A continual learning framework for adaptive defect classification and inspection

Wenbo Sun, Raed Al Kontar, Judy (Jionghua) Jin, and Tzyy-Shuh Chang

ABSTRACT
Machine-vision-based defect classification techniques have been widely adopted for automatic quality inspection in manufacturing processes. This article describes a general framework for classifying defects from high volume data batches with efficient inspection of unlabeled samples. The concept is to construct a detector to identify new defect types, send them to the inspection station for labeling, and dynamically update the classifier in an efficient manner that reduces both storage and computational needs imposed by data samples of previously observed batches. Both a simulation study on image classification and a case study on surface defect detection via 3D point clouds are performed to demonstrate the effectiveness of the proposed method.

KEYWORDS
3D point cloud data; continual learning; defect classification; out-of-distribution learning

1. Introduction
Recent development of advanced sensing and high computing technologies has enabled the wide adoption of machine vision to automatically inspect products’ dimensional quality for efficient process control and reducing the manual inspection cost. The process control procedure requires effective data analysis methods to provide reliable inspection results. In this article, we consider a high-volume manufacturing system that uses machine vision for automatic classification of product defects. Here classification implies both: identifying a defect and classifying its corresponding type. Going beyond the conventional automatic defect detection techniques, we would like the proposed system to identify defect types for root cause diagnosis of the target manufacturing system.

As a motivating example, we consider the scenario where batches of three-dimensional (3D) point cloud data are independently collected from a manufacturing process. The 3D point cloud data is obtained by measuring the 3D location of points on the product surface using a 3D scanner. The location measurements can then be used for fast classification of surface defects, and thus provide timely feedback for process control. Figure 1 (right) shows some exemplar surface defects on a wood product and the corresponding 3D point cloud measurements.

The 3D point cloud measurements have a set of defining characteristics that should be considered in the development of defect classification techniques. (i) The data size of 3D point cloud measurements is very high. In the aforementioned example, the 3D scanner performs a grid scan with step sizes of 0.3 mm (mm) in X-axis and 1 mm Z-axis on each surface, producing 20 points on each 1 mm³ cube on average. Consequently, a single wood part of length 2.85 meters incurs a data size of 82.7 megabytes. The points on the part are scanned and inspected sequentially in batches. As a result, defect classification techniques should be able to learn from both old and current batches while accounting for limited storage and computational capabilities needed to achieve continuous process monitoring. (ii) Product defects can be divided into different types/classes based on their shapes and sizes, which may occur at different surface locations. As a result, it is infeasible to pre-collect all types of defects as training samples. Furthermore, it is very common to have new defect types that evolve with time. Therefore, it is critical to continuously learn new defect types and augment the classification schemes with the ability to classify them. (iii) Manual data labeling for 3D point cloud measurements is extremely time-consuming and tedious as it requires identifying 3D locations of all individual scanned
points within the range of defects. Therefore, it is desirable to only request manual inspection and data labeling for new defect types while avoiding or reducing data labeling for previously well-trained old defects.

Machine vision algorithms have been widely investigated for industrial product surface defect detection. The traditional industrial product surface defect detection methods extract features on product texture (Jing et al. 2014), color (Ren et al. 2019) and shape (Wang, Fu, and Gao 2019) based on engineering knowledge, and then apply statistical inference or signal processing techniques to separate the features on normal and defective samples (Chen et al. 2021). These methods highly depend on the definition of the features and may not be generally applicable to the cases when there is no sufficient engineering knowledge to define the features. Some early works are related to adaptive learning of new process fault classes (Jin and Shi 2001) or multimode process baselines (Grasso, Colosimo, and Tsung 2017) from continuous monitoring of production data. These methods focused on the profile signals, thus their models and decision rules cannot be directly applied to 3D point cloud data. To overcome this limitation, deep-learning-based defect detection methods have been developed in recent years, which can be mainly divided into unsupervised methods and supervised methods. The unsupervised methods include the reconstruction-based methods and the embedding similarity-based methods. The former class (e.g., Bergmann et al. 2018) train a deep neural network to reconstruct normal samples and generate alarms when the reconstruction errors of incoming samples are above a threshold. The latter class (Cohen and Hoshen 2020; Defard et al. 2021) project samples into the embedded vector space and detect abnormal embedded vectors. The detection decision can be made by a engineering-knowledge-based threshold or a statistical control chart. However, the unsupervised methods cannot be directly applied to solve the target continual anomaly problem due to the following reasons. (i) the unsupervised methods do not make use of the defect class information and may lead to low detection accuracy; (ii) in the continuous process we would like to detect a new anomaly class directly, as such, running an unsupervised model with all the data we have from previous classes and the few data points of the new class will result in a substantially unbalanced learning scheme; (iii) it is not practically possible to update a new unsupervised learning model after each batch of observations.

Indeed, recent supervised classification techniques have been developed for defect detection (Jovančević...
et al. 2017; Hackel, Wegner, and Schindler 2017) or object recognition (Qi et al. 2017) based on 3D point cloud data. However, they require one to pre-label all the 3D point cloud data before training the classification model, which is infeasible for a high-volume manufacturing system and makes it difficult to account for new defect types that are either missed at initial labeling or evolve with time. In this article, we take an alternative route through continual learning of a defect classifier. Specifically, we present a statistical framework that can automatically detect new defect types, which are sent to the inspection station, and dynamically update the classifier in an efficient manner that reduces both storage and computational needs imposed by data samples of previously observed batches. Although machine vision is widely used for quality inspection in many manufacturing processes, it is often the case that machine vision may not always provide accurate inspection results, thus an inspection station is built to manually confirm whether the alarmed products are truly defective. In this article, we assume that the inspection station provides 100% correct labeling results for the alarmed samples but those inspections are time consuming or costly. Our idea and framework is illustrated in Figure 1. Figure 1 shows the flowchart of our proposed approach for inspecting a current batch having two old defect types (small dent & corner crack) that have been seen in the previous batches, and one new defect type (long crack) that occurs for the first time. Our approach is consisted of two modules. One is a new-defect detector that can automatically detect old defect types and separate them from new ones. In turn, efforts for data labeling of old defect types can be reduced. The other module is an updated defect classifier that gives the new inspection results. This updated classifier is intended to efficiently update the old model while relaxing the storage and computational needs for old defect types.

Regarding the two modules. The first module separates new defect types from old ones. An intuitive way is to monitor the probabilities or likelihoods of the current batch data falling within the previously trained defects. This will generate an alarm for new defect types when all the probabilities are low. However, state-of-the-art classifiers often over-estimate such probabilities, and thus over-confidently classify a sample from a new defect type into an old defect type (Nguyen, Yosinski, and Clune 2015). To resolve this issue, one may exploit recent out-of-distribution detection techniques. The basic idea is to construct a score that separates in-distribution samples (here samples from old defect types) from out-of-distribution samples (here samples from new defect types). Such score functions include but are not limited to the ODIN score (Liang, Li, and Srikant 2017), Mahalanomis-distance-based score (Lee et al. 2018; Hsu et al. 2020), energy-based score (Liu et al. 2020), and feature space singularity distance (Huang et al. 2020). These score functions are often parametrized by some tuning parameters, whose values are often trained by optimally separating the samples in the new classes from those in the old classes. However, a critical gap of applying the out-of-distribution technique in the proposed framework is that the training samples from the new defect types are not available, leading to a poor new-defect detection performance.

