Optimizing Wafer Classification in Industrial Manufacturing Using Particle Swarm Optimization and Deep Learning

Pisit Suwannoot[1] and Poom Konghuayrob[2*]

[1]Department of Robotics and AI, School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand
[2]Department of Electrical Engineering, School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand

(Received February 4, 2025; accepted August 1, 2025)

Keywords: particle swarm optimization, pattern classification, convolutional neural network (CNN), wafer classification

In this study, we examine the application of convolutional neural networks (CNNs) for wafer pattern classification, with a focus on enhancing training efficiency and model performance. To achieve this, particle swarm optimization (PSO) is employed to improve the model performance while reducing its complexity, a critical factor in production environments. By minimizing the number of layers, the proposed method accelerates training, reduces resource consumption, and enhances defect detection accuracy. Wafer failure patterns are classified into four categories: vertical, rectangular, edge, and horizontal. The approach achieves an impressive *F*1-*score* of 0.988, significantly surpassing the traditional CNN's score of 0.83. By integrating PSO, the method considerably improves the visual inspection process for hard disk drives, contributing to high-quality production. This optimization not only streamlines workflows but also enables manufacturers to address issues more rapidly, aligning with Industry 4.0's objectives of automation and intelligent monitoring.

1. Introduction

Wafers are essential materials for reader and writer components in hard disk manufacturing and undergo processes such as circuit fabrication, coating, cleaning, and polishing.(1) Engineers perform visual inspections to detect defects, such as contamination or anomalies, that can impact the performance of read and write operations in final devices.

By incorporating AI and computer vision, automation streamlines wafer inspections, enhancing quality control and reducing manual tasks in manufacturing.(2) Researchers have developed various methods of classifying wafer defects. Shin and Yoo used some compressed lightweight models such as EfficientNetV2, ShuffleNetV2, and MobileNetV3 to predict wafer map bin classification.(3) Nakazawa and Deepak(4) provided a technique for detecting and segmenting defect patterns using convolutional neural network (CNN) models such as fully

*Corresponding author: e-mail: poom.ko@kmitl.ac.th

https://doi.org/10.18494/SAM5551

convolutional network, segmentation network, and U-shaped network. Unlike traditional methods, deep convolutional neural networks perform end-to-end training without intermediate steps, processing wafer map images to classify defects. For a more detailed analysis of parameters, such as the location, size, and orientation of defect clusters, engineers require segmented defect clusters, particularly during the early phases of technology development. Shim *et al.*(5) used CNN with active clusters and uncertain estimation patterns with high accuracy while reducing the cost of manual annotation by an engineer. Kim and Kang(6) used a self-supervised learning-based dynamic wafer bin map clustering method, called the convolutional autoencoder method, which combines deep learning and pseudo-labeled data for obtaining dynamic patterns, offering robust, scalable clustering and improved defect pattern classification compared with benchmark models. Kaitwanidvilai *et al.*(7) introduced a parameter transfer learning (PTL) approach to improve virtual metrology (VM) by enabling knowledge transfer across factories and recipes, enhancing prediction accuracy and cutting retraining costs for adaptable wafer defect classification. Similarly, Rungtalay and Kaitwanidvilai(8) developed a dual-stage framework combining novelty detection and supervised learning to identify rare or unseen patterns in imbalanced data, addressing challenges of infrequent defects in evolving production. Both studies highlight adaptive, high-accuracy inspection methods suited for Industry 4.0 manufacturing. The analysis of large amounts of data collected from the production process and the results can help engineers make the right decision in classifying failure patterns. Implementing deep learning (DL) in production is challenging, particularly owing to issues related to high costs, long training times, and model complexity.(9,10) The computational resources in the training stage required for training large models can be expensive.(11,12) Focusing on the training periods further increases the operational cost because it is costly to train and deploy large DL models.(13)

The main objective of this study is to develop an optimization method by using particle swarm optimization (PSO) to fine-tune the key hypothesis of neural networks before setting parameter training in production. This research is focused on the training stage for wafer classification in the hard disk drive process, emphasizing the optimization of batch size, hidden size, and the number of convolutional (Conv) layers. The two primary objectives of the objective function approach are to achieve fitness functions to optimize model performance on the validated dataset and to reduce model complexity by minimizing the number of Conv layers, thereby striking a balance between the performance and complexity of the model.

