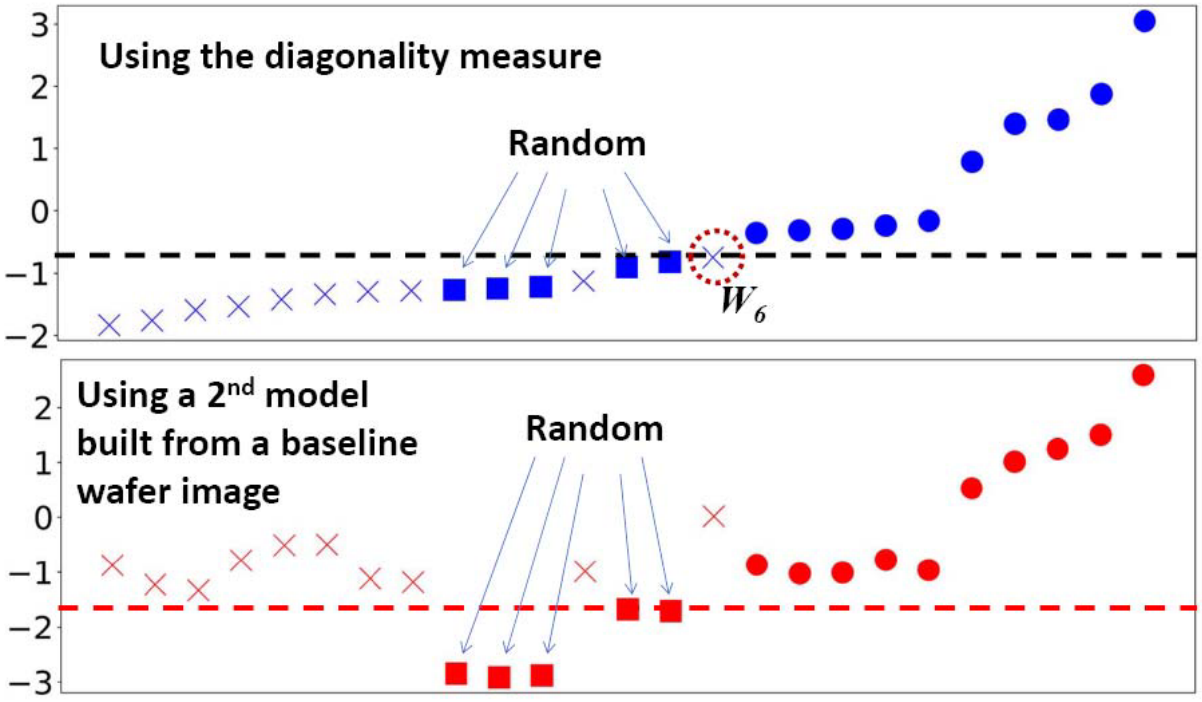

Figure 6.20: Errors calculated based on the diagonality measure.

In Figure 6.19, this Δ value is also shown for each of the six wafer images. It is interesting to note that W_6 and W_{11} were the two images causing difficulty in Figure 6.15 previously. Now W_{11} has a Δ value of 0.849, which is much bigger than the Δ value of W_6 at 0.474. Then, the top plot in Figure 6.20 shows the Δ values for all W_1 to W_{25}, which are also sorted and presented in natural log scale. The result is similar to that shown in Figure 6.17, and is slight better since 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 2^{nd} model and call it a baseline model, M_{base}. In this experiment, M_{base} is simply based on a wafer with no fails, i.e. no yellow dots at all. The bottom plot in Figure 6.20 shows the Δ values for W_1 to W_{25} based on 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.

6.1.9 Measuring diagonality on G_j excluding entry $[0,0]$

Observe in Figure 6.19 that the $[0,0]$ entry in most of the projected core matrices have the largest values. In Equation 6.1, this large entry effectively makes the Δ value smaller. However, if our goal is to differentiate, say W_3, W_6 from W_{11}, W_{12}, W_{22}, it seems that using the large values in the $[0,0]$ entry of G_{11}, G_{12}, G_{22} is not desirable. This motivates us to try a 2^{nd} distance measure by removing the entry $[0,0]$ from the G_j matrix in the Δ calculation:

$$\Delta'_{M \to W} = \frac{\Sigma_{\forall i \neq j} G[i,j]^2}{\Sigma_{\forall i, i \neq 0} G[i,i]^2} \tag{6.2}$$

Figure 6.21 then shows the sorted Δ' values for W_1 to W_{25}. 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 recognition results discussed so far.

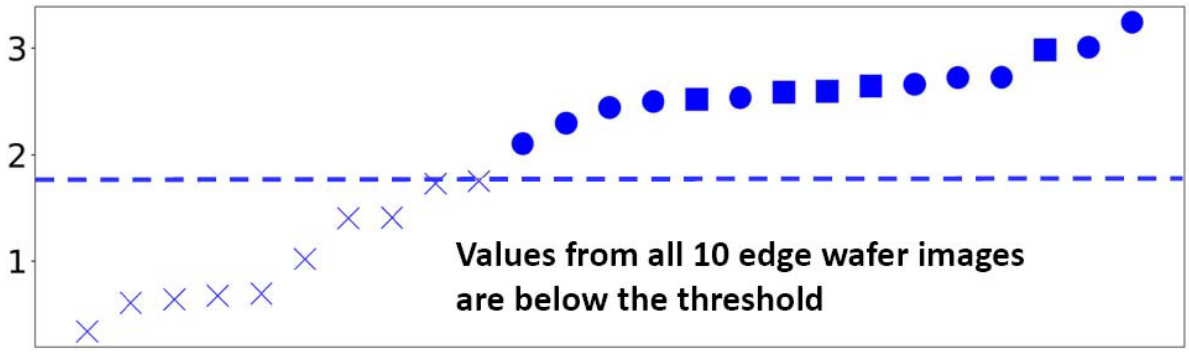

Figure 6.21: Errors calculated based by removing entry $[0,0]$.

6.1.10 Performing tensor-based recognition

To summarize, we have define two distance functions, Δ and Δ' such that they measure the similarity between two wafer images based on the *diagonality* property demon-

strated throughout the previous sections. Given a wafer matrix W_A, we first build a tensor model T_A using the method demonstrated in Figure 6.5 and 6.8. Specifically, a T_A model contains the projection matrices of wafer W_A, such that when T_A is applied to another wafer matrix W_B, the result is a *core matrix* $M_{A\to B}$. If W_B is similar to W_A, $M_{A\to B}$ would have relatively large absolute values in the diagonal entries and close-to-zero values in non-diagonal entries. Otherwise, $M_{A\to B}$ would have large values in non-diagonal entries. Hence, this method utilizes this diagonality property to define a *directional* distance measure Δ from W_A to W_B as:

$$\Delta_{A\to B} = \frac{\Sigma_{\forall i\neq j} M_{A\to B}[i,j]^2}{\Sigma_{\forall i} M_{A\to B}[i,i]^2} \tag{6.3}$$

Note, Equation 6.3 is effectively the same as Equation 6.1, except the core matrix G' is now replaced with a more explicit M which can indicate the directional calculation. Now for Δ', as indicated by Figure 6.5 and 6.8, the large value in entry $[0,0]$ can effectively make the Δ 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 Δ calculation by ignoring the entry $[0,0]$ as an alternative similarly measure. This modified calculation is our Δ':

$$\Delta'_{A\to B} = \frac{\Sigma_{\forall i\neq j} M_{A\to B}[i,j]^2}{\Sigma_{\forall i, i\neq 0} M_{A\to B}[i,i]^2} \tag{6.4}$$

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

$$\delta_{A\to B} = \frac{\Delta_{A\to B}}{\Delta_{A\to B} + \Delta_{base\to B}} \tag{6.5}$$

The distance between W_A and W_B is then calculated as the average of $\delta_{A\to B}$ and $\delta_{B\to A}$. Figure 6.22 uses a wafer A and its distances to three wafers B, C, D as an example to illustrate these distance calculations.

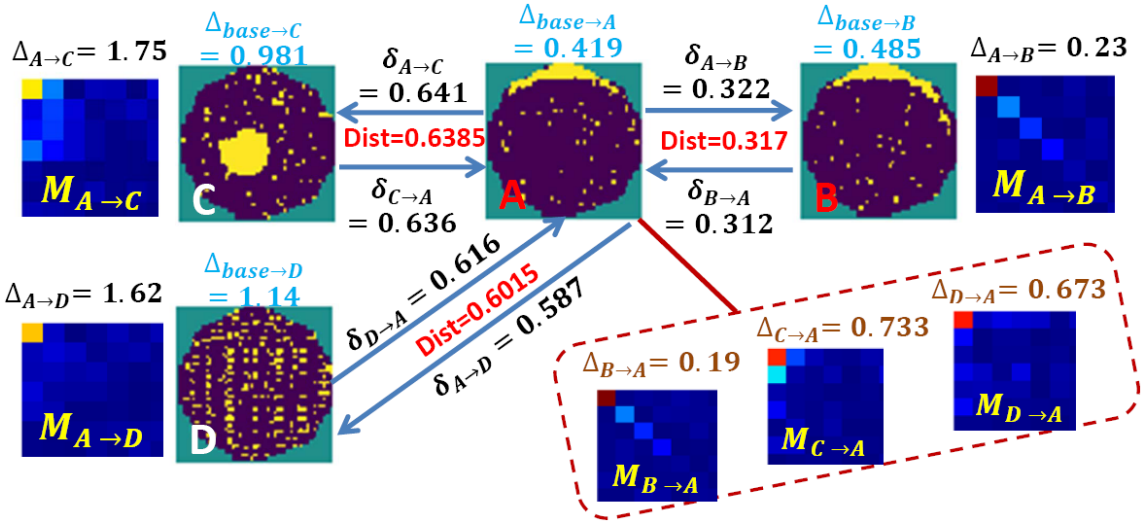

Figure 6.22: Illustration of the similarity metric (Each M matrix is actually bigger and only the top left portion is shown).

Notice that wafer A and wafer B are similar. The matrix $M_{A\to B}$ has large values on the diagonal entries and almost zero on the non-diagonal entries, thus $\Delta_{A\to B}$ is 0.23. The matrix $M_{B\to A}$ on the bottom shows a similar property with $\Delta_{B\to A}$ = 0.19. On the other hand, $M_{A\to C}$ and $M_{A\to D}$ do not have the property because they are not similar to wafer A. Their Δ 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.

To perform recognition with many M models, given an input wafer W_a, it would be recognized by a model M only if it is closer to that M than to any of the baseline models (i.e. having the smallest distance value among all). In other words, W_a will be assigned to a cluster of similar wafers identified by M. To generalize this, if we have a set of recognition models $M_1, \ldots, M_n$ and a set of baseline models, a wafer would be considered

recognized by the model M_i if that wafer has the smallest distance to M_i. Because we have two distance measures, Δ and Δ', for each type of wafer images, we can effectively have two recognition models based on the two measures. Suppose M_i, M_i' are the two models for recognizing the same type of image. Suppose in application their recognized sets are S_i, S_i'. By taking an intersection $S_i \cap S_i'$, this improves the robustness of the tensor-based recognition. To be demonstrated in the later sections, their intersection $S_i \cap S_i'$ is used as the recognized set from the tensor-based model for that type of wafer images. In other words, $S_i \cap S_i'$ is a lower-bound model derived for that type of wafer images.

6.2 Filtering Wafers with No Pattern

Having discussed how to derive an lower-bound model in the previous section, we can now focus on the other capability mentioned at the start of this chapter: filtering out wafers with no patterns. For the rest of this section, let us suppose this filtering capability is implemented as a step in a DSML software methodology. Let us further suppose "Jay" is the name of the DSML software provider.

The purpose of this filtering step is to filter out all wafers that do not contain a pattern, i.e. showing only random fails. It is important to keep in mind that the goal is not to filter out all random-failing wafers. Instead, the goal is to filter out random-failing wafers if their randomness can be decided rather easily. It is perfectly acceptable if some random-failing wafers slip through the filtering and get passed to the subsequent steps. As we will demonstrate in the next section, they increase the burden of the training step (because of the calculation of pairwise distances), but should not affect the end outcome much. Therefore, the purpose of the filtering step is more for reducing the cost than for improving the recognition. One key assumption here is that Jay does not have any

actual wafer images with random fails to train a recognizer. As a result, Jay needs to create some samples to help train a "no pattern" recognizer to perform the filtering. To be demonstrated in later sections, Jay will be using Generative Adversarial Networks (GANs) [31] as an approach to build recognizers for the to-be-deployed DSML software.

6.2.1 Generating artificial samples

It is straightforward to generate in-class samples containing random fails. Following an assumed yield distribution, a yield can be randomly picked and after that, the location of a fail can be randomly picked. In this generation, Jay assumes the average yield is 95%. A more interesting issue is how to generate out-of-class samples. There are three types of non-randomness Jay considers: (1) regularity, (2) concentration, and (3) catastrophe. For catastrophe, it is easy to generate the samples by enforcing a low yield, e.g. 5%. For the other two, Jay needs to use a rule for the generation. Regularity means the locations of the fails show a regular pattern in a particular direction. For example, horizontally there exists a constant **D** such that the distances between many pairs of failing dies are all **D**. Therefore, to generate a sample, Jay can randomly pick a direction and then randomly pick a **D**. Typically, a wafer would always contain some random fails. Hence, in the generation a regularity wafer will be stacked with a random wafer to obtain a sample. In the generation, four directions are considered: horizontally, vertically, and along the two 45-degree lines.

Concentration means there is a cluster of fails. There can be two types: one concentrating around a center point and the other spread across a line or an edge. In the generation, Jay can randomly pick a type and like the regularity, stack a generated wafer with a random wafer to obtain a sample. Figure 6.23 shows some examples of the generated in-class and out-of-class samples. Because out-of-class samples are not used in the

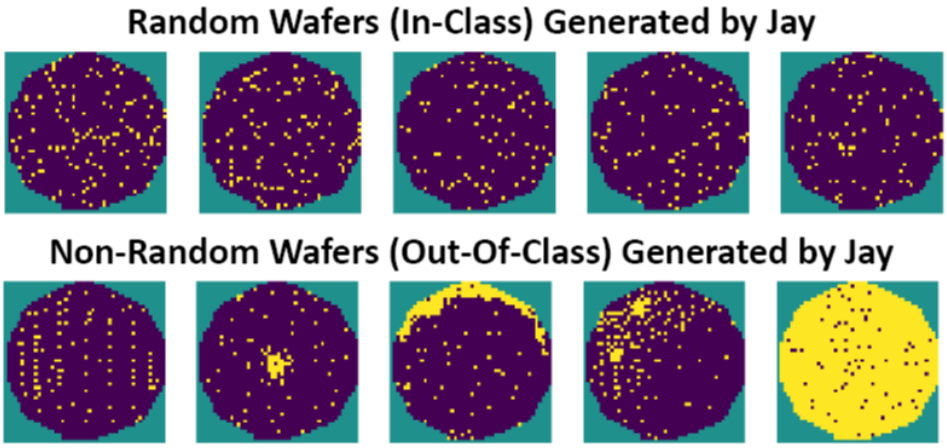

Figure 6.23: Jay's generated in-class and out-of-class samples.

training and only used in the validation, the list does not need to be complete. We only need some out-of-class examples to guide the training of a "no pattern" recognizer.

6.2.2 Rule checking on the filtered set

Jay knows that a GANs recognizer cannot be 100% accurate. Because it is important to not filter out a wafer where there is a chance that later steps might discover a pattern on it, the recognized set of random wafers by the GANs model needs to go through another checking process. This checking process applies two rules. The first rule looks for a regularity and the second rule looks for a concentration. Their specifics are described below.

[Rule 1]: For every fail on a wafer, calculate the distance value d as the distance to its nearest fail location along a given directional line (there are four possible directional lines as mentioned above). Let the number of fails with distance value d_j be f_j (frequency). Find the largest frequency number FR_{max} and let its distance be d_{most}. If $FR_{max} > th1$ and $d_{most} > 1$, then the wafer is considered non-random. The threshold $th1$ depends on the number of dies per wafer. The default value is 15.

[Rule 2]: Treat each fail as a vertex. Along the four possible direction lines, if there is also a fail then create an edge between these two fails. Given the graph, find the largest connected component. If $CC_{max} > th2$, where CC_{max} is the size of this largest connected component, then the wafer is considered non-random. The threshold $th2$ also depends on the number of dies per wafer. The default value is 20.

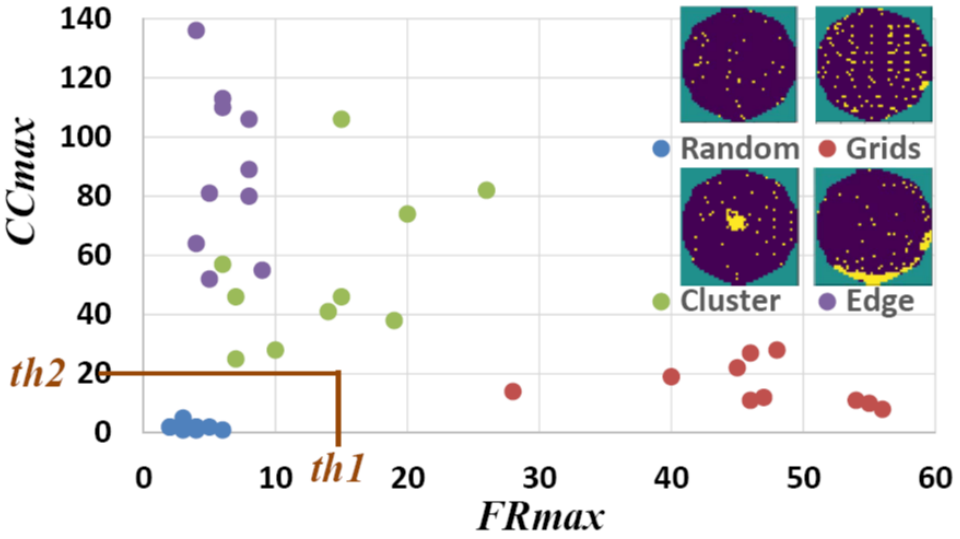

Figure 6.24: FR_{max} and CC_{max} on four types of wafers (10 each).

To see the effects of the rules on actual wafers, Figure 6.24 shows the FR_{max} and CC_{max} values across four types of wafers (10 wafers each, with one example image shown): random fails, grid pattern, concentrated cluster, and edge pattern. As seen, using both FR_{max} and CC_{max} can easily separate a random wafer from a non-random one.

6.3 Initial Deployment of a DSML Solution

Now that we have attained a way to derive lower-bound models using tensor-based methods, and a methodology to train a recognizer for filtering out "no pattern" wafers, we can attempt to deploy a DSML solution and observe the impacts of these capabilities' in deployment. Suppose during this deployment, "Jay" is the DSML solutions provider

and "Nik" is the DSML expert. Figure 6.25 illustrates the workflow employed by such a DSML software.