The second module is to classify defect types via a classification model. With the accumulated samples of newly labeled defect types, the model for defect classification should be continuously updated. Under constrained computational resources, only limited samples from the old defect types and the old model parameters can be stored and used to update the model. This is accomplished via continual learning techniques. The idea of continual learning dates back decades ago, to mixed-effects modeling where instead of re-learning a new model, empirical Bayes is used to updated random effects conditioned on the new observed data. One of the seminal works in this area is (Gebrael et al. 2005) who tests the developed model on bearing degradation signals. In contrast, our work focuses on classification, rather than regression tasks. Some works in continual learning of classification models include, Ruping (2001), who updates a support vector machine in a batch learning mode, and Ozawa, Pang, and Kasabov (2008), who extends incremental principal component analysis to classify chunks of training samples. To deal with high volume and high dimensional data set, deep-neural-network-based continual learning techniques have been developed, which build deep neural networks that can learn new classes while retaining the classification accuracy of old classes. For example, Xiao et al. (2014) and Roy, Panda, and Roy (2020) propose two hierarchical architectures to allow deep neural networks grow sequentially when new classes of samples arrive. Kochurov et al. (2018), Kirkpatrick et al. (2017) and Li and Hoiem (2018) impose regularization terms on the loss function to maintain classification accuracy for old classes. Rebuffi et al. (2001) stores representative samples from the old classes and uses them to update the model. Rusu et al. (2016) and Mallya and Lazebnik (2018) introduce additional model
components to draw the connection of model parameters between the old and new classes. Wu et al. (2019) and He et al. (2020) propose methods to update the model in an online manner when the old and new classes arrive simultaneously, and investigate the data imbalance issue between the old and new classes. However, these continual learning techniques rely on fully labeled samples in both of the old and new classes, which is not available in the target manufacturing application. Moreover, without additional treatments to the existing continual learning methods, the learnt classifier tends to over-confidently classify samples from incoming new classes into the classes in the training data set.

To overcome the shortcomings of the existing methods, this article presents an adaptive defect classification approach that integrates both out-of-distribution learning and continual learning together. The key contribution is in learning a "Look-ahead" defect classifier that aims to separate the known defects and potential unknown new defects in future data batches. Indeed, the engineering setting of having an inspection station to confirm machine-vision results is the key motivation behind this new framework. To the best of our knowledge, the proposed approach is the first anomaly detection work under this new engineering setting, which is highly demanded in manufacturing industry for automatic quality inspection using a machine vision system.

In particular, the contributions of the article are three-folds. (a) Going beyond the existing continual learning methods, we propose a data-driven strategy for essential data labeling, that is, required only for new defect types; (b) Different from the existing out-of-distribution learning techniques, we develop a systematic way to train a classifier that simultaneously updates and learns new defect types from inspections and avoids over-confident incorrect classification of samples from unseen classes. Our approach does not require all samples from previous batches to be stored or re-trained. (c) provide a guideline for selecting auxiliary out-of-distribution samples in different engineering applications. The proposed method is generic and compatible with any type of classifier and out-of-distribution score function. All these demonstrate important new contributions and the engineering significance of the proposed method.

The rest of article is organized as follows. In Section 2, we will describe the notations, data structure, and objective. In Section 3, the proposed method is elaborated for adaptive defect classification. Section 4 and Section 5 demonstrate the effectiveness of the method by using a simulation study in image classification and a case study in surface defect classification for 3D point clouds, respectively. The article then ends with a conclusion in Section 6.

2 Notations, data structure and objective

In this article, we consider a high-volume discrete manufacturing process that has a machine vision system to automatically detect potential defects and an inspection station is followed to confirm whether those alarmed parts are truly defective. It should be clarified that those alarmed samples by the machine vision system will be correctly labeled in the inspection station, while those un-alarmed samples will not go through the inspection station, thus no labeling. During production, the products are discretely produced and the inspection results come in batches. In each batch, the test results are assumed as independent samples. We call this data setting as discretized data batches, which is commonly seen in many manufacturing applications. In this situation, the inspection results are considered as independent samples.

The space of measurement data is denoted by $\mathcal{X}$. {Suppose independent measurement samples arrive in multiple batches $\{0, \ldots, T\}$ from a discrete manufacturing process. Let $X^{(t)} = \{x^{(t)}_1, \ldots, x^{(t)}_{n^{(t)}}\}$ denote the set of $n^{(t)}$ measurement data at batch $t \in \{0, \ldots, T\}$, where $x^{(t)}_i \in \mathcal{X}$ is the $i$-th sample in $X^{(t)}$. In general, $x^{(t)}_i$ can be any type of measurement data that are commonly seen in manufacturing processes, such as profile signals, images, time-series, etc. Given that the measurement data can take any form, such as vectors, matrices or tensors, we denote vectors, matrices or tensors by using lowercase boldface letters, and denote a set of vectors, matrices or tensors by using uppercase boldface letters throughout the article. Let $y^{(t)}_i$ denote the class label corresponding to $x^{(t)}_i$. In particular, $y^{(t)}_i = 0$ when sample $i$ is non-defective, and $y^{(t)}_i$ takes a positive integer value corresponding to the defect class when the sample $i$ is defective. Our major objective is to predict the quality label $y^{(t)}_i$ via a classifier $f : \mathcal{X} \rightarrow \mathbb{R}$ for defect classification.

Let $Y^{(t)}$ denote the set of all the class labels up to batch $t$. When the defect types come in a class-incremental form, that is, $Y^{(t)} \cap Y^{(t-1)} = \emptyset$, the classifier $f$ trained at the previous batches cannot predict the newly appeared defect types from $Y^{(t)} \backslash Y^{(t-1)}$. To classify these new defects, it requires to update the classifier $f$ with labeled samples from a subset of $X^{(t)}$. Given that inspecting all the $X^{(t)}$’s class labels are expensive, we would like to focus on the samples
whose labels belong to the set $\mathcal{Y}(t) \subset \mathcal{Y}^{(t-1)}$. To do this, a new-defect detector $g : \mathcal{X} \to \{0, 1\}$ is developed to screen the newly appeared defects. Specifically, we would like $g(x^{(t)}_i)$ to return 1 if $y^{(t)}_i \in \mathcal{Y}(t) \setminus \mathcal{Y}^{(t-1)}$, and to return 0, otherwise.

Now we add the superscript $(t-1)$ to the classifier $f$ and the new-defect detector $g$ to denote the trained functions prior to batch $t$. The proposed statistical framework for adaptive defect classification works as follows. At batch $t = 0$, $f^{(0)}$ and $g^{(0)}$ are initially trained using historical data set. At batch $t > 0$, $g^{(t-1)}$ is firstly applied to $X^{(t)}$ to detect potential new defect types, that is, to select $\tilde{X}^{(t)} = \{x : g^{(t-1)}(x) = 1, x \in X^{(t)}\}$. The samples of $\tilde{X}^{(t)}$ are then sent to the inspection station for labeling, and denoted by $\tilde{Y}^{(t)}$. These labeled samples are further used to update the classifier $f^{(t)}$ and the new-defect detector $g^{(t)}$. The procedure is repeatedly applied until the last batch $T$ is processed. A flowchart of the above procedure is provided in Figure 2. The top panel (above the brace) illustrates the overall methodology flowchart among all the data batches. The bottom panel (below the brace) depicts the detailed information flow within each data batch. The shaded dashed boxes highlight the key modules. The key functions of the proposed algorithm is highlighted in the square brackets. After model update, the classifier $f^{(t)}$ can be used to classify all the defect types appeared in data batches $1, \ldots, t$.

The proposed statistical framework is well established once the updating strategy of $f^{(t)}$ and $g^{(t)}$ is determined. In the next section, we will elaborate the technical details when updating $f^{(t)}$ and $g^{(t)}$.

3. Method

In this section, we decompose the proposed algorithm into three sub-functions and explain how they are used for adaptive defect classification. Section 3.1 provides an overall structure of the proposed algorithms. Section 3.2 to 3.4 elaborate the algorithmic details in three functions. Section 3.5 discusses the practical considerations including the choices of models, auxiliary out-of-distribution samples and the processes for tuning parameters.