2. Data, Materials, and Methods

2.1 Dataset preparation

In this study, a dataset of 7000 wafers was utilized to represent potential defects and surface patterns, as shown in Fig. 1. After wafer manufacturing, the wafer is classified visually by an expert team. Following labeling, data preparation involved splitting the dataset into 5000 wafers (70%) for training and 2000 wafers (30%) for testing. The wafers were classified into the following four categories, as shown in Fig. 2, on the basis of surface features: vertical pattern

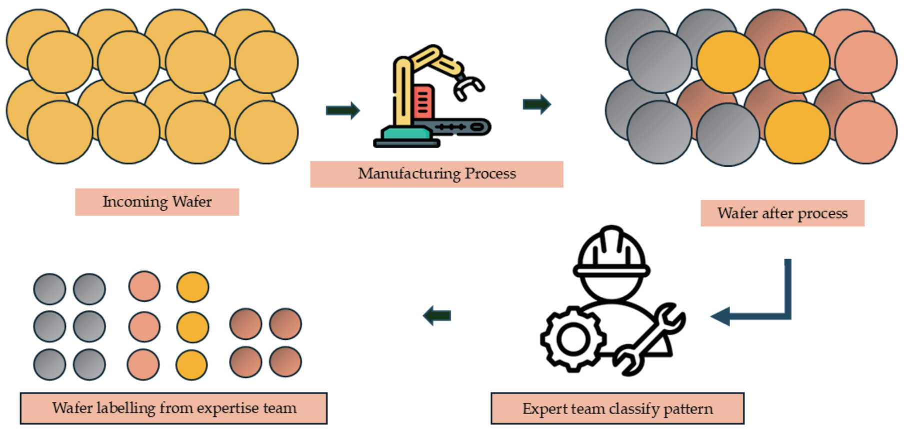

Incoming Wafer
Manufacturing Process
Wafer after process
Expert team classify pattern
Wafer labelling from expertise team

Fig. 1. (Color online) Preparation of dataset of manufacturing process.

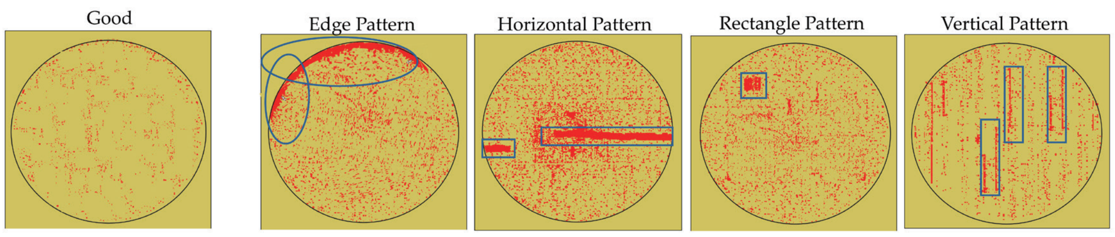


Fig. 2. (Color online) Examples of wafer failure patterns in the labeling process.

(distinct vertical lines indicating specific processes), horizontal pattern (prominent horizontal lines suggesting alignment issues), edge pattern (defects primarily at wafer edges; crucial for quality control), and rectangle pattern (rectangular shapes indicating design features or defects). Table 1 illustrates the number of defective wafers for each failure pattern as applied to this research.

2.2 CNN

CNN is a popular DL model widely used for image classification and computer vision. It excels in tasks such as object detection, image classification, and segmentation, making it essential for real-time video analysis.(14–16) The architecture design concept of CNN includes four types of layers: Conv layer, pooling layer, activation functions, and full connector (FC).

2.2.1 Conv layer

This layer consists of several convolution kernels that extract different features from various local areas of the input data, each neuron acting as a kernel with three-dimensional components of length (L), width (W), and depth (D), where the depth of the output volume represents the

Table 1
Number of wafers for failure patterns in training and test datasets.

	Vertical pattern	Rectangle pattern	Horizontal pattern	Edge pattern
Train	335	281	301	288
Test	68	84	57	63

number of filters. The number of channels in CNN layers enhances its feature extraction capability but also raises computational complexity and cost. The resulting feature map, as shown in Fig. 3, is produced through convolution operations, effectively capturing local spatial patterns such as edges, corners, and textures within the input data.(17–20) The equation of convolution can be expressed mathematically as

$$Y_{i,j,d} = \sum_{c=1}^{D_{in}} \sum_{m=1}^{k_w} \sum_{n=1}^{k_h} X_{i+m-1,j+n-1,c} \cdot F_{m,n,c,d} + b_d. \tag{1}$$

2.2.2 Pooling layer

After generating feature maps, a pooling (down-sampling) layer is introduced to reduce dimensionality while preserving essential information. Pooling functions, such as max pooling and average pooling, operate by scanning over the input feature map and summarizing regions by selecting key values. As shown in Fig. 4, this process simplifies data, reduces image resolution, retains the number of filters, and lowers computational costs. Pooling also helps decrease feature map size, minimize overfitting, and preserve critical information.(21)

2.2.3 Activation functions

Activation functions are essential in neural networks as they introduce nonlinearity to enable the learning of complex data patterns. They transform the weighted sum of a neuron's inputs into an output signal passed to the next layer. Without activation functions, neural networks would be restricted to modeling only linear relationships. Nonlinear activation functions are widely used as they allow neural networks to approximate any function and capture the nonlinear relationships found in real-world data.(21,22) Below are some of the most widely used activation functions.

