Arab American University

Faculty of Graduate Studies

Silicon Wafer Defects Classification Using Deep Learning Techniques

By

Hussein Salah Younis

Supervisor

Dr. Amjad Rattroot

This thesis was submitted in partial fulfilment of the requirements for the Master's degree in Computer Science

March /2024

Thesis Approval

Silicon Wafer Defects Classification Using Deep Learning Techniques

By

Hussein Salah Younis

This thesis was defended successfully on 18/4/2024
and approved by:

Committee members	Signature
1. Dr. Amjad Rattroot: Supervisor	Dr Amjad RATTROUT [illegible]
2. Dr. Mujahed Eleyat :Internal examiner	[illegible]
3. Dr. Rashid Jayousi :External examiner	R. Jayousi

Declaration

I Hussein Salah Hussein Younis, hereby declare that the work presented in this thesis has not been submitted for any other degree or professional qualification and that it is the result of my independent work.

The Name of The Student: Hussein Salah Hussein Younis

Student ID: 202120178

Signature: [illegible]

Date: 27/7/2024

Acknowledgments

In the name of Allah, the Most Gracious, the Most Merciful

I dedicate this work to my father, Dr. Salah Younis, a constant source of inspiration and motivation. His dedication and determination have set a high standard for me. I am grateful for his support and hope to make him proud. Special thanks to my supervisor, Dr. Amjad Rattroot, for his invaluable guidance. I also acknowledge the faculty members and colleagues whose support and expertise have greatly influenced this thesis. Thank you all for your contributions.

To my family and friends, I express my heartfelt thanks for your unwavering support and encouragement at every stage of this thesis. Your moral support has been instrumental in my ability to achieve this milestone.

Finally, I eagerly anticipate presenting the findings of this thesis with pride and confidence, with the hope that it will contribute to knowledge and advancement in the field of study.

Once again, I would like to express my gratitude to everyone for their valuable contributions and continuous support.

Thank You

Hussein Younis

Abstract

The manufacturing of semiconductor wafers is a complex process that is prone to defects. In this study, we present DefectClassifierX, an automated pattern classification system that uses a convolutional neural network model based on the GoogLeNet architecture and leverages CUDA for faster training and testing speed. We aim to improve defect classification in the semiconductor manufacturing process by accurately classifying single and mixed wafer defect patterns. To validate our approach, we conducted thorough experimentation using the newly introduced dataset called "WM-300K+ wafer map [single and mixed]," which consists of 36 different defect patterns. The experiment results show that the precision, recall, and F1-score for testing our model were all measured at 0.97, indicating excellent performance. Also, the results demonstrate a remarkable level of accuracy, with an average classification accuracy of 99.9% for both single and mixed defect types. Our approach outperforms previous studies in wafer defect pattern classification and has the potential to significantly improve the efficiency and effectiveness of wafer defect analysis in semiconductor manufacturing. Additionally, we utilized hyperparameter tuning with Optuna and implemented a patience stop mechanism for improved convergence. Moreover, we incorporated the AdamW optimizer to further enhance the model's performance. DefectClassifierX is compatible with multiple operating systems, ensuring accessibility for a broader user base. While our results are encouraging, further research is needed to address limitations regarding dataset quality, computational resource requirements, and data augmentation techniques. Additionally, it is important to evaluate the model using real wafer map images for practical applicability assessment.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ML	Machine Learning
AI	Artificial Intelligence
DL	Deep Learning
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
CBAM	Convolutional Block Attention Module
ANN	Artificial Neural Network
CCE	Categorical Cross-Entropy
Relu	Rectified Linear Activation Function
C+EL	Center with Edge-Loc
C+ER	Center with Edge-Ring
C+L	Center with Loc
C+S	Center with Scratch
L+S	Loc with Scratch
D+S	Donut with Scratch
D+EL	Donut with Edge-Loc
D+ER	Donut with Edge-Ring
D+L	Donut with Loc
EL+L	Edge-Loc with Loc
EL+S	Edge-Loc with Scratch
ER+L	Edge-Ring with Loc
ER+S	Edge-Ring with Scratch
C+EL+S	Center with Edge-Loc with Scratch
C+ER+S	Center with Edge-Ring with Scratch
C+EL+L	Center with Edge-Loc with Loc
C+ER+L	Center with Edge-Ring with Loc
C+L+S	Center with Loc with Scratch
D+EL+S	Donut with Edge-Loc with Scratch
D+ER+S	Donut with Edge-Ring with Scratch
D+EL+L	Donut with Edge-Loc with Loc
D+ER+L	Donut with Edge-Ring with Loc