3.1. Adaptive defect classification

To begin with, we specify the definitions of the classifier and new-defect detector as follows. Let $|\mathcal{Y}^{(t)}|$ denote the total number of labeled classes until batch $t$. The classification results depend on the predicted probabilities of a sample falling in each class $j$, which is denoted by $q_j(\cdot; \theta^{(t)}) : \mathcal{X} \to \mathbb{R}$, with $\theta^{(t)}$ representing the model parameter of the classifier. Let $\mathbf{q}$ be the vector that collects all the $q_j$’s, for $j = \{1, \ldots, |\mathcal{Y}^{(t)}|\}$. Note that $\mathbf{q}$ can be the predicted probability function
in any classifier. The classifier \(f^{(t)}\) returns the class label as the maximal element in the vector \(q\), which is

\[
f^{(t)}(x) = \arg \max_j q_j(x|\theta^{(t)}).
\]

The new-defect detector is established based on a score function that evaluates the probability whether a sample comes from a new defect or not. Let \(s(\cdot|\phi^{(t)})\) denote the score function, where \(\phi^{(t)}\) is the model parameter of the score function. \(s\) maps the predicted probability function \(q\) to a real-valued score such that the scores of new defects are separable from those of old defects. The parametric form of \(s\) can be flexibly selected from any out-of-distribution score functions. For example, the Mahalanobis-distance-based score measures the Mahalanobis distance between a sample \(x\) and the closest class-conditional Gaussian distribution. In particular, let \(\bar{\mu}_j\) and \(\Sigma\) denote the empirical mean and covariance matrix of \(q(x)\) for samples of defect type \(j\), respectively, that is,

\[
\begin{align*}
\bar{\mu}_j &= \frac{1}{N_j} \sum_{j:y=j} q(x_i), \\
\Sigma &= \frac{1}{N_j} \sum_{j:y=j} \sum_{i:y=j} (q(x_i) - \bar{\mu}_j)(q(x_i) - \bar{\mu}_j)^T,
\end{align*}
\]

where \(N_j\) is the number of training samples with defect type \(j\). The Mahalanobis-distance-based score is then formally expressed as the negative distance from \(q(x|\theta^{(t)})\) to the empirical mean of the softmax scores \(\bar{\mu}_j\) with considering the empirical covariance \(\Sigma\).

\[
s(q(x|\theta^{(t)})|\phi^{(t)}) = \max_j s(q(x|\theta^{(t)})|\mu_j) = \max_j \left( q(x|\theta^{(t)}) - \bar{\mu}_j \right)^T \Sigma^{-1} \left( q(x|\theta^{(t)}) - \bar{\mu}_j \right). \tag{3}
\]

In this way, a low \(s(q(x|\theta^{(t)})|\phi^{(t)})\) implies that \(q(x|\theta^{(t)})\) is far from the center of the softmax scores of old samples. Therefore, we set an upper threshold \(\tau^{(t)}\) for \(s(q(x|\theta^{(t)})|\phi^{(t)})\), and report detection results when \(s(q(x|\theta^{(t)})|\phi^{(t)})\) is above the threshold. That is, to define the new-defect detector \(g^{(t)}\) as:

\[
g^{(t)}(x) = I\{s(q(x|\theta^{(t)})|\phi^{(t)}) \leq \tau^{(t)}\}, \tag{4}
\]

where \(I\) is the indicator function. Eq. [4] shows that the new-defect detector \(g^{(t)}\) depends on the classifier’s prediction \(q\), implying that the new-defect detector should be updated with the classifier among different data batches.

As is shown in the following Algorithm 1, the proposed method can be decomposed into three sub-functions to incrementally update information from the measurement data \(X^{(t)}\) at batch \(t\). First, the previously trained new-defect detector \(g^{(t-1)}\) is used to detect the measurement data of new defect types in \(X^{(t)}\). The inspection station labels these samples, combines the labeled data with a given number of data of old defect types, and returns the combined data set as \(D^{(t)}\). Second, the combined data set \(D^{(t)}\) is fed to a continual learner to update the parameters of the classifier \(\theta^{(t)}\), the parameter of the new-defect detector \(\phi^{(t)}\), and the threshold of the score function \(\tau^{(t)}\). Lastly, the classifier and new-defect detector are updated according to Eqs. [1] and [4], respectively. At the end of each batch \(t\), \(f^{(t)}\) learns the new defect types in \(D^{(t)}\), and \(g^{(t)}\) will identify the new defect types beyond \(\gamma^{(t)}\). In the following subsections, we will elaborate the technical details in three sub-functions “Sample Inspection”, “Continual Training”, and “Model Update”.

**Algorithm 1**: Continual anomaly detection

**input**: a set of measurement data \(\{X^{(t)}; t = 0, ..., T\}\), an initial set of class labels \(\gamma^{(0)}\), a pre-trained classifier \(f^{(0)}\), an pre-trained new-defect detector \(g^{(0)}\).

**for** \(t \in \{1, ..., T\}\)

\[D^{(t)}, \gamma^{(t)} \leftarrow \text{Sample Inspection}(g^{(t-1)}, X^{(t)}, \gamma^{(t-1)}, \{D^{(k)}; k = 0, ..., t - 1\})\]

\[\theta^{(t)}, \phi^{(t)}, \tau^{(t)} \leftarrow \text{Continual Training}\left(\{D^{(k)}; k = 0, ..., t\}, \theta^{(t-1)}, \phi^{(t-1)}, \tau^{(t-1)}\right)\]

\[f^{(t)}, g^{(t)} \leftarrow \text{Model Update}(\theta^{(t)}, \phi^{(t)}, \tau^{(t)})\]

**end**

**output**: the updated classifier \(f^{(T)}\) and new-defect detector \(g^{(T)}\)

### 3.2. Sample inspection

At each batch \(t\), the first step is let all the unlabeled measurement data \(X^{(t)}\) go through the new-defect detector to identify the samples of potential new defects, and then send them to the inspection station for labeling. Note that the unlabeled measurement data at each batch could be a mixture of old and new defects. Following the definition of the new-defect detector, we store the samples in \(\tilde{X}^{(t)} = \{x : g^{(t-1)}(x) = 1\}\) and send them to the inspection station to get the corresponding labels \(\tilde{Y}^{(t)}\). During inspection, \(\gamma^{(t-1)}\) is continuously updated to include all the new defect types that have not been seen in the previous batches, resulting in the updated set \(\gamma^{(t)}\). The data set \(D^{(t)}\) then combines the newly labeled samples with the old samples in the previous batches, which will be further used to update the classifier and new-defect detector in the next step. Here we recommend to keep a given number of samples in the previous data batches because it can improve the classifier and new-defect detector’s performance after updating
the models. The determination of the number of samples in the previous data batches will be discussed in Section 3.5. Let $D_{old}^{(t)}$ denote the data set of old samples we keep in the previous data batches. The proposed algorithm for sample inspection is displayed in Algorithm 2.

It is worth noting that the performance of $g^{(t-1)}$ affects the efficiency of defect labeling and model update. In particular, for a sample $(x, y)$ with $g^{(t-1)}(x) = 1$ and $y \in \mathcal{Y}^{(t-1)}$ (i.e., false alarm), the inspection station labels an old defect type. Such a labeled sample $(x, y)$ has a small contribution to the improvement of the classifier and new-defect detector’s performance, making the inspection less efficient. For a sample $(x, y)$ satisfying $g^{(t-1)}(x) = 0$ and $y \notin \mathcal{Y}^{(t-1)}$ (i.e., misdiagnosis), the sample of new defect type $(x, y)$ remains unlabeled, and thus is not used for model update. An overly-high misdiagnosis rate leads to a small sample size of new classes, and may hence result in a poor classification performance on new defects. In practice, unlike traditional statistical process control techniques, the misdiagnosis rate is not necessarily near perfect, and should be determined based on the requirement on the sample size of each new class. For instance, suppose 10,000 samples under a new defect type arrive in a batch, a new-defect detector with 50% misdiagnosis rate can still send 5,000 samples for labeling, which is typically essential to train a classifier that learns the new defects.