1. **Sigmoid Function**: The input is in the range of 0–1, which is suitable for binary classification. This transformation is calculated as

$$f(x) = \frac{1}{1+e^{-x}}. \tag{2}$$

2. **Tanh Function**: This function, centered on zero, outputs values between −1 and 1, offering symmetric outputs and nonlinearity with faster convergence than the sigmoid function, but still faces gradient diffusion issues. This transformation is calculated as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{3}$$

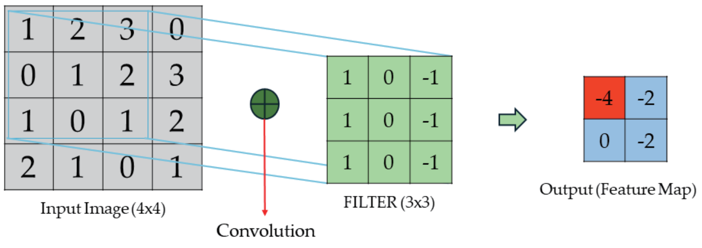


Fig. 3. (Color online) Convolution layer calculation using a visual map.

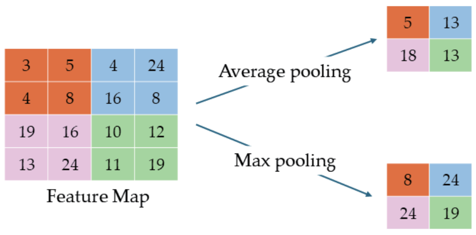


Fig. 4. (Color online) How pooling layers work.

3. **ReLU Function**: This is a popular nonlinear function in CNN. The input will convert from a value to 0 when $x < 0$ (negative input) and return to itself when $x > 0$ (positive input). The performance is better than those of the sigmoid and tanh functions owing to computation simplification. This transformation is calculated as

$$f(x) = \max(0, x). \tag{4}$$

2.2.4 Full connector

Fully connected (FC) layers, also known as dense layers, are essential parts of CNN. Located at the end of the network, their main job is to take the features extracted from the input data and produce final predictions by providing probabilities for each class. For example, in a study that classifies wafers into four patterns, the output layer would have four neurons, one for each pattern.

After the CNN processes the data through Conv and pooling layers, it flattens the output into a long vector. This vector is then fed into the FC layer, which helps capture important information from the previous layers. This setup allows the FC layer to improve the model's accuracy in tasks such as multiclass image classification by converting complex features into clear class probabilities.(23,24)

2.3 PSO

PSO is a swarm-based optimization technique. The algorithm maintains two key positions for each particle: the personal best position (P_{Best}) and the global best position (G_{Best}).(25,26) P_{Best}

represents the best solution that a particle has discovered so far on the basis of its own exploration of the search space. G_{Best} represents the best solution found by any particle in the entire swarm at any given time during optimization. These candidate solutions are updated in accordance with their fitness to the optimization problem. The algorithm begins with the initialization of a swarm of particles, each randomly positioned within the search space and assigned a random velocity. Following this, the fitness of each particle is evaluated using a defined objective function and then its velocity and position are updated on the basis of its own best known position as well as the best known position of the entire swarm. This collaborative approach allows particles to adjust their movements toward the most promising areas of the search space. The process of evaluation and update is iteratively repeated until a stopping criterion is met, such as reaching a maximum number of iterations or achieving satisfactory convergence. The mathematics of the movement of each PSO is defined as

$$v_{i(t+1)} = \omega \cdot v_{i(t)} + c_1 \cdot r_1 \times \left\{ \left(P_{Best(t)} - x_{i(t)} \right) + c_2 \cdot r_2 \times \left(G_{Best(t)} - x_{i(t)} \right) \right\}. \tag{5}$$

Here, $\boldsymbol{v}$ is the velocity of particle i at time t, ω is the inertia weight of balance between the local and global explorations, c_1 and c_2 are acceleration coefficients of the learning rate, r_1 and r_2 are random numbers between 0 and 1, P_{Best} is the personal best position of particle i, and G_{Best} is the global best position of the swarm. Each particle position is updated as

$$x_{i(t+1)} = x_{i(t)} + v_{i(t+1)}. \tag{6}$$

In this study, we employed PSO to identify the optimal configuration of hyperparameters—specifically, batch size, hidden size, and number of Conv layers to maximize model performance on a test dataset. Experimental results demonstrated that the PSO-optimized hyperparameters significantly enhance the performance of CNN models compared with baseline configurations. This approach not only streamlines the hyperparameter tuning process but also provides valuable insights into the relationship between model architecture and performance metrics. The proposed research is focused on the three key hyperparameters of the model architecture shown below and applies PSO to optimize them effectively.