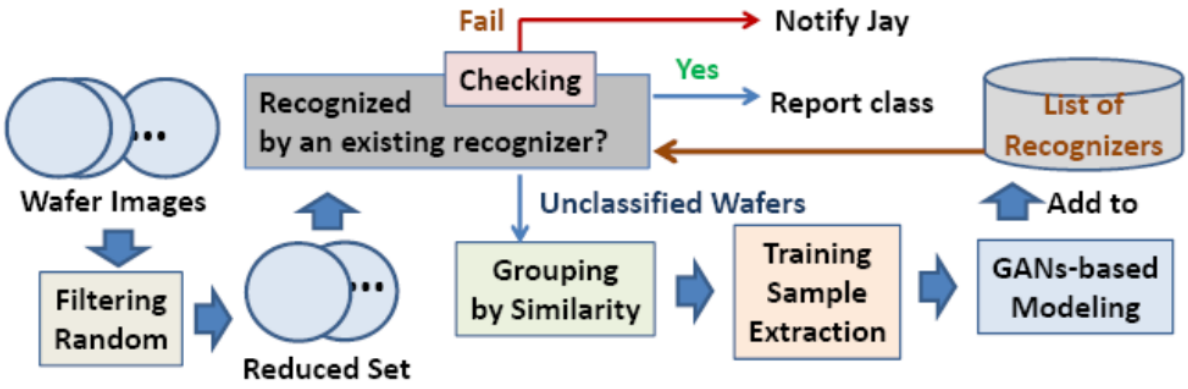

Figure 6.25: Workflow employed by the DSML software.

As discussed in Section 6.2, 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. 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. Otherwise, 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 deployment. Failing this Checking box results in a notification sent to Jay. After the recognition box, unrecognized 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.

6.3.1 Grouping by similarity

This step implements a clustering scheme in order to facilitate the training procedure of a GANs-based recognizer, which will be detailed in the next section. Our choice of the clustering algorithm is the Hierarchical DBSCAN (HDBSCAN) [75] 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 in terms of a GANs-based recognizer. The most subtle part in the cluster scheme is the definition of the distance function $Dist()$ that measures a distance between two images. Recall Section 6.1.10 which summarized the intuition behind the δ distance metric (Equation 6.5) derived from tensor-based methods. For implementing $Dist()$ between two wafers A and B, we use the average of $\delta_{A \to B}$ and $\delta_{B \to A}$.

6.3.2 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 our other work [76], 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 a Generative Adversarial Network (GAN) [31], there are two neural networks: the generator **G** and the discriminator **D**. The GANs modeling follows the same approach and training strategy presented in our initial GANs related work [77]. The generator's input is a randomly generated 100-dimension latent vector $\vec{v}$. The training is iterative, where 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

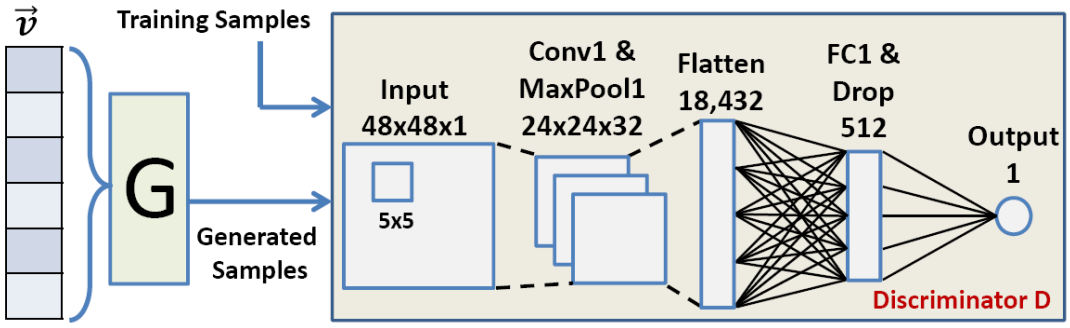

Figure 6.26: Discriminator neural network used in GANs modeling.

model for separating the generated samples from the training samples.

In the initial version of the software, the discriminator has a simple architecture as shown in Figure 6.26 (as compared to the complex architecture employed in [77]). Among several considerations behind this choice, the most important reason is due to the fact that training GANs can be tricky [78][79]. For instance, balancing the convergence between **G** and **D** can be quite a challenge, and often **D** might 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. The choice makes the capacity of **D** smaller than the capacity of **G**. This idea is inspired by the regularization ideas proposed in [80][81]. 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.

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 [76]. The training samples are used as the in-class sample set S_{in}. To calculate separability, we need an out-of-class sample set 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 [76], 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 [76] 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).

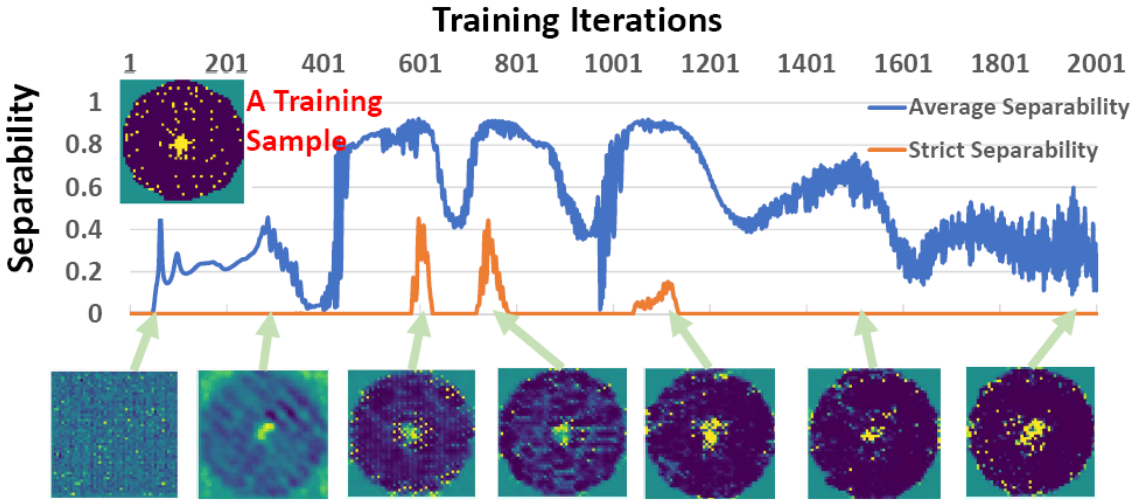

Figure 6.27: Illustration of a GANs training process over 2K iterations.

To illustrate the training process, Figure 6.27 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 S_{out} are used only in calculating the separability, and not used in the training of **D**. Hence, **D** is not optimized in terms of the separability measure 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 use the separability measure to select the best **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 (whose job is to separate the generated samples from the training samples) starts to over-fit 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 S_{out}.

6.4 Applying Tensor Methods as a Checker

6.4.1 Result from the first 50 lots

At first, the software flow depicted in Figure 6.25 is run on the first 50 lots. Figure 6.28 illustrates the result for each box. In this run, 9 recognizers are learned. Table 6.1

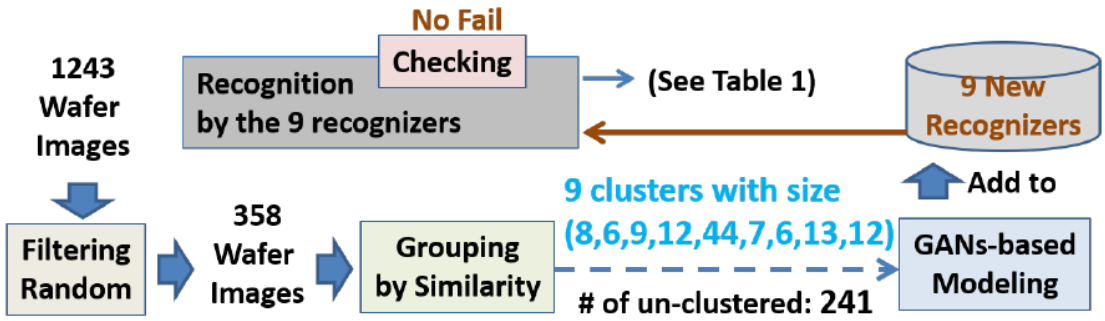

Figure 6.28: Workflow illustration for the first 50 lots.

then summarizes the recognition results by these recognizers.

Notice in Table 6.1, the number of wafers recognized in each concept is generally larger than the corresponding cluster size shown in Figure 6.28, except for Concept 5. As explained in our prior work [82], 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 6.29 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 [76], a Tensor model is more specific and hence, ends up separating Concepts 1-4 as different

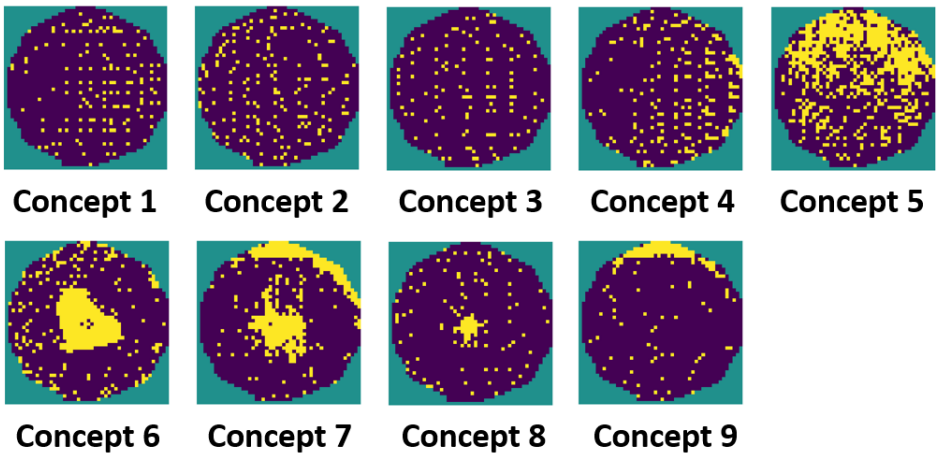

Figure 6.29: Typical example wafer image from each concept.

concepts.

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 6.30. 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 as unreasonable when he or his software script tries to utilize them.

In that regard, the numbers of false positives for Concepts 8 and 9 seem high. Figure 6.31 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 i is denoted as C_i).

Figure 6.30: False positive examples.

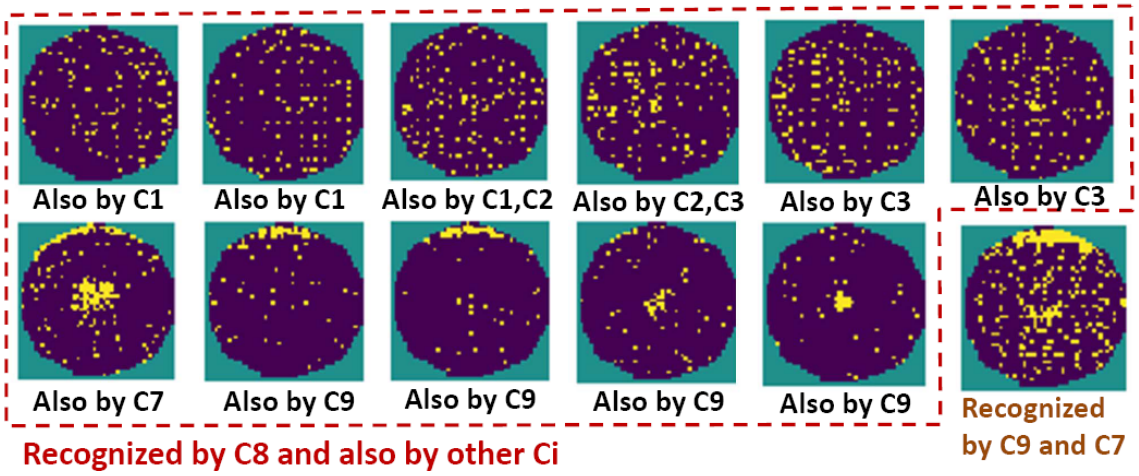

Figure 6.31: Other false positives for Concepts 8 and 9.

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 6.28. 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.

6.4.2 Result from the remaining 287 lots

The software continues its run on the remaining 287 lots. Figure 6.32 illustrates the results through the workflow. The difference between Figure 6.28 and Figure 6.32 is that, now in Figure 6.32 we have 9 recognizers already in the Recognition box.

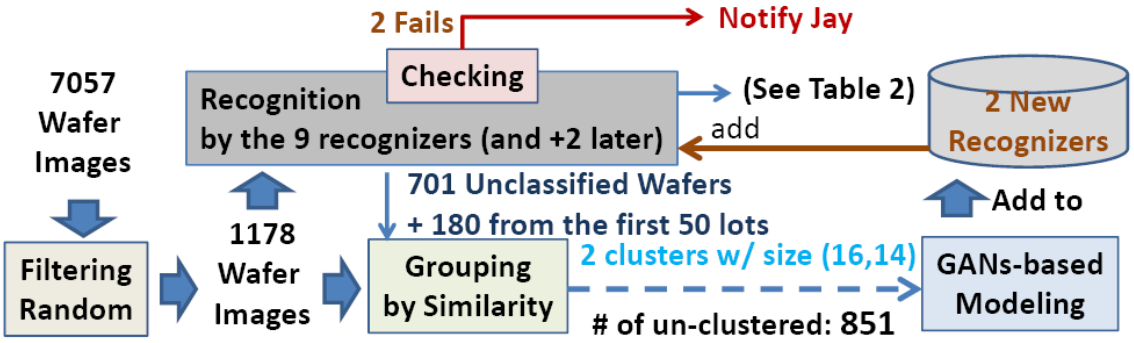

Figure 6.32: Workflow illustration for the remaining 287 lots.

Table 6.2 summarizes the results from the Recognition box. This time, recognition for Concept 8 and Concept 9 fails the Checking box, courtesy of the lower-bound tensor-based models. As a result, Jay is notified with the specific issue by the 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 6.33 shows what these new types of false positives look like. Notice that after the recognition, two more concepts are

identified with the two newly learned recognizers. These two new concepts are indicated at the bottom of Figure 6.33.

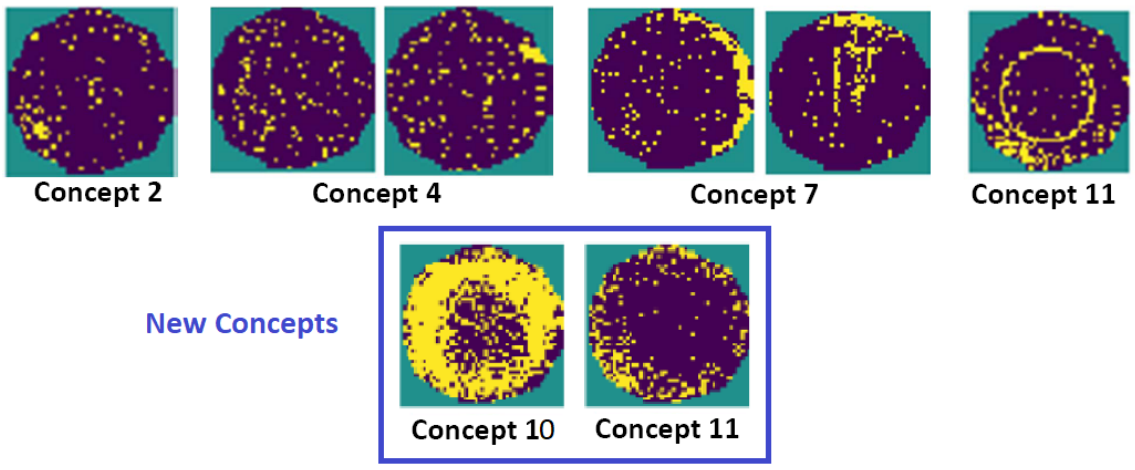

Figure 6.33: Additional false positive examples.

6.4.3 Implementation of the Checking Box

Following the Tensor modeling method discussed in Section 6.1.10, 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 i, the software maintains a GANs recognizer C_i and a corresponding Tensor model T_{C_i}. Let S_{C_i} be the set of wafers recognized by C_i, and let $S_{T_{C_i}}$ be the set of wafers classified by T_{Ci} as in-class to Concept i. The general idea for the Checking box is that it implements a *containment check* to see if $S_{T_{C_i}} \subseteq S_{C_i}$. This containment check is considered failing if more than one wafer that is in $S_{T_{C_i}}$ but not recognized to be in 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 [76].

Recall Equations 6.3 and 6.4 from Section 6.1.10 for the similarity metrics Δ and Δ' respectively. Let S and S' be the set of wafers in a concept bin based on Δ and Δ', respectively. The set $S \cap S'$ is used by the containment check.

Table 6.3: Containment Set Generation - First 50 lots

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

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 6.4: Containment Set Generation - Remaining 287 lots

Concept	1	2	3	4	5	6	7	8	9	10	11
1^{st} **Binning,** S	22	69	4	0	34	22	5	97	25	23	43
2^{nd} **Binning,** S'	18	58	4	1	15	41	3	48	22	22	37
Intersection Set $S \cap S'$	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 6.34 shows examples binned into a Concept i by the 1^{st} binning but not by the 2^{nd} binning, i.e. for those cases where there is a difference between the 1^{st} binning number and the intersection number. Comparing to Figure 6.29 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.

6.4.4 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

Figure 6.34: Examples in 1^{st} binning set S but not in 2^{nd} binning set S'

recognizers C8 and C9 fail because they have more than one escapes from the containment check set. Figure 6.35 shows the 7 escapes from C8 during the containment check.

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

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.

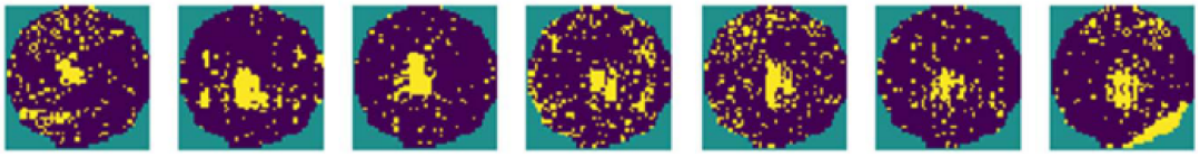

Figure 6.35: The 7 escapes by the GANs recognizer C8 for Concept 8.

6.4.5 Improving the deployed software solution

Jay is notified of the issues regarding Concept 8 & 9 in the deployed solution. Due to confidential information about the production, Nik cannot send Jay raw wafer data to help Jay improve the deployed software. Instead Nik employs tensor-based methods and sends Jay transformed training data extracted for Concept 8 & 9, which effectively

provides Jay with data for harder learning problems than the original. For more details regarding this transformation, please see [83].

Upon receiving the data, Jay makes the first empirical improvement by modifying the GANs architecture, adding in another CNN layer and reduced the size of a fully-connected layer. Figure 6.36 shows the improved separability for Concept 8's training by using the new architecture. In additional, Jay also sees from each training run in Figure

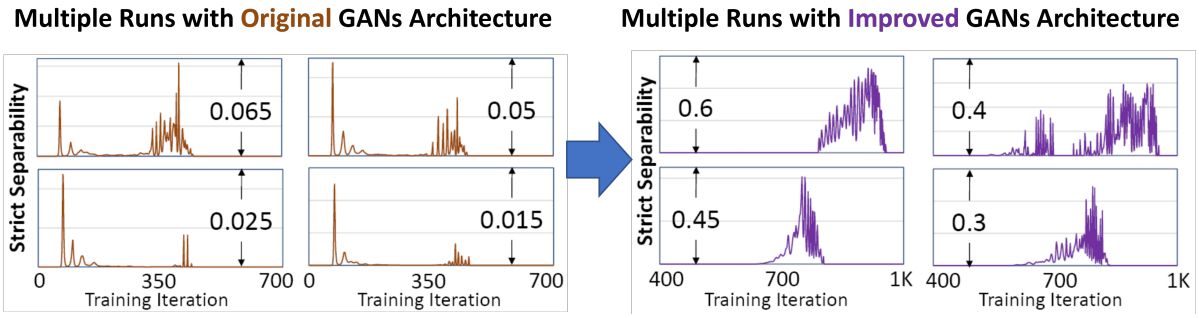

Figure 6.36: Improved separability seen in Concept 8 training.