**Algorithm 2**: Sample Inspection

**input**: the previous detector $g^{(t-1)}$, the new data batch $X^{(t)}$, the previous set of labels $\mathcal{Y}^{(t-1)}$, the previous batches $\{D^{(k)}, k = 0, ..., t - 1\}$

**initialize**

$\tilde{X}^{(t)} = \{\}$

$\tilde{Y}^{(t)} = \{\}$

$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)}$

for $x \in X^{(t)}$ do

if $g^{(t-1)}(x) == 1$ then

get label $y$ of sample $x$ from the inspection station

$\tilde{X}^{(t)} \leftarrow \tilde{X}^{(t)} \cup x$

$\tilde{Y}^{(t)} \leftarrow \tilde{Y}^{(t)} \cup y$

if $y \notin \mathcal{Y}^{(t)}$ then

$\mathcal{Y}^{(t)} \leftarrow \mathcal{Y}^{(t)} \cup y$

end

end

$D^{(t)} \leftarrow (\tilde{X}^{(t)}, \tilde{Y}^{(t)}) \cup D_{old}^{(t)}$

**output**: Dataset with labeled new defects and updated set of known labels: $D^{(t)}, \mathcal{Y}^{(t)}$

### 3.3. Continual training

In this sub-function, the classifier and new-defect detector are simultaneously updated based on the labeled data set $D^{(t)}$ and an auxiliary out-of-distribution data set $\tilde{D}^{(t)}$. The loss function for model training is proposed as follows:

$$L(\theta, \phi, \tau^{(t-1)}) = L(\theta, \phi, \tau^{(t-1)}) + \lambda_{new} \sum_{(x, y) \in \tilde{D}^{(t)}} \max\{0, \tau^{(t-1)} - s(q(x, y)|\theta)\}$$

where $L_{cont}(\theta)$ represents a loss function in continual classification to be discussed shortly. The second and third terms in Eq. [5], which encourages the score function $s$ to be greater than the threshold $\tau^{(t-1)}$ for any known defects from $D^{(t)}$, and to be smaller than the threshold $\tau^{(t-1)}$ for any auxiliary out-of-distribution samples from $\tilde{D}^{(t)}$. In this way, the trained classifier not only learns the new classes in $D^{(t)}$, but also separates the known defects and potential new defects in future data batches. This indeed is reminiscent of the slack term in support vector machine (Suykens and Vandewalle 1999) which aims to achieve an optimal separation across classes.

Now regarding $L_{cont}(\theta)$, we propose to penalize the cross-entropy classification loss $CE$ with a penalty term on the normalized Gaussian distance between the updated and original model parameters. This is given as

$$L_{cont}(\theta) = L_{class}(\theta) + \lambda_{prior} L_{prior}(\theta),$$

$$L_{class}(\theta) = \sum_{(x, y) \in D^{(t)}} CE(q(x, y)|\theta),$$

$$L_{prior}(\theta) = \frac{p}{\sum_{k=1}^{p} F_k (\theta_k - \theta_k^{(t-1)})^2},$$

where $\theta_k$ represents the $k$-th dimension of $\theta$, $p$ represents the dimension of the parameter space, and $F_k$ is the $k$-th diagonal element of a Fisher information matrix. Here $L_{prior}$ constraints $\theta$ to be close to the Laplace approximation of the posterior Gaussian distribution with mean given by the previous model parameter $\theta^{(t-1)}$ and a diagonal precision specified as the diagonal elements of the Fisher information matrix $F_k$’s. Given that the cross-entropy classification loss is proportional to the log-likelihood function, the diagonal elements of the Fisher information matrix can be computed as
where $|D^{(t)}|$ is the number of samples in the data set $D^{(t)}$.

The fundamental idea of the penalty term $L_{\text{prior}}$ is to penalize large deviations from the previous classifier parameterized by $\theta^{(t-1)}$, when training using $D^{(t)}$, so that the classifier with the updated model parameters can still perform well on samples from the old tasks. Specifically, the Fisher information $F$ contains information on the local curvature around the model parameter $\theta$. Thus, discriminative penalties reflecting the local curvature are applied to the elements of $\theta$ in such a way that a greater penalty is given to an element with a more rapid gradient. It implies that the penalty term restricts important parameters to stay around $\theta$, whereas allowing relatively non-informative parameters to move to learn new data. As a result, by minimizing $L_{\text{prior}}$, the model learns new data without spoiling what it has learnt from the previous tasks. Note that this does not create a significant computation load compared to the optimization without the penalty term as $F$ is readily computed from first-order derivatives. Adding this penalty term in turn allows updating the classifier while (i) maintaining previous accuracy of old defect types and (ii) reducing storage, computation and inspection needs as $D^{(t)}$ only contains data from new defect types and a limited number of samples of the old defect types.

Since we place the same weight $\lambda_{\text{out}}$ on the second and third terms in Eq. [5], solving the optimization problem cannot control the tradeoff between the false alarm and misdetection rates as was discussed in Section 3.2. Therefore, after updating $\theta^{(t)}$ and $\phi^{(t)}$, the threshold parameter $\tau^{(t)}$ should be tuned to achieve the desired new-defect detection performance. For example, we can specify the $\tau^{(t)}$ as a cutoff that guarantees an over $\eta\%$ true positive rate (e.g., $\eta$ can be set as 80), that is, to compute

$$\tau^{(t)} = \sup_{\tau} \left\{ \tau : \Pr \left[ s(q(x|\theta^{(t)})|\phi^{(t)}) \leq \tau | x \in D^{(t)} \right] \geq \eta \% \} \right.$$

[8]

To this end, the parameters $\theta^{(t)}$, $\phi^{(t)}$ and $\tau^{(t)}$ are updated based on the inspected samples, which can be fed into the definitions of $f^{(t)}$ and $g^{(t)}$ to update the classifier and new-defect detector for future data batches. The algorithm is summarized in Algorithm 3.

### Algorithm 3: Continual Training

**Input**: labeled samples $\{D^{(t)}, t = 0, \ldots, T\}$, previous parameters $\theta^{(t-1)}$, $\phi^{(t-1)}$, $\tau^{(t-1)}$.

**Initialize**
- Out-of-distribution dataset $\tilde{D}^{(t)}$.

**Training in one epoch**:
- for $(X_{\text{in}}, Y_{\text{in}}) \in D^{(t)}$ do
  - Randomly select $(X_{\text{out}}, Y_{\text{out}}) \in \tilde{D}^{(t)}$.
  - Forward Propagation:
    - $X \leftarrow [X_{\text{in}}, X_{\text{out}}]$.
    - $Y \leftarrow [Y_{\text{in}}, Y_{\text{out}}]$.
    - Evaluate $q(X|\theta^{(t)})$, $s(q(X|\theta^{(t)})|\phi^{(t)})$.
  - The value of Loss:
    - Evaluate $L(\theta^{(t)}, \phi^{(t)}, \tau^{(t-1)})$ in Eq. [5].
  - Backward Propagation:
    - Evaluate $\frac{\partial L(\theta^{(t)}, \phi^{(t)}, \tau^{(t-1)})}{\partial \theta^{(t)}, \phi^{(t)}, \tau^{(t-1)}}$.
- Update $\tau$ in Eq. [8].

**Output**: updated model parameters $\theta^{(t)}$, $\phi^{(t)}$, $\tau^{(t)}$.

### 3.4. Model update

Lastly, the classifier $f^{(t)}$ and new-defect detector $g^{(t)}$ are updated based on the trained model parameters $\theta^{(t)}$, $\phi^{(t)}$, $\tau^{(t)}$ according to the definitions in Eqs. [1] and [4], which is shown in Algorithm 4. $g^{(t)}$ will be used for identifying new defects and defect classification for future data batches, respectively.

