WaferCap: Open Classification of Wafer Map Patterns using Deep Capsule Network

Abhishek Mishra*, Mohammad Ershad Shaik†, Mohammad Ershad Shaik*, Suman Kumar*, Anup Das*, Nagarajan Kandasamy*, and Nur A. Touba†
*Electrical and Computer Engineering Department, Drexel University, Philadelphia, PA 19104, USA
†Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712, USA

***Abstract*—In integrated circuit design, analysis of wafer map patterns is critical to enhance yield and detect manufacturing issues. With the emergence of novel wafer map patterns, there is increasing need for robust artificial intelligence models that can both accurately classify seen patterns and while also detecting ones not seen during training, a capability known as open world classification. We develop a novel solution to this problem: WaferCap, a Deep Capsule Network designed for wafer map pattern classification and equipped with a rejection mechanism. When evaluated using the WM-811k dataset, WaferCap significantly surpasses existing methods, achieving 99% accuracy for fully seen patterns while demonstrating robust performance in open-world settings by effectively detecting unseen wafer map patterns.**

***Index Terms*—wafer map pattern classification, capsule network, open world classification**

I. Introduction

Analysis of wafer map patterns enables early detection and elimination of defective chips during integrated circuit (IC) design, leading to cost savings. This analysis also offers vital visual insights that are essential to pinpoint process-related errors. Automated techniques for wafer map pattern analysis based on machine learning (ML) have been developed.

Supervised learning using the K-nearest neighbor algorithm has been used to obtain ML models to classify defective wafer maps using features derived from spatial signatures [1], [2]. Support vector machines have been used for classification, in which the machine is trained using features extracted through radon and various geometric transformations [3], [4]. More recently, the image recognition capabilities of convolutional neural networks (CNNs) have been used to identify wafer map patterns, achieving state-of-the-art results [5]–[8].

Unsupervised learning techniques that cluster wafer maps based on defect patterns have also been developed [9]–[12]. Once the wafer map collections are obtained, test engineers manually label each map according to its defect pattern. Nero et al. propose an unsupervised approach using generative adversarial networks to train a set of recognizers to identify wafer map patterns [13]. However, this method requires substantial manual tuning during the learning process.

The supervised wafer map pattern classification approaches described above make the *closed-world assumption*: defect classes appearing in the test data must also have appeared in the training data. However, real-world scenarios frequently challenge this assumption. As the IC design cycle progresses and wafer map patterns are continually analyzed to improve the yield, a classifier trained on seen patterns may not be efficacious enough to classify new (unseen) patterns, because over time, new wafer map patterns evolve [5]. Thus, in practice, a classifier should accurately categorize the patterns of the wafer map seen based on the classes observed during training, while also identifying the unseen patterns that do not belong to any existing class, a problem called *open classification* [14], [15].

We develop a novel ML-based approach to address open wafer pattern classification. Our architecture called **WaferCap** uses a deep capsule network with dynamic routing in an open world setting, equipped with rejection capability. In addition to achieving state-of-the-art results compared to existing methods, our approach also addresses two critical challenges.

Rejecting unseen patterns. Occasionally, wafers may present patterns not previously seen by the model during training. WaferCap is designed to reject these new patterns, which may be further inspected by test engineers to find insights. This capability relates to lifelong learning; without the ability to identify & adapt to new wafer map patterns, a model is limited in its potential for continual self-improvement [14], [16].

Handling data imbalance. Wafer map patterns can vary in frequency of occurrence, leading to imbalanced datasets. Stable manufacturing processes often produce wafer maps that are predominantly pattern-free. Maps exhibiting defect patterns comprise only a tiny portion of the entire dataset. This imbalance can skew the predictions of the model, favoring the more common patternless wafers. Thus, rarer but potentially problematic patterns might be overlooked, allowing systematic failures to go undetected. We address this imbalance using proposed WaferCap with a dynamic routing algorithm, which effectively captures spatial relationships in patterns and focuses on subtle features for enhanced recognition, achieving a 97% classification accuracy, which further improves to 99% with our data augmentation method.

As noted earlier, there is a large amount of recent work using CNNs for wafer map pattern classification, which achieves high accuracy. Our choice of deep capsule network addresses a basic limitation of CNNs. While they excel at feature detection, CNNs are poor at capturing hierarchical relationships

Abhishek Mishra and Mohammad Ershad Shaik are co-first authors with equal contribution.

979-8-3503-6378-4/24$31.00 ©2024 IEEE

among features. They are invariant of translation, meaning that they can recognize patterns regardless of their position within the image. In contrast, capsule networks can discern the position of objects, capturing relative positioning and spatial information. Unlike CNNs, which may lose information through pooling operations, dynamic routing capsule networks can effectively extract spatial attributes such as size, orientation, and relative position, all while requiring fewer training data. WaferCap draws inspiration from the advancements achieved by capsule networks in diverse applications, including image recognition and natural language processing [17]–[19].