6.36, there seems to be some variability due to the statistical nature of the training process. Specifically, Jay is concerned about the stability of the training. This motivated Jay to adopt *ensemble learning*, where the best 50 recognizers, in terms of separability, out of 100 trained are kept to perform recognition collectively. For more information on the ensemble learning considerations, please refer to [83].

6.4.6 Results from re-deploying the improved software

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.

For Concept 9, there is still one false positive and one escape from the containment check. They are actually different wafers and Figure 6.37 shows what they look like.

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	$\surd$	$\surd$

$\surd$: model passing the containment check

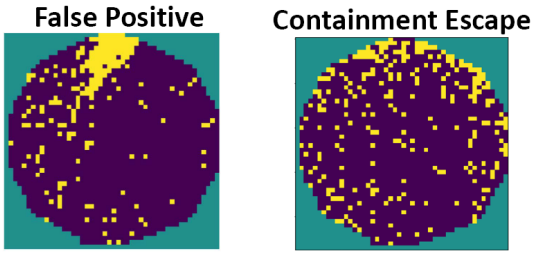

Figure 6.37: The false positive and escape for Concept 9 in Table 6.

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 50 Concept 9 recognizers.

The software is also run on a 2^{nd} 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 2^{nd} product is much larger than the die size of the 1^{st} 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, where Figure 6.38 shows two examples of their five training wafers to illustrate what they look like. The table in Figure 6.39 summarizes the results from the recognition and containment check. Notice that even though many wafers are not filtered out by the

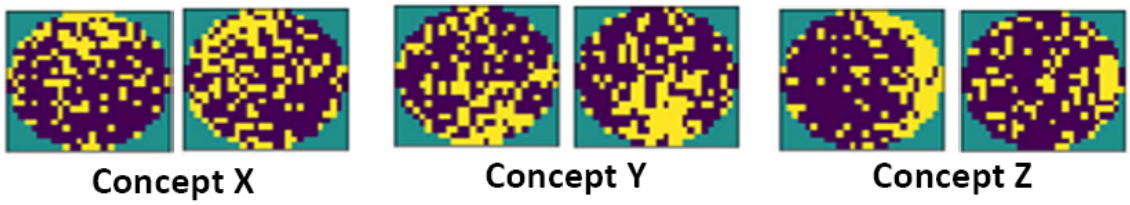

Figure 6.38: Example wafers from Concepts X, Y, and Z.

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 6.39 also shows what the false positives look like, to confirm there is no systematic surprise in the result.

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	√	√	√

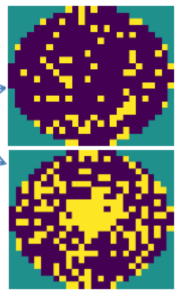

Figure 6.39: Summary table, the false positive and the escape.

6.5 Summary

In this chapter, we demonstrate two capabilities raised in Section 5.4. One capability is being able to filter out wafer maps without a pattern. Specifically, we describe a methodology to generate in-class and out-of-class samples to facilitate the training of a "none" pattern recognizer. In addition, we stress using rules to check the filtered out wafers to ensure they indeed have no pattern, where two examples rules are demonstrated.

The primary focus of this chapter remains on the second capability, which is obtaining a lower-bound clustering model of wafer samples. This capability was enabled by the development of tensor-based recognizers, which leveraged properties of tensor decomposition. We show that tensor-based recognizers are able to cluster very similar wafer samples as the same wafer type, where this cluster can be used as a containment check to ensure the robustness of a deployed DSML solution. By being able to utilize the containment property of tensor-based results with respect to GANs-based results, we can infer tensor-based methods can produce lower-bound models. In addition, the use of tensor-based methods facilitated the 5 training samples requirement for the deployed GANs-based DSML solution.

Chapter 7

Per-Sample-Based Recognizers

"Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less."

— Marie Curie

In the last chapter, we demonstrate tensor-based methods can obtain lower-bound models for the WMPR dataset. However, we cannot also expand to an upper-bound model based on the tensor-based methods. We also described a methodology to train a "none" pattern recognizer in the previous chapter. But this methodology is hard to generalize, as we would need to considered many types of unseen failures when generating samples for this recognizer. Furthermore, the above capabilities are just two out of the total four mentioned in Section 5.4. If we wanted to enable all four capabilities, what is the essential approach? Looking at the GANs-based models from the previous chapter, when building a recognizer, it has a requirement of needing at least 5 training samples. Ultimately, if we take this to the extreme, we need to be able to build a recognizer based on a single wafer sample. This allows us the maximum flexibility to accomplish

our requirements discussed in Section 5.4. In this chapter, we will demonstrate how to realize this per-sample-based recognizer. In the following chapter, we discuss how to leverage this flexibility and provide some initial application examples.

7.1 The Learning Problem

The learning problem can be stated as the following: We are given a single sample s exhibiting a *feature* of interest. The goal is to build a model M_s such that given future samples $s_1, \ldots, s_n$, $\forall s_i$, $1 \leq i \leq n$, $M_s(s_i)$ would return "yes" if s_i exhibits the same feature as s.

Figure 7.1: The goal is to learning a recognizer with one training sample.

In machine learning, a closely-related problem is *one-shot learning* [84] which also intends to learn with one training sample. One-shot learning is a form of *transfer learning* [85]. To apply one-shot learning, we start with a model $M()$ which is already learned to classify wafer plots into different categories based on their feature. The single training sample s is given to represent a *new* category. Then, the model $M()$ is enhanced based on the new sample.

In our work, the requirement is different though. For each category, we intend to build a recognizer independent of other recognizers. Hence, the learning has to "start from scratch". A practical implication of independent models is being able to facilitate debugging and maintenance of the recognition software. If there is a deficiency in recognition of one category of plots, we can simply focus on that particular recognizer without

concerning other recognizers.

In the above problem statement, the exact meaning of the word "feature" is to be defined by a learning algorithm. Typically, s_i and s are treated as having the same feature if they are *similar*. However, different algorithms might measure this *similarity* differently.

A learning algorithm of this sort essentially comprises two elements. The first can be thought of as a projection method that maps each sample to a score. For example, a score indicates how close a given sample s_i is to be considered as the same category of the training sample s. For the most part, this projection method is the learning algorithm. However, just obtaining the scores is not enough. To decide a sample s_i should be treated as in-class or out-of-class, a *threshold* would also be needed. In practice, threshold can be decided by extensive experiment. However, this is only feasible if sufficient data are available. If not, the problem can become challenging.

7.2 Threshold Decision Problem

With only one training sample, it is not surprising that threshold selection is a challenge. Take the wafer plot shown in Figure 7.1 as an example. To accomplish the first element, i.e. the projection method, we can simply choose a *similarity measure* that computes a similarity score between two wafer plots. For example, suppose each plot is represented as a 48×48 matrix. Each entry contains one of the three values -1 (purple), 0 (green), and $+1$ (yellow).

Given two wafer plot matrices W_1, W_2, a simple way is to use their square distance, $\| W_1 - W_2 \|^2$, i.e. subtracting each entry individually and take the sum of their square. Another way is to focus on the yellow entries ("+1" entries) where the feature is defined. First we replace every "−1" with "0" to make the matrix binary. Let U be the number

of "1" entries in $W_1 \vee W_2$. U captures the size of the union set of the yellow dots. Then, let I be the number of "1" entries in $W_1 \wedge W_2$. I captures the size of the intersection set of the yellow dots. We let the similarity score be $\frac{I}{U}$. We use $IoU(W_1, W_2)$ to denote this score.

In our dataset, there are six plots with the "ring" feature. These six plots are shown in Figure 7.2. We treat them as the *in-class* samples. Then, if any other plot is considered by a recognizer as in-class, it is treated as a false positive.

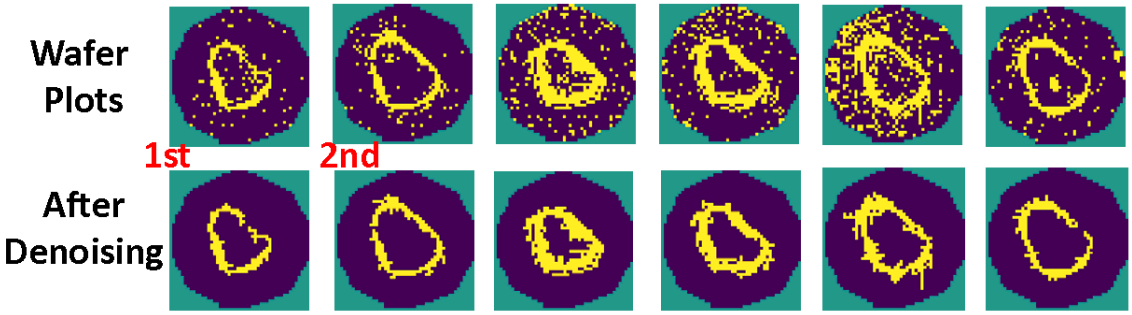

Figure 7.2: The six plots with "ring", before and after denoising.

Before applying $IoU()$, each wafer plot goes through a denoising step which removes random and scatter yellow dots while maintaining the feature. This is typical in image processing. Figure 7.2 shows the images before and after the denoising step. In the rest of the paper, all wafer plots shown have been denoised.

Let W be the first wafer plot shown in Figure 7.2. Suppose it is the one training sample given to us. First, we have $IoU(W, W) = 1$. For any other W_i, unless $W_i = W$ exactly, we have $IoU(W, W_i) < 1$. The minimum IoU score is 0.

Suppose we have 25 other wafers to help in finding a threshold. Figure 7.3-(a) shows the IoU values for the 25 wafers. The values are shown along the y-axis. Note that for ease of visualization, the blue dots are spread out along the x-axis direction which has no physical meaning.

Suppose we know all blue dots are out-of-class samples. There can be two (reasonable)

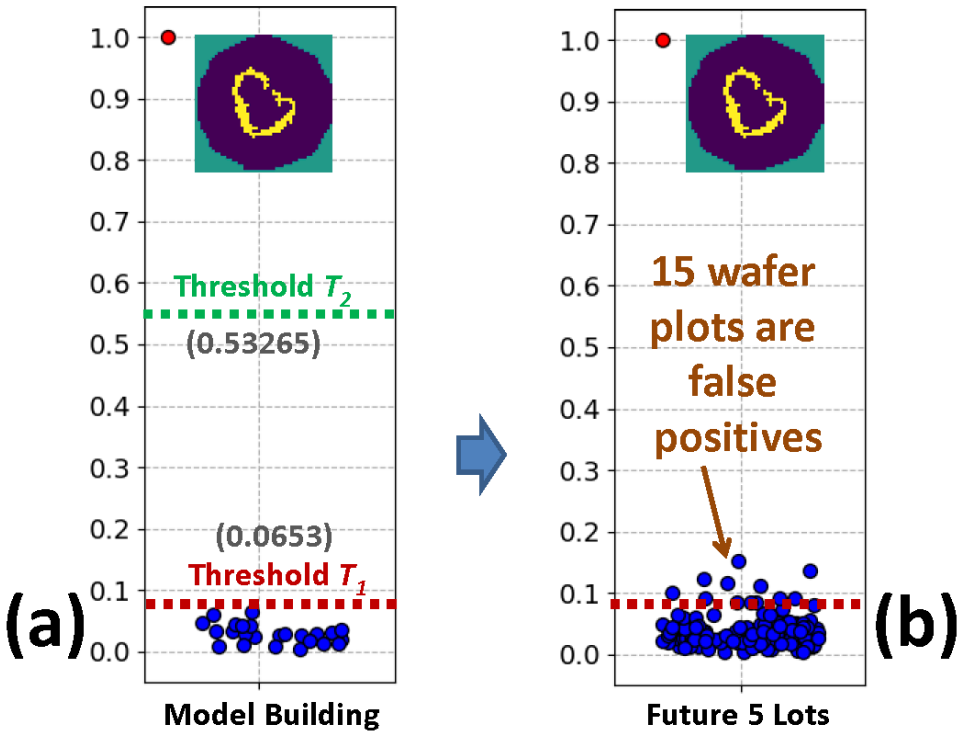

Figure 7.3: Model building: Selecting a threshold.

options to set the threshold. Threshold T_1= 0.0653 is set by the blue dot with the highest score. Another option is to set the threshold at the middle point, i.e. T_2= $\frac{1+0.0653}{2}$ = 0.53265. Note that setting a threshold close to the red dot is not an option in this plot because we would not know how close is "close".

Suppose we apply the model to check on future 125 wafers where none is in-class. Figure 7.3-(b) shows the result. If T_1 is taken, it will result in 15 false positives. If T_2 is taken, it will have no false positive. But result in Figure 7.4 below shows that taking T_2 will miss all the in-class plots.

Suppose the second wafer plot in Figure 7.2 is also available to us. Figure 7.4-(a) shows its IoU score. This score, 0.1823, now can be our third threshold option, T_3. The figure shows that if a different set of 125 wafer plots is considered, T_3 would result in 12 false positives.

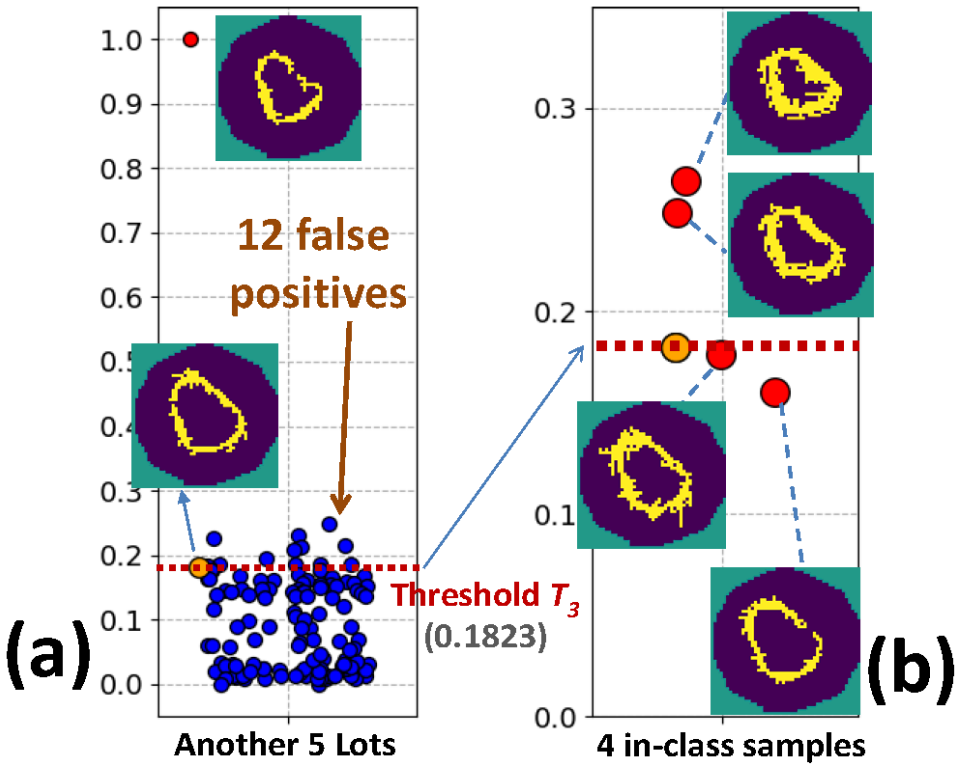

Figure 7.4: What if we add a second training sample.

Figure 7.4-(b) shows the IoU scores for the remaining four in-class plots. Observe that even with T_3, the model would have treated two in-class plots as out-of-class (below the threshold line). If threshold $T_2 = 0.53265$ from Figure 7.3 is used, the model would have missed all in-class plots.

The above example illustrates the challenge for deciding a threshold when limited data is available. Unless the data show where the highest blue dot (out-of-class) or the lowest red dot (in-class) might be, there seems no effective way to optimize a model against false positive or false negative.

In the above example, we assume there are out-of-class samples to help decide the threshold. Even this assumption is considered unrealistic in our work. Suppose our software is deployed already. Then, the learning depicted in Figure 7.1 needs to be carried out online. Without acquiring labels for other samples through other means,

the software would not know if they are in-class or out-of-class, i.e. the learning cannot simply treat unlabeled samples as out-of-class.

7.3 Manifestation Learning

Suppose a sample s is given to represent a concept in domain $\mathscr{A}$. The idea of *Manifestation Learning* is that, we would try to recognize the concept by using a pre-trained recognizer M_b for a different concept in a different domain $\mathscr{B}$. The recognizer M_b is trained with a dataset originated from the alternative domain, where there are plenty of training samples available.

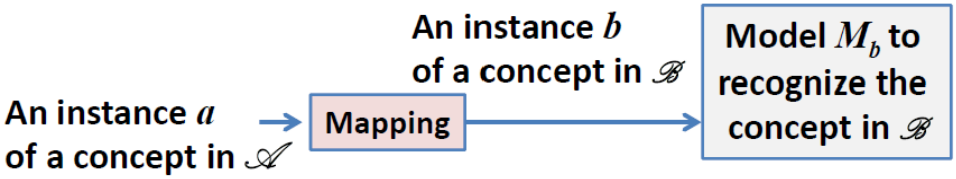

Figure 7.5: The basic idea of Manifestation Learning.

Figure 7.5 illustrates the idea. Given an instance to be recognized for a concept in $\mathscr{A}$, we simply *manifest* the instance into a concept in $\mathscr{B}$ domain and use M_b for the recognition. To realize this idea, at the minimum, it requires building a mapping function that maps every instance in $\mathscr{A}$ to an instance in $\mathscr{B}$. Note, like one-shot learning, manifestation learning can also be thought of as a special form of *transfer learning* [85].

7.3.1 The Manifestation Space

Recent advances in the AutoEncoder (AE) approach [64], in particular the Variational AutoEncoder (VAE) [32], provide a means for realizing the proposed idea. An AE has three essential components: an encoder function, a probabilistic decoder model, and a coded *latent space*.

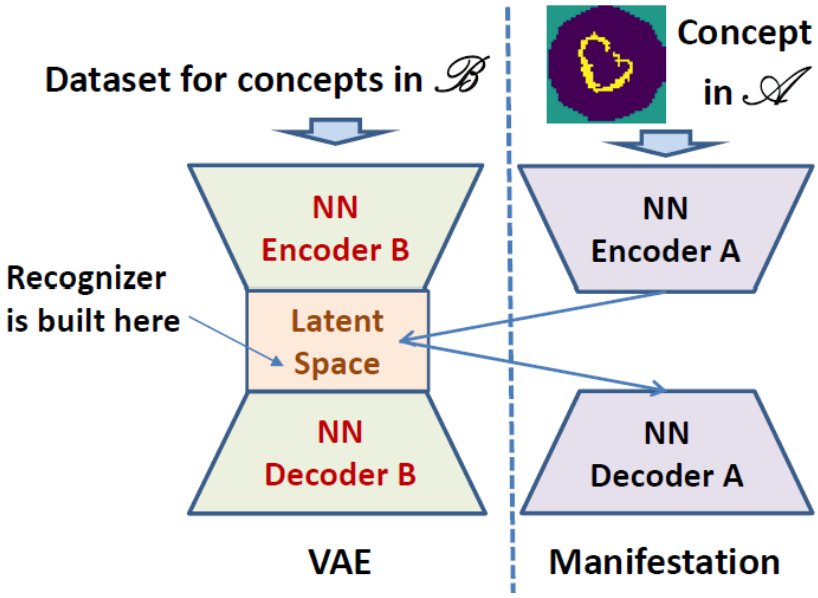

Figure 7.6: Setup for Manifestation Learning.