### Algorithm 4: Model update

**Input**: updated parameters $\theta^{(t)}$, $\phi^{(t)}$, $\tau^{(t)}$.

- $f^{(t)}(x) \leftarrow \arg \max q(x|\theta^{(t)})$.
- $g^{(t)}(x) \leftarrow \mathbb{I}\{s(q(x|\theta^{(t)})|\phi^{(t)}) \leq \tau^{(t)}\}$.

**Output**: updated classifier and detector $f^{(t)}$ and $g^{(t)}$.

### 3.5. Practical considerations

In this subsection, we discuss some practical details when implementing the proposed approach to real engineering applications. The first practical issue is the choice of the auxiliary out-of-distribution data set $\tilde{D}^{(t)}$. Adversarial out-of-distribution samples are commonly used in the out-of-distribution learning literature. For example, the adversarial samples can be generated via projected gradient descent (PGD, Madry et al. 2017) or fast gradient sign method (FGSM, Goodfellow, Shlens, and Szegedy 2014). However, these adversarial training techniques are designed to detect images under random disturbances rather than new types of defects with certain patterns. Under our
The back-propagation algorithm, the number of epochs, and the computational time and complexity during model updating are dominated by the multiplication of the number of matrix operations. Hence, model updating can be done when the production is not running, thus not interfering with production.

Second, it is recommended to utilize the limited storage space and improve the model performance via efficiently handling the previous and current data batches. In this setting, the principle is to require the samples in the previous data set first, and train the new-defect detector to separate the other old types of defects from the left-out type of defect. Examples of selecting such an auxiliary out-of-distribution data set will be given in the following two sections.

Algorithm 3 with a combination of the sampled old defects and the newly labeled defects. On the other hand, the classifier may have a poor prediction performance for the new defect types when the samples in the new defect types are not enough. To resolve this issue, we recommend to keep the classifier functioning for a continuous system performance guarantee and only update the classifier when the sample size of the new defect types exceeds a pre-specified threshold. When a particular defect type lacks training samples, the corresponding samples should be merged into the out-of-distribution data set $\tilde{D}^{(t)}$ and re-train the new-defect detector. In this way, the detector can identify more samples in the target new defect type, send them to the inspection for labeling until a desired sample size of the new defect is achieved. Moreover, updating the algorithm can be done when the production is not running, thus not interfering with production.

In addition, the computational time and complexity of the proposed approach are dominated by the multiplication of the number of matrix operations during the back-propagation algorithm, the number of epochs, and the number of samples in the training data set $D^{(t)}$. The space complexity of the proposed approach is dominated by the multiplication of the number of samples in the training data set $D^{(t)}$ and the size of each sample. Comparing to the existing multitask learning approaches, the proposed method drops the data set $\{D^{(k)},k=1,\ldots,t\}$ and only updates the model parameter based on the most recent data set $D^{(t)}$, which significantly improves the learning efficiency.

Since both of the computation time and space complexities highly depend on $|D^{(t)}|$, we recommend to firstly specify an upper limit of $|D^{(t)}|$ to satisfy the constraints on computational resources, then balance the tradeoff between the sample sizes of the old and new defects to achieve the desired classification accuracy.

Lastly, the hyper-parameters $\lambda_{\text{load}}$ and $\lambda_{\text{prior}}$ in the loss function should be tuned before implementing the proposed algorithm to all the data batches. To reduce computational time, we suggest to tune the hyper-parameters based on the first several batches of data that contains new defect types. Specifically, the hyper-parameters should be selected to achieve the lowest classification error in the test data set, while identifying over a pre-specified percentage (e.g., 50%) of the new types of defects for new data batches. Detailed discussion on the hyper-parameter tuning procedure can be found in Appendix B.

4 Simulation study

4.1. Performance of the trained detector and classifier

In this section, we demonstrate the effectiveness of the proposed approach by a simulation study on the Modified National Institute of Standards and Technology (MNIST) data set (Deng 2012). The objective is to train a classifier that learns the digit labels from images of handwritten digits. To fit the MNIST data set into the adaptive defect classification situation, we assume that a classifier is pre-trained based on the images with digits {0, 1}, and then updated based on the images with other digits in three data batches sequentially. Batches 1, 2 and 3 only include digits {2, 3}, {4, 5}, and {6, 7}, respectively. In each class, we extract 4-dimensional features from the flattened $784 \times 1$ images via a trained deep auto-encoder, and then divide 80% of the samples into the training data set, and the rest into the test data set using “Split” (Vakayil et al. 2021; Joseph and Vakayil 2022). We leave the images with digits {8, 9} as a potential choice of the out-of-distribution data set. As shown in Figure 3, we presume that the images are not labeled at each batch, and apply the proposed approach to detect images from new classes and label these images in the inspection station. Afterwards, the labeled images are fed back to the continual learner for model updating. In Figure 3, the top panel shows exemplar images with different class labels in each batch. The bottom panel shows the ratio of detected samples (the values before “/”) among all the samples in the new classes (the values after “/”).
Given the limited storage space, we keep 3,000 images in each old class when processing the current data batch. We select the benchmark classifier Residual Network (ResNet, He et al. 2016) with two blocks, each of which is consisted of two convolution layers with kernel size 3. We tune the hyper-parameters following the procedure in Section 3.5. The resultant $\lambda_{\text{prior}} = \lambda_{\text{ood}} = 1$. We compare the new class detector’s performance based on the ODIN and Mahalanobis-distance-based scores. We set $\eta = 80$ in Eq. [8] to allow for 20% false positive rate in detecting new classes. We train the baseline model with digits \{0, 1\}, train the out-of-distribution detection with digits \{8, 9\}, and test the detection performance in the three sets of new digits \{2, 3\}, \{4, 5\} and \{6, 7\}. The detector based on the ODIN score detects 84.25%, 86.85% and 80.40% samples in the three sets of new digits, respectively. The detector based on the Mahalanobis-distance-based score detects 88.95%, 88.92%, and 88.14%, respectively. We select the ODIN score in the following analysis because Mahalanobis-distance-based score requires more computational time for a complex deep neural network although the Mahalanobis-distance-based score show a slight better detection performance than the ODIN score.

The proposed approach is applied to the MNIST data set under the above setting. The classifier is trained 100 epochs for each batch of data. The performance is evaluated based on the false negative rates of the detector $g^{(t)}$ and the prediction accuracy of the classifier $f^{(t)}$ in the test data set. The true negative rates in all the three batches are displayed at the lower panel of Figure 3, which implies that the trained detector can detect more than 48.8% of the samples in the new classes, guaranteeing that more than 2,500 samples of each new class are used for model update. Figure 4 shows the prediction accuracy of different classes in different batches. The solid line depicts the test accuracy of class \{0, 1\} at different epochs during baseline training and the three batches. The dashed line represents the test accuracy of class \{2, 3\} at different epochs in the three batches. The dot-dashed line shows the test accuracy of class \{4, 5\} at different epochs in batches 2 and 3. The dotted line represents the test accuracy of class \{6, 7\} at different epochs in batch 3. Each task is trained for 100 epochs. The test accuracy of the old tasks drops to relatively lower values at the initial 20 to 30 epochs after each 100 epochs. This is because the corresponding model parameter deviates from that in the previous task while still not reaches the minimal point of the proposed loss function. However, after training for enough epochs, the test accuracy of both the old and new tasks increase to above 0.97, thus demonstrating the effectiveness of the proposed method. After 100 training epochs, the prediction accuracy is above 97% for all the classes, demonstrating that the proposed continual learner identifies the new class labels without forgetting the old ones. The prediction accuracy results
also validate our arguments in Section 3.2 that the new-defect detector is not required to have a near-perfect performance in catching all the samples of new classes. Instead, we can train a good classifier with enough samples from new classes, classify the remaining unlabeled samples in the old data batches, and then add the classified samples for the future data batches.