Key contributions of our paper include:

- Domain-specific application of a dynamic routing capsule network to capture spatial information between features for better recognition of wafer-map patterns.
- Rejection capability that can be helpful in identifying new wafer-map patterns or dealing with data drift problems.
- Domain-specific data augmentation, which allows for automated generation of a desired amount of data, especially for underrepresented wafer map patterns. The augmentation step preserves the structural properties of the original wafer patterns.

The accuracy achieved by WaferCap under both closed & open world classifications is evaluated on WM-811k dataset [3]. For closed-world classification, WaferCap outperforms current state-of-the-art approaches: without any data augmentation, WaferCap achieves an average classification accuracy of 97%, which improves to 99% with the proposed data augmentation method. It achieves superior accuracy on underrepresented patterns compared to existing methods. The results also demonstrate the efficacy of WaferCap in open classification when some fraction of the patterns belong to unseen classes.

The WaferCap code repository and the datasets used to obtain the reported results are publicly available at https://github.com/abhishekkumarm98/WaferCap.

The paper is organized as follows. Section II formulates the problem. Section III develops WaferCap and Section IV elaborates on the data augmentation technique. Section V presents the key results and we conclude the paper in Section VI.

II. Open Classification of Wafer Map Patterns

We formulate the wafer map pattern classification problem and introduce basic concepts related to capsule networks.

A. Problem Formulation

Consider wafer datasets $\mathbf{W} = \{W_1, W_2, \ldots, W_n\}$ and classes $\mathbf{C} = \{C_1, C_2, \cdots, C_n\}$ where every instance within $W_i \in \mathbf{W}$ is labeled with the same class $C_i \in \mathbf{C}$. Figure 1 shows the wafer map patterns or classes of interest: Center, Donut, Edge-Location, Edge-Ring, Random, Location, Near-Full, Scratch, and No-Pattern. Grey pixels signify a good die that has successfully passed all wafer tests, yellow pixels indicate a bad die that has failed some test, and blue pixels represent regions outside the wafer. The training set $\mathbf{T} = \{(w_i, C_i)\}_1^N$, contains N examples; w_i is an example from the wafer dataset associated with label C_i. The number

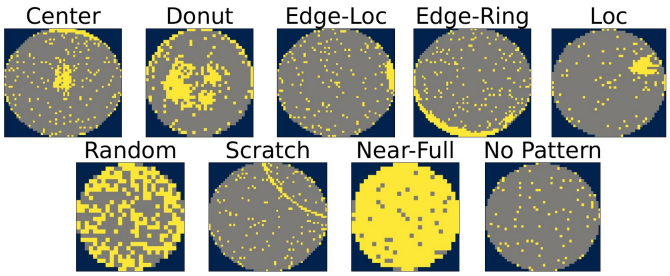

Fig. 1: Examples of different wafer pattern types.

of training (seen) pattern classes within $\mathbf{T}$ is n. The testing set $\mathbf{T_e}$, however, contains classes $\{\mathbf{C} \cup C_{rej}\}$, where C_{rej} is the collection of all novel or unseen pattern classes.

We wish to train a classifier $M(w)$ using the training set $\mathbf{T}$ and evaluate its performance on the testing set $\mathbf{T_e}$. This classifier should either assign a pattern from $\mathbf{T_e}$ to one of the seen classes from $\mathbf{C}$ or reject the pattern by assigning it to the unseen class C_{rej}. Therefore, our aim is to construct a $(n+1)$ classifier $M(w)$ with classes $\{C_1, C_2, ..., C_n, C_{rej}\}$, allowing it to perform open-world classification.

B. Capsule Network With Dynamic Routing

WaferCap is a variation of a capsule network with dynamic routing [17]. A capsule network captures crucial information related to the spatial and relative positioning of patterns within the wafer map for better recognition performance. The capsule network was originally proposed to overcome limitations inherent to CNNs where the pooling operations reduce the feature map's spatial dimensions, resulting in information loss. Also, pooling layers without learning parameters limits the ability to model complex interactions between features. Thus, a CNN may not be able to identify small objects since the corresponding details are too fine to be captured at the reduced spatial resolution of the feature maps (due to pooling).

Capsule networks are a neural network architecture composed of a series of layers, called capsule layers, consisting of many capsules. Each capsule is a set of neurons sensitive to a particular feature of the input image, such as the presence of an edge or a specific shape, and captures both the likelihood and parameters of a feature. The output of the capsules (features) in a given layer is then passed on to the next layer, where they are combined and passed up the hierarchy to higher-level capsules, which extract more abstract concepts such as the identity of an object. It uses vector-based capsules instead of scalar-based neurons in CNNs. The output vector of a capsule is called the activity vector, whose magnitude represents the probability of detecting a feature and its orientation represents its properties. The structure of the network is stratified, where capsules are attached in layers, beginning with the primary capsule layer representing basic features, and then passing through dynamic routing-by-agreement layers that represent complex features and objects. Here, dynamic routing serves as a parallel attention mechanism that allows each capsule at one level to attend to some active capsules at the lower level and to ignore others. This allows the model to pay attention to subtle or small objects within the image.

