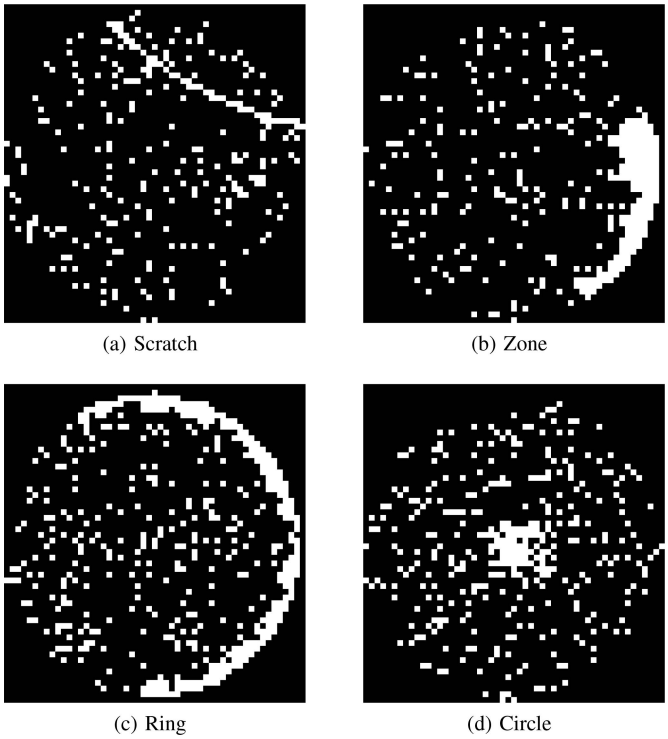
Memory-Augmented Convolutional Neural Networks With Triplet Loss for Imbalanced Wafer Defect Pattern Classification

Yunseung Hyun and Heeyoung Kim

***Abstract*—A wafer bin map (WBM) represents the wafer testing results for individual dies on a wafer using a binary value that represents pass or fail. WBMs often have specific defect patterns, which occur because of assignable causes. Therefore, the identification of defect patterns in WBMs aids in understanding the root causes of process failure. Previous studies on the classification of WBM defect patterns have demonstrated effective performances. However, in previous studies, the effect of class imbalance on the WBM defect patterns was not considered, although in practice, it is more reasonable to assume that there is a significantly large number of WBMs that lack defect patterns because they occur when there are process faults. In this paper, we propose memory-augmented convolutional neural networks with triplet loss for classifying defect patterns in highly imbalanced WBM data. We use a triplet loss-based convolutional neural network as an embedding function to obtain a well-separated low-dimensional space according to the defect patterns. We then use a memory module to balance the number of WBMs between defect-pattern classes. We train the proposed model end-to-end to learn the embedding function and update the memory simultaneously. We then validate the proposed model using simulated WBM data. The proposed model demonstrates a high classification performance and effective embedding results for imbalanced WBM data.**

***Index Terms*—Class imbalance, convolutional neural network, defect pattern classification, memory module, semiconductor manufacturing, wafer bin map.**



(a) Scratch (b) Zone (c) Ring (d) Circle

Fig. 1. Illustration of systematic defect patterns.

I. Introduction

SEMICONDUCTOR-MANUFACTURING processes consist of four major steps of wafer fabrication, wafer test, packaging, and final testing. Wafer fabrication is the procedure of building integrated circuits (ICs) on a semiconductor (e.g., silicon) wafer and comprises hundreds of complicated processes such as oxidation, photolithography, and etching. A wafer is cut into many dies, each containing a copy of the IC. After the wafer fabrication, the wafer test is performed to verify the functionality of individual dies. According to the wafer test results, each die of a wafer can be assigned a binary value, e.g., a value of 0 or 1 for good dies or defective dies, respectively. The spatial map of a wafer thus obtained is called a wafer bin map (WBM).

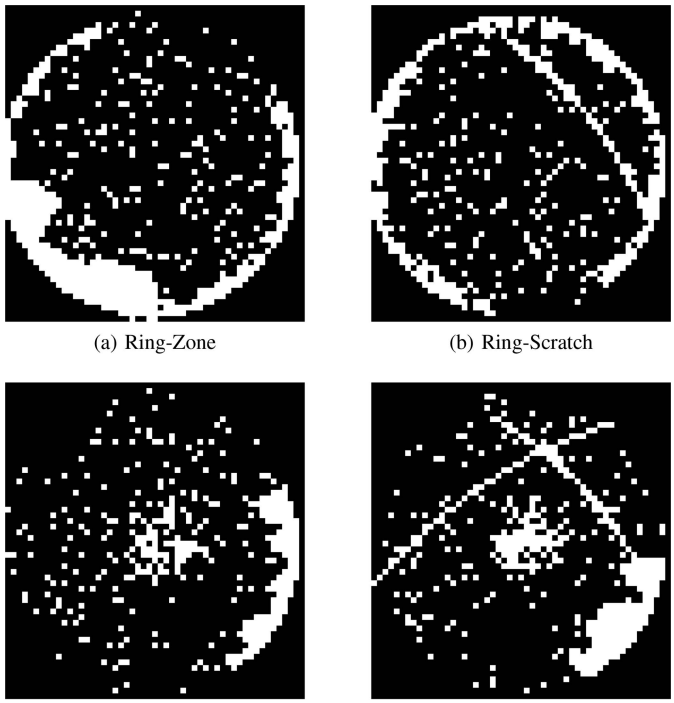
The defects in WBMs can be classified into two types, namely, random defects and systematic defects. The former are randomly distributed over a wafer, while the latter form a specific defect pattern. Random defects mainly occur owing to random causes, e.g., dust particles attached to a wafer surface, and thus, long-term strategies should be developed to eliminate these defects. In contrast, systematic defects occur owing to assignable causes. For example, in Figure 1, which illustrates typical patterns of systematic defects, the scratch pattern presented in (a) is caused by agglomerated particles and the hardening of the pad during the chemical–mechanical planarization (CMP); the zone pattern in (b) is caused by non-uniform or uneven cleaning ; the ring pattern in (c) is caused

Manuscript received May 22, 2020; revised June 25, 2020; accepted July 18, 2020. Date of publication July 21, 2020; date of current version October 29, 2020. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) under Grant 2018R1C1B6004511. *(Corresponding author: Heeyoung Kim.)*

The authors are with the Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea (e-mail: heeyoungkim@kaist.ac.kr).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSM.2020.3010984



(a) Ring-Zone (b) Ring-Scratch

(c) Circle-Zone (d) Circle-Zone-Scratch

Fig. 2. Examples of mixed-type defect patterns.

by a layer-to-layer misalignment during the storage-node process; and the circle pattern in (d) is caused by a uniformity issue occurring during the CMP [1]. Therefore, the identification of systematic defect patterns in WBMs aids in identifying process failures and understanding their root causes.

Recently, as the semiconductor circuit line width has become small, the wafer fabrication process has become complicated. Consequently, various defect patterns could be present in a single wafer (i.e., *mixed-type* defect patterns occur), which makes it difficult to detect and classify the defect patterns. Figure 2 illustrates the various mixed-type defect patterns. In Figure 2(a), the ring and zone patterns are mixed; in Figure 2(b), the ring and scratch patterns are mixed; in Figure 2(c), the circle and zone patterns are mixed; and in Figure 2(d), the circle, zone, and scratch patterns are mixed. The identification of mixed-type defect patterns in WBMs is more difficult than that of a single defect pattern, because we do not know how many defect patterns are generally mixed in a wafer.

In the extant literature, the problem of the identification of mixed-type defect patterns in WBMs has been addressed as a classification or clustering problem. To overcome the difficulties in identifying defect patterns due to the occurrence of global random defects, which act as noise, and owing to the unknown number of patterns mixed over a wafer, in many previous studies, a filtering method was applied to remove global random defects, grouping of systematic defects into clusters was performed, and the defect clusters were classified as specific patterns. For example, [2] first applied spatial filtering [3] for removing global random defects and then used the *k*-means algorithm and hierarchical clustering in combination to cluster systematic defects. The defect clusters are then classified into their specific patterns (e.g., linear, elliptical, and ring) using the Gaussian expectation-maximization algorithm and spherical shell algorithm. Reference [4] also applied spatial filtering for removing global random defects and then used a kernel-based eigen decomposition and support vector clustering in combination for grouping systematic defects into clusters. Reference [5] used the k nearest neighbor method to remove global random defects and then used a similarity-based clustering method [6] to cluster the systematic defects. Reference [1] developed connected-path filtering (CPF) to remove global random defects and then used the infinite warped mixture model to group systematic defects into clusters. In particular, they showed that CPF is particularly advantageous for detecting scratch patterns, which are difficult to detect using spatial filtering. Reference [7] directly performed classification of WBMs comprising mixed-type defect patterns using convolutional neural networks (CNNs) without removing random defects and grouping systematic defects into clusters in advance. Furthermore, they proposed an individual classification model for each of the mixed-type patterns, i.e., they considered a multi-class problem as a multi-label problem. They demonstrated the effective performance of CNNs even in cases comprising many global random defects. Similarly, [8] and [9] used CNNs to classify mixed-type defect patterns. Although not dealing with mixed-type patterns, [10] applied generative adversarial networks for classification of imbalanced single-type defect patterns. Reference [11] used Dirichlet process variational autoencoders to cluster wafer defect patterns.