D+L+S	Donut with Loc with Scratch
EL+L+S	Edge-Loc with Loc with Scratch
C+L+EL+S	Center with Loc with Edge-Loc with Scratch
C+L+ER+S	Center with Loc with Edge-Ring with Scratch
D+L+EL+S	Donut with Loc with Edge-Loc with Scratch
D+L+ER+S	Donut with Loc with Edge-Ring with Scratch
GPU	Graphical Processing Unit
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
Apis	Application Programming Interfaces
Urls	Uniform Resource Locators
Adamw	Adam With Weight Decay Regularization
Adam	Adaptive Moment Estimation
ROC	Receiver Operating Characteristic
TP	True Positives
TN	True Negatives
FP	False Positives
FN	False Negatives
TPR	True Positive Rate
FPR	False Positive Rate

Chapter One

Introduction

This chapter serves as an introduction to the study. It provides background information on the topic and presents the research problem and questions to be addressed. The aims and objectives of the study are outlined, highlighting the purpose and goals. The significance and motivation behind the study are discussed, emphasizing its importance. Additionally, the chapter delves into the research scope, defining the boundaries and extent of the study. Lastly, the organization of the thesis is described, providing an overview of how the subsequent chapters are structured.

Background Information

The demand for electronic devices has significantly increased owing to the Fourth Industrial Revolution (Industry 4.0), advancements in semiconductor manufacturing and Internet of Things devices[1], [2]. Fortune Business Insights statistics indicate that the global consumer electronics market is expected to see substantial growth reaching 989.37 billion USD by 2027 [3]. Electronic devices are composed of integrated circuits containing various electronic components such as resistors, transistors, and diodes [4]. These components and their connections are built on a semiconductor wafer typically composed of single-crystal Si, as shown in **Figure 1** [5].

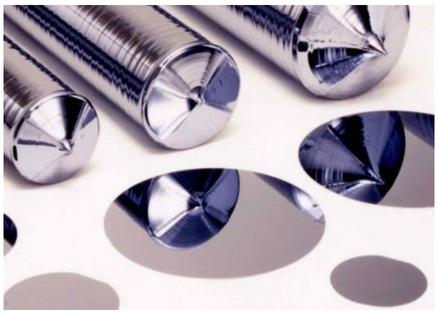


Figure 1: *A sample of silicon ingots and wafer surfaces* [1]

The fabrication process of integrated circuits involves the use of a thin, circular slice of material known as a wafer [6]. This process involves the transformation of raw materials or components into a final product. Semiconductor wafer manufacturing forms the core of integrated circuit production, is highly complex, as illustrated in **Figure 2** [7].