Figure 7.6 illustrates our setup to realize the manifestation learning idea. Given a dataset for learning about a concept in $\mathscr{B}$ (which contains many in-class and out-of-class samples), an encoder and a decoder are trained. The encoder and decoder in the setup are both neural network models. For an input sample x, the encoder maps x onto a point in the latent space. This point essentially can be seen as a *code* $\vec{z}$ for x. For example, if the latent space is defined with 16 dimensions, $\vec{z}$ is a vector of size 16. Then, the decoder reconstructs the input x from the code $\vec{z}$. A main training objective is to minimize the reconstruction error.

After the training, each sample in the dataset is mapped to a code in the latent space. In manifestation learning, one sample s for a concept in $\mathscr{A}$ is given. This sample is used in training an encoder and a decoder separately. This manifestation encoder maps s to a selected code in the latent space. This selected code corresponds to a particular sample from a selected concept in $\mathscr{B}$. The decoder then reconstructs the sample from the code. The training objective for the decoder is also to minimize the reconstruction

error. However, the training objective for the encoder is different from before, i.e. now the objective is to map s to a selected code.

For a selected concept in $\mathscr{B}$, its recognizer is built in the latent space where each sample is represented as a vector. The recognizer can be built using a traditional learning technique such as SVM [17]. More importantly, there are sufficient data for learning the model. Detail will be discussed later in Section 7.4. In Figure 7.6, we call the latent space as a *Manifestation Space*. This is to differentiate our use of the space from that of traditional AE and VAE.

7.3.2 VAE implementation

VAE replaces the deterministic encoder function in AE with a posterior model $q(\vec{z}|x)$. The effect is that, instead of learning a code for a sample as a vector in the latent space, VAE learns codes represented as probabilistic distributions in the space. When each sample is represented as a probability distribution, effectively the latent space becomes continuous. This enables information to be learned between two sample points and allows samples to be drawn randomly from the latent space.

Note that our VAE implementation did not follow the original VAE [32] which uses KL divergence to regularize learning of the latent space. As indicated in a number of works [86][87], this can lead to an uninformative latent code and also the tendency to over-fit samples, especially on a small dataset. We use the InfoVAE proposed in [88] which uses Maximum Mean Discrepancy (MMD) [89] to measure divergence and regularize the latent space.

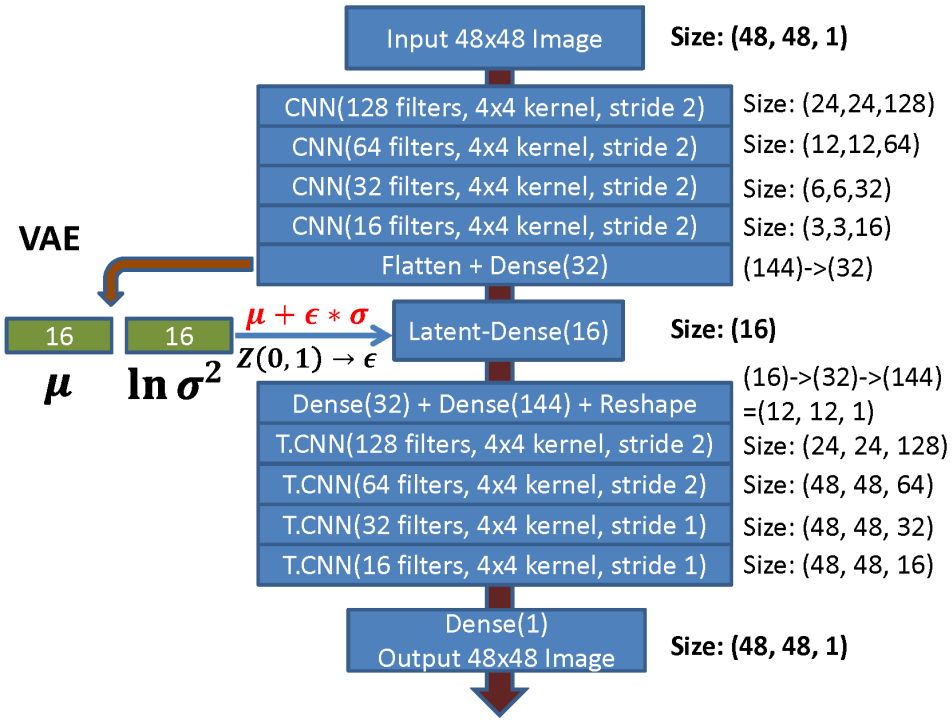

Figure 7.7: Convolutional Neural Networks for VAE.

7.3.3 The neural network architecture

Figure 7.7 illustrates the architecture of the neural networks for implementing VAE. Convolutional neural networks (CNN) are used to implement the encoder and decoder. The encoder comprises four convolutional layers denoted as "CNN()". The decoder comprises four transposed convolutional layers denoted as "T.CNN()". The input image is a $48 \times 48 \times 1$ tensor, i.e. a 48×48 image with one channel. Unlike a RGB image with three channels, our image has one channel with three possible values, -1, 0, and $+1$.

The first CNN layer essentially converts the input image into a 24×24 image with 128 channels, i.e. a $24 \times 24 \times 128$ tensor. Each 24×24 matrix is the result of applying a *filter* on the original input image. Each filter uses a 4×4 kernel (window) to "scan" the input image. The stride length is 2, i.e. during scanning the window moves every 2 pixels. The effect is to reduce the original image size by half.

Another way to understand the CNN layer's operation is by thinking in terms of a *convolutional matrix* [90], which is defined by the kernel and the stride length. In the example, the convolutional matrix can be denoted as $C_{576\times2304}$ where $576 = 24 * 24$ and $2304 = 48 * 48$. The 48×48 input image matrix is flatten into a column vector $\vec{v}_{2304}$. Then the convolution layer computes the product $C_{576\times2304} * \vec{v}_{2304}$. This product is a vector of 576 values which can then be reshaped into a 24×24 matrix.

Similarly, the other three CNN layers below the first CNN layer further reduces the image size, from 24×24 to 12×12, then to 6×6, and then to 3×3. The number of channels is equal to the number of filters used in each layer. Each filter is a $i \times i \times k$ tensor where $i \times i$ is the kernel size and k is the number of channels from the previous layer. After four CNN layers, the result is a 16-channel 3×3 image, i.e. a $3 \times 3 \times 16$ tensor. This tensor is flatten into a vector of size 144. The last layer of the encoder is a fully connected (FC) layer that connects 144 inputs to 32 outputs.

The latent space has 16 dimensions. If it were AE, another FC layer would have been used, to reduce the size from 32 to 16. In training, each sample would be mapped into a vector of size 16 in the latent space. In VAE, two 16-dimensional vectors are used, denoted as μ and $\ln \sigma^2$. In each training cycle, a multivariate constant ϵ is randomly drawn from the distribution $Z(0, 1)$, the 16-dimensional Gaussian distribution with mean 0 and standard deviation 1. Then, $\mu + \epsilon * e^{\frac{\ln \sigma^2}{2}}$ is used as the resulting vector in the latent space (Note that $e^{\frac{\ln \sigma^2}{2}} = \sigma$). Effectively, each sample would correspond to a probability distribution in the latent space. This distribution is assumed to be Gaussian, represented by its mean vector μ and the vector $\ln \sigma^2$. Note that the reason to use $\ln \sigma^2$ instead of its standard deviation σ directly is a trick to enable the training [32].

The decoder takes a 16-dimensional vector from the latent space and produces a 48×48 image. First, it uses a 16-to-32 FC layer, followed by a 32-to-144 FC layer to expand the 16-dimensional vector to a vector of size 144. The vector is then reshaped

into an 12×12 image with 1 channel. A transposed convolutional layer is similar to a convolutional layer except that instead of using a convolutional matrix, a T.CNN uses the transpose of a convolutional matrix.

For example, in CNN suppose we multiply a convolutional matrix $C_{144\times576}$ to the flatten vector of a 24×24 matrix. This gives a vector of size 144, equivalent to a 12×12 matrix. In T.CNN we would multiply $(C_{144\times576})^T$ to the flatten vector of a 12×12 matrix. This gives a vector of size 576, equivalent to a 24×24 matrix. The last T.CNN layer produces a 48×48 image with 16 channels. A FC layer is then used to convert the 16 channels into 1 channel. In VAE training, the reconstruction error, i.e. the difference between the input image and the output image, is minimized.

7.3.4 An example of Manifestation Space

TensorFlow [91] is used to implement the neural networks shown in Figure 7.7. One of the datasets we experimented was from the MNIST handwritten digit database [22]. The dataset comprises samples for individual digits "0" to "9". In our study, for each digit we had >5000 samples. Figure 7.8 shows ten examples for digits "0", "1", and "2".

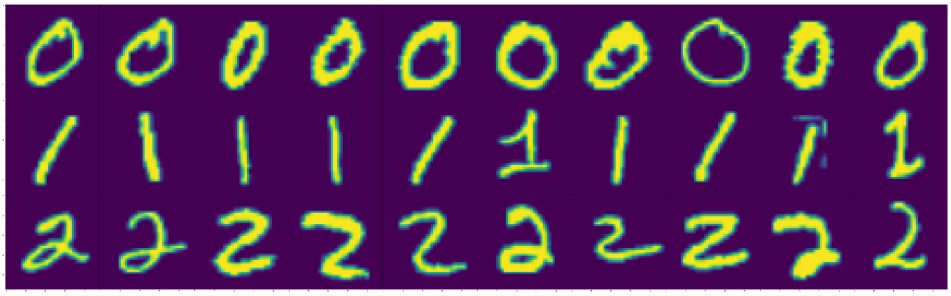

Figure 7.8: Examples of "0", "1", and "2" from the MNIST dataset.

We took 5000 samples for each of the three digits to form a dataset of 15K images. We trained a VAE model with these 15K images. We were interested in how the samples distributed in the latent space. The latent space has 16 dimensions. Hence, we used the

t-SNE technique [92] to project the latent space onto a 2D space for ease of viewing. Figure 7.9 shows the result. For clarity, only 100 samples from each class are randomly selected to be shown. It is interesting to observe that samples from each digit exhibit a cluster in the latent space.

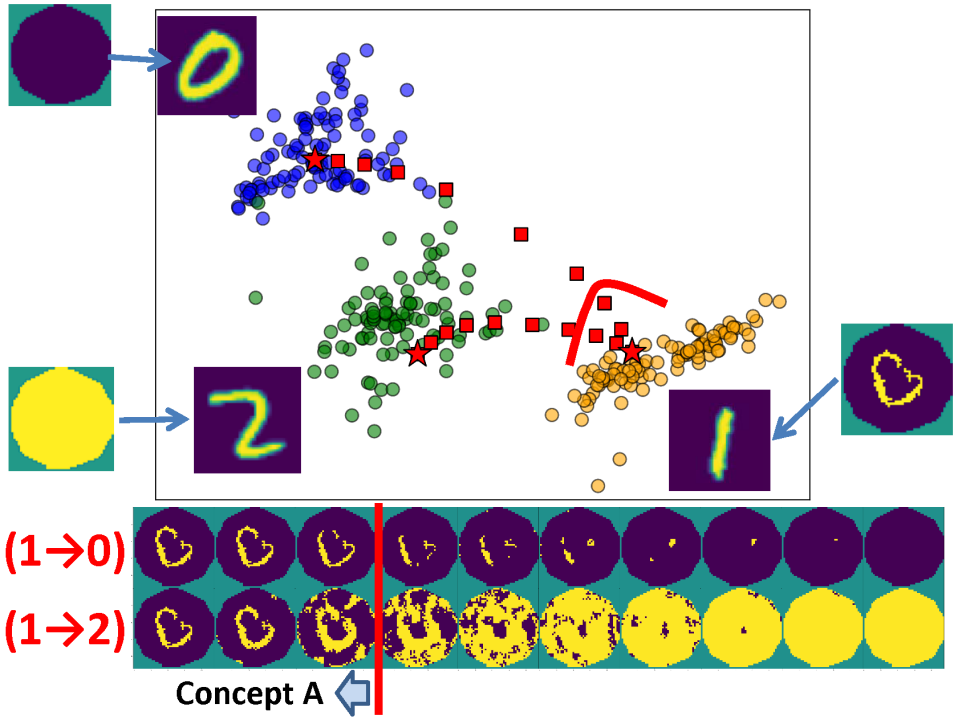

Figure 7.9: A Manifestation Space example.

Recall that in VAE, each sample in the latent space corresponds to a probability distribution. Hence, it is important to note that the points shown in Figure 7.9 is the result by randomly drawing an ϵ (see Figure 7.7) for each sample. As a result, if we tried to re-generate the plot in Figure 7.9 with another run, the exact location of a sample's point could change due to the random drawing. However, exhibition of the three clusters would always be there.

It is also important to note that during VAE training, the images were used without their label. Hence, the training is unsupervised. Therefore, the clustering property shown

in Figure 7.9 is the result of the VAE training. Note that this property might not be there if a different dataset or a different learning setup is used. Later in Section 7.4 we will discuss this property in more detail.

Suppose we want to map a wafer plot to a sampled point in the latent space. As shown in Figure 7.6, we train a separate encoder and decoder (Encoder A and Decoder A in the figure). As shown in Figure 7.9, this is done for three wafer plots: the first "ring" wafer plot shown in Figure 7.2, a generic no-fail ("blank") plot, and a generic all-fail ("full") plot. Each plot is mapped to the "center" of a digit cluster. In Figure 7.9, the centers are marked by a red star "⋆".

To check how well the manifestation decoder works, Figure 7.9 also shows the sequence of images produced by two *walks*. The first walk moves from digit "1" to digit "0". Observe that the resulting images gradually change from "ring" to "blank". The second walk moves from digit "1" to digit "2". Also observe that the resulting images gradually change from "ring" to "full".

In the two walks, observe that the first three plots all exhibit the "ring" feature. They can all be treated as examples of concept $\mathscr{A}$, i.e. concept "ring". In the latent space, those "ring" plots correspond to points closer to the center of the digit "1" cluster. If we could define a region around the digit "1" cluster, in Manifestation Learning this region could also be used to define a region for concept $\mathscr{A}$. Then, recognizing concept $\mathscr{A}$ becomes simply to check if an input image, after going through the manifestation encoder, results in a point inside the concept region or not.

7.3.5 Another example of Manifestation Space

Before we discuss how to model a *concept region* in a manifestation space, in this section we show another example of manifestation space. This example shows some

generality of the Manifestation Learning idea.

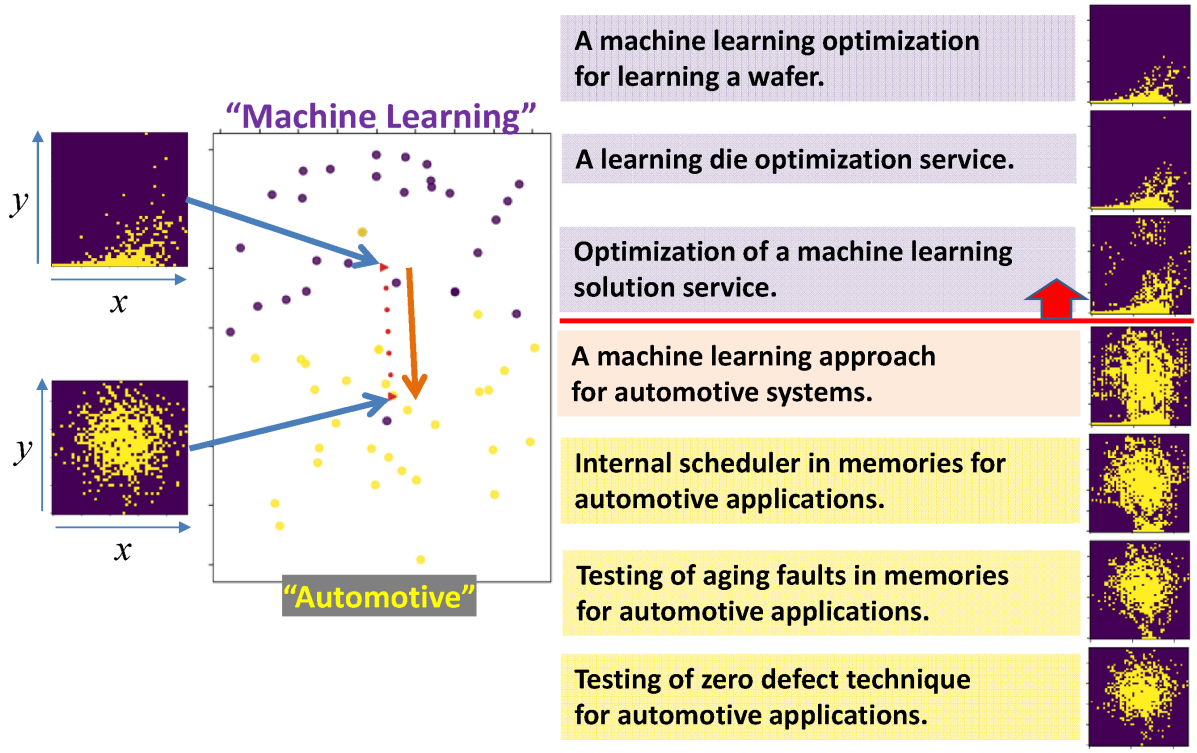

Figure 7.10: A sentence-based manifestation space.

Figure 7.10 shows a manifestation space learned based on 60 paper titles from past International Test Conference (ITC), where 29 titles are categorized as "Machine Learning" and 31 titles are categorized as "Automotive" (two tracks of ITC in recent years). We use the VAE approach proposed in [93] to train a VAE model, where the CNNs in Figure 7.7 are replaced with LSTMs [94].

Then, we take two plots for training the manifestation encoder and decoder, which follow the network architecture shown in Figure 7.7. The first plot (on the top) is the same correlation plot shown as Figure 3-(a) in our previous paper [77]. The x-axis is the average e-test value and the y-axis is the number of fails. Each yellow dot represents a wafer lot. It is a correlation plot example used in [77] to show that, as the e-test value becomes larger, it tends to result in more fails. The second plot is a manually-created,

generic counter example showing no correlation between x and y.

The manifestation encoder is trained to map the correlation plot to the center of the "Machine Learning" cluster. The no-correlation plot is mapped to the center of the "Automotive" cluster. As before, by walking a path from "Machine Learning" to "Automotive" in the manifestation space, we can generate various plots using the manifestation decoder. In addition, we can generate various paper titles using the VAE's decoder. Note that in the figure, none of the generated titles is the same as the original titles used in training the VAE.

It is interesting to observe that the top three titles are about "Machine Learning". The bottom three are about "Automotive". The middle one is a hybrid. Also observe that the top three plots all show a similar correlation trend. Suppose we have a model that can recognize a paper title of "Machine Learning" category. Then, this model could also be used to recognize plots with a correlation trend.

7.4 The Concept Region

To model a *concept region* in a manifestation space, we need to decide on the following four aspects:

- (*Dataset*): We need to choose a labeled dataset that comprises multiple classes of samples, where for each class there are sufficient samples for the VAE training.
- (*Learning Setup*): To learn a model in the manifestation space, we need to choose to employ a supervised learning or an unsupervised learning setup.
- (*Algorithm*): Based on the learning setup, we choose a particular learning algorithm.
- (*Target Class*): Through experiment, we choose a particular class as our *target class* for modeling the concept.