Although previous studies have presented effective classification performances in the case of WBMs with mixed-type defect patterns, they have not taken into consideration the *class imbalance*, i.e., situations wherein some classes have a significantly greater number of samples than other classes. For example, when using CNNs, [7] used training samples of equal size for all the mixed-type patterns including "no systematic pattern." However, it is more reasonable to assume that we obtain a significantly larger number of WBMs lacking systematic defect patterns in practice, because systematic patterns are caused by process faults. Similarly, we would obtain more WBMs comprising a single defect pattern than WBMs comprising mixed-type defect patterns. In general, when dealing with imbalanced data, conventional classification methods tend to be biased toward the majority class [12].

In the extant literature, two main approaches to the classification of imbalanced data have been studied. One approach comprises balancing the data by over-sampling the minority-class instances or under-sampling the majority-class instances or both. For example, the synthetic minority over-sampling technique (SMOTE) [13] and Tomek links [14] are widely used for over-sampling and under-sampling, respectively. The SMOTE synthesizes minority-class samples by using nearest neighbors, while Tomek links remove majority-class samples if they are close to the minority-class samples. Another approach comprises assigning various misclassification costs to different classes using cost-sensitive learning. For example, [15] introduced cost items into the learning framework of

AdaBoost [16]. Reference [17] proposed a cost-sensitive neural network that jointly optimizes class-dependent costs and neural network parameters. Although these approaches have shown effective performances in many examples of imbalanced data classification, they also have some limitations. For example, over-sampling may cause overfitting with unwanted noise. Under-sampling may result in the loss of valuable information while discarding majority-class samples. Using cost-sensitive learning, it is difficult to find a cost matrix that fits the data well. These difficulties can become more apparent if the data are high-dimensional, such as that in the case of WBMs. Recently, [10] applied generative adversarial networks to classify wafer defect patterns from an imbalanced dataset. They addressed the class imbalance by using an adaptive generative controller that balances the sample size according to the classification accuracy while training the model. For this method, a sufficient amount of samples are needed for each class to synthesize WBMs for imbalanced learning. However, we usually have a small number of WBMs with mixed-type defect patterns, while we have a large number of WBMs without systematic defects. Therefore, it is possible that this model synthesizes inappropriate samples with bias for classes of mixed-type defect patterns [18].

In this paper, we propose a new method for classifying highly imbalanced WBM data comprising mixed-type defect patterns. The proposed method has three key characteristics. First, to extract low-dimensional features from high-dimensional WBM data, we use a CNN as an embedding function, which learns a low-dimensional embedding space where WBMs are well separated according to their defect patterns. With deep structures, CNNs can have high expressive powers and can extract useful representations. This idea had been borrowed from FaceNet [19], which learns a mapping from high-dimensional face images to a Euclidean space where distances directly correspond to face similarity using the triplet loss that takes a triplet of a reference sample, a positive sample (in the same class as that of the reference), and a negative sample (in different classes from that of the reference) as an input. The triple loss is used to minimize the distance between the reference sample and positive sample while maximizing the distance between the reference sample and negative sample. By selecting the triples with the same class probability, the training data for CNNs can be balanced. Recently, there have been attempts to use the triplet loss for imbalanced data classification [20], [21]. Similarly, we train CNNs with the triplet loss to extract low-dimensional embeddings from high-dimensional WBMs based on WBM similarity. Using the low-dimensional embeddings, we can achieve a much greater representational efficiency and, thus, a more effective classification performance.

Second, to balance the majority- and minority-class samples, we use a key-value memory module [22] that was originally proposed for remembering rare events in one-shot learning using neural networks. We use the memory module to store the information of the minority-class samples effectively. More specifically, representative examples of the low-dimensional embeddings learned by a CNN are stored in the memory as the keys with the corresponding labels as the values. The class for a new example can then be predicted as that of the nearest neighbor key from the memory. To prevent examples of the majority class from occupying more memory than the minority class during the training, we allocate the same memory size for each class. In this manner, the keys used for the prediction become balanced among the classes, which improves the classification accuracy for the imbalanced data.

Third, we train the proposed model end-to-end by formulating the loss function such that it simultaneously takes into consideration the embeddings of the CNN and the classification of the memory. The embedding and classification processes are interactive. Thus, the CNN is trained with the classification results from the memory for better embedding, while the memory is trained with the embedding results from the CNN for better classification. Therefore, it is necessary to optimize the loss used to learn both the embedding and classification processes end-to-end. When extracting features from WBMs, CNNs are known to be robust to global random defects, and previous studies to use CNNs for defect pattern classification demonstrated effective performance without a denosing step [7]. Similarly, we do not require a denoising step to remove random defects; raw data are directly used for embedding and classification.

Our contributions are summarized as follows. First, to extract low-dimensional features from high-dimensional WBMs while considering class imbalance, we use CNNs with the triplet loss. In particular, by selecting triples with the same class probability, we force the CNNs to extract features from balanced data. Second, to improve the classification accuracy for imbalanced data, we use a key-vale memory module. In particular, we modify the memory module to allocate the same memory size for each class to balance the low-dimensional features learned by CNNs among the classes. Third, to interactively learn the embedding and classification processes, we train the proposed model end-to-end. The proposed model demonstrated effective classification performance in imbalanced WBM data in our experiments in Section III.

The rest of this paper is organized as follows. Section II presents the proposed model in detail. Section III presents the validation of the proposed model via experiments. Finally, the conclusions of this study are presented in Section IV.

II. Memory-Augmented Convolutional Neural Networks With Triplet Loss

A. Overview of the Proposed Method

Figure 3 presents a schematic of the proposed method, i.e., a memory-augmented CNN with triplet loss. The proposed method consists of two major components. The first component is a CNN, which learns a mapping from high-dimensional WBMs to a low-dimensional space where distances directly correspond to WBM similarity, i.e., the WBMs comprising the same defect pattern have small distances between each other, while WBMs comprising different defect patterns have large distances between each other. Therefore, in the low-dimensional embedding space, there must be a clear separation

of the WBMs according to their defect patterns. The second component is a memory module that learns representative examples of the low-dimensional embeddings learned by a CNN for each class. The representative examples are stored in the memory as the keys with the corresponding class labels as the values. In particular, the same memory size is allocated to each class, such that the keys stored in the memory are balanced among the classes. Finally, while predicting the class label of a new WBM, the new WBM is mapped into the low-dimensional space using the embedding function that was already learned by the CNN. The low-dimensional embedding of the new WBM is then compared with the keys in the memory, and the class of the new WBM is finally predicted as the value of the nearest neighbor key from the memory. We present each component in detail in Sections II-B and II-C.

B. CNNs With Triplet Loss

CNNs are widely used for the classification of image data, and are often trained using the cross-entropy loss to learn a softmax classifier. However, in this study, a CNN is used as an embedding function rather than a classifier; the CNN is trained to map high-dimensional WBMs to a low-dimensional space. To train CNNs for this purpose, we use the triple loss [19] rather than the cross-entropy loss along with the softmax function.

The triplet loss has been widely used to learn low-dimensional embeddings from high-dimensional images in the structure of a CNN. The CNN is trained such that the squared Euclidean distance in the embedding space directly corresponds to a measure of image similarity using the triplet loss. Let us suppose that we have an embedding $f(x)$ from an image x into a d-dimensional Euclidean space, and we consider a triplet of a reference image (called the anchor and denoted by x_i^a), an image belonging to the same class as the reference image (called the positive and denoted by x_i^p), and an image belonging to a different class from the reference sample (called the negative and denoted by x_i^n). The triplet loss is used to ensure that an image is closer to all the other images of the same class than it is to images of different classes in the embedding space, which is represented using the following relationship:

$$\left\|f(x_i^a) - f(x_i^p)\right\|_2^2 + \alpha < \left\|f(x_i^a) - f(x_i^n)\right\|_2^2, \quad \forall (x_i^a, x_i^p, x_i^n) \in \mathcal{T},$$

where α is a margin that is enforced between the positive and negative pairs , $\mathcal{T}$ is the set of all possible triplets in the training set with cardinality N, and $\|\mathbf{x}\|_2 = \sqrt{\sum_i |x_i|^2}$ is the L_2-norm of vector $\mathbf{x}$.

The triplet loss that is being minimized is then given as

$$L_1 = \sum_i^N \left[\left\|f(x_i^a) - f(x_i^p)\right\|_2^2 - \left\|f(x_i^a) - f(x_i^n)\right\|_2^2 + \alpha \right]_+, \quad (1)$$

where N is the number of all possible triplets, and $[x]_+ = \max(x, 0)$.

For realizing efficient learning, it is important to select triplets that violate the constraint presented in (1) because considering all possible triplets is time consuming, and many triplets would easily satisfy the constraint. There are two variations that can be made for a better triplet selection. The first variation comprises the set from which triples are generated. One option for this is to generate triplets from the whole training data. This option can always generate the best triplets for learning but has a high computational cost. Another option is to generate triplets within a mini-batch. This option is computationally efficient but requires that the training data be carefully split into mini-batches.