4.2. Choice of auxiliary out-of-distribution data set

There are multiple choices of auxiliary out-of-distribution samples. For illustration purpose, we compare the detection performance of two detectors (g), one is trained using the adversarial samples generated from projected gradient descent (PGD, Madry et al. 2017) as the out-of-distribution data set ($\mathcal{D}^{(i)}$), the other is trained using the images with labels in $\{8, 9\}$ as the out-of-distribution data set. The comparison results are illustrated in Figure 5 with a baseline model trained with samples in class $\{0, 1\}$. The three rows depict the histograms of the score function $s$ for the in-distribution training data, the out-of-distribution training data, and the out-of-distribution test data, respectively. On the left panel of Figure 5, the generated adversarial samples do not resemble the actual out-of-distribution samples, which results in a detector that cannot identify out-of-distribution samples with labels $\{2, 3\}$. On the other hand, as the samples in classes $\{8, 9\}$ share the similar out-of-distribution pattern as the samples in classes $\{2, 3\}$ (shown on the right panel of Figure 5), a detector that separates the auxiliary samples in $\{8, 9\}$ from $\{0, 1\}$ can also separate the actual out-of-distribution samples in $\{2, 3\}$ from $\{0, 1\}$.

5. Case study

In this section, we apply the proposed method to a 3D point cloud data set generated by scanning a long-cuboid-shaped wood part with a 3D laser scanner. There exists five major types of defects on the four side surfaces of the wood part: small dent, corner crack, big dent, long crack, and non-smooth texture. We manually highlight the defective areas after importing the 3D point clouds into the software “MeshLab” (shown in the top panel of Figure 6). In Figure 6, the top panel illustrates a segment of the original part, and an exemplar segment of the 3D point cloud data. The light points represent the center points of the normal class. The dark points represent the center points of different defects. The middle panel shows a normal surface and four types of defects. The bottom panel shows the ratio of detected samples (the values before “/”) among all the samples in the new classes (the values after “/”). The majority of the wood part of length 2.85 meters is non-defective, with less than 20 defective areas in each defect class. However, it is not feasible to train a deep neural network by directly using the point clouds from such a limited number of defective areas. We hence adopt the following data-augmentation strategy to increase the sample size and balance the sample in each defect class, which is implemented in the three steps. (i) For each class, including the four types of defects and the normal surface, we sample 10,000 points from the highlighted areas in each class as the center points. (ii) Given each center point, we sample 300 points that come from the same highlighted area and lie within 10 mm from the center point. The distance threshold 10 mm is specified based on our prior knowledge on the sizes of defects. We treat the 300 points as one sample, which is stored as a $300 \times 3$ 3D point cloud matrix with a class label. (iii) We treat each 3D point cloud matrix as an image to employ the existing image classification techniques. Due to the fact that different permutations of the rows of a 3D point cloud matrix represent the same 3D object, we sequentially sort each 3D point cloud matrix based on the third and second columns (representing the Z and Y coordinates in the Cartesian coordinate system, respectively). In this way, our samples are invariant to the permutation variation, which is a key challenge in the area of 3D point cloud.
To this end, we have 10,000 samples for each data class. Each sample contains a $300 \times 3$ image and a class label. Similar to the analysis in Section 4, in each class, we extract 10-dimensional features from the flattened $900 \times 1$ images, and then divide the samples into a training data set of 8,000 samples and a test data set of 2,000 samples using “Split” (Vakayil et al. 2021; Joseph and Vakayil 2022).

To simulate the scenario where high-volume point clouds come from multiple data batches, we assume that samples from the normal surface, small dent, and corner crack classes are available in the initial batch. Then the samples from the non-smooth texture and long crack classes come in two batches sequentially. In these two batches, we presume the sample labels are not known until sending to the inspection station. For the reasons indicated in Section 4, we use the samples from the big dent class as auxiliary out-of-distribution samples for training the new-defect detector. The proposed approach is then applied for adaptive defect classification of the 3D point cloud data batches. Specifically, we train the classifier as a neural network for image classification, which includes two convolution layers and three fully connected layers. Each convolution layer is consisted of a 2D convolution with kernel size 3 and a maxpooling layer of size 3. The first convolution layer outputs 6 features, the second convolution layer outputs 16 features. The output dimensions of the three fully connected layers are 120, 84, and 7, respectively. In each batch, the neural network is trained for 200 epochs with a learning rate 0.0001. Based on the test performance, we set $\lambda_{ewc} = \lambda_{aud} = 0.1$. To guarantee the classification performance, we keep 3,000 out of 8,000 training samples from the previous batches when learning the new batch.

As was mentioned in the introduction section, we are interested in detecting defects and identifying their types. Therefore, the performance of the proposed approach is evaluated based on three criteria – (i) the new-defect detector should detect enough new defects for model update, (ii) the classifier should separate defects from normal surfaces, and (iii) the classifier should also identify the exact defect types. For criterion (i), we check the proportions of detected new types of defects in the training data set of each data batch, which are illustrated in the bottom panel of Figure 6. In the first batch, the new-defect detector identifies 4,940 out of 8,000 non-smooth texture samples as the new type of defects that have not been seen in the baseline data set. In the second batch, the new-defect detector screens 5,643 long crack samples from the total 8,000 training samples. As was indicated in Section 3.2, these proportions are not necessarily to be close to 1 once the sample size of the newly detected defects is enough for updating the model.

For criterion (ii), we check the number of non-defective test samples that are incorrectly classified as defects, as well as the number of defective test samples...
that are incorrectly recognized as normal. In both of the two data batches, the number of mis-classified cases is 4 among 2000 normal samples in the test data set. In data batch 2, 1 sample of long crack are incorrectly recognized as normal surfaces among a total of 8,000 defect samples. These results indicate the trained classifier is powerful in detecting surface defects.

For criterion (iii), the classification performance is illustrated in Table 1, which is the confusion matrix of the classifier in the test data set after training all the data batches. The proposed method has an overall test accuracy of \( \frac{1996 + 1913 + 1811 + 1690 + 1915}{2000 + 2000 + 2000 + 2000 + 2000} = 93.25\% \). Note here that the accuracy of catching a defect (i.e., Normal vs non-normal) is 99.99\%. Yet the ability of the model to separate all the classes is 93.25\%. Comparing to the number of mis-classified cases in criterion (ii), the defect classification has a relatively higher classification error, which mainly arises from the strategy of generating the 3D

Table 1. Confusion matrix of the classifier.

| Class          | Normal (true) | Small dent (true) | Corner crack (true) | Long crack (true) | Texture (true) |
|----------------|---------------|-------------------|---------------------|-------------------|----------------|
| Normal (predict) | 1996          | 0                 | 0                   | 1                 | 0              |
| Small dent (predict) | 4             | 1913              | 69                  | 29                | 6              |
| Corner crack (predict) | 0             | 11                | 1811                | 246               | 5              |
| Long crack (predict) | 0             | 46                | 86                  | 1690              | 74             |
| Texture (predict) | 0             | 30                | 34                  | 34                | 1915           |

![Figure 7. Exemplar mis-classified samples.](image)

Figure 7. Exemplar mis-classified samples.
produces reasonable results matching with the human classification performance. However, the mis-classification performance is not as good as the defect detection becomes larger. In the last row, when the points are sampled from a small segment of the long crack, they are likely to be classified as other types of defects, which gives majority of the mis-classification error. In the third row, a non-smooth texture could be classified as other types of defects when the surface variation becomes larger. In the last row, when the points are drawn from one surface on the corner crack, they are identified as a small dent and a non-smooth texture, respectively. The sample at block (2, 4) resembles a long crack, giving rise to a mis-classification error. In the third row, a non-smooth texture could be classified as other types of defects when the surface variation becomes larger. In the last row, when the points are drawn from one surface on the corner crack, they are likely to be classified as other types of defects, which gives majority of the mis-classification cases in Table 1. In all, due to the vague boundary among different types of defects and the variation in sampling the points from point clouds, the overall classification performance is not as good as the defect detection performance. However, the mis-classification examples in Figure 7 still implies that the classifier produces reasonable results matching with the human knowledge on the defect shapes.