1. **Batch size**: This is the number of training examples used in one iteration of the model. Hwang *et al.*(27) provided a technique for determining the optimal batch size for DL models applied to time series data with fast Fourier transform (FFT), suggesting that a systematic approach can significantly enhance model accuracy and training efficiency.
2. **Number of Conv layers**: The Conv layer is a fundamental component of CNN and uses filters (kernels) to scan over input images and detect features such as edges, textures, and patterns.(28) The input is typically a multichanneled image (e.g., grayscale or RGB). The convolution operation involves scanning a kernel (a small matrix with randomly initialized weights) across the image, computing dot products at each position, and summing the results to produce a feature map that highlights specific patterns. Stride and padding are crucial factors that affect the feature map's dimensions.(29)

3. **Hidden size**: The hidden size parameter in the FC layers of CNN determines the number of features in the hidden state, affecting the model's ability to learn patterns. A small hidden size may lead to underfitting, whereas a large one increases the number of parameters, raising computational costs and training time. Techniques such as dropout, which randomly disables neurons to prevent overfitting, and pruning, which removes unnecessary connections, can help balance model efficiency and resource management. This balance enables the CNN to effectively learn complex patterns while remaining computationally manageable.(29)

2.4 Evaluation method

To evaluate the optimization model, an objective function is designed on the basis of the $F1$-*score*, which serves as a measure of a classification model's performance on imbalanced datasets, as well as the complexity of the DL model.(30) The number of Conv layers in the neural network is also considered in this evaluation. The $F1$-*score* ranges between 0 and 1, where a value of 1 indicates perfect **precision** and **recall**, whereas a value of 0 signifies no precision.(31) By effectively accounting for both precision and recall, the $F1$-*score* provides a balanced assessment of the model's ability to classify instances accurately. The calculation methodology is

$$F1\text{-}score = 2 \times \left(\frac{Precision \times Recall}{Precision + Recall} \right). \tag{7}$$

The **precision** and **recall** of the model can be calculated using Eqs. (8) and (9), respectively, where TP = true positives (correctly predicted positive cases), TN = true negatives (correctly predicted negative cases), FP = false positives (incorrectly predicted positive cases), and FN = false negatives (incorrectly predicted negative cases).

$$Recall = \frac{(TP)}{(TP + FN)} \tag{8}$$

$$Precision = \frac{(TP)}{(TP + FP)} \tag{9}$$

The PSO objective is calculated using two factors, namely, model performance ($F1$-*score*) and model complexity, based on the number Conv and hidden layers from Eqs. (10) and (11), respectively. These two components are combined in Eq. (12), with all parameters normalized to a 0–1 scale to ensure fair comparison despite differing value ranges. This ensures that each contributes equally to the overall complexity metric.

C = Number of Conv layers from PSO searching space
H = Number of hidden size from PSO searching space
C_{min}, C_{max}: Lower, upper bounds of Conv layer in PSO searching space
H_{min}, H_{max}: Lower, upper bounds of hidden size in PSO searching space

$$C_{norm} = \frac{C - C_{min}}{C_{max} - C_{min}} \tag{10}$$

$$H_{norm} = \frac{H - H_{min}}{H_{max} - H_{min}} \tag{11}$$

$$Complexity\, matrix\, of\ model = \frac{C_{norm} + H_{norm}}{2} \tag{12}$$

An equation that combines the following two important evaluation criteria is created: the performance of the model represented by the $F1$-*score* in Eq. (7) and the complexity matrix of the model in Eq. (12). The objective function value can be determined as

$$PSO_{Objective} : \frac{F1}{Complexity\, matrix\, of\ model} \tag{13}$$

To make the results easier to understand, the values from the objective function are normalized to a scale of [0, 1]. On this scale, a value of 1 indicates that the model is performing at its best while keeping complexity to a minimum. This approach allows us to effectively evaluate and optimize the model's performance while considering its complexity.