The second variation is the strategy of selecting the positive and negative . One possible strategy, called *Batch Hard*, comprises the use of the hardest triplet, i.e., the farthest sample of the same class as the positive and the closest sample of different classes as the negative. These hardest triplets can contribute to a more significant reduction in the training loss than other possible triplets. However, this strategy may result in bad local minima that every sample maps to 0 (i.e., $f(x) = 0$) early in training. Another possible strategy, called *Batch All*, comprises the use of all triplets except the triplets having a zero loss (i.e., $\left\|f(x_i^a) - f(x_i^p)\right\|_2^2 + \alpha < \left\|f(x_i^a) - f(x_i^n)\right\|_2^2$). According to [23], this strategy is more stable in earlier steps; however, the performance is poorer than that obtained on using the hardest triplets.

In consideration of the computational efficiency and class imbalance, we generated triples within a mini-batch using the *Batch All* triplet selection strategy. When configuring the mini-batch, we sample the WBMs of each class uniformly such that all the class probabilities are equal in the mini-batch. More specifically, we first randomly select a class, and we then randomly sample the WBMs having the selected class label from the training data. Using this sampling procedure, we encourage class balance in the mini-batch and ensure the stable learning of the model.

C. Memory Module

Reference [22] proposed a key-value memory module that effectively memorizes rare events. The memory module $\mathcal{M}$ consists of multiple slots, and each slot consists of three components: a matrix $\mathbf{K}$ of memory keys, vector $\mathbf{V}$ of memory values, and vector A of the age of items stored in the memory:

$$\mathcal{M} = (\mathbf{K}_{m \times d}, \mathbf{V}_m, \mathbf{A}_m), \quad (2)$$

where m is the number of slots in the memory, and d is the dimension of the low-dimensional embeddings learned by the CNN. Here, the keys are outputs of a selected layer of a CNN, and the values are the ground-truth labels for the corresponding keys. To make the memory module more effective for the class imbalance problem, we allocate an equal number of slots to each class in contrast to the original proposal in [22]. We thus force $\mathbf{V}$ to contain an equal number of possible values (i.e., labels) as its entry.

The memory operates via three major steps. Let us suppose that an input vector $\mathbf{q}$ is given as a query with the correct label c, and $\mathbf{q}$ is normalized, i.e., $\|\mathbf{q}\|_2 = 1$. The nearest neighbor NN of $\mathbf{q}$ in $\mathcal{M}$ is defined as the keys that maximize the dot

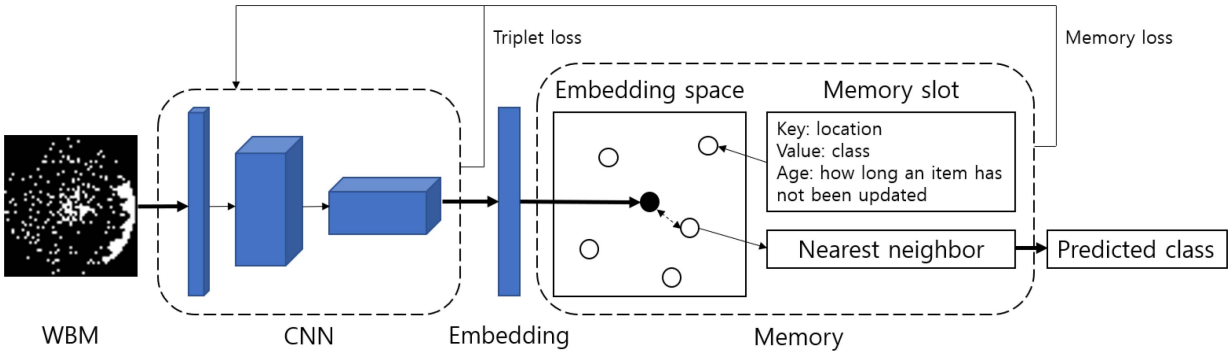
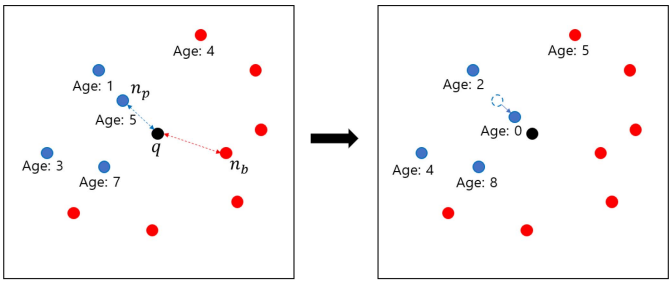
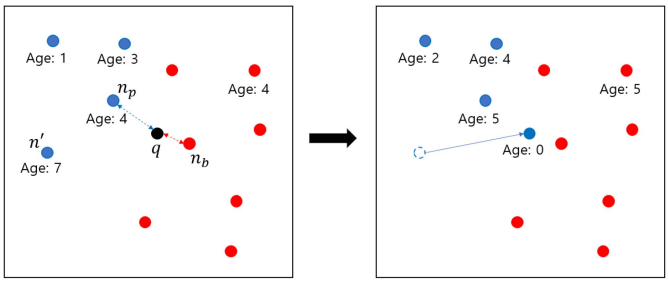


Fig. 3. A schematic of the proposed method.



(a) Case 1 : the memory returns the correct value.



(b) Case 2 : the memory returns an incorrect value

Fig. 4. Two cases of the memory update procedure.

product with $\mathbf{q}$:

$$NN(\mathbf{q}, \mathcal{M}) = \text{argmax}_i \ \mathbf{q} \cdot \mathbf{K}[i]. \tag{3}$$

First, given a query $\mathbf{q}$, two neighbors of $\mathbf{q}$ are selected from the memory $\mathcal{M}$: the positive neighbor n_p, which is the nearest neighbor of $\mathbf{q}$, while satisfying $\mathbf{V}[n_p] = c$, and the negative neighbor n_b, which is the nearest neighbor of $\mathbf{q}$, while satisfying $\mathbf{V}[n_b] \neq c$.

The memory loss is then defined as

$$L_2 = \left[\|\mathbf{q} - \mathbf{K}[n_p]\|_2^2 - \|\mathbf{q} - \mathbf{K}[n_b]\|_2^2 + \beta \right]_+, \tag{4}$$

where β is a margin enforced between the positive and negative neighbors. To ensure that the memory loss L_2 is consistent with the triple loss L_1 in (1), which is based on the L_2 norm, we change the memory loss to that shown in (4) instead of using the cosine similarity proposed in [22]. The loss is propagated to the CNNs, such that the CNNs learn to generate $\mathbf{q}$ such that it is close to the positive neighbor but far from the negative neighbor.

Lastly, the memory is updated using the query $\mathbf{q}$ and the corresponding class label c in the order in which the query $\mathbf{q}$ enters the module. Figure 4 illustrates the update procedure of the memory. In the figure, the black point represents the query, blue points represent the memory slots having the same value as that of the query, and red points represent the memory slots having different values from that of the query. The update procedure differs according to whether the memory returns the correct value. First, the memory can return the correct value, i.e., $\mathbf{K}[n_p]$ is closer to $\mathbf{q}$ than $\mathbf{K}[n_b]$ is, as in Figure 4(a). This means that n_p can well represent the information contained in $\mathbf{q}$. In other words, the information contained in n_p and $\mathbf{q}$ can be considered similar. Therefore, instead of assigning a new memory slot to $\mathbf{q}$, we incorporate the information of $\mathbf{q}$ into n_p by combining the keys of n_p and $\mathbf{q}$ using the normalized average. That is, we update the key of n_p by computing the normalized average of the current key and $\mathbf{q}$, and we reset the age:

$$\mathbf{K}[n_p] \leftarrow \frac{\mathbf{q} + \mathbf{K}[n_p]}{\|\mathbf{q} + \mathbf{K}[n_p]\|_2}, \ \mathbf{A}[n_p] \leftarrow 0. \tag{5}$$