7.4.1 Dataset selection

To choose a dataset, we considered various options. Because our goal is to capture a concept based on a wafer plot, it would seem intuitive to consider using a dataset comprising multiple classes of wafer plots. For example, the work in [83] shows 11 classes of wafer plots found in the dataset with 8300 wafers. After investigating the possibility, we decided not to use the dataset because the numbers of wafer plots across the 11 classes differ significantly. Some have much fewer samples while the largest class has over a hundred. The imbalance and insufficiency in terms of the number of samples are of the major concern.

One crucial property considered for a manifestation space is that multiple classes of samples exhibit clustering as that shown in Figure 7.9. This is important because it enables us to model a target class region more easily. We also considered using the CIFAR-10 dataset [95]. However, while we could train a VAE model with a good reconstruction accuracy, we found it more difficult to observe the desired clustering property in the latent space. This was true even with using only two classes of samples.

For example, Figure 7.11 shows the latent space with 100 samples each from two classes of CIFAR-10: cat and automobile. The training is based on 5000 samples in each class. The reconstruction accuracy is 96.002%. The dimension of the latent space is set at 1024 to achieve the accuracy. As seen, points from the two classes show no clustering. Figure 7.10 shows another option based on a sentence-based manifestation space. While it is interesting, we found that training with the sentence-based VAE [93] could be much trickier than training a CNN-based VAE. After all the experiments, we settled with the MNIST handwritten digit dataset [22]. The dataset we used comprises 10 classes of image samples. Each class has 5000 samples. The dataset is balanced and also has a sufficiently large number of samples from each class.

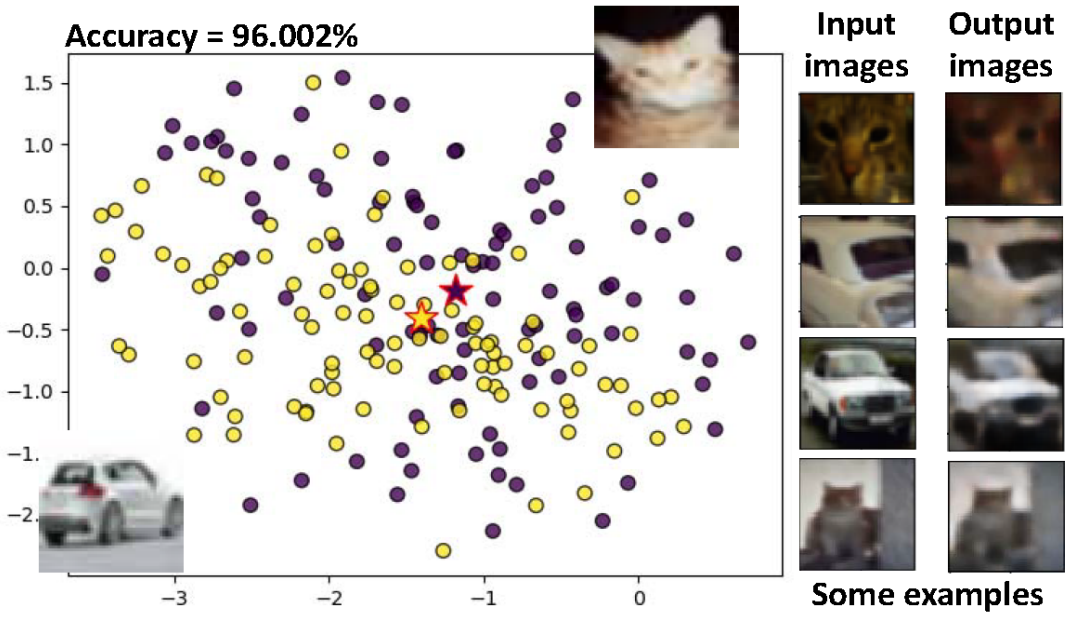

Figure 7.11: Latent space shows no clustering with CIFAR-10 samples.

Similar to Figure 7.9, Figure 7.12 shows the 2D projection plot of the latent space. Again, the figure shows 100 samples for each digit. As seen, samples of the same digit are clustered. In terms of the clustering property, this was the best result observed in all of our experiments. Hence, we chose this space as our manifestation space.

7.4.2 Learning setup and the algorithm

Each sample in the MNIST dataset is labeled by its digit. From the manifestation space, we can obtain a dataset where each sample is represented as a vector of size 16. Each vector has a label and hence intuitively, to learn a concept region for a target class, say digit "1", we can treat it as a supervised binary classification problem, i.e. to treat "1" as one class and the rest of digits as the other class.

After some initial study, we decided not to follow this approach. The main reason is that we desire our model for a concept region to have no false positive (This property will be utilized later in Section 7.6). A supervised learning algorithm usually is not designed

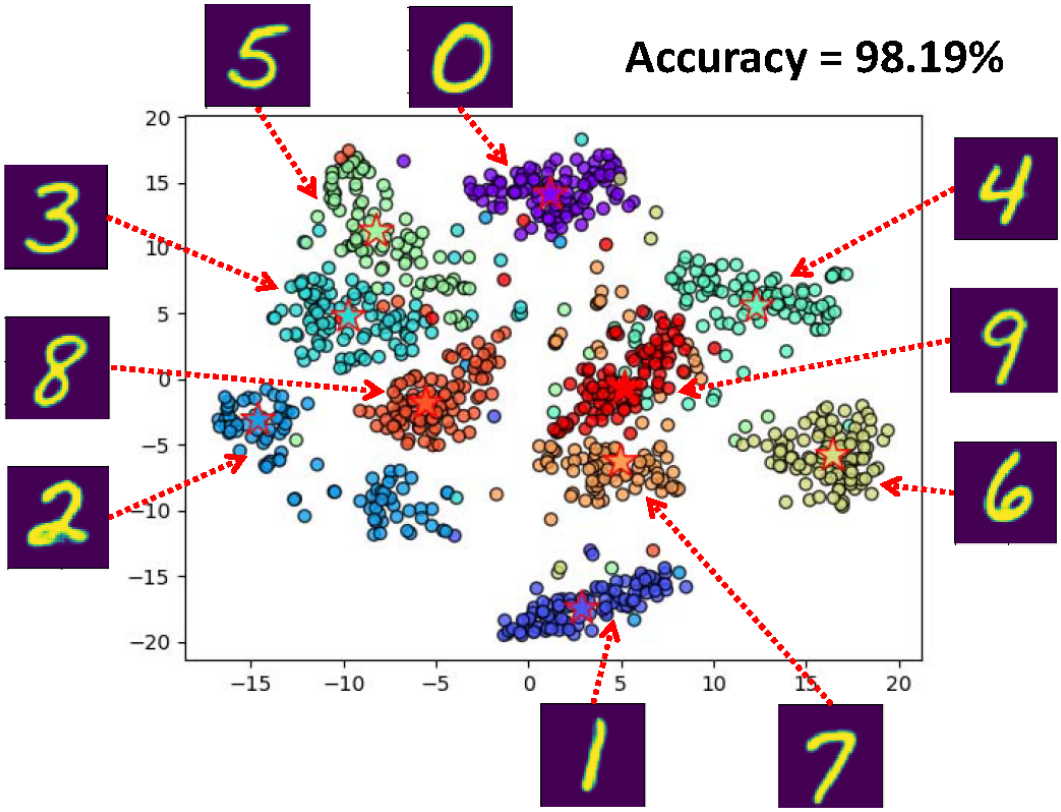

Figure 7.12: Clusters in the manifestation space with MNIST samples.

to ensure no error with respect to a given class. Hence, in our implementation we chose to treat it as an unsupervised learning problem.

Suppose we want to learn a model to capture the concept region for the digit "1" in the manifestation space. With an unsupervised setting, when learning the model we would only use the samples of digit "1". The learning algorithm we chose was the SVM one-class algorithm [24], in particular the implementation from the Scikit-Learn package [96].

To run SVM one-class, user needs to supply value to the parameter called ν ("nu"). The ν parameter provides an upper bound on the fraction of samples that are classified as out-of-class by the model. Note that in an actual implementation, this bound is not strictly followed because of the use of a soft margin, i.e. if a sample falls outside but

very close to the decision boundary (within a margin), it would not be considered as a violation of the bound.

For example, Figure 7.13 shows the result by running SVM one-class on the 5000 samples of digit "1". We set $\nu = 0.01$. This means that the model should classify no more than 50 samples as outliers. The model comprises two elements: (1) for each sample a *score* is calculated, and (2) a threshold is provided to classify samples into in-class (above the threshold) and out-of-class (below). Note that in SVM this threshold is always set at the value 0. However, the Scikit-Learn implementation moves this threshold by a positive bias to facilitate plotting with a log scale.

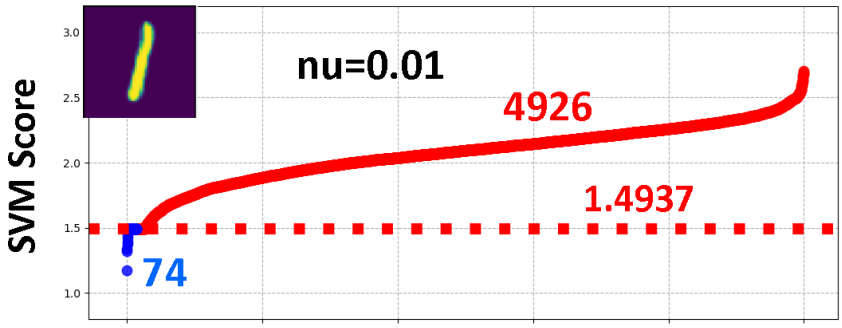

Figure 7.13: Resulting SVM scores on digit "1" samples.

As seen in Figure 7.13, 74 samples are considered out-of-class, i.e. below the threshold. In our context, we can interpret this as a recognition model capturing a concept region that includes 4926 samples of digit "1".

Next, we apply the digit "1" model to 5000 samples of digit "0". Figure 7.14 shows the result. While the model does leave 4977 samples outside the concept region, it includes 23 samples as in-class. These 23 samples are false positives. As mentioned above, we desire no false positive for a model. Hence, this model would not work.

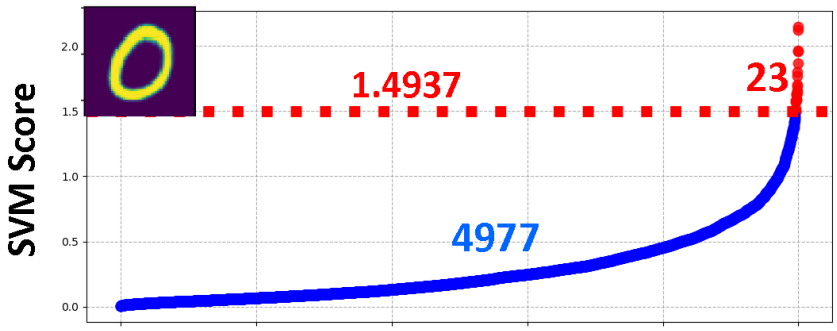

Figure 7.14: Applying the SVM model on digit "0" samples.

7.4.3 Target class selection

To search for a model that works, we use all the out-of-class samples as the *validation set*. The search is based on changing ν. We would search for a minimal ν value that results in a model with no false positive, i.e. the model treating all samples in the validation set as out-of-class. Figure 7.15 shows the result based on the model found with $\nu = 0.28$ for digit "1". The model includes 3599 samples of digit "1" as in-class. The model classifies *all* samples from other digits as out-of-class.

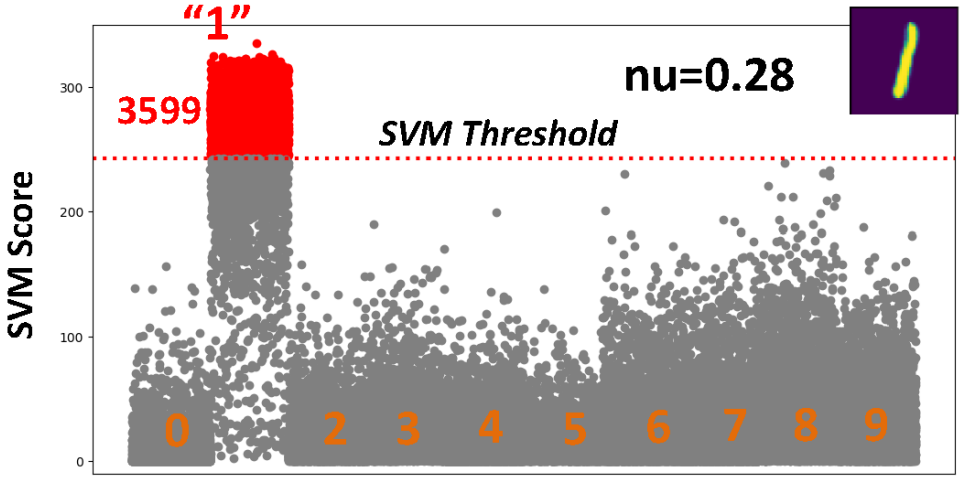

Figure 7.15: SVM scores by the digit "1" model, across all samples.

In contrast, Figure 7.16 shows the model for digit "0". In order to achieve no false positive, the ν is set at 0.5, resulting in excluding half of the samples of digit "0" from

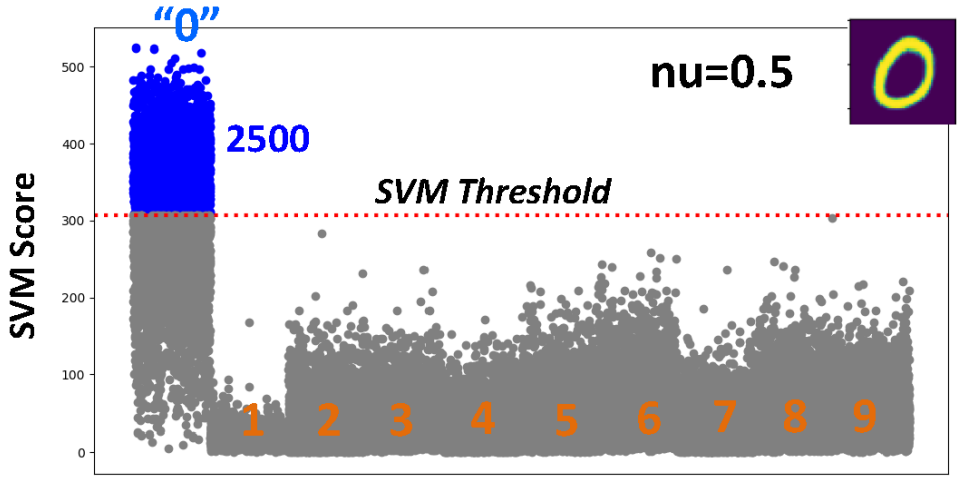

Figure 7.16: SVM scores by the digit "0" model, across all samples.

the concept region (below the threshold).

Table 7.1: The resulting ν for the ten digit classes

0	1	2	3	4	5	6	7	8	9
0.50	0.28	0.82	0.89	0.88	0.94	0.64	0.88	0.79	0.79

Table 7.1 shows the ν values found across the ten digit classes where the resulting model has no false positive. The search started from ν =0.01 with an increment of 0.01. The result shows that samples of digit "1" provide the best model, meaning that the model captures the largest number of in-class samples in the concept region.

This result is consistent to the fact that the cluster of digit "1" is more separated from others, which can be somewhat observed in Figure 7.12. Suppose for each cluster we calculate its "center" which is the average of all the sample vectors in the manifestation space. For each digit, we calculate the average (Euclidean) distance from its center to the other 9 centers. Table 7.2 shows such average distance for each digit. As seen, digit "1" has the largest average distance to others.

More separation of a cluster to other clusters enables building a better model, i.e. capturing more in-class samples inside the concept region. Because of this observation,

Table 7.2: Average distance to the other 9 digits

0	1	2	3	4	5	6	7	8	9
5.64	6.16	5.37	5.01	5.27	4.87	5.28	5.35	4.46	4.47

we chose digit "1" as our *target class* for manifestation. In other words, for a given concept $\mathscr{A}$ to be learned we would map its training sample onto a sample of digit "1" in the manifestation space.

7.4.4 Manifesting to a digit "1" sample

Refer back to Figure 7.6. To manifest the "ring" wafer plot (concept $\mathscr{A}$), we use the plot to train a manifestation encoder and a manifestation decoder. When training the encoder, we need to select a point in the concept region of digit "1". In our implementation, we select the point that has the largest SVM score. We train the encoder to map the wafer plot to the point. After that, we train the decoder to reconstruct the wafer plot from the point. After manifestation training, we then check how the resulting recognizer performs on the other 8299 wafer plots.

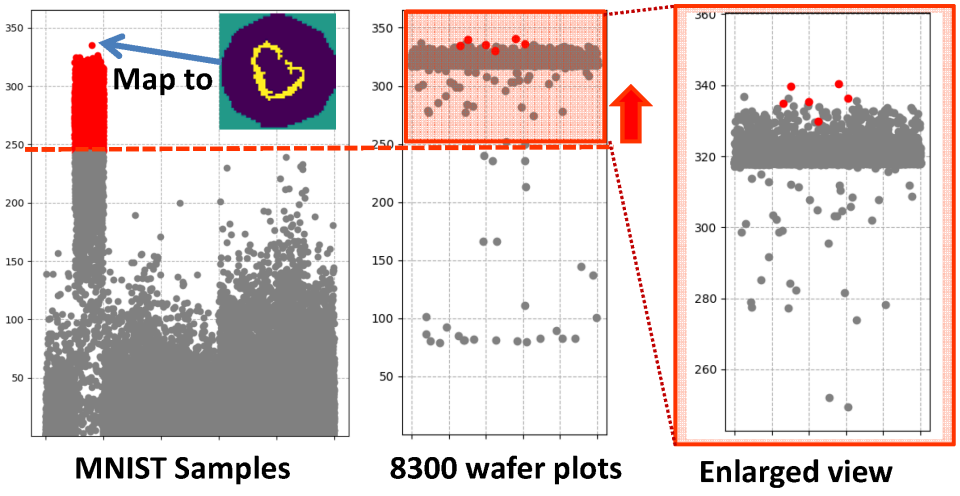

Figure 7.17: Initial manifestation that does not work.

Figure 7.17 shows the result for the 8300 wafer plots. Each wafer plot corresponds

to a point in the manifestation space. The SVM model calculates a score for the point. The six "ring" plots in Figure 7.2 are shown as six red dots "•". All others are shown as grey dots.

What we desire to see is that the six red dots are above the threshold and none of the grey dots are above it. In contrast, we see that majority of the grey dots are also above the threshold, an undesirable result. In the enlarged view, however, we do see that the scores of the six wafer plots are higher. The problem is that the scores for other plots are also high. Hence, we need to find a way to lower the scores for all the other plots.

7.5 Generic Counter Examples

Figure 7.18 shows three wafer plot examples and the corresponding plots generated by the manifestation decoder. It is interesting to see that regardless of the input image, the output image always shows a similar "ring" feature. As explained below, the problem, though, is not with the decoder, but is with the encoder.

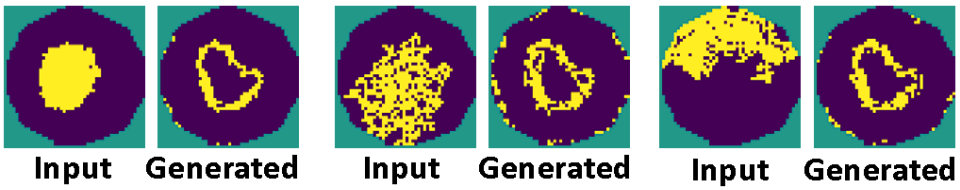

Figure 7.18: Checking the decoder's output - 3 examples.