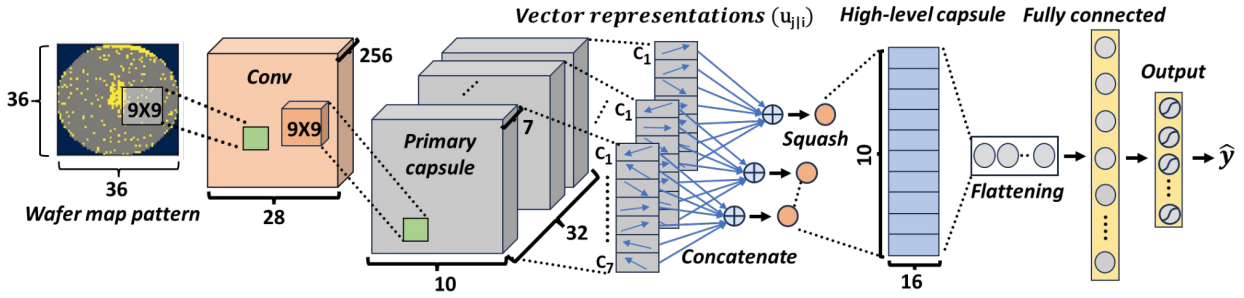

Fig. 2: Architecture of WaferCap.

III. The WaferCap Architecture

This section describes in greater detail the WaferCap architecture, shown in Fig. 2.

A. Convolutional Layer

The input wafer map is reshaped to the dimension of $(1, 36, 36)$ and processed using a 9×9 kernel with a stride length of one and no padding to extract 256 feature maps. Each point within the feature map is computed as

$$\sum_{d} \sum_{m} \sum_{n} IW_d(c_x - m, c_y - n) K_d(m, n), \qquad (1)$$

where d iterates over the depth of both the input (IW) and the learnable filter (K). The coordinates of a specific point within the wafer are given by c_x and c_y, and $K_d(m, n)$ is the filter element at position (m, n) at depth d. The convolutional layer extracts salient features from the input and provides an output of dimension $(batch\_size, 256, 28, 28)$. The ReLU activation function is subsequently applied element-wise, introducing nonlinearity to the extracted features. Since the wafer map is a grayscale image, the depth is set to $d = 1$.

B. Capsule Layer

Features extracted by the convolutional layer are fed into the primary capsule layer which plays an important role in transitioning from the scalar-output feature detectors (typical of CNNs) to vector-output capsules. The aim is to preserve instantiated parameters such as the local order of pattern features. This layer retains the relative positioning and sequencing of features, capturing the inherent spatial structure of wafer map patterns, such as the pixel arrangements. For instance, in the Center wafer map pattern type, the bad dies are centrally clustered with respect to the pattern's edge. This spatial relationship is conserved by the capsule layer. Preserving these features' local order accurately identifies the pattern type while also ensuring that the model uses the hierarchical relationships within the data to improve its predictions.

Capsule networks automatically learn child-parent (part-whole) relationships, which refers to the hierarchical relationship between simpler (child or part) and more complex (parent or whole) features in the data. In the Edge-Ring pattern, for example, distinct features such as the ring-like bad dies at the edge and the randomly distributed bad dies throughout the pattern are simpler features (parts) that collectively form the more complex Edge-Ring pattern feature (whole). The capacity of capsule networks to discern these child-parent relationships captures these hierarchical structures within the data. The capsule layer encodes not only the presence of specific features, but also their spatial orientation and relationship to other features. It inherently understands that certain features (e.g., ring at the edge and random features spread over pattern) must be present in specific spatial relationships to one another for a pattern to be recognized as a specific pattern type (eg., Edge-Ring). This provides a more nuanced understanding of the data and results in better generalization, especially in scenarios where traditional CNNs may fail. In simpler terms, while traditional networks may just look for the presence of certain features, capsule networks try to understand how these features fit together in a larger picture, making them more adept at understanding the holistic nature of data.

Dynamic routing ensures that the output of each primary capsule is sent to the appropriate capsule in the higher-level layer and is formulated mathematically as follows. The total output to a high-level capsule j, $\mathbf{x_j}$, is a weighted sum over all prediction vectors $\hat{u}_{j|i}$ from the primary capsules in the preceding layer. It is produced by multiplying the output u_i of each primary capsule by a weight matrix K_{ij} as

$$\hat{u}_{j|i} = K_{ij} u_i + \hat{b}_{j|i} \in R^d, \qquad \mathbf{x_j} = \sum_{i=1} c_{ij} \hat{u}_{j|i}, \qquad (2)$$

where $\hat{b}_{j|i}$ a bias term and c_{ij} are coupling coefficients determined by the iterative routing process. The prediction vector $\hat{u}_{j|i}$ can be intuitively understood as encoding the spatial relationship between features extracted by primary capsules. The coefficients c_{ij}, calculated by the dynamic routing mechanism shown in Algorithm 1, determine the contribution of each prediction vector $\hat{u}_{j|i}$ to $\mathbf{x_j}$. This decides the degree of influence of capsule i on capsule j based on how aligned their outputs are. In essence, these equations describe the mechanism by which lower-level capsules send their output (transformed by learned weight matrices) to higher-level capsules, where the weights are determined by the dynamic routing process. The goal is to ensure that the information is sent to the capsules in the next layer in a way that preserves the hierarchical and spatial relationships present in the data.