In contrast, as in Figure 4(b), the memory can return an incorrect value, i.e., $\mathbf{K}[n_p]$ is further away from $\mathbf{q}$ than $\mathbf{K}[n_b]$ is. This means that $\mathbf{K}[n_p]$ cannot well represent the information contained in $\mathbf{q}$, and $\mathbf{q}$ can provide new information for class c. Then, we assign a new memory slot to $\mathbf{q}$. For a memory slot to store $\mathbf{q}$, we pick the memory slot n' having the maximum age among those of the same class as $\mathbf{q}$. This restriction to the same class as that of $\mathbf{q}$ ensures that the memory values are balanced among the classes during training. The key and value of n' are then updated as $\mathbf{q}$ and c, respectively, and the age is reset:

$$\mathbf{K}[n'] \leftarrow \mathbf{q}, \ \mathbf{A}[n'] \leftarrow 0. \tag{6}$$

Through this memory update strategy, the memory keeps incorporating new queries on its memory keys. As a result, one memory key becomes the normalized average of multiple queries of samples, and therefore, one memory key can represent multiple samples in the training data. After the update procedure, the age of all the non-updated slots is increased by one. Finally, the value of c of the input vector $\mathbf{q}$ is predicted using the nearest neighbors method:

$$\hat{c} = \mathbf{V}[NN(\mathbf{q}, \mathcal{M})].$$

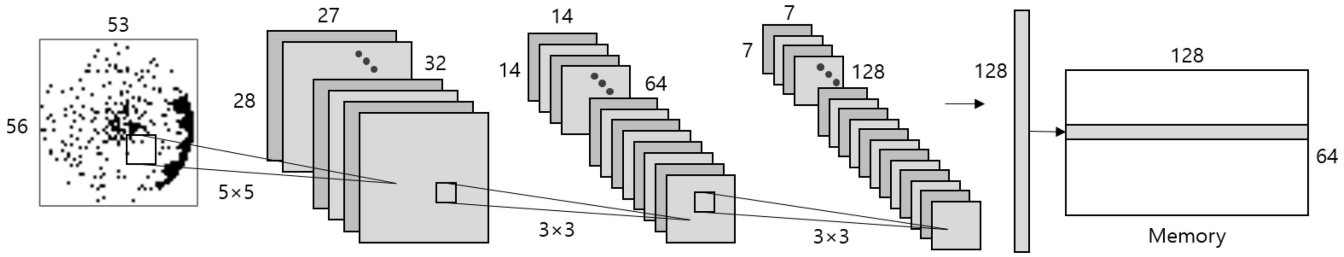


Fig. 5. Structure of a memory-augmented CNN with the triple loss used in this study.

TABLE I
CONFIGURATION OF THREE TRAINING SETS, NAMELY SET A, SET B, AND SET C, AND ONE TEST SET

Type	No. of Classes	No. of WBMs for Set A	No. of WBMs for Set B	No. of WBMs for Set C	No. of WBMs for Test set
No pattern	1	2900	2180	200	300
Single-type	4	80	800	800	1200
Mixed-type	11	220	220	2200	3300
Total	16	3200	3200	3200	4800

D. Training

We propose the use of a unified learning framework for the proposed method. Considering the interactive processes of the embeddings of the CNN and classification of the memory, we train the proposed model end-to-end using the triplet loss and memory loss jointly. Once a low-dimensional embedding learned by the CNN comes into the memory module as a query, the memory returns the memory loss L_2 to the CNN. The CNN is then trained using backpropagation with a total loss L, which is the weighted sum of the triplet loss L_1 and the memory loss L_2, i.e., $L = w_1 L_1 + w_2 L_2$, where $w_1, w_2 \geq 0$ and $w_1 + w_2 = 1$. To investigate the impact of different weights on the performance, we performed experiments with various ratios between w_1 and w_2, but there was no significant difference in test accuracy. In this study, we used 1:1 ratio of w_1 and w_2. During the backpropagation, we fix a key matrix $\mathbf{K}$ in L_2.

For example, we consider a classification problem for 56×53 WBMs of 16 classes (the data are described in detail in Section III-A). Figure 5 illustrates the structure of a memory-augmented CNN with the triple loss used in this study. The CNN learns an embedding $\mathbf{q} = f(x)$ using input WBMs from a mini-batch, where $x \in \mathbb{R}^{56\times53}$ is a WBM, and $\mathbf{q} \in \mathbb{R}^{128}$ is a low-dimensional embedding. We constrain the embedding with $\|\mathbf{q}\|_2 = 1$ for realizing stable learning. The 128-dimensional embedding $\mathbf{q}$ then becomes a query for the memory module of 64 slots. It should be noted that each of the 16 classes takes an equal number of four slots for handling the imbalanced data appropriately, as discussed in Section II-C. In terms of the architecture of the CNN, we use three convolutional layers. The first convolutional layer consists of 32 filters of size 5×5 and stride 1. The second and third convolutional layers consist of 64 and 128 filters, respectively, both of size 3×3 and stride 1. Following each convolutional layer, we apply the rectified linear unit (ReLU) activation function and 2×2 max-pooling. After the last max-pooling, we apply a fully connected layer with the output dimension $d = 128$. This output becomes the input to the memory model. The memory then returns the memory loss L_2 to the CNN, and the CNN is trained to minimize the total loss L from the backpropagation. To train the model, we used adaptive moment estimation (Adam) [24] with a learning rate of 0.0001 and a gradient clipping threshold of 5.0, with the batch size of 128.

In this study, we optimized the number of slots in the memory as a hyperparameter in a heuristic manner. More specifically, we considered four cases of 32, 64, 96, and 128 slots and evaluated the corresponding classification performances. In the case of 32 slots, the memory module resulted in a low classification performance. In other cases, there was no significant difference in the performances; thus, we determined that the use of 64 slots of the memory module provided the most efficient computation . As discussed in Section II-C, we set the memory values such that they were fixed and balanced among the classes. We also attempted to use floating memory values by using the memory updating rule without a restriction to the same class when selecting n' in (6). However, the number of slots allocated to the minority classes then became zero during the training. Other hyperparameters were similarly optimized in a heuristic manner. In the loss function, α and β were both set as 0.4. Also, we initialized the memory key matrix $\mathbf{K}$ as a zero matrix.

III. EXPERIMENTS

A. Data

To evaluate the performance of the proposed method, we performed experiments using simulated datasets. We simulated the data according to the WBM data-generation procedure used in [7]. The size of each WBM was 56×53, i.e., each WBM contained a total of 2968 dies. We considered all the 16 possible mixed-type defect patterns generated from a combination of circle, ring, zone, and scratch patterns. Examples of the generated data are presented in Figures 1 and 2. To investigate the impacts of the degree of imbalance on the classification performances, we generated three different datasets for training and one dataset for testing. Each training set comprised

a total of 3200 WBMs with different degrees of imbalance. The test set comprised a total of 4800 WBMs and was class-balanced so that the performance of the proposed method could be properly evaluated not only for majority classes but also for minority classes. The same test set was used with the three training sets for fair comparison.

Table I summarizes the configuration of three training sets—namely, Sets A, B, and C—and one test set. We categorized the WBMs of the 16 classes into three types: WBMs with no defect patterns, WBMs with a single defect pattern, and WBMs with mixed-type defect patterns (i.e., two or more defect patterns). Set A consisted of 2900 WBMs with no defect patterns and 20 WBMs for each of the four possible single-type defect patterns and the 11 possible mixed-type defect patterns. We considered this set as the severe class-imbalance dataset. Set B consisted of 2180 WBMs with no defect patterns, 200 WBMs for each of the four possible single-type defect patterns, and 20 WBMs for each of the 11 possible mixed-type defect patterns. We considered this set as the moderate class-imbalance dataset. Lastly, Set C was the class-balance dataset and consisted of 200 WBMs for each of all the 16 classes. For Sets A, B, and C, the class-imbalance degree was measured as 145, 109, and 1, respectively, using the imbalance ratio (IR), which is calculated as the ratio of the sample size of the largest majority class to that of the smallest minority class. A larger value of IR indicates a higher degree of imbalance.

B. Evaluation

For each experiment, we trained the proposed model using 5000 iterations, such that the loss was converged. For each iteration, we constructed one mini-batch as discussed in Section II-B to train the model. We checked whether overfitting occurred by evaluating the training and validation losses. The validation loss was evaluated on a separate dataset of balanced 512 samples. Figure 6 presents the training and validation losses calculated using the proposed model trained using Sets A, B, and, C, where the training and validation losses are indicated by the blue and orange lines, respectively. We can see that the validation loss converged properly without signs of overfitting.

For each different training set in Table I, we repeated the experiments 10 times and determined the averaged performance of the proposed model. For each repeated experiment, we regenerated the training and test datasets to evaluate the performance of the proposed method over various training and test sets. We considered three performance measures of accuracy, precision, and recall, which were calculated as follows:

- *Accuracy:* $\text{Acc} = \frac{\sum_i TP_i}{N}$,
- *Precision:* $\text{Pre}_i = \frac{TP_i}{TP_i+FP_i}$,
- *Recall:* $\text{Rec}_i = \frac{TP_i}{TP_i+FN_i}$,

where N is the total number of all WBMs in the test set; TP_i is the number of true positives, i.e., correctly classified WBMs of the ith class; FP_i is the number of false positives, i.e., WBMs misclassified as the ith class; and FN_i is the number of false negatives, i.e., WBMs of the ith class that are misclassified.