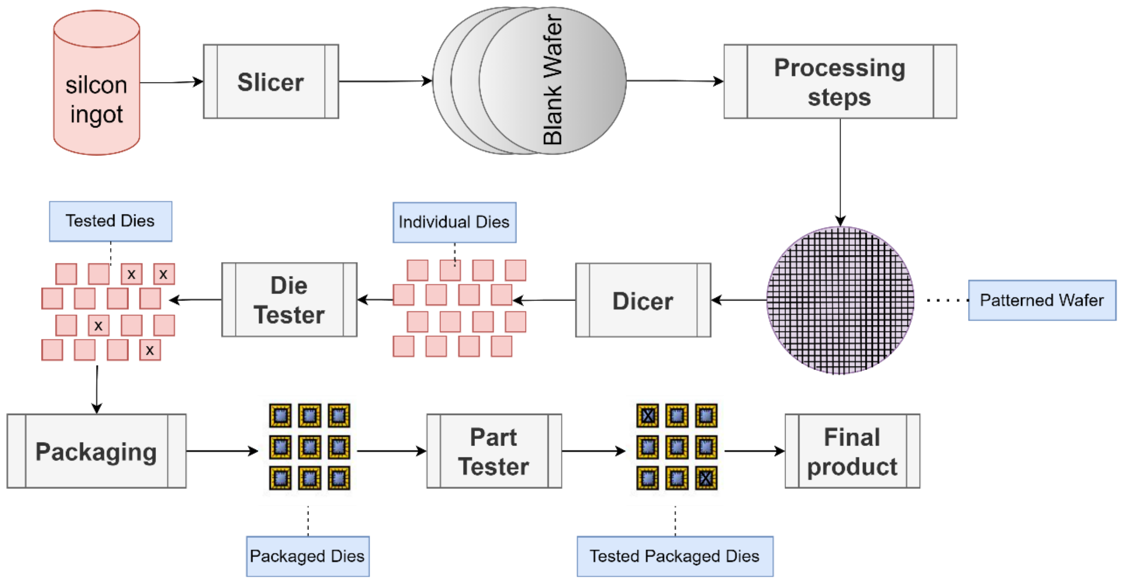


Figure 2: *Overview of general processes in integrated circuits manufacturing.*

In the context of silicon wafer manufacturing, the processes include general steps such as Wafer processing, Oxidation, Photomask, Etching, Film Deposition, Interconnection, Testing and Packaging. The details of these steps are as follows:

1. **Wafer processing:** In this process, silica sand is heated to separate silicon and carbon monoxide until ultra-high-purity electronic-grade silicon (EG-Si) is obtained. Then the “EG-Si” is melted and cast into a large cylinder form called “ingot.” After that, the “ingot” is sliced into a certain thickness called "die" which is an unprocessed raw wafer [8].

[1] Image was uploaded by Yoshifumi Uesaka from Nikkei staff writer (source Image URL)

2. **Oxidation**: In this process, a protection layer is added to the wafer surface to protect it from chemical impurities, current leakage, and wafer slipping during the etching phase [9].
3. **Photomask**: In this process, circuit patterns are printed onto wafers using photomask technology, which uses ultraviolet irradiation [10].
4. **Etching:** This is a very important phase that includes removing all oxide film and unwanted parts from the wafer's surface using wet or dry etching, depending on the materials used [11].
5. **Film Deposition:** This process uses a variety of techniques, including chemically vaporized deposition (CVD), atomic layer deposition (ALD), and physical vapour deposition (PVD), to produce film materials on a semiconductor wafer at the molecular level [10].
6. **Interconnection**: In this process, the wafer components and layers are connected electrically to allow the transmission of signals from one layer to another. Aluminium (Al) and copper (Cu) are primarily used in the interconnection process [12].
7. **Testing**: In this process, different types of testing are performed to ensure that the wafer's quality meets semiconductor manufacturing standards. All components are performed normally, defects in the wafer are identified and eliminated, and defective components are replaced [13].
8. **Packaging:** This is the final process in which individual wafer chips are encapsulated for protection, and an electrical connection is added before wafer dicing, which allows the attachment of many integrated circuits into a single wafer [14].

Semiconductor wafer manufacturing is a series of costly, complicated, and highly disciplinary processes in which the incidence of defects is significant. Complex and time-consuming diagnosis, inspection, and analysis processes are required in each phase to detect defects, which helps semiconductor engineers track and handle the source of failure in each phase before reaching the final production stage and ensure that the components are aligned correctly and operated correctly in a desired manner [15].

Typically, the semiconductor manufacturing process takes up to 26 weeks [16]. In addition, the training process for operators or engineers to manually classify defects with an accuracy of 90% takes up to 9 months [17]. During the wafer manufacturing process, there are two main sources of defects in the wafer: dust particles in the manufacturing environment, equipment and/or human errors [18]. **Figure 3** shows an example of a real defect appearing at the edge of a semiconductor wafer.