2.5 Methodology

In the context of the CNN model, Fig. 5 shows the CNN architecture in this study. Conv layers, with the number determined by the PSO-optimized architecture, extract fundamental features such as edges and textures. The pooling layer reduces dimensionality, and the data then flows through three FC layers. The FC1 layer takes the flattened input with the number of neurons defined by the PSO-optimized hidden size. The FC2 layer further transforms the

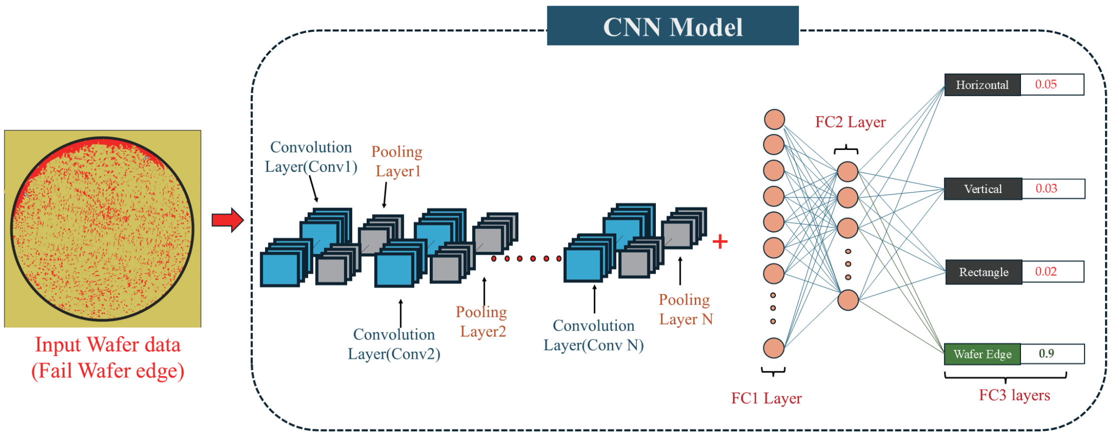


Fig. 5. (Color online) Structure of CNN applied to wafer clustering.

representation with additional neurons, while the FC3 layer produces the final classification results with neurons corresponding to the number of classes (e.g., four for wafer pattern classification).

To optimize the model using PSO, initial particles are placed within the search space as potential solutions. Key parameters, essential for the PSO algorithm's performance, must be set accordingly.

- Cognitive coefficient (c_1) = 2
- Social parameter (c_2) = 2
- Inertia weight (ω) = 0.9

Table 2 provides a step-by-step guide for using PSO to optimize the hyperparameters of DL models, which is our purpose. This process helps us find the best settings to improve the model's performance while minimizing the complexity of the model.

The optimization process is repeated for 10 iterations (n_iter = 10) with 10 starting points (particles) (n_par = 10). Each model undergoes training for 20 epochs. For each particle, the model's hyperparameters, such as batch size, hidden size, and the number of Conv layers, are randomly generated within specified lower and upper bounds. The variable bounds for these hyperparameters were set as follows: batch size in [30, 200], hidden size in [20, 200], and the number of Conv layers in [1, 5]. Each particle was initialized with random values uniformly sampled within these bounds. After training, the model is evaluated on a test dataset to calculate the objective function, as described by Eq. (13). The best objective function (P_{Best}) achieved for each particle is updated, and the global best (G_{Best}) across all iterations is determined.

The process is continued by updating the velocity and position of each particle based on the PSO Eqs. (5) and (6), which incorporates the P_{Best} and G_{Best} values. This step guides the particles toward regions of the search space with better objective function values. The optimization

Table 2
PSO pseudo-code algorithm.

```
Input: n_iter, n_par, Upper, lower bounds of batch size, hidden size, number of layers.
Output: Best Objective function(13)
Start
Initial swarm parameters [c1, c2, $\omega$]
Set $G_{Best}$ = 0, $P_{Best}$ = 0
While < n_iter:
    While < n_par:
        PSO Generate batch size, hidden size, Conv layers
            While < Epoch:
                Train model in and update hyperparameters
                Evaluate model on testing data (Calculate Cost)
                If Current cost >= $P_{Best}$:
                    $P_{Best}$ = Current cost
                End If
            End While
    If $P_{Best}$ >= $G_{Best}$:
        $G_{Best}$ = $P_{Best}$
    End While
End While
Stop
```

continues until a stopping criterion is met, such as a maximum number of iterations or a target objective function value. Finally, the model with the hyperparameters corresponding to G_{Best} is selected as the optimized model. By leveraging the PSO's swarm intelligence, this approach efficiently explores the hyperparameter space to find an optimal configuration, potentially enhancing the model's performance on the given task.

3. Results

The initial parameters were set using the algorithm and model settings discussed in the Methodology section, and PSO was implemented for the DL model. Figure 6 shows the calculation of the objective function, as described by Eq. (13), for determining the P_{Best} of each particle at iteration 9.