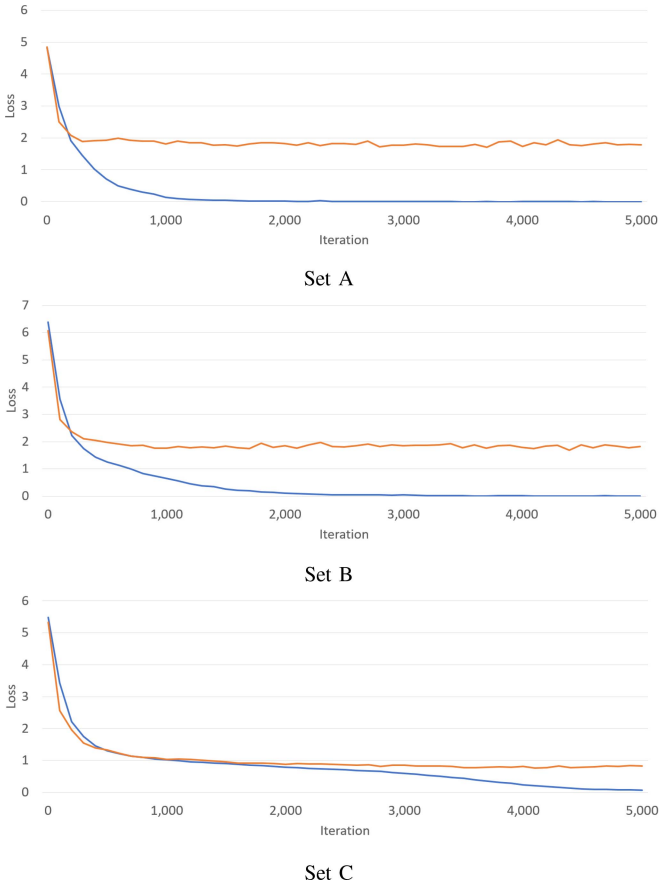


Fig. 6. The training and validation losses for Sets A, B, and C (blue: training loss, orange: validation loss).

TABLE II
CHARACTERISTICS OF EACH COMPETING METHOD

Model	Loss	Dimension of embeddings	Classifier
CNN-softmax	cross-entropy	128	softmax
CNN+memory	cross-entropy+memory	128	kNN
CNN+kNN	triplet	128	kNN
A-SVM	adaptive cost	-	SVM
Proposed	triplet+memory	128	kNN

TABLE III
MEAN TEST ACCURACY OF EACH COMPETING METHOD OVER 10 EXPERIMENTS. STANDARD ERRORS IN PARENTHESES

Set	CNN –softmax	CNN +memory	CNN +kNN	A-SVM	Proposed
A	0.630 (0.027)	0.633 (0.027)	0.680 (0.025)	0.525 (0.008)	**0.785** (0.022)
B	0.662 (0.012)	0.721 (0.019)	0.793 (0.017)	0.488 (0.008)	**0.823** (0.013)
C	**0.903** (0.014)	0.897 (0.009)	0.866 (0.009)	0.630 (0.013)	0.888 (0.016)

It should be noted that accuracy is defined over the entire test data, whereas precision and recall are class-specific.

We compared the performance of the proposed method with those of four different competing methods. The first competing method is a CNN with a softmax classifier trained using the cross-entropy loss. This is a common method of the use of CNNs for supervised learning. We denote this method as

TABLE IV
PRECISION AND RECALL OF THE COMPETING METHODS ON SET A

Class	CNN–softmax		CNN+memory		CNN+kNN		A-SVM		Proposed	
	Pre	Rec	Pre	Rec	Pre	Rec	Pre	Rec	Pre	Rec
None	**0.583**	**1.000**	0.454	**1.000**	0.372	**1.000**	0.383	**1.000**	0.559	**1.000**
	(0.048)	(0.000)	(0.044)	(0.000)	(0.028)	(0.000)	(0.013)	(0.000)	(0.043)	(0.000)
Z	0.660	0.591	0.629	0.474	0.689	0.574	0.532	0.461	**0.877**	**0.693**
	(0.040)	(0.101)	(0.086)	(0.076)	(0.063)	(0.141)	(0.029)	(0.058)	(0.067)	(0.029)
R	0.723	0.784	0.697	0.841	**0.790**	0.868	0.514	0.710	0.761	**0.990**
	(0.062)	(0.109)	(0.057)	(0.100)	(0.065)	(0.089)	(0.025)	(0.125)	(0.050)	(0.014)
RZ	0.537	0.533	0.697	0.513	0.658	0.521	0.500	0.499	**0.766**	**0.767**
	(0.072)	(0.127)	(0.081)	(0.130)	(0.087)	(0.103)	(0.038)	(0.080)	(0.065)	(0.085)
S	0.737	0.336	0.597	0.197	**0.916**	0.191	0.569	0.011	0.808	**0.470**
	(0.099)	(0.108)	(0.214)	(0.104)	(0.120)	(0.056)	(0.241)	(0.007)	(0.108)	(0.081)
SZ	0.621	0.541	0.635	0.516	0.838	0.646	0.590	0.427	**0.875**	**0.724**
	(0.065)	(0.101)	(0.070)	(0.108)	(0.070)	(0.101)	(0.038)	(0.058)	(0.054)	(0.102)
SR	0.601	0.624	0.693	0.600	0.678	**0.731**	0.536	0.501	**0.917**	0.693
	(0.100)	(0.096)	(0.094)	(0.110)	(0.057)	(0.095)	(0.052)	(0.077)	(0.058)	(0.091)
SRZ	0.585	0.486	0.638	0.636	0.679	0.580	0.573	0.439	**0.851**	**0.736**
	(0.074)	(0.101)	(0.086)	(0.105)	(0.052)	(0.114)	(0.045)	(0.057)	(0.051)	(0.101)
C	0.718	0.914	0.686	0.922	**0.762**	0.965	0.605	0.774	0.693	**0.997**
	(0.066)	(0.048)	(0.087)	(0.065)	(0.055)	(0.039)	(0.032)	(0.060)	(0.070)	(0.009)
CZ	0.655	0.566	0.691	0.591	0.759	0.710	0.554	0.548	**0.863**	**0.757**
	(0.063)	(0.075)	(0.061)	(0.088)	(0.071)	(0.085)	(0.029)	(0.064)	(0.054)	(0.088)
CR	0.704	0.785	0.695	0.827	0.698	0.819	0.512	0.717	**0.776**	**0.971**
	(0.053)	(0.088)	(0.054)	(0.067)	(0.071)	(0.067)	(0.028)	(0.084)	(0.053)	(0.026)
CRZ	0.493	0.467	0.604	0.473	0.535	0.603	0.517	0.444	**0.815**	**0.682**
	(0.091)	(0.109)	(0.123)	(0.072)	(0.082)	(0.066)	(0.041)	(0.076)	(0.085)	(0.144)
CS	0.769	0.708	0.793	0.750	0.828	**0.812**	0.674	0.506	**0.860**	0.742
	(0.096)	(0.080)	(0.080)	(0.101)	(0.087)	(0.078)	(0.043)	(0.080)	(0.069)	(0.087)
CSZ	0.593	0.547	0.595	0.536	0.778	0.635	0.596	0.451	**0.873**	**0.725**
	(0.032)	(0.078)	(0.112)	(0.101)	(0.085)	(0.093)	(0.060)	(0.037)	(0.042)	(0.061)
CSR	0.603	0.553	0.658	0.540	0.747	**0.689**	0.536	0.434	**0.880**	0.668
	(0.054)	(0.087)	(0.081)	(0.125)	(0.083)	(0.070)	(0.053)	(0.065)	(0.093)	(0.095)
CSRZ	0.570	0.539	0.615	0.559	0.718	0.608	0.586	0.472	**0.821**	**0.800**
	(0.082)	(0.136)	(0.091)	(0.164)	(0.070)	(0.061)	(0.036)	(0.074)	(0.085)	(0.120)

CNN–softmax. The second method comprises learning lower-dimensional embeddings using a CNN with the triplet loss and then performing classification on the embeddings using the k nearest neighbor method with $k = 3$. This model is similar to the proposed method in terms of the use of the CNN as an embedding function with the triple loss, but it does not include a memory module. We denote this method as CNN+kNN. The third method comprised the use of a CNN trained with the cross-entropy loss and combined with a memory module. The output of the final layer is used as the query to the memory module, and the final prediction is returned as the nearest neighbor in the memory. This method is similar to the proposed method in terms of the use of a memory module, but it comprises the use of the cross-entropy loss instead of the triple loss. We denote this method as CNN+memory. For the CNN–softmax, CNN+kNN, and CNN+memory methods, we used a CNN of the same structure as that used in our proposed method. The fourth method comprises the use of a support vector machine with adaptive class weights [25]. We denote this method as A-SVM. We implemented this method using the SVM package in [26]. The characteristics of each competing method are summarized in Table II.