6 Conclusion

A new adaptive defect classification framework is proposed for high-volume independent data batches. The integration of a continual learner and an out-of-distribution detector enables an effective inspection of unlabeled samples and updating the model upon the arrival of new defect data. The simulation and case study have shown the effectiveness of the proposed method.

In this article, we conduct anomaly detection based on the point-cloud data. The assumption is that the point cloud data cover the current features we are interested in. As such, our goal is to identify point cloud data that neither belong to the non-defective group or the to the set of anomalies we have. We do agree that one may use data other than point cloud to enhance predictive power when needed and available. In real practice, it is recommended to determine the data source for anomaly detection based on the domain engineering knowledge. Moreover, the hyperparameter determination are also sensitive to different product manufacturing processes and measurement environmental conditions. So, in practice, those parameters need to be trained for each given process/product under the fixed measurement conditions by following the similar training procedure proposed in the article. The main purpose of this article is to develop a generic training procedure that can be applied to different processes/products rather than providing the specific parameters for this particular process/product. It is recommended to tune the hyper-parameters following the procedure in Appendix B. A general guideline of the choice of the hyper-parameters for different defect characteristics is expected in the future work.

The imaging process can play a key role in model performance. Strong contrasts, dirt on the image, or different zooms can indeed affect performance. This has indeed been seen in the deep network literature. The good news is that there has been quite some literature recently on the automatic standardization of images and their features, often via invariance. We also plan to investigate this route in our future work.

The quality inspection in the case study is to check surface defects of a wood log that has four flat surfaces. This allows us to transform the 3D point clouds into surface defects’ images and train a image classifier. It should be clarified that the proposed statistical framework can be incorporated with any commonly-used classifier including for 3D point clouds. If defect detection is made on complex 3D objects, we recommend to apply a 3D point cloud classification tool as a classifier (e.g., PointNet, Qi et al. 2017).

The proposed statistical framework is tailored for the discretized data batches. However, it can be extended to the cases where data batches are generated from continuous manufacturing processes. Take the roll-to-roll manufacturing as an example, such as paper or fabric manufacturing process, we can divide the part surface into equally large non-overlapped grids, and then continuously inspect grids via a laser scanner. Our proposed approach can be applied to the scanned 3D point cloud data in each grid for defect classification during continuous production. The model is then updated when enough samples in the new defect types are collected. It is worth noting that when continuous monitoring data are collected, future works need to address the auto-correlations among the samples within each grid.
We end by noting that although a practical approach for selecting the auxiliary out-of-distribution data set has been provided in Section 3.5, it relies on whether we have additional old defect types to leave out, which may not be available in real applications. As a future work, a systematic approach should be developed to automatically generate out-of-distribution samples that resemble the patterns of samples in the new defect types. For example, a valid out-of-distribution sample for the MNIST data set should contain some random handwritten scratches instead of random mosaics.

Disclosure statement

No potential conflict of interest was reported by the authors.

About the authors

Dr. Wenbo Sun is a research faculty at the University of Michigan Transportation Research Institute and an affiliated faculty at the Michigan Institute for Data Science. He is interested in developing statistical methodologies for uncertainty quantification and decision making in complex engineering systems. He has been actively working on real-world problems in vehicle safety, assistant driving system, smart manufacturing, and health care.

Dr. Jionghua (Judy) Jin is a professor in the Department of Industrial and Operations Engineering and the Director of Manufacturing Program of Integrative Systems and Design Division at the University of Michigan. Her research interests are in data fusion and analytics in quality engineering with primary applications in manufacturing. Her research targets in-situ quality control for defect prevention by integrating design and manufacturing through optimizing operations and maintenance decisions concurrently.

Dr. Raed Al Kontar leads a data science group at the industrial & operations engineering (IOE) department at the University of Michigan. He is also an affiliate with both the Michigan Institutes for Data Science and Computational Discovery and Engineering. He works on developing data science methods for solving engineering problems.

Dr. Tzyy-Shuh Chang obtained his doctoral degree from the University of Michigan in 1995. The focus of his research was Systems Design and Integration. His specialty is systems that comprise mechanical design, electronic hardware, sensors, cameras, computer interfaces, and software.

ORCID

Raed Al Kontar http://orcid.org/0000-0002-4546-324X

Data availability statement

The data that support the findings of this study are available from OG Technology Inc. Restrictions apply to the availability of these data, which were used under license for this study. Data are available from the author, with the permission of OG Technology Inc.

References

Bergmann, P., S. Löwe, M. Fauser, D. Sattlegger, and C. Steger. 2018. Improving unsupervised defect segmentation by applying structural similarity to autoencoders. arXiv preprint arXiv:1807.02011.

Chen, Y., Y. Ding, F. Zhao, E. Zhang, Z. Wu, and L. Shao. 2021. Surface defect detection methods for industrial products: A review. Applied Sciences 11 (16):7657. doi: 10.3390/app11167657.

Cohen, N., and Y. Hoshen. 2020. Sub-image anomaly detection with deep pyramid correspondences. arXiv preprint arXiv:2005.02357.

Defard, T., A. Setkov, A. Loesch, and R. Audigier. 2021. PaDiM: A patch distribution modeling framework for anomaly detection and localization. Paper presented at the International Workshops and Challenges, virtual event, January 10-15. International Conference on Pattern Recognition Part IV: 475–489. Cham, Switzerland: Springer.

Deng, L. 2012. The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Processing Magazine 29 (6):141–2.

Gebral, N. Z., M. A. Lawley, R. Li, and J. K. Ryan. 2005. Residual-life distributions from component degradation signals: A bayesian approach. IIE Transactions 37 (6):543–57. doi: 10.1080/07408170590929018.

Goodfellow, I. J., J. Shlens, and C. Szegedy. 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572

Grasso, M., B. M. Colosimo, and F. Tsung. 2017. A phase i multi-modelling approach for profile monitoring of signal data. International Journal of Production Research 55 (15):4354–77. doi: 10.1080/00207543.2016.1251626.

Hackel, T., J. D. Wegner, and K. Schindler. 2017. Joint classification and contour extraction of large 3d point clouds. ISPRS Journal of Photogrammetry and Remote Sensing 130:231–45. doi: 10.1016/j.isprsjprs.2017.05.012.

He, J., R. Mao, Z. Shao, and F. Zhu. 2020. Incremental learning in online scenario. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 13926–35.

He, K., X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770–8.

Hsu, Y.-C., Y. Shen, H. Jin, and Z. Kira. 2020. Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10951–60.

Huang, H., Z. Li, L. Wang, S. Chen, B. Dong, and X. Zhou. 2020. Feature space singularity for out-of-distribution detection. arXiv preprint arXiv:2011.14654.

Jin, J., and J. Shi. 2001. Automatic feature extraction of waveform signals for in-process diagnostic performance improvement. Journal of Intelligent Manufacturing 12 (3): 257–68. doi: 10.1023/A:1011248925750.
Jing, L., D. Tingting, S. Dan, and X. Chunbo. 2014. A review on surface defect detection. *J Front Comput Sci Technol* 8 (9):1041–8.

Joseph, V. R., and A. Vakayil. 2022. Split: An optimal method for data splitting. *Technometrics* 64 (2):166–76. doi: 10.1080/00401706.2021.1921037.

Jovanc’evi’, I.-H.-H. Pham, J.-J. Orteu, R. Gilbas, J. Harvent, X. Maurice, and L. Br’ehes. 2017. 3d point cloud analysis for detection and characterization of defects on airplane exterior surface. *Journal of Nondestructive Evaluation* 36 (4):1–17. doi: 10.1007/s10921-017-0453-1.

Kirkpatrick, J., R. Pascanu, N. Rabinowitz, G. Desjardins, H. Soyer, J. Zaremba, V. Merel, M. Jenkins, and A. Rusu. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America* 114 (13):3521–6. doi: 10.1073/pnas.1611835114.