The contour plot in Fig. 7 allows for an intuitive understanding of how variations in batch size and hidden size impact the objective result of the PSO function. Observing the surface

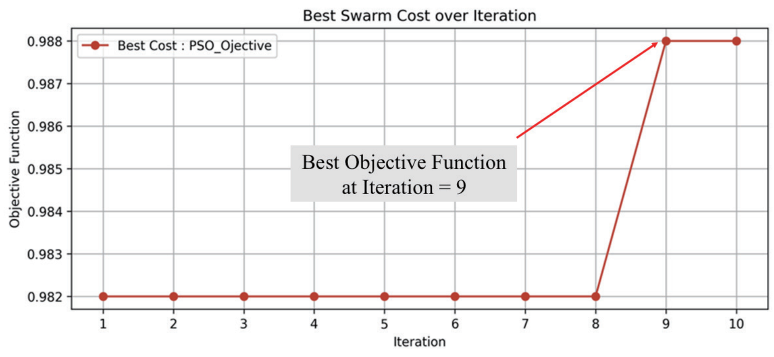


Fig. 6. (Color online) Best PSO objective function determined by iteration.

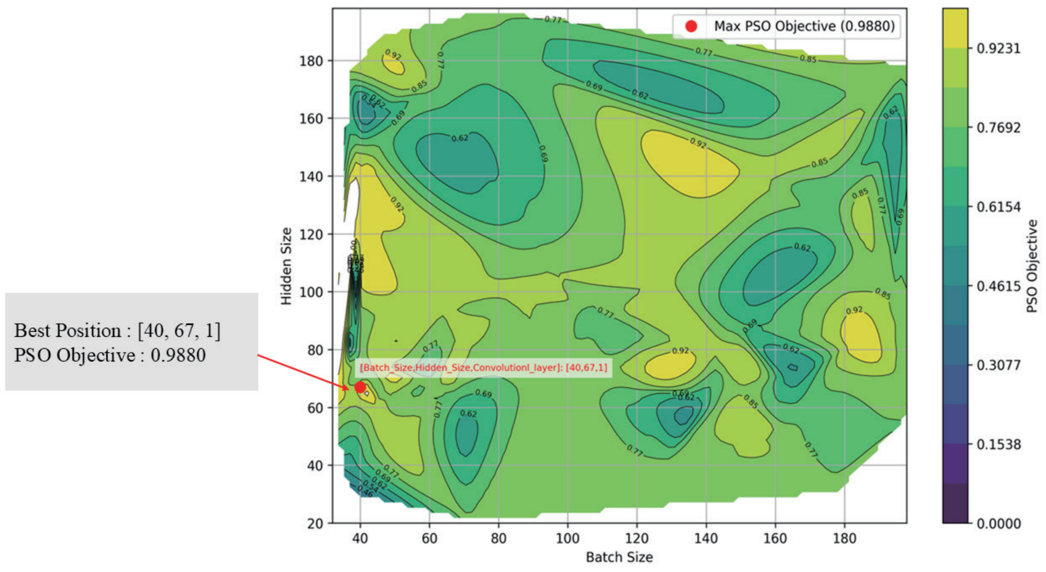


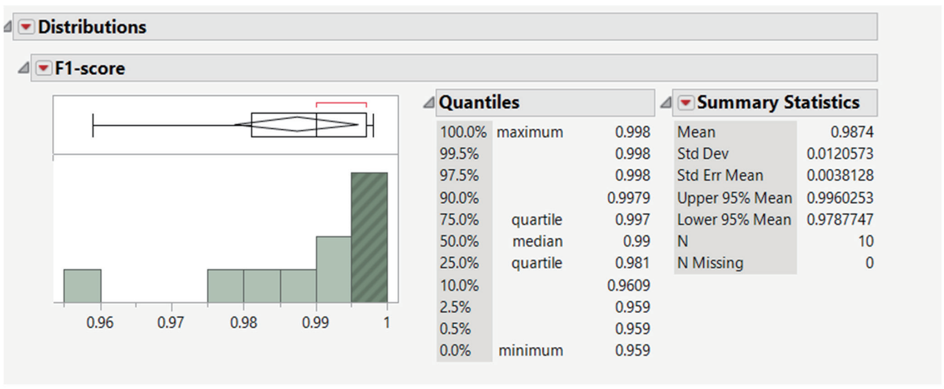
Fig. 7. (Color online) Contour plot of batch size, hidden size, and objective result obtained by PSO.

formed by the red plotted points in the contour graph can help in the identification of optimal configurations for batch size, hidden size, and the number of Conv layers that yield the best performance of the PSO-optimized model. The optimal model configuration was found at the PSO-determined position of (40, 67, 1), corresponding to a batch size of 40, a hidden size of 67, and 1 Conv layer. This configuration yielded the best performance in this study. To validate the effectiveness and stability of the selected hyperparameters, the experiment was repeated 10 times using the same settings. The results, summarized in Table 3, confirm consistent performance. Figure 8 presents the statistical analysis of the *F1-scores* across these repeated trials, demonstrating the reliability and reproducibility of the PSO-optimized configuration.