C. Results

Table III summarizes the average of the accuracy for each of the competing methods for Sets A, B, and C over 10 experiments, with the standard errors in parentheses. For each set, the best score is indicated in boldface. It can be observed that, for Sets A and B, the proposed method achieved a higher performance than other methods. This demonstrates that the proposed method is effective in dealing with imbalanced data. For Sets A and B, CNN+kNN demonstrated a better performance than CNN–softmax, which shows that the use of CNN as an embedding function is more advantageous than its use as a classifier for imbalanced data classification. The proposed method comprised the use of an additional memory module as compared with CNN+kNN. The comparison between CNN+kNN and the proposed method demonstrates the advantage of using the memory module for classification of imbalanced data. The cause of this advantage may be that the memory acts as a down-sampling method. In the memory, each class takes the same number of memory slots, and one memory key represents multiple samples in the training data. That is, the memory learns the class representation while maintaining the class balance, which could result in a higher classification performance of the proposed method. In contrast, the comparison between CNN+memory and the proposed method shows the advantage of using the triplet loss instead of the cross-entropy loss for the classification of imbalanced data. It should be noted that the cross-entropy loss evaluates the class probabilities for each sample individually and is then averaged over all the samples. The cross-entropy

TABLE V
PRECISION AND RECALL OF THE COMPETING METHODS ON SET B

Class	CNN–softmax		CNN+memory		CNN+kNN		A-SVM		Proposed	
	Pre	Rec	Pre	Rec	Pre	Rec	Pre	Rec	Pre	Rec
None	**0.886**	0.997	0.877	**1.000**	0.625	**1.000**	0.580	0.970	0.790	**1.000**
	(0.062)	(0.008)	(0.057)	(0.000)	(0.049)	(0.000)	(0.044)	(0.025)	(0.052)	(0.000)
Z	0.588	**0.974**	0.630	0.956	0.690	0.813	0.445	0.806	**0.867**	0.900
	(0.023)	(0.028)	(0.065)	(0.043)	(0.081)	(0.095)	(0.022)	(0.032)	(0.049)	(0.058)
R	0.634	0.996	0.601	**1.000**	**0.713**	0.996	0.342	0.998	0.700	**1.000**
	(0.047)	(0.009)	(0.034)	(0.000)	(0.032)	(0.008)	(0.012)	(0.004)	(0.055)	(0.000)
RZ	0.658	0.551	0.768	0.372	**0.861**	0.627	0.521	0.355	0.843	**0.749**
	(0.092)	(0.075)	(0.102)	(0.077)	(0.084)	(0.100)	(0.021)	(0.061)	(0.080)	(0.078)
S	0.806	0.864	0.770	**0.885**	**0.880**	0.736	0.684	0.439	0.817	0.839
	(0.090)	(0.087)	(0.065)	(0.055)	(0.053)	(0.082)	(0.036)	(0.118)	(0.063)	(0.070)
SZ	0.797	0.205	0.824	0.297	0.957	0.646	0.637	0.077	**0.958**	**0.678**
	(0.140)	(0.083)	(0.137)	(0.110)	(0.047)	(0.087)	(0.099)	(0.036)	(0.040)	(0.084)
SR	0.742	0.385	0.834	0.480	0.875	0.622	0.488	0.033	**0.886**	**0.700**
	(0.081)	(0.093)	(0.111)	(0.079)	(0.068)	(0.081)	(0.117)	(0.021)	(0.058)	(0.065)
SRZ	0.668	0.543	0.738	0.648	0.901	0.711	0.602	0.383	**0.956**	**0.721**
	(0.062)	(0.126)	(0.069)	(0.045)	(0.052)	(0.088)	(0.040)	(0.064)	(0.030)	(0.081)
C	0.627	0.997	0.611	**1.000**	**0.678**	0.991	0.410	0.997	0.669	**1.000**
	(0.066)	(0.009)	(0.057)	(0.000)	(0.054)	(0.020)	(0.010)	(0.002)	(0.073)	(0.000)
CZ	0.618	0.486	0.826	0.643	0.798	**0.798**	0.524	0.397	**0.867**	0.793
	(0.092)	(0.101)	(0.070)	(0.090)	(0.050)	(0.068)	(0.013)	(0.036)	(0.039)	(0.091)
CR	0.733	0.775	0.776	0.929	0.774	0.959	0.518	0.603	**0.790**	**0.985**
	(0.059)	(0.096)	(0.077)	(0.060)	(0.053)	(0.048)	(0.034)	(0.079)	(0.042)	(0.015)
CRZ	0.534	0.555	0.739	0.639	0.781	0.743	0.548	0.453	**0.817**	**0.791**
	(0.106)	(0.141)	(0.091)	(0.099)	(0.094)	(0.101)	(0.029)	(0.040)	(0.071)	(0.069)
CS	0.826	0.506	0.813	0.594	**0.889**	**0.786**	0.443	0.008	0.841	0.781
	(0.094)	(0.140)	(0.093)	(0.130)	(0.073)	(0.054)	(0.331)	(0.009)	(0.072)	(0.067)
CSZ	0.562	0.556	0.702	**0.714**	0.894	0.700	0.600	0.354	**0.962**	0.681
	(0.087)	(0.094)	(0.103)	(0.125)	(0.088)	(0.072)	(0.034)	(0.030)	(0.038)	(0.091)
CSR	0.608	0.583	0.765	0.695	**0.864**	**0.776**	0.537	0.454	0.863	0.770
	(0.065)	(0.112)	(0.083)	(0.146)	(0.071)	(0.088)	(0.039)	(0.075)	(0.051)	(0.059)
CSRZ	0.621	0.527	0.757	0.563	0.889	**0.766**	0.609	0.483	**0.948**	0.695
	(0.063)	(0.105)	(0.072)	(0.134)	(0.062)	(0.078)	(0.025)	(0.051)	(0.057)	(0.101)

loss can then be very small even if the classifier produces poor predictions for minor classes and is dominated by the most prevalent class. Therefore, it would not be advisable to use the cross-entropy loss in the classification of imbalanced data. In contrast, in the case of Set C, CNN–softmax demonstrated the best performance, although the proposed method also demonstrated a comparable performance. This shows that the use of the softmax classifier with the cross-entropy loss is more advantageous for the classification of balanced data. The performance of A-SVM was the worst for all three sets; this may be because it uses raw data without feature extraction, which is not effective for handling high-dimensional image data.

Tables IV, V, and VI summarize the average precision and recall of each competing method for each class on Sets A, B, and C, respectively. The values in parentheses represent the standard errors. In the three tables, the label "None" denotes the class without any patterns, and "C," "R," "S," and "Z" denote the classes of the circle, ring, scratch, and zone patterns, respectively. The label "RZ" denotes the class comprising a mixture of ring and zone patterns, and other classes of mixed-type defect patterns are similarly denoted. For each class, the best scores are indicated in boldface. In Tables IV, V, and VI, we observe the same tendency of the precision and recall as that of the accuracy observed in Table III. That is, for Sets A and B, which are imbalanced datasets, overall, the proposed method achieved a significantly higher precision and recall than the other competing methods. The performance gap between CNN+kNN and the proposed method for Set B tended to be smaller than that for Set A. This shows that the advantage of balancing the classes through the memory is reduced as the class imbalance is alleviated. In the case of Set C, all the methods except A-SVM produced a comparable precision and recall.

Furthermore, we investigated whether the embeddings learned by the CNN for our method were formed such that they were suitable for the classification task. For 640 randomly selected testing WBMs, we visualized the embeddings learned using the proposed method with the true label and compared them with those of CNN–softmax, CNN+memory, and CNN+kNN in Figure 7, where the various colors indicate different clusters. The embeddings learned using each method were originally 128-dimensional, but the dimensions were further reduced to two using t-distributed stochastic neighbor embedding (t-SNE) to map the embeddings into two-dimensional spaces [27]. In both CNN+kNN and the proposed method, the embeddings were directly learned using CNN as an embedding function, but in CNN+kNN, only the triplet loss was used, while in the proposed method, a combination of the triple loss and memory loss was used. In both