WaferCap uses dynamic routing between the primary-capsule layer and the high-level fully-connected capsule layer. The mechanism is shown in Algorithm 1 which is adopted from Sabour et al. [17]. The input argument Lo_{ij} stands for

Algorithm 1 Dynamic routing between capsules.

Input: $\hat{u}_{j|i}$, Lo_{ij}, R, l
for all capsule i in layer l and capsule j in layer $l+1$ **do**
  $Lo_{ij} \leftarrow 0$
end for
/*Routing iterations*/
for r = 1, …, R **do**
  for all capsule i in layer l **do**
    $c_{ij} \leftarrow \frac{exp(Lo_{ij})}{\sum_k exp(Lo_{ik})}$
  end for
  for all capsule j in layer $l+1$ **do**
    $\mathbf{x}_j \leftarrow \sum_i c_{ij}\hat{u}_{j|i}$
    $v_j \leftarrow \text{Squash}(\mathbf{x}_j)$
  end for
  for all capsule i in layer l and capsule j in layer $l+1$ **do**
    $Lo_{ij} \leftarrow Lo_{ij} + \hat{u}_{j|i} \cdot v_j$
  end for
end for
Output: v_j

initial logits (before softmax) needed for routing and R is a number of routing iterations. Dynamic routing generates a nonlinear mapping in an iterative fashion, ensuring that the output from each capsule is properly routed to its appropriate parent capsule in the subsequent layer. Another advantage is the ability to modulate the connection strength between capsules. This flexible connectivity proves more effective compared to earlier routing methods such as max-pooling in CNN that detects whether a feature is present in any position, but loses spatial information about the feature.

The primary-capsule layer has seven capsules, where each capsule itself is a convolutional layer with 32 channels and a 9×9 kernel with a stride of 2 and no padding. This results in an output dimension of $(32\times10\times10)$. Thus, the combined output from this layer layer has shape $(batch\_size, 7, 32, 10, 10)$. The Squash function introduces non-linearity, modulates vector magnitude between 0 and 1 to represent feature presence probability, preserves the vector's orientation which encodes feature properties, and limits the vector length. It is defined as

$$\text{Squash}(\mathbf{x_j}) = v_j = \frac{\|\mathbf{x_j}\|^2}{1+\|\mathbf{x_j}\|^2} \cdot \frac{\mathbf{x_j}}{\|\mathbf{x_j}\|} \tag{3}$$

where $||\mathbf{x_j}||$ is the magnitude of the vector $\mathbf{x_j}$ of capsule j.

C. High-Level Fully-Connected Capsule Layer

This layer consists of one 16-dimensional capsule and accepts an input of shape $(batch\_size, 7, 32, 10, 10)$ from the primary capsule layer, producing an output of $(batch\_size, N_c, 16)$. Each capsule receives input from all capsules in the preceding layer and generates an activity vector whose length captures a condensed representation. Here, N_c denotes the total number of capsules in the layer, which has been set to 10 based on model tuning. Subsequently, the input $(batch\_size, N_c, 16)$ is flattened and fed into a fully connected layer, undergoing a linear transformation.

D. Fully-Connected Layer

The fully-connected layer performs two operations on the flattened 1D data: a linear transformation and a nonlinear transformation. These transformations combine features learned by previous layers, allowing the network to recognize complex and nonlinear patterns during training to better approximate the desired function. We first perform a linear transformation, followed by a non-linear one as

$$z_1 = \text{ReLU}(W_{fc}x_{fc} + b_{fc}), \tag{4}$$

where x_{fc} represent the input vector to the fully connected layer. The weight matrix for this layer is denoted by W_{fc}, where each column corresponds to the weights associated with a specific neuron in the layer. The bias vector is represented by b_{fc}. After applying the linear and nonlinear transformations, the layer outputs a vector z_1.

E. One-Versus-All Sigmoid Layer

Conventional multi-class classifiers often employ softmax for the final output layer [20]–[22]. However, this approach lacks a rejection capability since the predicted probability for each class is normalized over all training or seen classes. In contrast, we employ a one-versus-all sigmoid layer as the output layer, consisting of n sigmoid functions for the n seen classes. For the i-th sigmoid function associated with class C_i, the proposed capsule network treats all the examples where $y = C_i$ as positive and all other examples where $y \neq C_i$ as negative. Each label C_i corresponds to a one-versus-all binary classification task, leveraging shared representations derived from the capsule network. The one-vs-all sigmoid layer offers a nuanced representation of all classes (including both seen and unseen ones) and enables the one class to form a good decision boundary. Following the receipt of input z_1 from the fully connected layer, this layer yields n probabilistic outputs, each ranging between 0 and 1.