The proposed method was evaluated by comparing it with a traditional CNN without hyperparameter tuning, as well as other state-of-the-art DL methods including GRU, ResNet, VGG16, RNN, and Autoencoder. All models were trained using the same dataset, input size, number of training epochs (20), optimization algorithm (Adam Optimizer), and learning rate (0.001), as shown in Table 4. All experiments were performed on a workstation equipped with an NVIDIA RTX 4000 Ada Generation. The results indicate that the proposed method achieves the lowest computational complexity among all the models, as evidenced by both the training time and prediction processing time on 40 samples, as shown in Table 4, but also shows maximum performance represented by the *F*1-*score* result.

Table 3
Results of repeated experiment based on PSO objective function.

Experiment no.	PSO objective result	*F*1-*score*	*Precision*	*Recall*
1	0.99	0.99	0.99	0.99
2	0.982	0.984	0.982	0.982
3	0.997	0.997	0.997	0.997
4	0.99	0.99	0.99	0.99
5	0.996	0.996	0.996	0.996
6	0.998	0.998	0.998	0.998
7	0.959	0.963	0.957	0.959
8	0.987	0.987	0.988	0.987
9	0.978	0.978	0.979	0.978
10	0.997	0.997	0.997	0.997



Quantiles

100.0%	maximum	0.998
99.5%		0.998
97.5%		0.998
90.0%		0.9979
75.0%	quartile	0.997
50.0%	median	0.99
25.0%	quartile	0.981
10.0%		0.9609
2.5%		0.959
0.5%		0.959
0.0%	minimum	0.959

Summary Statistics

Mean	0.9874
Std Dev	0.0120573
Std Err Mean	0.0038128
Upper 95% Mean	0.9960253
Lower 95% Mean	0.9787747
N	10
N Missing	0

Fig. 8. (Color online) Summarized statistical analysis of results obtained from repeated experiments.

Table 4
Performance characteristics of proposed and traditional deep learning methods.

Model	Training time (s)	Processing time (ms)	*F*1-*score*	*Precision*	*Recall*
Proposed method	80	16.840	0.988	0.988	0.966
Traditional CNN	408	31.256	0.83	0.832	0.849
GRU	109	27.800	0.687	0.677	0.743
RNN	171	294.04	0.692	0.677	0.750
VGG16	1313	93.800	0.364	0.278	0.527
RestNet	4845	191.602	0.762	0.791	0.762
Auto Encoder	441	728.400	0.853	0.995	0.746

4. Conclusions

PSO is an effective method for identifying the optimal settings for DL models and enhances their performance while reducing training time and complexity. PSO not only reduces training time but also maintains model performance by effectively tuning hyperparameters such as batch size and hidden size. This capability is particularly beneficial in factory settings where computational resources may be limited and the rapid deployment of new systems is essential. This reduction in complexity helps lower the costs associated with implementing parameters in classification models before transitioning to production, which is crucial for businesses.

The wafer clustering model offers significant advantages in enhancing the process performance of hard disk manufacturing. By leveraging advanced algorithms to analyze and classify wafer data, engineers can effectively monitor wafer processing stages in real time. This proactive approach enables the early detection of anomalies during and after the production process, allowing for timely interventions that can prevent potential issues during production. Moreover, adopting such intelligent systems not only improves operational efficiency but also transforms traditional practices. For example, by automating the inspection process, reliance on manual visual assessments is significantly reduced. This shift minimizes human error and streamlines workflows, allowing engineers to focus on more critical tasks requiring their expertise. In summary, integrating wafer clustering models enhances performance metrics and fosters a smarter, more efficient manufacturing environment.

Acknowledgments

This research was made possible through funding received from the School of Engineering, King Mongkut's Institute of Technology Ladkrabang for industrial purposes (Grant number KREF016328).