TABLE VI
PRECISION AND RECALL OF THE COMPETING METHODS ON SET C

Class	CNN–softmax		CNN+memory		CNN+kNN		A-SVM		Proposed	
	Pre	Rec	Pre	Rec	Pre	Rec	Pre	Rec	Pre	Rec
None	**0.970**	0.982	0.921	**0.997**	0.834	**0.997**	0.687	0.895	0.867	**0.997**
	(0.037)	(0.022)	(0.056)	(0.008)	(0.053)	(0.010)	(0.053)	(0.049)	(0.070)	(0.008)
Z	0.908	**0.921**	0.894	0.909	0.883	0.905	0.590	0.589	**0.932**	0.900
	(0.052)	(0.045)	(0.064)	(0.070)	(0.059)	(0.054)	(0.040)	(0.082)	(0.038)	(0.058)
R	**0.913**	0.943	0.885	**0.994**	0.863	0.950	0.606	0.760	0.856	0.976
	(0.030)	(0.037)	(0.029)	(0.014)	(0.032)	(0.022)	(0.048)	(0.121)	(0.048)	(0.044)
RZ	0.862	0.862	0.844	0.862	0.822	0.836	0.612	0.508	**0.873**	**0.900**
	(0.073)	(0.059)	(0.078)	(0.053)	(0.061)	(0.093)	(0.034)	(0.047)	(0.067)	(0.060)
S	**0.938**	**0.964**	0.900	0.958	0.914	0.914	0.718	0.676	0.903	0.924
	(0.053)	(0.047)	(0.061)	(0.048)	(0.048)	(0.043)	(0.037)	(0.087)	(0.054)	(0.070)
SZ	0.902	**0.862**	0.910	0.773	0.946	0.790	0.646	0.482	**0.976**	0.832
	(0.043)	(0.074)	(0.066)	(0.094)	(0.030)	(0.056)	(0.053)	(0.101)	(0.030)	(0.067)
SR	0.879	0.898	**0.933**	**0.922**	0.801	0.856	0.579	0.663	0.890	0.854
	(0.052)	(0.044)	(0.041)	(0.036)	(0.049)	(0.049)	(0.053)	(0.103)	(0.037)	(0.074)
SRZ	0.893	**0.824**	0.928	0.760	0.937	0.694	0.660	0.492	**0.958**	0.798
	(0.043)	(0.076)	(0.061)	(0.089)	(0.070)	(0.051)	(0.036)	(0.061)	(0.041)	(0.059)
C	**0.986**	0.958	0.914	0.992	0.831	0.974	0.638	0.930	0.823	**0.997**
	(0.019)	(0.042)	(0.027)	(0.013)	(0.073)	(0.038)	(0.040)	(0.027)	(0.042)	(0.009)
CZ	0.882	0.911	0.885	**0.934**	0.860	0.876	0.603	0.599	**0.897**	0.868
	(0.030)	(0.048)	(0.030)	(0.035)	(0.060)	(0.033)	(0.037)	(0.070)	(0.042)	(0.066)
CR	**0.939**	0.940	0.911	0.977	0.840	0.949	0.608	0.759	0.868	**0.985**
	(0.023)	(0.037)	(0.045)	(0.022)	(0.059)	(0.030)	(0.048)	(0.063)	(0.035)	(0.018)
CRZ	0.800	**0.867**	0.844	0.824	0.796	0.853	0.618	0.471	**0.852**	0.858
	(0.042)	(0.029)	(0.062)	(0.082)	(0.075)	(0.083)	(0.021)	(0.067)	(0.063)	(0.103)
CS	**0.948**	**0.986**	0.893	0.964	0.895	0.900	0.732	0.566	0.892	0.908
	(0.036)	(0.015)	(0.073)	(0.041)	(0.034)	(0.047)	(0.030)	(0.096)	(0.049)	(0.056)
CSZ	0.896	**0.836**	0.926	0.767	0.899	0.781	0.667	0.504	**0.946**	0.767
	(0.042)	(0.033)	(0.066)	(0.072)	(0.050)	(0.083)	(0.024)	(0.080)	(0.048)	(0.049)
CSR	0.872	0.905	**0.884**	**0.923**	0.857	0.874	0.570	0.662	0.856	0.853
	(0.062)	(0.033)	(0.037)	(0.048)	(0.046)	(0.065)	(0.028)	(0.065)	(0.050)	(0.064)
CSRZ	0.887	**0.785**	**0.937**	0.746	**0.937**	0.746	0.648	0.520	0.931	0.756
	(0.021)	(0.067)	(0.038)	(0.071)	(0.047)	(0.071)	(0.028)	(0.063)	(0.061)	(0.072)

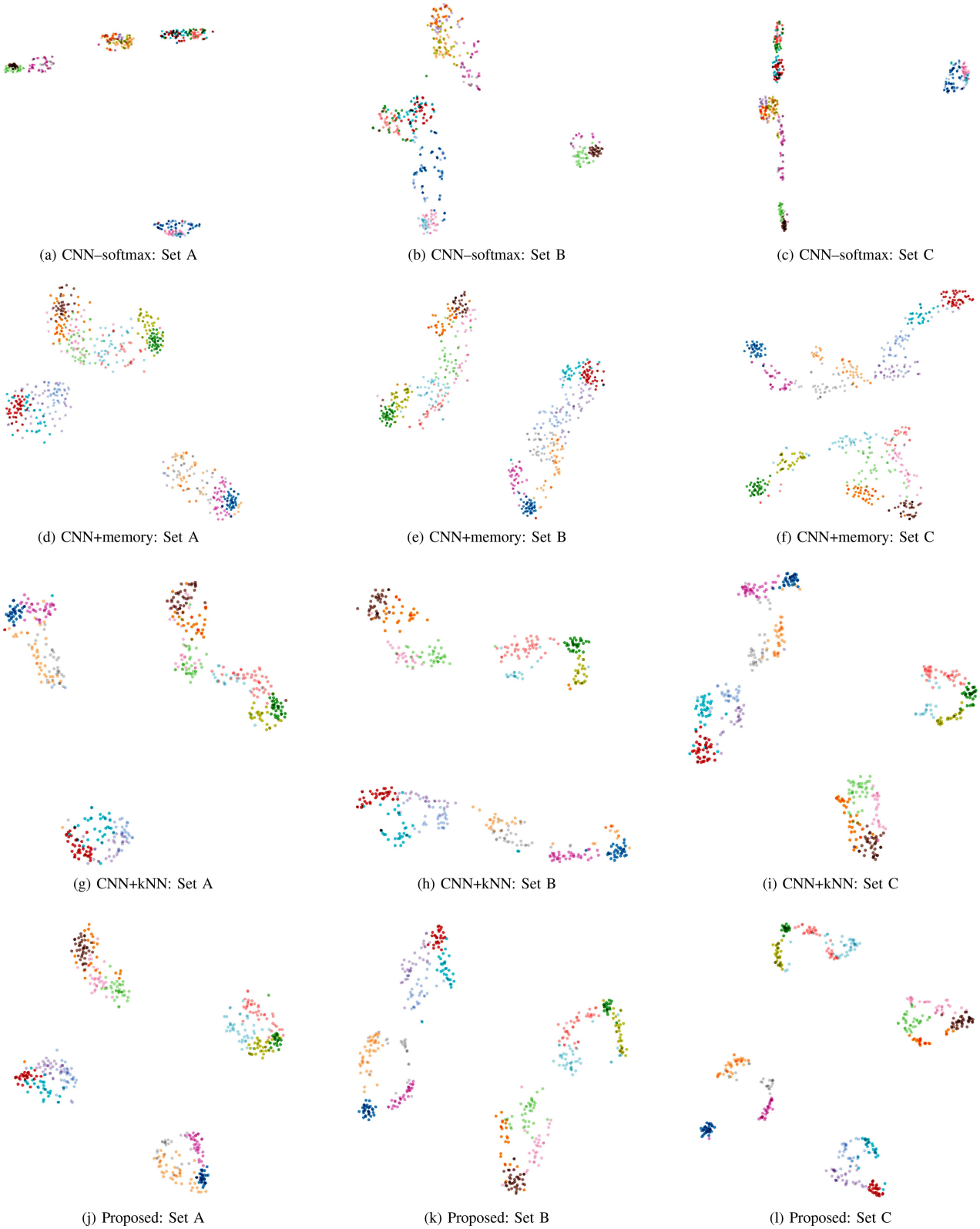
CNN–softmax and CNN+memory, the outputs of the final layer of the CNN were used as the embeddings, but only the cross-entropy loss was used in CNN–softmax, while a combination of the cross-entropy loss and memory loss was used in CNN+memory

It can be observed that, in the first and second rows of Figure 7, it is difficult to determine a clear centroid of each cluster in the distribution of the embeddings leaned using CNN–softmax and CNN+memory for all three sets. In contrast, as shown in the second and third rows in Figure 7, the embeddings learned using CNN+kNN and the proposed method were separated into clusters with clearer boundaries than those in the cases of CNN–softmax and CNN+memory . This is probably because CNN+kNN and the proposed method learn the embeddings directly using the triplet loss, while CNN–softmax and CNN+memory do not. On using the proposed method, the embeddings of each cluster were even more densely distributed than that in the case of CNN+kNN. This is probably because the additional memory loss for the proposed method forced the WBMs to become embedded around the representative values. Moreover, from left to right in Figure 7, the WBMs from the same class were embedded more closely, and the boundaries between the classes were clearer. This is probably because the performance of the CNN improved as the classes become more balanced.

Interestingly, as shown in the last two rows of Figure 7, for CNN+kNN and the proposed model, four clusters were configured in the embedding space, each of which comprised four classes. This may be because two dimensions are insufficient for representing various defect patterns; it seems that at least four dimensions are required, if we consider the presence of four distinct unit patterns (circle, ring, scratch, and zone). To interpret the embeddings more intuitively, we attempted to study the embedding results of the proposed method in greater detail. Figure 8 presents the details of the embedding results of the proposed method for Set C. It can be observed that there are four clusters that are divided according to the presence of ring and circle patterns, and furthermore, each cluster includes four different classes that are further divided according to the presence of scratch and zone patterns. This demonstrates that the proposed method effectively learned low-dimensional embeddings that are intuitively interpreted.

IV. CONCLUSION

In this article, we proposed a method for classifying WBMs, which comprise high-dimensional, highly imbalanced data, wherein a large number of WBMs had no patterns and a small number of WBMs had defect patterns. The proposed



(a) CNN–softmax: Set A (b) CNN–softmax: Set B (c) CNN–softmax: Set C

(d) CNN+memory: Set A (e) CNN+memory: Set B (f) CNN+memory: Set C

(g) CNN+kNN: Set A (h) CNN+kNN: Set B (i) CNN+kNN: Set C

(j) Proposed: Set A (k) Proposed: Set B (l) Proposed: Set C

Fig. 7. Visualization of the embeddings learned by each competing method using the t-SNE.

method uses CNN as an embedding function for learning low-dimensional features that are embedded in high-dimensional WBMs. In particular, the CNN is trained using the triplet loss, which is used to learn a well-separated low-dimensional space according to the defect patterns. The learned embeddings are then fed into a memory module. The memory module stores representatives of the embeddings in its memory slots while balancing the number of memory slots assigned for each class.