IV. Data Augmentation

During manufacturing, wafer maps that do not show patterns are produced much more frequently than those exhibiting patterns. Thus, certain patterns occur with varying frequencies, leading to an imbalanced dataset. This can skew the model predictions, causing it to favor the more common No-Pattern wafer while often misclassifying the more problematic patterns [23]–[25].

To mitigate the imbalance issue, we use a domain-specific data enhancement approach based on bijective functions, summarized in Algorithm 2. The algorithm is inspired from [26]–[28], that can generate a desired number of analogous data from underrepresented patterns while preserving the underlying properties of the original pattern. From each class

Algorithm 2 Bijective-function based data augmentation.

$TP \leftarrow \{\}$ /* Initialize new wafer-pattern set */
for $j = 1 : n$ **do**
  for $template$ in W^{C_j} **do**
    $angle \leftarrow random(0, 360)$
    $i \leftarrow random(1, k)$
    $I \leftarrow \tau_i(template, angle)$
    $TP \leftarrow TP \cup I$ /* Add new pattern to the set */
  end for
end for

of underrepresented patterns, we randomly select template images, and to introduce diversity, subject these templates to various geometric transformations, generating new patterns in the process. Here, n represents the number of classes of interest, $\mathbf{T}$ denotes the set of training wafer map patterns, and $W^{C_i} \subset \mathbf{T}$ indicates the set of templates used for each class. Let $\{\tau_1, \tau_2, \ldots, \tau_k\}$ represent k distinct *geometric transformations*. Each transformation, τ_i, employs a bijective function defined as $\tau_i : template \longrightarrow I$. We use the following affine transformations: random rotation, horizontal flip, and vertical flip. We also combine these transformations. Since a geometric transformation is any bijection of the set of template images to itself or to another such set with some salient geometrical underpinning, this one-to-one mapping between template images and the corresponding transformed images generates diverse array of new and unique wafer map patterns.

V. Results

WaferCap is implemented in Pytorch. Its performance is evaluated using the WM-811k dataset which consists of 811,457 wafer maps collected from 46,293 lots. The dataset exhibits nine distinct wafer map patterns: Center, Donut, Edge-Location, Edge-Ring, Random, Location, Near-Full, Scratch, and No-Pattern, which were previously shown in Fig. 1. Each pattern in the dataset comes with varying image sizes, but consistently contains three pixel levels: 2 for bad dies, 1 for good dies, and 0 for absent dies. Given this variety in sizes, we resize all patterns to a uniform dimension of 36×36 pixels. Referring back to Fig. 1, gray pixels represent good dies that pass all wafer tests, yellow pixels signify bad dies failing one or more tests, while blue pixels denote areas outside the wafer.

The WM-811k dataset contains 172,950 wafer maps labeled by domain experts. Thus, for a fair comparison against state-of-the-art (SOTA) approaches [3], [5], [8], [29], [30], only labeled data is used in this work and are grouped into different sets based on various training/test ratios.

In our experiments, the training batch size is set to 256. All routing logits Lo_{ij} are initialized to zero, and the number of iterations is kept to 3. We use the AdamW optimizer which has a learning rate of 1e-3 and the loss function

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{N} &- \mathbf{I_{DF}}(y_j = C_i)\log p(y_j = C_i) \\ &- \mathbf{I_{DF}}(y_j \neq C_i)\log(1 - p(y_j = C_i)),\end{aligned} \tag{5}$$

where $\mathbf{I_{DF}}$ is the indicator function and $p(y_j = C_i) = \mathrm{Sigmoid}(z_{j,i})$ represents the probability output from i^{th} sigmoid function on the j^{th} wafer map pattern sample's i^{th}-dimension of z. During testing, predictions from n sigmoid functions are reinterpreted to achieve rejection capability:

$$\hat{y} = \begin{cases} \textit{reject}, & \text{if } \mathrm{Sigmoid}(z_i) < th_i, \forall C_i \in \mathbf{C} \\ \underset{C_i \in Y_n}{\operatorname{argmax}}\, \mathrm{Sigmoid}(z_i), & \text{otherwise} \end{cases}$$

The predicted probability $\mathrm{Sigmoid}(z_i)$ from the i^{th} sigmoid function is compared to a threshold th_i for class C_i. If all predicted probabilities for a sample fall below their respective thresholds, it is rejected. Otherwise, its class prediction is determined by the sigmoid function with the highest probability. In simpler terms, if none of the seen classes yields a probability above its respective threshold, the sample is classified as new or unseen; otherwise, it is classified to the seen class with the highest probability.