References

1 C. Xanthopoulos, P. Sarson, H. Reiter, Y. Makris: IEEE Int. Test Conf. (2017) 1. https://doi.org/10.1109/TEST.2017.8242040
2 S. H. Huang and Y. C. Pan: Comput. Ind. **66** (2015) 1. https://doi.org/10.1016/j.compind.2014.10.006
3 E. Shin and C. D. Yoo: Sensors **23** (2023) 1926. https://doi.org/10.3390/s23041926
4 T. Nakazawa and V. K. Deepak: IEEE Trans. Semicond. Manuf. **32** (2019) 250. https://doi.org/10.1109/TSM.2019.2897690

5 J. Shim, S. Kang, and S. Cho: Expert Syst. Appl. **183** (2021) 115429. https://doi.org/10.1016/j.eswa.2021.115429
6 D. Kim and P. Kang: IEEE Trans. Semicond. Manuf. **34** (2021) 444. https://doi.org/10.1109/TSM.2021.3107720
7 S. Kaitwanidvilai, C. Sittisombut, Y. Huang, and S. Bom: Processes **13** (2025) 962. https://doi.org/10.3390/pr13040962
8 N. Rungtalay and S. Kaitwanidvilai: Sens. Mater. **36** (2024) 1487. https://doi.org/10.18494/SAM5036
9 E. Oluwasakin, T. Torku, S. Tingting, A. Yinusa, S. Hamdan, S. Poudel, N. Hasan, J. Vargas, and K. Poudel: Mach. Learn. Appl. **13** (2023) 100483. https://doi.org/10.1016/j.mlwa.2023.100483
10 B. Shah and H. Bhavsar: Procedia Comput. Sci. **215** (2022) 202. https://doi.org/10.1016/j.procs.2022.12.023
11 G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. W. M. van der Laak, B. van Ginneken, and C. I. Sánchez: Med. Image Anal. **42** (2017) 60. https://doi.org/10.1016/j.media.2017.07.005
12 A. Paleyes, R. G. Urma, and N. D. Lawrence: ACM Comput. Surv. **55** (2022) 29. https://doi.org/10.1145/3533378
13 G. Menghani: ACM Comput. Surv. **55** (2023) 37. https://doi.org/10.48550/arXiv.2106.08962
14 A. Devarakonda, M. Naumov, and M. Garland: arXiv:1712.02029 (2017). https://doi.org/10.48550/arXiv.1712.02029 (accessed August 2024).
15 M. M. Taye: Computers **12** (2023) 91. https://doi.org/10.3390/computers12050091
16 N. Sharma, V. Jain, and A. Mishra: Procedia Comput. Sci. **132** (2018) 377. https://doi.org/10.1016/j.procs.2018.05.198
17 A. Krizhevsky, I. Sutskever, and G. E. Hinton: Commun. ACM **60** (2017) 84. https://doi.org/10.1145/3065386
18 J. Liu and Y. Jin: J. Autom. Intell. **2** (2023) 175. https://doi.org/10.1016/j.jai.2023.10.002
19 X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, and M. Parmar: Artif. Intell. Rev. **99** (2024) 99. https://doi.org/10.1007/s10462-024-10721-6
20 C. L. Yang, Z. X. Chen, and C. Y. Yang: Sensors **20** (2020) 168. https://doi.org/10.3390/s20010168
21 M. M. Taye: Computation **11** (2023) 52. https://doi.org/10.3390/computation11030052
22 Y. Wang, Y. Li , Y. Song, and X. Rong: Appl. Sci , **10** (2020) 1897. https://doi.org/10.3390/app10051897
23 S. H. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee: Neurocomputing **378** (2020) 112. https://doi.org/10.1016/j.neucom.2019.10.008
24 T. N. Sainath, A. R. Mohamed, B. Kingsbury, and B. Ramabhadran: 2013 IEEE Int. Conf. Acoustics, Speech and Signal Processing (2013) 8614. https://doi.org/10.1109/ICASSP.2013.6639347
25 Y.-L. Chen, N.-C. Wang, M.-Y. Chen, Y.-F. Huang, and Y.-N. Shih: Sens. Mater. **26** (2014) 325. https://doi.org/10.18494/SAM.2014.992
26 W. Kanjanapruthipong, P. Prasitmeeboon, and P. Konghuayrob: Sens. Mater. **36** (2024) 1377. https://doi.org/10.18494/SAM4825
27 J. S. Hwang, L. S. Soo, G. J. Won, and L. C. Ki: Sustainability **16** (2024) 5936. https://doi.org/10.3390/su16145936
28 Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner: Proc. IEEE **86** (1998) 2278. https://doi.org/10.1109/5.726791
29 L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan: J. Big Data **8** (2021) 53. https://doi.org/10.1186/s40537-021-00444-8
30 T. Tziolas, T. Theodosiou, K. Papageorgiou, A. Rapti, N. Dimitriou, and D. Tzovaras: 2022 13th Int. Conf. Information, Intelligence, Systems & Applications (IISA) (2022) 1–8, https://doi.org/10.1109/IISA56318.2022.9904402
31 A. Alzammam, H. Binsalleeh, B. AsSadhan, K. G. Kyriakopoulos, and S. Lambotharan: 2019 Int. Conf. Advances in the Emerging Computing Technologies (AECT) (2020) 1–6. https://doi.org/10.1109/AECT47998.2020.9194155