The manifestation encoder is trained with only one wafer plot of concept $\mathscr{A}$. Hence, the encoder lacks information on where to map a wafer image different from concept $\mathscr{A}$. To improve the encoder's mapping, we use a *generic* counter example to differentiate from concept $\mathscr{A}$.

One generic counter example is the no-fail wafer plot, i.e. the "blank" wafer image. Figure 7.19 shows the result by adding this image in training the manifestation model.

The training now has two wafer images. The blank image is mapped to a point of digit "0" in the manifestation space. Instead of mapping to the point with the largest SVM score, it maps to the point with the medium SVM score. The reason "0" is chosen is because in the manifestation space, the distance between the center of digit "1" and the center of digit "0" is the largest (e.g. see Figure 7.12). Through experiment, we observed that the further away the digit we chose for this mapping, the better the result would be.

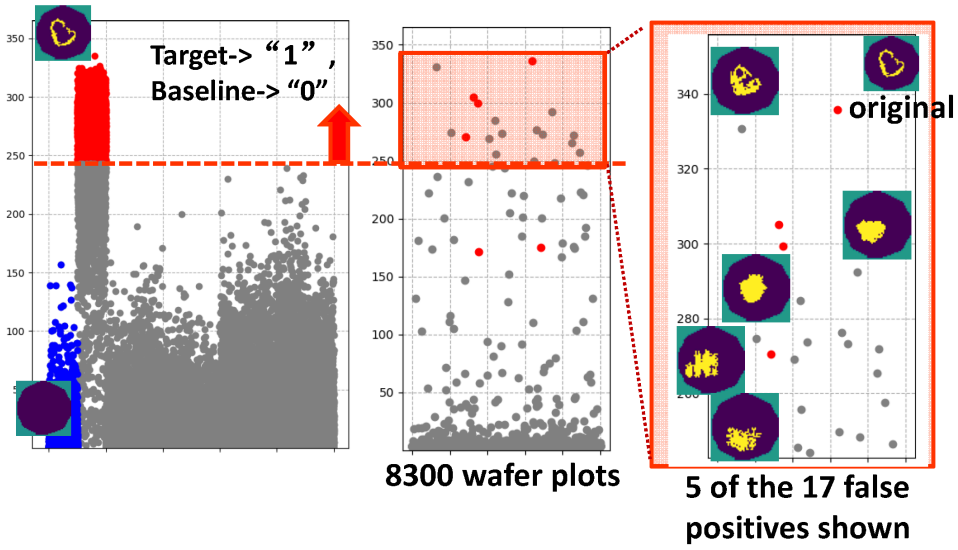

Figure 7.19: Add a no-fail wafer image, mapped to digit "0".

Comparing to Figure 7.17, observe in Figure 7.19 the significant effect by adding the blank wafer image. The resulting model classifies 21 wafer plots as in-class. Four out of the 21 plots are among those six containing a "ring", i.e. those shown in Figure 7.2. Hence, we consider the other 17 as false positives. The images for five of these 17 false positives are shown as examples.

To improve the result further, we consider adding a second generic counter example. This time we add an all-fail wafer plot, i.e. the "full" wafer image. This wafer image is mapped to digit "2" whose center is the 2nd farthest to the center of digit "1". Figure 7.20

shows the effect.

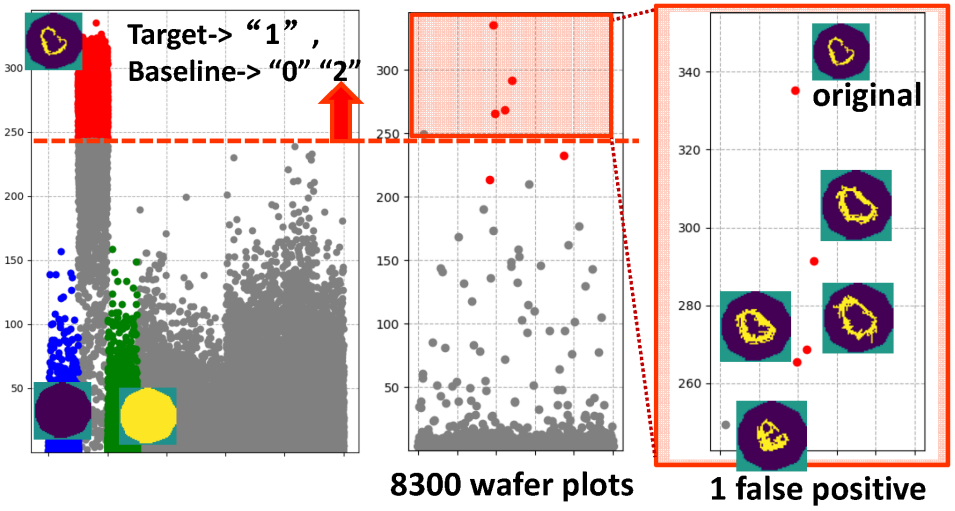

Figure 7.20: Add a no-pass wafer image, mapped to digit “2”.

The model classifies five images as in-class with only 1 false positive left. The four recognized in-class plots in Figure 7.19 are also kept as in-class. Their images are shown in Figure 7.20. The image of the one false positive is also shown in Figure 7.20. Effectively, adding the “full” wafer image removes 16 of the 17 false positives in Figure 7.19.

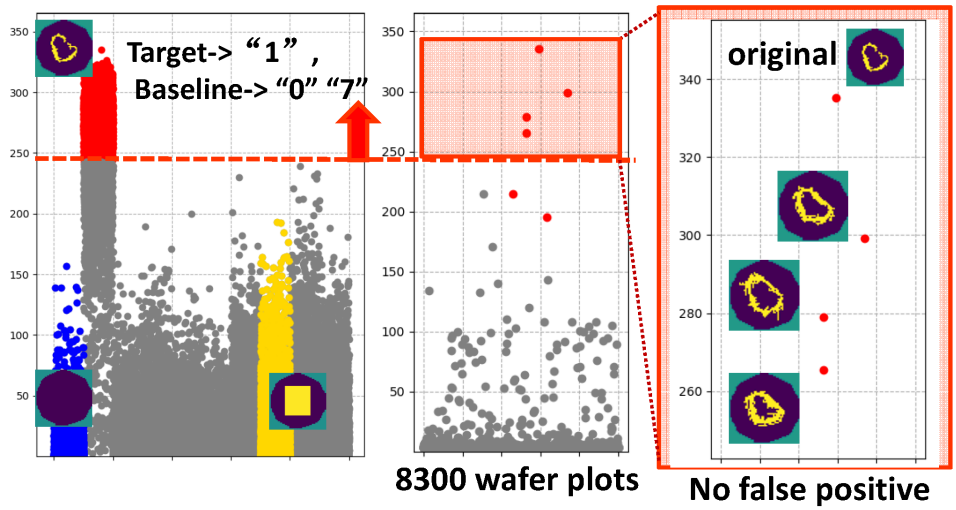

Figure 7.21: Add a bounding-box wafer image, mapped to digit “7”.

Figure 7.21 shows the best manifestation setup found in our study. Instead of using the all-fail wafer image, we use an image with all-fail in a bounding box. The bounding box is the smallest box containing the "ring" feature. Then, we fill the box with yellow pixels (excluding the area outside the wafer boundary if any). Because this bounding box image depends on the wafer plot, we map this image to digit "7" (instead of digit "2" as that in Figure 7.20). In Figure 7.12, we observe that digit "7" is the closest digit to digit "1".

As seen in Figure 7.21, the best model keeps the four in-class plots as in-class while removing the 1 false positive concerning us in Figure 7.20. We consider this as the most desirable result as it can recognize wafer plots similar to the given training wafer plot with no false positive.

7.6 Iterative Recognition

As shown in Figure 7.2, we consider six wafer plots having the same "ring" feature. The first one is used to train the manifestation model shown in Figure 7.21. The model recognizes three of the other five plots. This leaves two wafer plots unrecognized, i.e. two false negatives.

Because our manifestation setup is designed to achieve zero false positive, the false negative issue can be mitigated rather easily. The idea is that we would simply expand the recognition by iteratively building additional manifestation models based on those recognized wafer plots. Suppose in iteration i, a set S_i of wafers $\{W_1, W_2, \ldots\}$ are recognized where no W_j is recognized in a previous iteration. For each W_j, we build a manifestation model and apply it in the next iteration to see if there is any new wafer plot recognized.

This process continues until either (1) we reach a *closure* recognition set, or (2)

the recognized plots are "too far" from the original training wafer plot. A closure set $\{W_1, \ldots, W_k\}$ means that if we build a manifestation model for any W_i in the set, the recognized wafers are all in the set.

We found that iterative recognition can reach a closure set in a few iterations (e.g. three iterations) but not always. To avoid too many iterations, we can impose rules to prevent building a model from some recognized plots.

For example, such rules can be implemented by checking the size and location of a bounding box from a recognized plot, as comparing to the bounding box from the original wafer plot. In our implementation, if the size or location of the bounding box differs from the original bounding box by more than 50%, we would not build a manifestation model for the recognized plot.

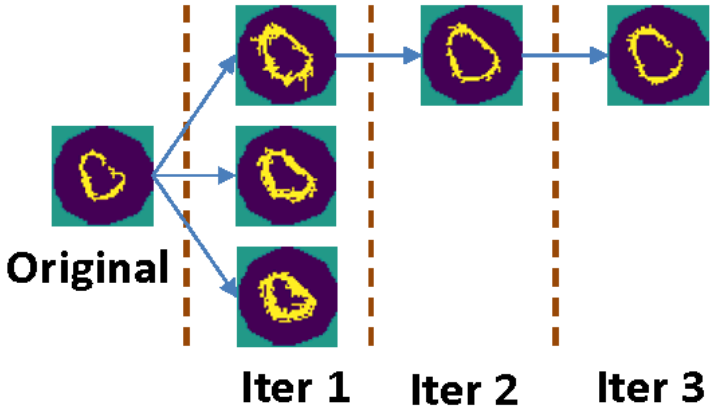

Figure 7.22: Iterative recognition for the "ring" concept.

Figure 7.22 shows the result of this iterative recognition process. The first iteration shows the same result as shown in Figure 7.21. Then, the two missing in-class plots are recognized in the 2nd and the 3rd iterations. With iterative recognition, all six wafer plots shown in Figure 7.2 are included. There is no false positive in the iterative recognition process, i.e. Figure 7.22 shows all the recognized plots. Furthermore, the process stops because it reaches a closure set.

7.6.1 Other classes of wafer plots

To validate the generality of the manifestation setup, we consider two additional classes of wafer plots: "center" and "edge" which were among the 11 classes of wafer plots reported in [83]. Figure 7.23 shows the iterative recognition result starting with a "center" wafer plot.

Again, in each iteration only those newly recognized wafer plots are shown. A manifestation model is built and applied based on each recognized plot, except those marked with a "▲". Those marked wafer plots were excluded by the rule checking mentioned before.

For those unmarked plots, if there is no outgoing line, it means all the recognized plots by the manifestation model are already included in the previous and current iteration(s). The figure shows all plots recognized in the process. Similarly, Figure 7.24 shows the

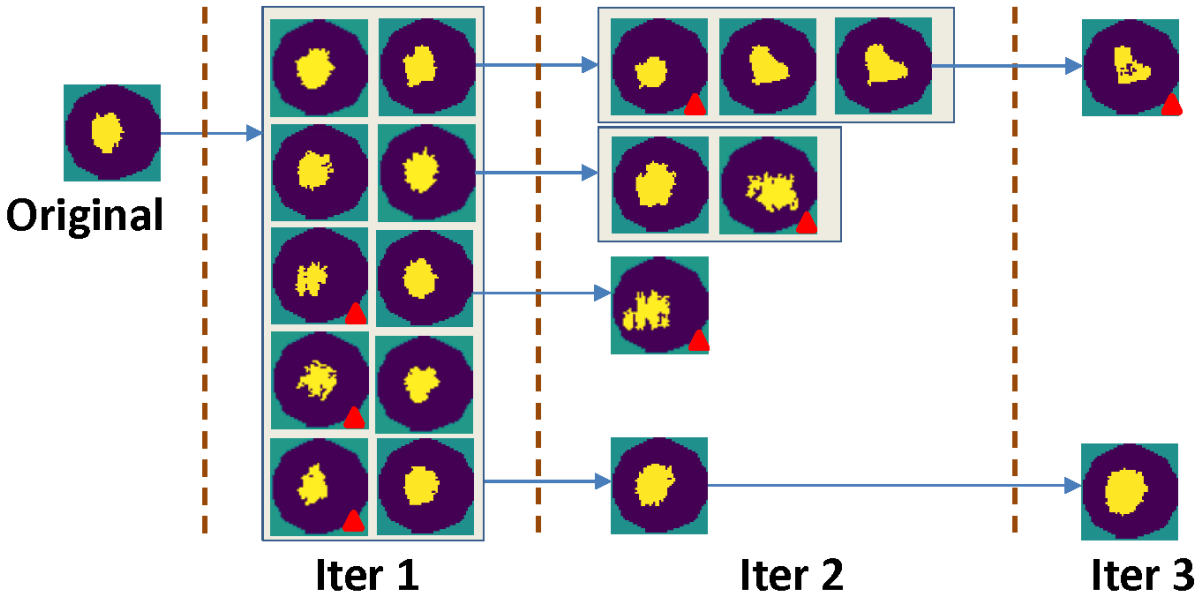

Figure 7.23: Iterative recognition for the "center" concept.

result starting with a wafer plot with an "edge" feature. Note that in the work [83], out of the 8300 wafers, in total there were 80 wafer plots reported as the "center" class

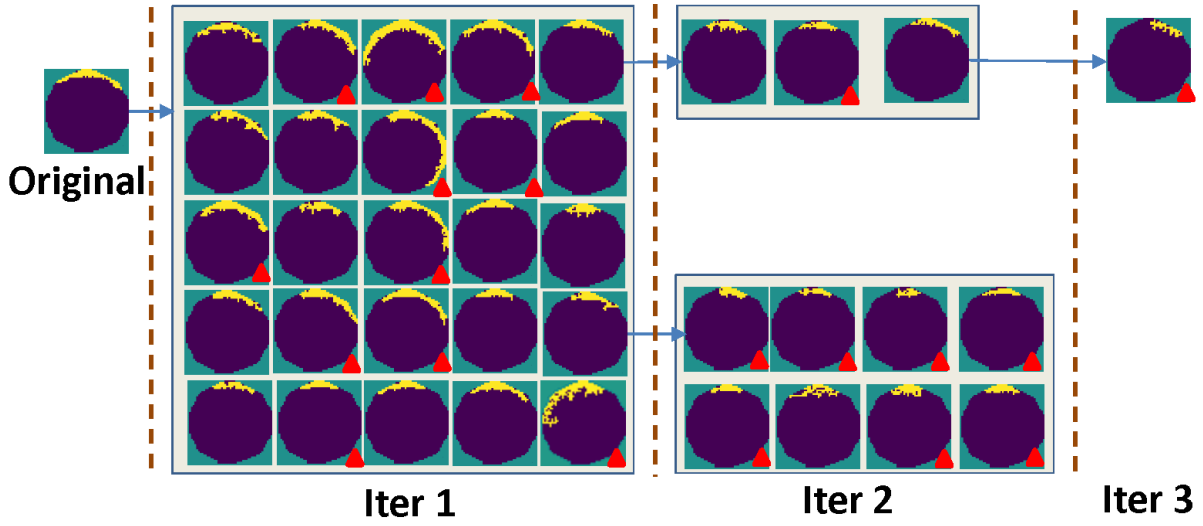

Figure 7.24: Iterative recognition for the "edge" concept.

and 108 wafer plots reported as the "edge" class. In contrast, Figure 7.23 shows 20 wafer plots and Figure 7.24 shows 38 wafer plots. None of them would be considered as a false positive from the classification reported in [83]. However, the manifestation models recognize much fewer wafer plots in each case. This is understandable because the recognition process starts with one particular wafer plot, and the process only tries to recognize plots similar to the given wafer plot.

7.7 Summary

In Section 7.2, we begin by discussing a similarity scoring method $IoU()$. Given a wafer plot, the method calculates a score for every wafer plot. As discussed in Section 7.4.2, in manifestation a SVM score is calculated for every wafer plot. Figure 7.25 shows a comparison for the top ranked plots using these two scoring methods.

For the six "ring" plots, the SVM scoring ranks them as 1,2,3,4,6, and 7. The 1st plot is the given plot. The model in Figure 7.21 puts the threshold between the 4th and the 5th plots. Hence, the top 4 plots are recognized as in-class. The IoU ranks the six plots

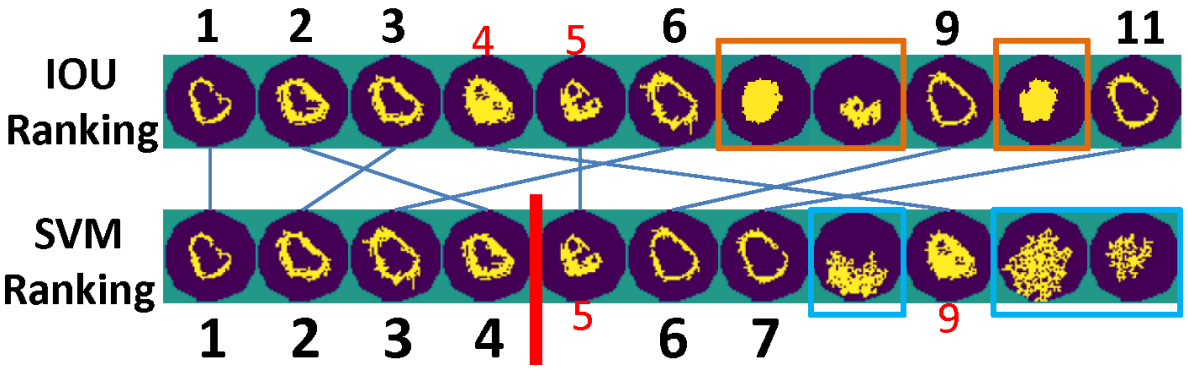

Figure 7.25: Ranking comparison: SVM scores Vs. IoU scores.

as 1,2,3,6,9, and 11. It is interesting to observe that if we assume the threshold is set to include all the top 11 plots as in-class, with both rankings, they both can capture all six "ring" plots as in-class. Also, both have exactly five false positives. The false positive sets, however, are different by three plots (the three plots highlighted in each ranking).

In view of the $IoU()$-based method discussed in Section 7.2, we see that the proposed Manifestation Learning effectively does the following four things:

1. It maintains the trend of an IoU ranking.
2. It fine-tunes the IoU ranking.
3. It automatically decides a threshold.
4. It ensures no false positive to enable implementation of an iterative recognition process.

It is interesting to note that if we consider IoU as intuitive and interpretable to a person, we see that Manifestation Learning keeps this intuition to a large extent. Enabled by our Manifestation Learning approach, let us now look at the implications of being able to train per-sample-based recognizers in the next chapter.

Chapter 8

Explainability-First DSML

"It is not enough to have a good mind; the main thing is to use it well."

— Rene Descartes

In Chapter 4, we have discussed the requirements for implementing a practical DSML solution in the context of WMPR. In Section 5.4, we have discussed the capabilities for meeting the first requirement. In Chapter 7, we present a novel method to enable learning a recognizer based on one wafer map. In this chapter, we will explain how such learning can enable an implementation of a practical DSML solution in view of the capabilities and requirements discussed before.