To determine an optimal threshold th_i for each seen class C_i, we use concepts from outlier detection theory [20], [31]. Assume this the predicted probabilities $p(y = C_i|x_j, y_j = C_i)$ for all training data of class i adhere to one side of a Gaussian distribution. We then synthetically generate the complementary half of the Gaussian distributed points (≥ 1). For every present point, $p(y = C_i|x_j, y_j = C_i)$, a mirrored point, $1 + (1 - p(y = C_i|x_j, y_j = C_i))$ is created, reflecting about a mean of one. Subsequently, the standard deviation σ_i is estimated using both the original and the mirrored points. In statistical terms, a point lying a α of standard deviations from the mean is deemed an outlier. Thus, the probability threshold is set as $th_i = max(0.5, 1 - \alpha\sigma_i)$. Importantly, due to this Gaussian fit, different classes, C_i, might possess unique classification thresholds, th_i. The optimal value of α is obtained during experimentation based on the specific characteristics of the data and the WaferCap model.

The accuracy and macro-average F1 score serve as the primary evaluation metrics. Accuracy evaluates classification performance across all wafer map patterns and is defined as the fraction of correctly classified samples to the total number of samples. However, data imbalance can influence this metric. For individual patterns, the macro-average F1 score $F1_{\text{macro}} = \frac{1}{n}\sum_{i=1}^{n} F1_i$ is the chosen metric. To evaluate open-world classification performance, we reserve certain classes as unseen during training and reintroduce them during testing. The fraction of classes used for training is varied between 25%, 50%, 75%, and 100% of the total classes, whereas testing uses all available classes. Using 100% of classes for training equals closed classification. For the WM-811K dataset, the 25% configuration involves training with 2 classes and testing against all 9 (with 7 being unseen during training). The evaluation metric relies on the average accuracy across the 2 trained classes and an additional one for rejection. Instances from the unseen classes are excluded from the validation set.

Table I shows the results of the open classification. Observe that as the proportion of classes used for training increases, average accuracy improves noticeably, ranging from 80% for 25% of seen classes to 99% when all classes are seen. Interestingly, the accuracy for seen classes initially drops from 98% to 83% as more classes are introduced, before rebounding to 99% in a conventional closed-world setting where all classes are seen. In contrast, the accuracy for unseen classes consistently improves. This behavior can be attributed to the changing balance between seen and unseen class examples during testing. In the 25% setting, testing involves examples from 2 seen classes (Center, Donut) and 7 unseen classes. Similarly, the 50% setting comprises test examples from 4 seen classes and 5 unseen ones. Since most test examples in the 25% and

TABLE I: Accuracy results under open classification (**S**: seen classes, **US**: unseen classes).

Percent of Seen Classes	Average Accuracy	Seen Classes Accuracy	Unseen Classes Accuracy	Wafer Map Patterns								
				Center	Donut	Edge-Loc	Edge-Ring	Local	Random	Scratch	Near-Full	No Pattern
25%	80%	98%	78%	S	S	US	US	US	US	US	US	US
50%	84%	85%	83%	S	S	S	S	US	US	US	US	US
75%	91%	83%	95%	S	S	S	S	S	S	S	US	US
100%	99%	99%	NA	S	S	S	S	S	S	S	S	S

TABLE II: Accuracy achieved by WaferCap (under 100% seen wafer map patterns) compared to SOTA approaches.

Reference	Model	Split Ratio	Avg. Accuracy	No Pattern	Center	Donut	Edge-Loc	Edge-Ring	Local	Random	Scratch	Near-Full
[5]	CNN	8:2	94%	98%	96%	73%	70%	96%	64%	57%	29%	40%
[5]	SVM [3]^C	8:2	91%	100%	79%	36%	50%	96%	2%	0%	38%	80%
[29]	MC32+MLP	8:2	96%	98%	96%	89%	87%	98%	70%	90%	17%	90%
Ours$_{\mathbf{woa}}$	WaferCap	8:2	97%	100%	100%	79%	81%	100%	82%	95%	2%	0%
Ours$_{\mathbf{wa}}$	WaferCap	8:2	**99%**	**100%**	**100%**	**96%**	**96%**	**100%**	**86%**	**95%**	**86%**	**100%**
Ours$_{\mathbf{wa}}$	WaferCap	8:2	**99%**	$^{F1}\mathbf{100\%}$	$^{F1}\mathbf{100\%}$	$^{F1}\mathbf{96\%}$	$^{F1}\mathbf{95\%}$	$^{F1}\mathbf{100\%}$	$^{F1}\mathbf{90\%}$	$^{F1}\mathbf{98\%}$	$^{F1}\mathbf{82\%}$	$^{F1}\mathbf{100\%}$
[8]	CNN	7:3	94%	98%	93%	83%	84%	98%	79%	84%	61%	93%
[29]	MC32+MLP	7:3	95%	99%	87%	89%	71%	94%	50%	71%	12%	95%
Ours$_{\mathbf{woa}}$	WaferCap	7:3	96%	100%	100%	68%	75%	100%	82%	1%	0%	0%
Ours$_{\mathbf{wa}}$	WaferCap	7:3	99%	100%	100%	96%	94%	100%	90%	99%	75%	75%
[30]	P^2-Net	8:1:1	96%	$^{R}99\%$	$^{R}93\%$	$^{R}81\%$	$^{R}73\%$	$^{R}97\%$	$^{R}60\%$	$^{R}91\%$	$^{R}29\%$	$^{R}90\%$
Ours$_{\mathbf{woa}}$	WaferCap	8:1:1	95%	$^{R}100\%$	$^{R}98\%$	$^{R}50\%$	$^{R}68\%$	$^{R}98\%$	$^{R}58\%$	$^{R}11\%$	$^{R}0\%$	$^{R}0\%$
Ours$_{\mathbf{wa}}$	WaferCap	8:1:1	97%	$^{R}100\%$	$^{R}98\%$	$^{R}96\%$	$^{R}65\%$	$^{R}97\%$	$^{R}72\%$	$^{R}85\%$	$^{R}81\%$	$^{R}100\%$