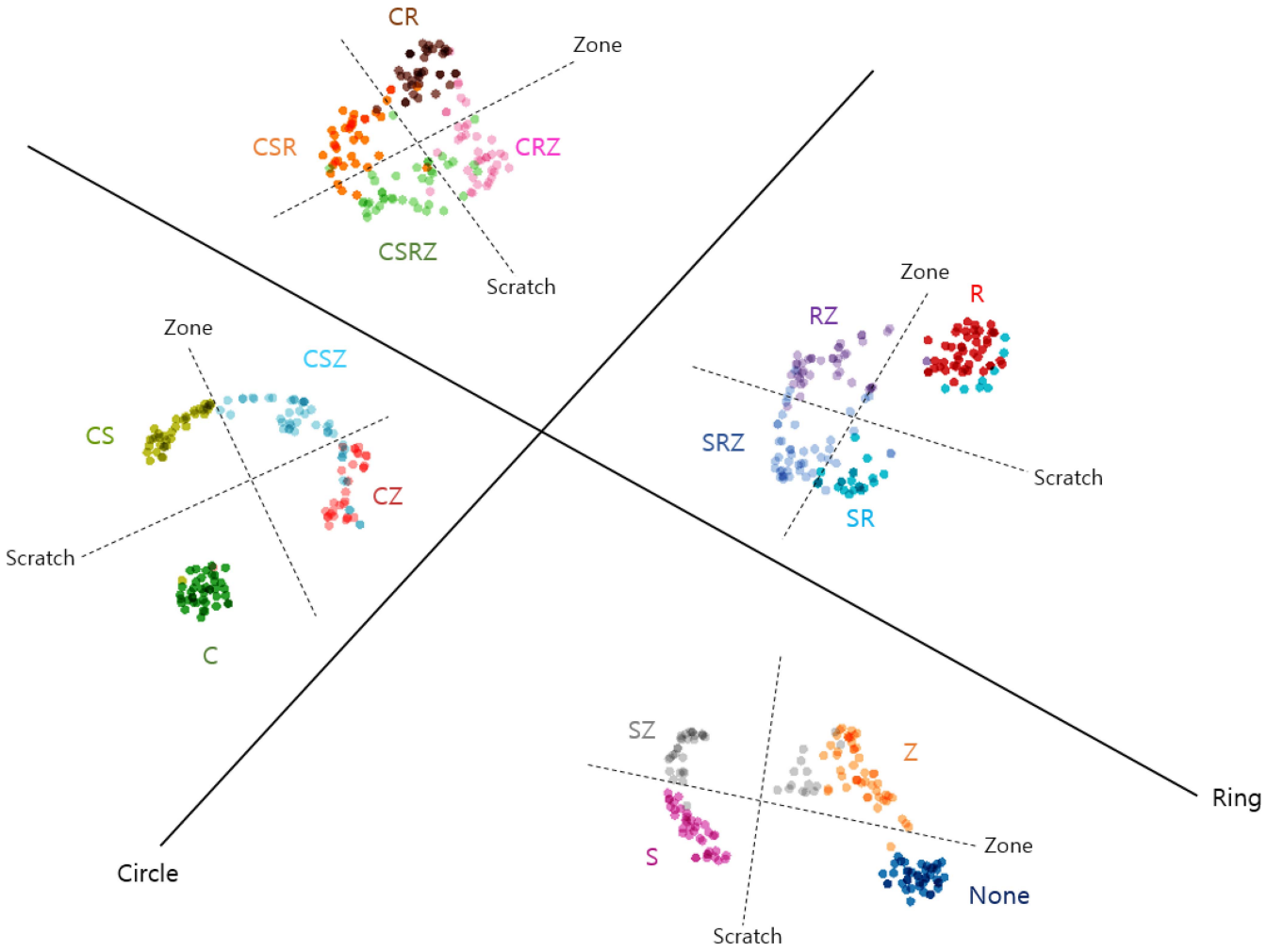


Fig. 8. t-SNE plot of the embeddings obtained using the proposed method for Set C.

The label predicted for a new WBM is then returned as the label of the nearest neighbor in the memory. We train the proposed model end-to-end, such that the embedding function and memory are learned interactively. As a result, the proposed model can learn embeddings that are suitable for classification. In the simulation study, the proposed method demonstrated a significantly better performance than the other competing methods for the classification of imbalanced data. The proposed method does not require any preprocessing steps. It should be noted that many existing methods for classifying WBM data require filtering methods for removing random defects. Moreover, in many examples of imbalanced-data classification, re-sampling was often applied to ensure that the data was balanced before the classification was performed. Without the requirement for these preprocesses, the proposed method is particularly attractive in practice. Although this study considered sixteen classes of defect patterns formed by combination of four unit patterns, more defect patterns may need to be considered in real applications. The increase in the number of classes may result in decreased performance. In this case, we may convert our multi-class classification problem into a multi-label classification problem and construct a memory-augmented CNN individually for each unit pattern class similar to [7]. By integrating the classification results of the individual models, we may improve the performance of the proposed method. Furthermore, in this manner, we could easily adapt the proposed model to the case comprising a different number of defect patterns without re-training it.

ACKNOWLEDGMENT

The authors thank the Referees and Editor for reviewing the manuscript and providing valuable comments.

REFERENCES

[1] J. Kim, Y. Lee, and H. Kim, "Detection and clustering of mixed-type defect patterns in wafer bin maps," *IISE Trans.*, vol. 50, no. 2, pp. 99–111, 2018.
[2] C.-H. Wang, W. Kuo, and H. Bensmail, "Detection and classification of defect patterns on semiconductor wafers," *IIE Trans.*, vol. 38, no. 12, pp. 1059–1068, 2006.
[3] R. C. Gonzalez and R. Woods, *Digital Image Processing*. Delhi, India: Pearson Educ., 2002.
[4] C.-H. Wang, "Separation of composite defect patterns on wafer bin map using support vector clustering," *Exp. Syst. Appl.*, vol. 36, no. 2, pp. 2554–2561, 2009.
[5] T. Yuan, W. Kuo, and S. J. Bae, "Detection of spatial defect patterns generated in semiconductor fabrication processes," *IEEE Trans. Semicond. Manuf.*, vol. 24, no. 3, pp. 392–403, Aug. 2011.

[6] M.-S. Yang and K.-L. Wu, "A similarity-based robust clustering method," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 4, pp. 434–448, Apr. 2004.
[7] K. Kyeong and H. Kim, "Classification of mixed-type defect patterns in wafer bin maps using convolutional neural networks," *IEEE Trans. Semicond. Manuf.*, vol. 31, no. 3, pp. 395–402, Aug. 2018.
[8] G. Tello, O. Y. Al-Jarrah, P. D. Yoo, Y. Al-Hammadi, S. Muhaidat, and U. Lee, "Deep-structured machine learning model for the recognition of mixed-defect patterns in semiconductor fabrication processes," *IEEE Trans. Semicond. Manuf.*, vol. 31, no. 2, pp. 315–322, May 2018.
[9] T. Nakazawa and D. V. Kulkarni, "Wafer map defect pattern classification and image retrieval using convolutional neural network," *IEEE Trans. Semicond. Manuf.*, vol. 31, no. 2, pp. 309–314, May 2018.
[10] 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 Trans. Semicond. Manuf.*, vol. 32, no. 3, pp. 310–319, Aug. 2019.
[11] J. Hwang and H. Kim, "Variational deep clustering of wafer map patterns," *IEEE Trans. Semicond. Manuf.*, early access, Jun. 23, 2020, doi: 10.1109/TSM.2020.3004483.
[12] S. Kim, H. Kim, and Y. Namkoong, "Ordinal classification of imbalanced data with application in emergency and disaster information services," *IEEE Intell. Syst.*, vol. 31, no. 5, pp. 50–56, Sep./Oct. 2016.
[13] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002.
[14] I. Tomek, "Two modifications of CNN," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, no. 11, pp. 769–772, Nov. 1976.
[15] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognit.*, vol. 40, no. 12, pp. 3358–3378, 2007.
[16] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.
[17] S. H. Khan, M. Hayat, M. Bennamoun, F. A. Sohel, and R. Togneri, "Cost-sensitive learning of deep feature representations from imbalanced data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3573–3587, Aug. 2017.
[18] M. Hu and J. Li, "Exploring bias in gan-based data augmentation for small samples," 2019. [Online]. Available: arXiv:1905.08495.
[19] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 815–823.
[20] C. Huang, Y. Li, C. Change Loy, and X. Tang, "Learning deep representation for imbalanced classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 5375–5384.
[21] C. Huang, Y. Li, C. L. Chen, and X. Tang, "Deep imbalanced learning for face recognition and attribute prediction," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, May 7, 2019, doi: 10.1109/TPAMI.2019.2914680.
[22] Ł. Kaiser, O. Nachum, A. Roy, and S. Bengio, "Learning to remember rare events," 2017. [Online]. Available: arXiv:1703.03129.
[23] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," 2017. [Online]. Available: arXiv:1703.07737.
[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: arXiv:1412.6980.
[25] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intell. Syst. Appl.*, vol. 13, no. 4, pp. 18–28, Jul./Aug. 1998.
[26] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.
[27] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.