8.1 Discovering Pattern Classes Based On Graph-Based Operations

Given wafer maps w_i, w_j, suppose we have learned their recognizers R_i, R_j. The benefit of having these two recognizers is that we can now define a *primitive* called pattern

equivalence $PE(w_i, w_j)$ and use it to decide if two wafer maps should be considered to have the same pattern class. Note that PE does not say that the two wafer maps are definitely having the same class. It says that they should be considered. We can implement PE by enforcing that R_i recognizes w_j and R_j recognizes w_i.

Once the PE primitive is defined, we can convert a given batch of wafer maps into a graph. In this graph, there is an edge between two wafer maps w_i, w_j if their $PE(w_i, w_j)$ is true. In this way, we obtain a graph as the starting point to discover pattern classes.

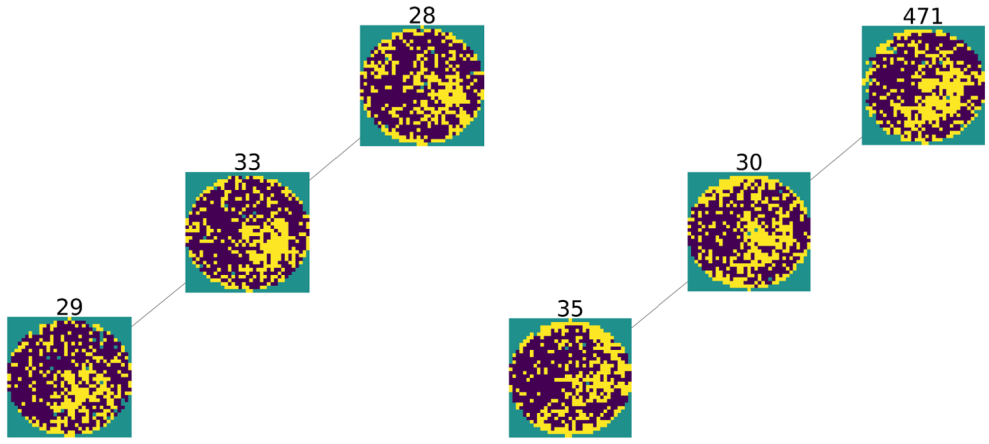

Figure 8.1: 2 example sub-graphs as a connected component

On this graph, we can easily discover connected components as sub-graphs. For example, Figure 8.1 shows two sub-graphs found in a batch of wafer maps. Figure 8.3 shows another sub-graph whose size is larger.

8.2 Lower-Bound and Upper-Bound

Partitioning the graph by its connect components provides us an upper-bound model. Between any two components, there is no edge. Hence, we know that there is no PE relationship between any pair of wafer maps across the two components. It is only an upper-bound model because two wafer maps in the same component can be connected

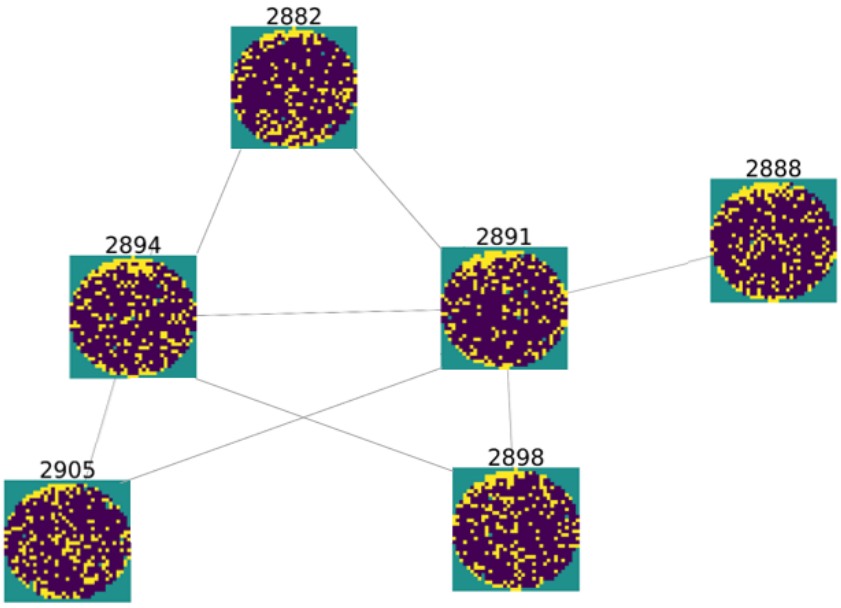

Figure 8.2: A larger sub-graphs as a connected component

through a sequence of wafer maps in between. For example, w_1 is connected to w_{100} because $PE(w_1, w_2), PE(w_2, w_3), \ldots, PE(w_{99}, w_{100})$ are all true. In this case, we cannot say that w_1 and w_{100} are likely to have the same pattern class (from a user's perspective). On the contrary, the chance is probably low for a user to consider them the same because their connection is established through so many intermediate wafer maps.

Given a connected component, we can find cliques inside the component. These cliques serve as the choices to define a lower-bound model. For example, Figure 8.3 shows a connected component where three cliques of size three are found. The assumption in our DSML solution is that wafer maps in a clique of size at least three are considered in the same pattern class. For example, suppose we choose clique A in the figure as our lower-bound model. Then, through our explainability-first interaction with the user, it might be decided that wafer map w should be added to the pattern class represented by clique A. This decision will trigger adding the other two wafer maps because they and w belong to two other cliques, respectively.

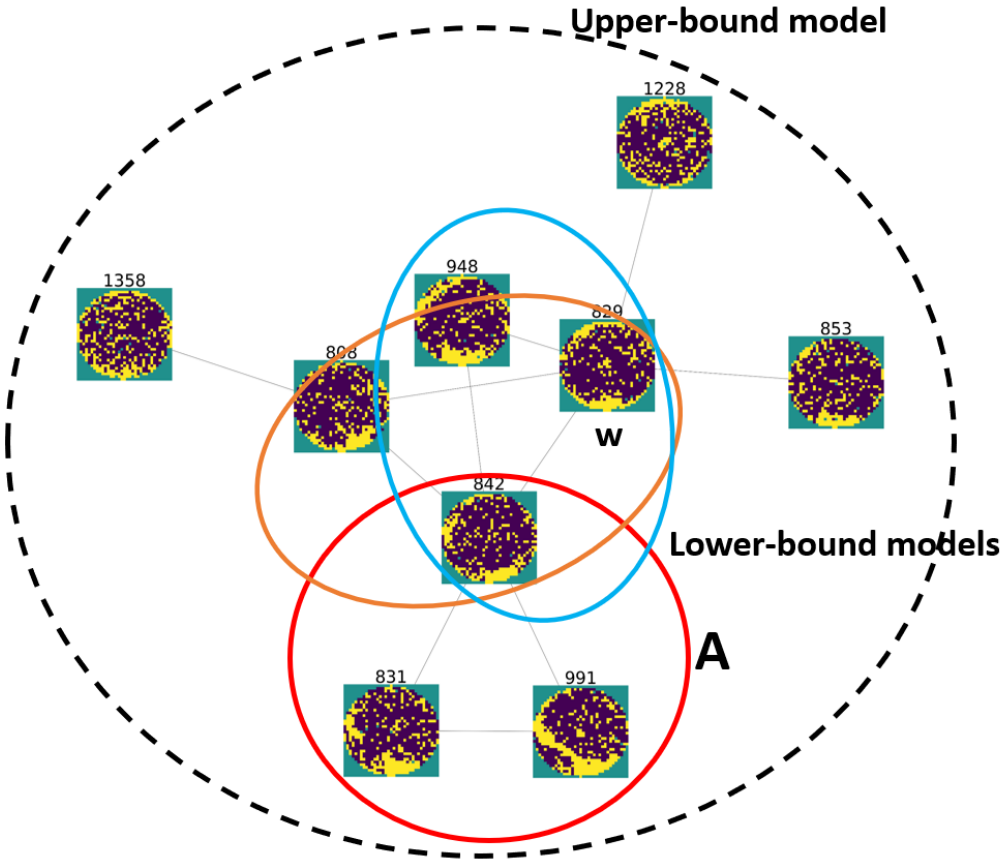

Figure 8.3: Example to show lower-bound and upper-bound models

8.3 Explainability-First Fine-Tuning

Given an upper-bound model and a lower-bound model, our goal is then to implement a workflow that facilitates a user to attain a final decision on the set of wafer maps that should be in the same pattern class. Figure 8.4 illustrates this workflow.

Given a clique C as a lower-bound model, our strategy to extend its coverage of other wafer maps is the following. We will use a set of features $f_1, \ldots, f_n$ to find a rule that explains the clique. We desire this rule to be understandable by the user. The learning problem to get to such a rule is similar to the rule learning problems discussed before in Section 1.2.1. In essence, we have a dataset where positive samples are those in the

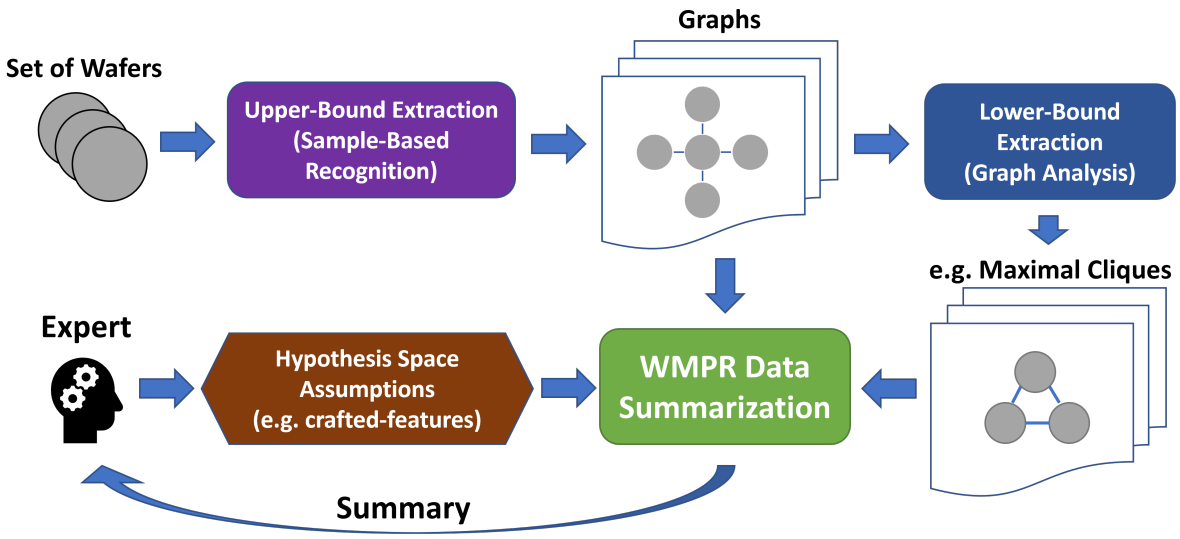

Figure 8.4: WMPR data summarization workflow, where sample-based primitives enable the extraction of the Upper-Bound and the Lower-Bound models.

clique and negative samples are those not in the connected component containing the clique. Those wafer maps not in the clique but in the connected component are ignored in this rule extraction process.

Feature	Type	Description
$Ratio_{area,cc}$	CC	Ratio of CC's area to wafer area
$Ratio_{peri,cc}$	CC	Ratio of CC's perimeter to wafer radius
$Dist_{max,cc}$	CC	Max. distance between CC's and wafer center
$Dist_{min,cc}$	CC	Min. distance between CC's and wafer center
$Ratio_{major,cc}$	CC	Ratio of major axis of CC's estimated ellipse to wafer radius
$Ratio_{minor,cc}$	CC	Ratio of minor axis of CC's estimated ellipse to wafer radius
$Ratio_{conv,cc}$	CC	Ratio of CC's convex hull defectivity
$Ratio_{ecc,cc}$	CC	CC's estimated ellipse eccentricity
$Count_{global}$	Statistical	Count of defective dies on wafer
$Ratio_{global}$	Statistical	Ratio of defective dies to all wafer dies
$Ratio_{local}$	Statistical	Ratio of defective dies in two outer-most rings of wafer
$Count_{lines}$	Statistical	Count of lines detected with Hough transform

Table 8.1: Features used to demonstrate the data summarization tool's feasibility.

In practice, our DSML solution can come with an initial set of features. However, we do not assume that this initial set is sufficient. The idea is to allow a user to create

and use their own features. This allows the user to ingest their perspective to the DSML solution and affect the explanation result. As a start, Table 8.1 lists a set of features considered in our current implementation of the envisioned DSML solution. Note that this is still an ongoing effort taken over by other students in our lab.

8.4 Initial Results

Based on the feature set summarized in Table 8.1, in this section we use two examples to illustrate the effect of the methodology depicted in Figure 8.4. The context is in the analysis of a sequence of selected wafer maps produced within a month from a production line. There are 1448 wafer maps in the consideration.

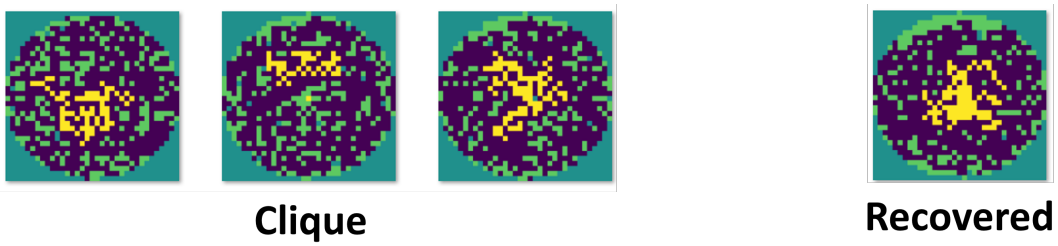

Figure 8.5: $Rule = ($ $Ratio_{global} \geq 0.31088,$ $Dist_{max,cc} \leq 0.72760,$ $Ratio_{major,cc} \leq 1.07521,$ $Ratio_{conv,cc} \leq 0.44943$ $)$

The first example is shown in Figure 8.5. The left shows the clique containing three wafer maps. After a rule explaining the clique is found, one wafer map in the corresponding connected component is found satisfying the rule. This wafer map is then added to the pattern class. The rule is stated in the caption of the figure.

The second example is shown in Figure 8.6. Again, the rule found one wafer map and added the wafer map into the pattern class. Note that the rules for the two examples both involve four features. Having more features in a rule can be seen as less interpretable for the user. The examples illustrate how the proposed methodology might be used to find

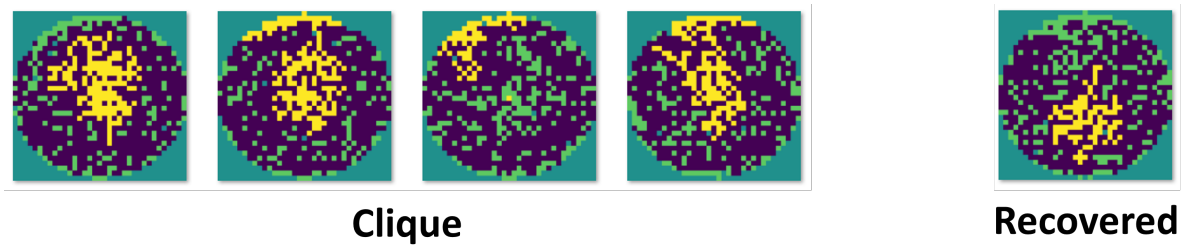

Figure 8.6: $Rule = ($ $Count_{lines} \geq 5.0$, $Ratio_{local} \leq 0.42366$, $Ratio_{conv,cc} \geq 0.57447$, $Dist_{min,cc} \leq 0.0$ $)$

a target model according to a user's perspective. The rules shown might not be ideal. However, getting to the results can help us think of a better set of features, in terms of more interpretable rules. This is how we expect a user to use the DSML solution. The solution summarizes the current results and presents possible options (e.g. a lower-bound and upper-bound pair). Then, the user can gradually learn from the results, create a new feature to refine results, and iteratively attain the results that are desirable. In this iterative process, the features created become the knowledge assimilated by the solution, which can be re-used in the future. In this way, there might be an initial ramp-up cost to use the software, but this cost shall diminish quickly over time.

8.5 Summary

In view of the four capabilities listed in Section 5.4, the methodology presented in Figure 8.4 can take care of the first three. It appears the fourth one is unattained. However, the same methodology can be used to attain the fourth capability. To illustrate this, Figure 8.7 shows examples of wafer maps that do not belong to any of the low-bound pattern classes. As seen, these wafer maps exhibit no clear pattern, i.e. their failing die distributions are random. In other words, without our per-sample-based analysis, those random wafer maps that cannot be grouped with another wafer map are automatically

left out. This is another benefit by enabling the per-sample-based analysis approach.

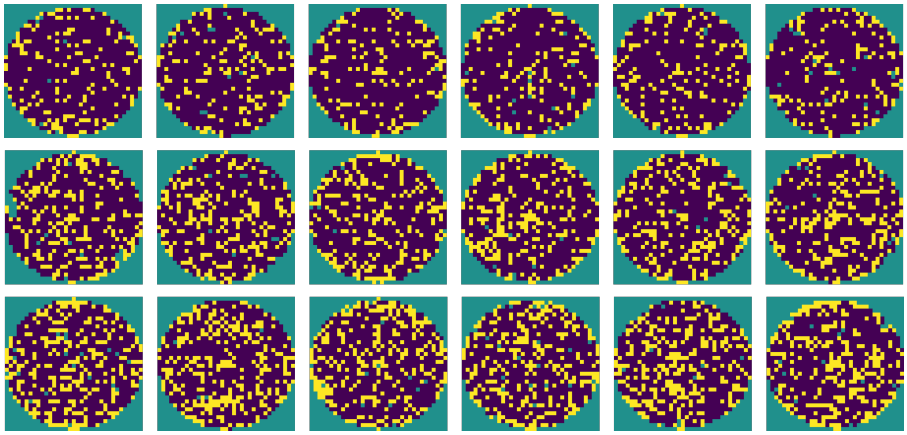

Figure 8.7: Examples of stand-alone wafer maps

In Section 4.5, we list three requirements and two questions in view of implementing a DSML solution in the context of WMPR. The proposed methodology above provides the capabilities to address the three requirements. In particular, the methodology does not intend to define a set of pattern classes automatically. It starts with two bounds, lower and upper, and allow the user to search for a desirable answer in between the two bounds. The methodology does not rely on an assumption that there is a golden answer to the definition of pattern classes. Different users can achieve their own golden answer using their respective feature set.

Using the proposed methodology as the foundation, other functions can be added to the software for the convenience of results review by the user. For example, one function can be to rank pattern classes according to their yield loss. Another function can be to summarize all current pattern classes in a single screen where one wafer map is selected to represent a pattern class, e.g. the one with the highest yield loss. Another function can be to allow a user to merge two pattern classes into a single class manually. For example, this recommendation might be based on the overlap of features in use to describe the two pattern classes. Simply put, the proposed methodology and the per-sample-based

capability provides a foundation that enables many useful functions to be built into the software, which can greatly enhance the added value brought to the user.

For the last two questions mentioned in Section 4.5, our DSML solution can easily address them. For example, all found pattern classes can be ranked according to their yield loss and presented to the user. The user can review from the top one with the highest yield loss. This facilitates the user to reach an answer to the first question ("Is there a pattern that represents a potential yield issue?) quickly.

Regarding the second question ("If yes, is the pattern new or has it occurred previously?"), as mentioned before it essentially demands the capability to perform a similar-pattern search on a given set of wafers. Implementing this search is straightforward with our per-sample-based recognition approach. The PE primitive defined earlier can enable such a search easily.

Overall, we see that the proposed methodology can enable us to implement a DSML solution in the context of WMPM which has all the capabilities to meet all the requirements discussed before. It is important to emphasize, though, this complete solution can only be possible after we have found a way to enable the per-sample-based analysis, i.e. learning a recognizer based on one wafer map.