* Under the Split Ratio column, the value before the colon indicates the training set ratio, while the value after the colon represents the ratio of either the test set or a validation and test set. Samples are randomly selected to populate each set. ***R***: Recall, ***F1***: F1 score
* ***woa***: without augmentation, ***wa***: with augmentation, ***C***: The authors of [5] used [3]'s methodology with their own training set.

50% settings come from unseen classes, that is why, for 25% setting, the seen class accuracy is high because it has only two classes whereas, for unseen classes, it has 78% accuracy. However, as more seen classes are introduced with unseen ones, the decision boundaries become more intertwined; in addition to differentiating between multiple seen classes, the model also has to detect the presence of any unseen class, leading to decreasing seen class accuracy. In the 100% setting, the absence of unseen classes to consider allows for higher accuracy in seen class predictions. Using Gaussian fitting helps to find better probability thresholds when many unseen classes are present (under 25% and 50% settings).

We benchmarked WaferCap against leading SOTA methods on the WM-811k dataset, as shown in Table II, under traditional closed classification. With data augmentation, WaferCap surpasses all other SOTA techniques across every pattern. Even without data augmentation, for an 8:2 split ratio, it surpasses the performance of [3], [5], [29] for the Center, Edge-Ring, Local, and Random patterns. For a 7:3 ratio, it surpasses [8], [29] for the No-Pattern, Center, Edge-Ring, and Local patterns. Finally, with an 8:1:1 split, WaferCap outpaces [30] for the No-Pattern, Center, and Edge-Ring patterns.

Methods such as [8], [29], [30], [32] resize wafer map patterns to 64×64, while [5] adjusts them to 256×256. In contrast, WaferCap operates on a more compact size of 36×36, which emphasizes its efficiency and superior capability in capturing the spatial relationships within wafer maps.

Al Rahman et al. use the original Capsule Network proposed by Sabour et al. to classify eight defect patterns — without including No Pattern — with 91.4% accuracy [32]. WaferCap targets all nine patterns, which is more reflective of real-world scenarios, and with appropriate modifications to the Capsule Network achieves significantly higher classification accuracy. Furthermore, WaferCap can perform classification in an open-world setting, whereas the approach developed by Al Rahman et al. does not have this capability.

TABLE III: Execution-time and storage overhead.

Reference	Average (ms/wafer)	Storage
SVM [3]	0.5^P	N/A
CNN [8]	40	N/A
MC32+MLP [29]	1.5^P	N/A
P^2-Net [30]	0.01-0.02	N/A
WaferCap	0.81^P	31.8MB

Table III lists the execution-time and storage costs incurred by the various approaches. Some implementations are parallelized (P). Though the hardware platforms are different, we report WaferCap's execution time using a comparable system (2.30GHz Intel Xeon CPU with an NVIDIA Tesla T4 GPU).

VI. Conclusion

This paper has developed WaferCap, a deep capsule network-based ML architecture for wafer map pattern classification. The results obtained using the WM-811k benchmark confirm that WaferCap outperforms existing SOTA methods for closed classification. It also demonstrates superior performance under open classification. We have released WaferCap in the open-source domain so that other researchers can reproduce our results and also improve upon its design. Future work will focus on classification of mixed wafer maps and improving WaferMap's computational efficiency.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No 2008167.

REFERENCES

[1] K. W. Tobin Jr, S. S. Gleason, T. P. Karnowski, S. L. Cohen, and F. Lakhani, "Automatic classification of spatial signatures on semiconductor wafer maps," in *Metrology, Inspection, & Process Control for Microlithography XI*, vol. 3050. SPIE, 1997, pp. 434–444.

[2] T. P. Karnowski, K. W. Tobin Jr, S. S. Gleason, and F. Lakhani, "Application of spatial signature analysis to electrical test data: validation study," in *Metrology, Inspection, and Process Control for Microlithography XIII*, vol. 3677. SPIE, 1999, pp. 530–541.