Kochurov, M., T. Garipov, D. Podoprikhin, D. Molchanov, A. Ashukha, and D. Vetrov. 2018. Bayesian incremental learning for deep neural networks. *arXiv preprint arXiv:1802.07329*.

Lee, K., K. Lee, H. Lee, and J. Shin. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *arXiv preprint arXiv:1807.03888*.

Li, Z., and D. Hoiem. 2018. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40 (12):2935–47. doi: 10.1109/TPAMI.2017.2773081.

Liang, S., Y. Li, and R. Srikant. 2017. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*.

Liu, W., X. Wang, J. D. Owens, and Y. Li. 2020. Energy-based out-of-distribution detection. *arXiv preprint arXiv:2010.03759*.

Madry, A., A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

Mallya, A., and S. Lazebnik. 2018. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7765–73.

Nguyen, A., J. Yosinski, and J. Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 427–36.

Ozawa, S., S. Pang, and N. Kasabov. 2008. Incremental learning of chunk data for online pattern classification systems. *IEEE Transactions on Neural Networks* 19 (6):1061–74. doi: 10.1109/TNN.2007.2000059.

Qi, C. R., H. Su, K. Mo, and L. J. Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 652–60.

Rebuffi, S.-A., A. Kolesnikov, G. Sperl, and C. H. Lampert. 2018. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010–7.

Ren, H., K. Tian, S. Hong, B. Dong, F. Xing, and L. Qin. 2019. Visualized investigation of defect in cementitious materials with electrical resistance tomography. *Construction and Building Materials* 196:428–36. doi: 10.1016/j.conbuildmat.2018.11.129.

Roy, D., P. Panda, and K. Roy. 2020. Tree-CNN: A hierarchical deep convolutional neural network for incremental learning. *Neural Networks* 121:148–60. doi: 10.1016/j.neunet.2019.09.010.

Ruping, S. 2001. Incremental learning with support vector machines. In *Proceedings 2001 IEEE International Conference on Data Mining*, 641–2. IEEE.

Rusu, A. A., N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Suykens, J. A., and J. Vandewalle. 1999. Least squares support vector machine classifiers. *Neural Processing Letters* 9 (3):293–300. doi: 10.1023/A:1018628609742.

Vakayil, A., R. Joseph, S. Mak, and M. A. Vakayil. 2021. Package ‘split’. *R Package Version* 1:1–5.

Wang, J., P. Fu, and R. X. Gao. 2019. Machine vision intelligence for product defect inspection based on deep learning and hough transform. *Journal of Manufacturing Systems* 51:52–60. doi: 10.1016/j.jmsy.2019.03.002.

Wu, Y., Y. Chen, L. Wang, Y. Ye, Z. Liu, Y. Guo, and Y. Fu. 2019. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 374–82.

Xiao, T., J. Zhang, K. Yang, Y. Peng, and Z. Zhang. 2014. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM International Conference on Multimedia*, 177–186. doi: 10.1145/2647868.2654926.

### Appendix A. Definition of the new-defect detector using ODIN score

When the ODIN score is employed, the score function is defined as

\[
s(x|\phi^{(i)}) = \max q_i(\tilde{x}|\phi^{(i)}),
\]

where \(\phi = \{T, \epsilon\}\) collects the temperature and disturbance parameters, \(q_i(\cdot|\phi^{(i)})\) represents the predicted probabilities after temperature scaling, and \(\tilde{x}\) represents the input under perturbation. Specifically, the temperature scaling function \(q_i\) is defined as

\[
q_i(x|\phi^{(i)}) = \frac{\exp \left( q_i(x|\phi^{(i)}) / T \right)}{\sum_{j=1}^{\left|\phi^{(i)}\right|} \exp \left( q_j(x|\phi^{(i)}) / T \right)}.
\]

The perturbed input can be computed as

\[
\tilde{x} = x - \epsilon \cdot \text{sign}(-\nabla_x \log q_{\text{true}}(x))(x|\phi^{(i)})
\]

where \(\text{sign}(\cdot)\) is the sign function, and \(\nabla_x\) is the gradient operator with respect to \(x\). The score function in Eq. [1] maps an input \(x\) into a real-valued score \(s\).

### Appendix B. Choice of the hyper-parameters

There are three key hyper-parameters to be tuned in the proposed method - \(\tau^{(i)}\), \(\lambda_{\text{ood}}\) and \(\lambda_{\text{prior}}\). The threshold \(\tau^{(i)}\) is tuned to guarantee an over \(\rho^{(i)}\) true positive rate as was shown in Eq. [8]. Here we only discuss the choices of \(\lambda_{\text{ood}}\) and \(\lambda_{\text{prior}}\).
The hyper-parameter \( \lambda_{\text{ood}} \) affects whether the score function can separate new defects from old ones. Since the score function is established on the trained classifier (as was shown in Eq. [4]), an over-parameterized deep neural network can be trained to simultaneously guarantee a low classification loss and an effective separation between old and new defects. In other words, the loss term \( L_{\text{cont}}(\theta) \) and the penalty term in Eq. [5] do not contradict each other, and thus the classifier’s performance should not be very sensitive to the choice of \( \lambda_{\text{ood}} \).

To validate this intuition, we fix \( \lambda_{\text{prior}} = 1 \), and test the proposed method in the MNIST data set by setting the value of \( \log_{10}(\lambda_{\text{ood}}) \) as \([-2, -1, 0, 1, 2, 3]\), respectively. The left panel of Figure A1 shows the test accuracy of the old class 0, 1 and the new classes 2, 3 and 4, 5 under different choices of \( \lambda_{\text{ood}} \), which indicates that the proposed method is not sensitive to the values of \( \lambda_{\text{ood}} \).

The hyper-parameter \( \lambda_{\text{prior}} \) encourages the optimizer to update the model parameter of the classifier around the vicinity of the model parameter, \( \theta^{(t-1)} \), in the previous task, which prevents the mis-classification of the samples in the old tasks. Intuitively, training with a small \( \lambda_{\text{prior}} \) results in a classifier with poor prediction performance on the samples in the old tasks. A large \( \lambda_{\text{prior}} \) ties the model parameter around the vicinity of \( \theta^{(t-1)} \), and thus results in a classifier that cannot learn the new tasks. In section 3.5, we suggest to draw a given number of samples in each of the old defect types within the memory budget, such that the classifier can still predict both the old and new tasks when the model parameter is not close to \( \theta^{(t-1)} \). This treatment introduces the robustness of the proposed method with respect to the choice of \( \lambda_{\text{ood}} \).

To validate the arguments above, we fixed \( \lambda_{\text{ood}} = 1 \), and varied the values of \( \log_{10}(\lambda_{\text{prior}}) \) in the set \([-2, -1, 0, 1, 2, 3]\). The right panel of Figure A1 shows the test accuracy of the old class 0, 1 and the new classes 2, 3 and 4, 5 under different values of \( \lambda_{\text{prior}} \). The figure illustrates that the method performance is poor when \( \lambda_{\text{prior}} \) is greater than 100. Given that the model performance is not sensitive to \( \lambda_{\text{ood}} \), we used the widely-used grid-search method to determine the univariate hyper-parameter, \( \lambda_{\text{prior}} \). We selected \( \lambda_{\text{prior}} = 1 \) because the algorithm under \( \lambda_{\text{prior}} = 1 \) returns the highest average prediction accuracy among all the classes, which is 0.9572.

Furthermore, we compared the model performance under different numbers of old samples that are brought back to the training data set. The comparison is visualized in Figure A2, where the averaged test accuracy for the three tasks increases with the sample size of old samples. Although more samples in the previous batch can improve the model performance, we would recommend stop introducing old samples whenever the test accuracy satisfies practitioner’s expectation or the number of old samples exceed the memory budget.