Chapter 9

Conclusion

"If a Bodhisattva has a mark of self, a mark of others, a mark of living beings, or a mark of life, then he is not a Bodhisattva."

— The Vajra Prajna Paramita Sutra, translation by Buddhist Text Translation Society

We start this thesis with an overview of ML's foundational developments between the 1950s to the 2000s, followed by an examination of deep learning's explosion starting from the early 2010s. For deep learning, we discuss its success in the general image classification problem and the natural language understanding problem. After introducing the application of ML in these two general contexts, we also introduce the application of ML in a domain specific context, semiconductor design and test. Specifically, we illustrate using application examples in Section 1.2, when applying domain specific machine learning (DSML), our focus seems to be on everything else but the learning algorithms. This observation is illustrated through three characteristics of applying DSML, 1) we lack the "care" samples, 2) we need to be able to answer "No models found", 3) we need to be able to check how applicable our model is for the current data.

Next, in Chapter 2, we dive into the unique aspects of DSML and articulate its difference between general ML. Specifically, we discuss that all tasks in DSML involve an iterative process, where the DSML expert is applying tools to be able to achieve a closure faster for their task at hand. In other words, the DSML experts are involved in an iterative debugging/improvement process, where they are searching for "surprising" information for their tasks.

Based on the new information discovered in an iteration, the DSML expert can actively adjust their assumptions about the underlying problem and establish the scope of the next iteration. In a sense, when the expert's assumptions are actively changing between immediate iterations ("local"), DSML problems approach the definition of *Local No-Free-Lunch*. This is all in contrast to general ML, where a set of hypothesis space assumptions are fixed about the underlying problem at hand, and ML algorithms tries to find the best generalized model in view of the current data.

Embracing the phenomenon of *Local No-Free-Lunch* in DSML, we aim to design data summarization tools to be able to facilliate an expert's iterative search process. Specifically, we aim to provide capabilities to address the three characteristics mentioned above, and also provide another capability to perform an unsupervised per-sample-based analysis (for the WMPR problem). In addition, we stress the importance of a DSML solution being able to provide clear added value for an existing company workflow, as this is crucial for its adoption.

In Chapter 3, we further dive into the challenges associated with applying general ML in view of DSML problems. From a high level, we can view the task of learning as a filtering process as depicted in Figure 3.2, where inconsistent hypotheses are filtered out by the data. When many hypotheses are left in this hypothesis space, the "simplest" hypothesis is chosen. This approach essentially treats Occam's razor as a first principle during learning. However, we show that from the view of Bayesian inference, Occam's ra-

zor is justified only when picking between hypothesis spaces, but not between hypotheses within the same fixed hypothesis space. Specifically, using a monomial learning example, we illustrate this risk of treating Occam's razor as the first principle in DSML, where problems are Local No-Free-Lunch.

That said, Occam's razor can still be used to guide the selection between hypothesis space assumptions. Practically, when tackling DSML problems, instead of enumerating all assumptions, the expert is more focused on obtaining a manageable set of initial assumptions. With these initial assumptions, the user can utilize their domain knowledge and iterate towards their solution. This motivates us to realize a software system to assist the user in attaining these initial assumptions.

Next, in Chapter 4, we introduce the Wafer Map Pattern Recognition (WMPR) problem as our DSML application context. We start with discuss the existing works which tackles the WMPR problem from a general ML view (e.g. multi-class classification). Then, specific issues are raised regarding taking a general ML approach for WMPR, ultimately emphasizing the risk of being biased by *Law of the Instrument*, as this rarely results in practical DSML solutions. By taking a DSML view, we discuss the WMPR problem's characteristics, such as the no-pattern class being analogus to "no models found". With these problem characteristics, we define the requirements for a WMPR DSML solution: 1) Defining a set of pattern classes is part of the WMPR problem. 2) There is no golden answer to the definition. 3) If two sets of classes are presented to a user as two options, the user can judge which is the preferred one. We also describe the end goals of an expert when interacting with our DSML solution, where they want to know 1) If a pattern representing a potential yield issue exists, and 2) If yes, is this pattern new?

Having established the WMPR problem as our DSML context, we next introduce Structural Risk Minimization (SRM) in Chapter 5. SRM is a learning principle that

emphasizes the establishment of a set of hypothesis spaces with an increasing complexity "structure". Starting from the search for a hypothesis in the least complex space, and increasing the complexity as needed, upon seeing enough samples, this ensures the arrival of the "first-fit" hypothesis space. This learning principle serves as an inspiration for some design consideration, which will be elaborated below.

Going back to the WMPR problem, we discuss the inherently unsupervised nature of its requirements. Specifically, taking the requirement of discovering a set of pattern classes, we can frame it as a clustering problem. From this perspective, we can see that the expert desires a transparent process of deciding some clustering between two extremes. Where all wafers can be considered their own cluster, and all wafers can be considered a single cluster. With this in mind, we aim to 1) develop a lower-bound cluster model, where no two wafer maps that have been put into the same pattern class will be separated in future analysis. Similarly, 2) for a upper-bound model, no two wafer maps that have not been put into the same pattern class shall be in the same class in future analysis. Ultimately, for a practical DSML solution, we want to develop these two intuitive models to 3) guide the expert to search for their own "best" model. In addition, to facilitate the expert's search, we aim to reduce the search space by 4) develop a way to filter out wafer maps with no pattern. If we examine our lower-bound, the target desired by the expert, and the upper-bound, these models naturally can be framed as an approximate SRM implementation of 3 increasingly complex hypothesis spaces.

Chapter 6 discusses the realization of two capabilities above, the development of a lower-bound and a way to filter out no-pattern wafer maps. For the lower-bound model, we employed properties of tensor decomposition, and developed tensor-based recognizers which can produce wafer clusters that are very similar within the cluster. We show that this cluster can be used as a checker to ensure more robustness for a deployed Generative Adversarial Networks (GANs-based) DSML solution. Specifically, tensor-based recog-

nition results can be used in a containment check against the GANs-based recognition results, where when the containment fails, a DSML technology provider is notified of the failure, where they can start implementing improvements for the deployed software. The feasibility of the containment check implies tensor-based models are lower-bound models. In addition, to address the second capability, we developed a rule-based methodology to generate artificial in-class and out-of-class samples to facilitate the training of a no-pattern class recognizer. We also stress the importance of implementing rule-based checkers to ensure that the filtered out wafers indeed does not contain patterns.

With both of these capabilities addressed, we are still missing two more capabilities mentioned above, the upper-bound model, and enabling the expert's search. In addition, tensor-based methods cannot extend to an upper-bound model, and the rule-based methodology is difficult to generalize to more types of failures. If we wanted to address all four capabilities at the same time, then we have to consider the extreme case, where a recognizer can be learned using just a single wafer sample.

Motivated by this, in Chapter 7, we detail the intuition and implementation of our novel per-sample-based learning approach, *Manifestation Learning*. With the enablement of per-sampled-based recognizers, in Chapter 8, we describe how they can fulfill each of the capabilities in a explainability-first DSML software. Specifically, relationship primitives (e.g. pattern equivalence) can be defined between all wafer samples, where graph analysis can take place to extract lower-bound models and upper-bound models. We demonstrate the feasibility of how the expert can using crafted-features (i.e. hypothesis space) to formulate rules and interpret their own grouping definition. We also show the fourth capability of filtering no-pattern wafer maps is also enabled naturally by the per-sample-based approach. Lastly, also enabled by the per-sample-based recognizers, we discuss implementations of different data summarization approaches to fulfill the expert's two usage models in this WMPR software mentioned previously.

9.1 A Philosophical Remark About This Work

In essence, DSML is fundamentally different from ML, for ML is driven by optimization while DSML is not. This is because DSML lives in a world that is constrained by (local) No-Free-Lunch. In a No-Free-Lunch world, optimization is not the ultimate solution. Continuous optimization can only lead to diverse outcomes that are difficult to comprehend. When that happens, some outcomes might be perceived as good and others perceived as bad, depending on one's own perspective.

We can observe this phenomenon when watching a practitioner trying to apply ML. For example, they might desire to employ a clustering tool to group wafer maps into pattern classes. They might try to use tools provided by a ML library such as Sickit-Learn [96]. They would find many different tools. They applied and saw the pros and cons of each tool from the result. At the end, they can look at the result and judge which one is good and which one is bad. However, if they have to do this every time, then those tools probably put more burden on them than actually helping them.

For example, if an engineer needs to review 500 wafer maps a day, probably it would be easier to review them directly (following a simple ranking by the yield loss), without invoking any ML tool. In other words, continuous optimization provides us with different ML tools that can produce diverse results. When a result makes sense, a person thinks the tool is useful. When a result does not make sense, a person thinks the tool is not useful. The tricky part is that, for the same tool, its result may make sense sometime and does not make sense some other time.

In a world of No-Free-Lunch, continuous optimization only introduces more complexity to the world, not an ultimate solution. In contrast, in our DSML development, the first thing we learned to accept, is that the world is No-Free-Lunch. Then, all the subsequent works were developed based on that assertion. In other words, in retrospect we

could say that the most important step in our development was when we decided to take away our desire to optimize.

Without the desire to optimize, we still would need a driver for the development. Searching for this driver eventually led us to focus on providing added value to the user, and at the same time to the understanding that this added value should be perceived from the user perspective (not ours). This driver then enabled us to see that ultimately to achieve the goal, we would need a per-sample-based recognition capability.

We therefore learned that the essential driver of DSML is not optimization but compassion, the compassion for other people, wanting to provide help to them in their perspective. The "help in their perspective" characterizes the meaning of compassion which is reflected in our proposed methodology to implement a practical DSML solution. We also learn that in our research and development process, the most important thing in reaching the goal is to take away the "self". A true compassion has no self. Otherwise, it becomes "desire to optimize compassion" which is just another form of continuous optimization and that eventually, will not work well in a world of No-Free-Lunch.

The lesson we learned seems to be compatible to the first stage in the quote from the Vajra Prajna Paramita Sutra, stated at the beginning of this chapter: "If a Bodhisattva has a mark of self, then he is not a Bodhisattva". There are still much to learn after realizing the importance of this first stage. It will be quite intriguing to contemplate what DSML might mean in view of the other three stages.

Bibliography

[1] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019.

[2] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system," *ACM Computing Surveys*, vol. 52, p. 1–38, Jan 2020.

[3] Google, "Search by people, things & places in your photos," *Google Photos Help*.

[4] "About face id advanced technology," *Apple Support*, Sep 2021.

[5] S. Ananthakrishnan, "Amazon scientists applying deep neural networks to custom skills," *Amazon Science*, Jul 2020.

[6] Google, "Translate images," *Google Translate Help*.

[7] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," 2021.

[8] E. Alpaydin, *Introduction To Machine Learning.* Mit Press, 4 ed., 2020.

[9] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[10] M. L. Minsky and S. Papert, *Perceptrons an introduction to computational geometry*. Cambridge, Mass. Massachusetts Institute Of Technology, 1990.

[11] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, p. 193–202, Apr 1980.

[12] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the National Academy of Sciences*, vol. 79, p. 2554–2558, Apr 1982.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, p. 533–536, Oct 1986.

[14] PyTorch, "Neural networks — pytorch tutorials documentation," *pytorch.org*.

[15] F. Chollet, "Keras documentation: Introduction to keras for researchers," *keras.io*, Apr 2020.

[16] S. Ji, "English: An illustration of kernel trick in svm. here the kernel is given by:," *Wikimedia Commons*, Jun 2017.

[17] C. Cortes and V. Vapnik, "Support-vector networks," in *Machine Learning*, pp. 273–297, 1995.

[18] V. N. Vapnik, *Statistical Learning Theory*. Wiley, Cop, 1998.

[19] S. Theodoridis and K. Koutroumbas, *Pattern recognition*. Amsterdam Elsevier/Acad. Press [20]11, 2008.

[20] adam.morey@nist.gov, "Special database catalog," *NIST*, Feb 1992.

[21] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171 – 1220, 2008.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, p. 2278–2324, 1998.

[23] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, 04 1998.

[24] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik, "Support vector clustering," *Journal of Machine Learning Research*, vol. 2, p. 125–137, Dec 2001.

[25] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, p. 1527–1554, Jul 2006.

[26] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems* (B. Schölkopf, J. Platt, and T. Hoffman, eds.), vol. 19, MIT Press, 2007.

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[28] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, p. 39–41, Nov 1995.

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (Red Hook, NY, USA), p. 1097–1105, Curran Associates Inc., 2012.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Proceedings of the International Conference on Neural Information Processing Systems*, vol. NIPS 2014, no. 1, p. 2672–2680, 2014.

[32] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2014.

[33] A. Ng and M. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in Neural Information Processing Systems* (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), vol. 14, MIT Press, 2002.

[34] hindupuravinash, "hindupuravinash/the-gan-zoo," *GitHub*, Sep 2018.

[35] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[36] C. Rosset, "Turing-nlg: A 17-billion-parameter language model by microsoft," *Microsoft Research*, Feb 2020.

[37] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[39] OpenAI, "Openai api," *beta.openai.com*.

[40] L.-C. Wang, "Experience of data analytics in eda and test—principles, promises, and challenges," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 6, pp. 885–898, 2017.

[41] L.-C. Wang and J. Shan, "Machine learning in test," in *IEEE Test Technology Education Program, Tutorial at IEEE International Test Conference*, 2020, 2021.

[42] Y. Katz, M. Rimon, A. Ziv, and G. Shaked, "Learning microarchitectural behaviors to improve stimuli generation quality," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 848–853, 2011.

[43] W. Chen, L.-C. Wang, J. Bhadra, and M. Abadir, "Simulation knowledge extraction and reuse in constrained random processor verification," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2013.

[44] J. Chen, B. Bolin, L.-C. Wang, J. Zeng, D. Drmanac, and M. Mateja, "Mining ac delay measurements for understanding speed-limiting paths," in *2010 IEEE International Test Conference*, pp. 1–10, 2010.

[45] L.-C. Wang, *Learning from Limited Data in VLSI CAD*, pp. 375–399. Cham: Springer International Publishing, 2019.

[46] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 2000.

[47] J. Tikkanen, S. Siatkowski, N. Sumikawa, L.-C. Wang, and M. S. Abadir, "Yield optimization using advanced statistical correlation methods," in *2014 International Test Conference*, pp. 1–10, 2014.

[48] N. Sumikawa, J. Tikkanen, L.-C. Wang, L. Winemberg, and M. S. Abadir, "Screening customer returns with multivariate test analysis," in *2012 IEEE International Test Conference*, pp. 1–10, 2012.

[49] N. Sumikawa, L.-C. Wang, and M. S. Abadir, "An experiment of burn-in time reduction based on parametric test analysis," in *2012 IEEE International Test Conference*, pp. 1–10, 2012.

[50] 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, 2016.

[51] 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, 2017.

[52] M. R. Nero, *Domain Specific Machine Learning - A No-Free-Lunch Perspective*. University of California, Santa Barbara, March 2022.

[53] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[54] D. Wolpert and W. Macready, "Coevolutionary free lunches," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 721–735, 2005.

[55] J. Schaffer, "What not to multiply without necessity," *Australasian Journal of Philosophy*, vol. 93, p. 644–664, Dec 2014.

[56] D. J. C. Mackay, *Information theory, inference, and learning algorithms*. Cambridge University Press, (Imp, 2003.

[57] A. H. Maslow, *The Psychology of Science*. New York: Harper & Row, 1966.

[58] M.-J. Wu, J.-S. R. Jang, and J.-L. Chen, "Wafer map failure pattern recognition and similarity ranking for large-scale data sets," *IEEE Transactions on Semiconductor Manufacturing*, vol. 28, no. 1, pp. 1–12, 2015.

[59] S. Helgason, *The Radon Transform*. Springer US, 1999.

[60] M. Fan, Q. Wang, and B. van der Waal, "Wafer defect patterns recognition based on optics and multi-label classification," in *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pp. 912–915, 2016.

[61] J. Yu and X. Lu, "Wafer map defect detection and recognition using joint local and nonlocal linear discriminant analysis," *IEEE Transactions on Semiconductor Manufacturing*, vol. 29, no. 1, pp. 33–43, 2016.

[62] M. Piao, C. H. Jin, J. Y. Lee, and J.-Y. Byun, "Decision tree ensemble-based wafer map failure pattern recognition based on radon transform-based features," *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 2, pp. 250–257, 2018.

[63] J. Yu, "Enhanced stacked denoising autoencoder-based feature learning for recognition of wafer map defects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 613–624, 2019.

[64] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, pp. 318–362. 1987.

[65] N. Yu, Q. Xu, and H. Wang, "Wafer defect pattern recognition and analysis based on convolutional neural network," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 566–573, 2019.

[66] J. Wang, Z. Yang, J. Zhang, Q. Zhang, and W.-T. K. Chien, "Adabalgan: An improved generative adversarial network with imbalanced learning for wafer defective pattern recognition," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 3, pp. 310–319, 2019.

[67] T.-H. Tsai and Y.-C. Lee, "A light-weight neural network for wafer map classification based on data augmentation," *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 4, pp. 663–672, 2020.

[68] M. Saqlain, Q. Abbas, and J. Y. Lee, "A deep convolutional neural network for wafer defect identification on an imbalanced dataset in semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 3, pp. 436–444, 2020.

[69] M. B. Alawieh, D. Boning, and D. Z. Pan, "Wafer map defect patterns classification using deep selective learning," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[70] M. Sewell, "Structural risk minimization," *www.svms.org*.

[71] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, p. 279–311, Sep 1966.

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

[73] 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.

[74] 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*, KDD '06, (New York, NY, USA), p. 374–383, Association for Computing Machinery, 2006.

[75] L. McInnes, J. Healy, and S. Astels, "hdbscan: Hierarchical density based clustering," *The Journal of Open Source Software*, vol. 2, 03 2017.

[76] A. Wahba, C. J. 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)*, pp. 1–4, 2019.

[77] M. Nero, C. Shan, L.-C. Wang, and N. Sumikawa, "Concept recognition in production yield data analytics," in *2018 IEEE International Test Conference (ITC)*, pp. 1–10, 2018.

[78] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.

[79] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[80] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 214–223, PMLR, 06–11 Aug 2017.

[81] D. Berthelot, T. Schumm, and L. Metz, "Began: Boundary equilibrium generative adversarial networks," *arXiv:1703.10717 [cs, stat]*, May 2017.

[82] 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)*, pp. 79–84, 2019.

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

[84] F.-F. Li, "Knowledge transfer in learning to recognize visual object classes," *International Conference on Development and Learning (ICDL)*, 2006.

[85] I. Goodfellow, Y. Benjio, and A. Courville, "15.2 transfer learning and domain adaptation," *Deep Learning*, 2016.

[86] e. a. Xi Chen, "Variational lossy autoencoder," 2016.

[87] D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, "Ladder variational autoencoders," *Advances in Neural Information Processing Systems*, vol. 29, pp. 3738–3746, 2016.

[88] S. Zhao, J. Song, and S. Ermon, "Infovae: Information maximizing variational autoencoders," 2017.

[89] A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample-problem," *Advances in Neural Information Processing Systems*, pp. 513–520, 2007.

[90] N. Shibuya, "Up-sampling with transposed convolution," 2017.

[91] M. Abadi and et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[92] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[93] e. a. Samuel R. Bowman, "Generating sentences from a continuous space," 2015.

[94] e. a. Samuel R. Bowman, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, 8 1997.

[95] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[96] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.