[3] 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 Trans. Semiconductor Manufacturing*, vol. 28, no. 1, pp. 1–12, 2014.

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

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

[6] R. di Bella, D. Carrera, B. Rossi, P. Fragneto, and G. Boracchi, "Wafer defect map classification using sparse convolutional networks," in *International Conf. Image Analysis & Proc. (ICIAP)*. Springer, 2019, pp. 125–136.

[7] D.-Y. Du and Z. Shi, "A wafer map defect pattern classification model based on deep convolutional neural network," in *IEEE International Conf. Solid-State & Integrated Circuit Technology (ICSICT)*, 2020, pp. 1–3.

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

[9] M. B. Alawieh, F. Wang, and X. Li, "Identifying wafer-level systematic failure patterns via unsupervised learning," *IEEE Trans. Cmputer-Aided Design Integrated Circuits & Syst.*, vol. 37, no. 4, pp. 832–844, 2017.

[10] F.-L. Chen and S.-F. Liu, "A neural-network approach to recognize defect spatial pattern in semiconductor fabrication," *IEEE trans. semiconductor manufacturing*, vol. 13, no. 3, pp. 366–373, 2000.

[11] C.-F. Chien, S.-C. Hsu, and Y.-J. Chen, "A system for online detection and classification of wafer bin map defect patterns for manufacturing intelligence," *J. Production Research*, vol. 51, no. 8, pp. 2324–2338, 2013.

[12] M. B. Alawieh, F. Wang, and X. Li, "Identifying systematic spatial failure patterns through wafer clustering," in *IEEE symp. circuits & syst. (ISCAS)*, 2016, pp. 910–913.

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

[14] B. Liu, "Learning on the job: Online lifelong and continual learning," in *Proc. AAAI conf. artificial intelligence*, vol. 34, no. 09, 2020, pp. 13 544–13 549.

[15] G. Fei and B. Liu, "Breaking the closed world assumption in text classification," in *Proc. Conf. North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 506–514.

[16] J. Yoon, S. J. Hwang, and Y. Cao, "Continual learners are incremental model generalizers," *arXiv preprint arXiv:2306.12026*, 2023.

[17] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *Advances neural information processing syst.*, vol. 30, 2017.

[18] W. Zhao, J. Ye, M. Yang, Z. Lei, S. Zhang, and Z. Zhao, "Investigating capsule networks with dynamic routing for text classification," *arXiv preprint arXiv:1804.00538*, 2018.

[19] J. Choi, H. Seo, S. Im, and M. Kang, "Attention routing between capsules," in *Proc. IEEE/CVF int'l conf. computer vision workshops*, 2019.

[20] A. Bendale and T. E. Boult, "Towards open set deep networks," in *Proc. IEEE conf. computer vision & pattern recognition (CVPR)*, 2016, pp. 1563–1572.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[22] A. K. Mishra and M. Chakraborty, "Does local pruning offer task-specific models to learn effectively?" in *Proceedings of the Student Research Workshop Associated with RANLP 2021*, 2021, pp. 118–125.

[23] Y. Lin, M. B. Alawieh, W. Ye, and D. Z. Pan, "Machine learning for yield learning and optimization," in *IEEE International Test Conf. (ITC)*, 2018, pp. 1–10.

[24] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?" in *IEEE conf. digital image computing: techniques & applications (DICTA)*, 2016, pp. 1–6.

[25] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent data analysis*, vol. 6, no. 5, pp. 429–449, 2002.

[26] A. K. Mishra, A. K. Das, and N. Kandasamy, "Built-in functional testing of analog in-memory accelerators for deep neural networks," *Electronics*, vol. 11, no. 16, p. 2592, 2022.

[27] A. K. Mishra, A. Das, and N. Kandasamy, "Online performance monitoring of neuromorphic computing systems," in *2023 IEEE European Test Symposium (ETS)*. IEEE, 2023, pp. 1–4.

[28] M. E. Shaik, A. K. Mishra, and Y. Kim, "Predicting the silent data error prone devices using machine learning," in *2023 IEEE 41st VLSI Test Symposium (VTS)*. IEEE, 2023, pp. 1–4.

[29] P. R. Genssler and H. Amrouch, "Brain-inspired computing for wafer map defect pattern classification," in *IEEE International Test Conf. (ITC)*, 2021, pp. 123–132.

[30] Y. Liao, R. Latty, P. R. Genssler, H. Amrouch, and B. Yang, "Wafer map defect classification based on the fusion of pattern and pixel information," in *IEEE International Test Conf. (ITC)*, 2022, pp. 1–9.

[31] L. Shu, H. Xu, and B. Liu, "Doc: Deep open classification of text documents," *arXiv preprint arXiv:1709.08716*, 2017.

[32] M. Abd Al Rahman, S. Danishvar, and A. Mousavi, "An improved capsule network (wafercaps) for wafer bin map classification based on dcgan data upsampling," *IEEE Transactions on Semiconductor Manufacturing*, vol. 35, no. 1, pp. 50–59, 2021.