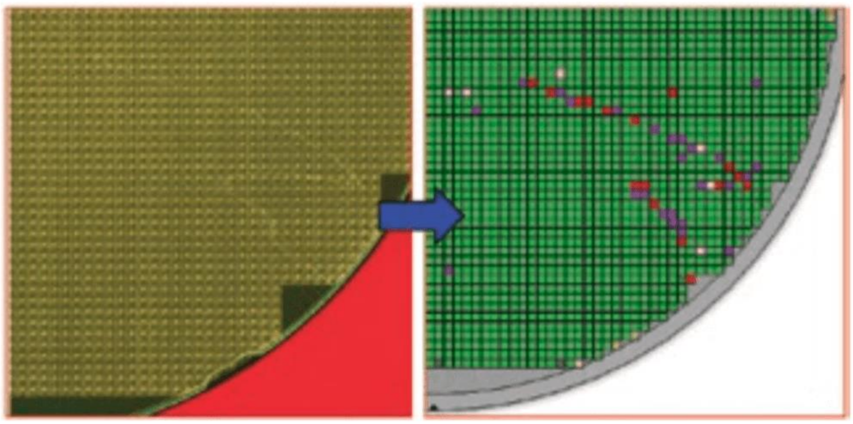


Figure 3: *A real sample of a defect on the edge of the semiconductor wafer.*[2]

In wafer manufacturing, there are typically two types of defect patterns: global defects, which are evenly spread across the entire wafer, and local defects, which show distinct spatial patterns. These spatial patterns can provide valuable information regarding specific manufacturing issues.

[2] Image was uploaded by Shannon Davis from Semiconductor Digest's (source Image URL)

Local defects are classified according to their distribution from the edge to the center of the wafer surface. The most common defect patterns are Center, Donut, Edge-Loc, Edge-Ring, Loc, Near-full, Random and Scratch, as shown in **Figure 4** [19].

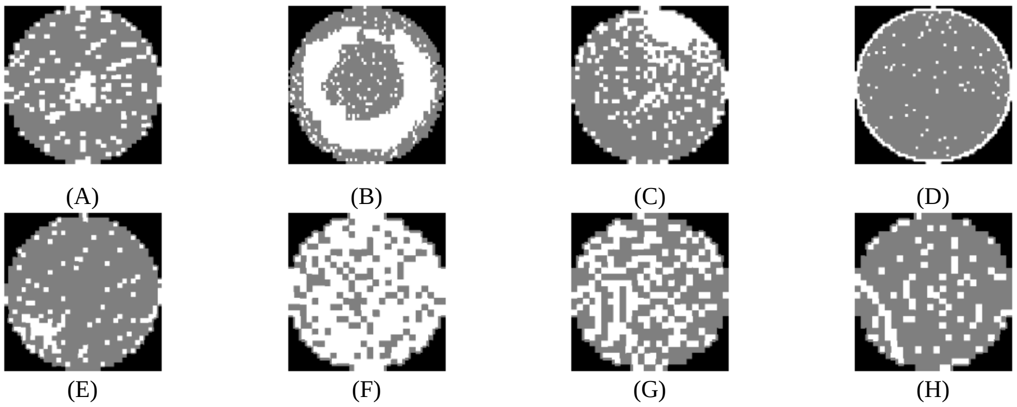


Figure 4: *Display of common defect patterns in wafer maps, including (A) Center, (B) Donut, (C) Edge-Loc, (D) Edge-Ring, (E) Loc, (F) Near-Full, (G) Random, and (H) Scratch.*

The various defects observed in the product can be attributed to various issues during the manufacturing process. For instance, a scratch often indicates mishandling by the machine, whereas Edge-Ring is typically associated with problems encountered during etching. The appearance of a center defect may signal complications arising from thin-film deposition techniques.[20]. **Table 1** provides a summary of different defect patterns encountered in the manufacturing process and their related issues. Each defect pattern is associated with specific causes and implications, ranging from alignment and etching problems to contamination, uneven deposition, material defects, insufficient material, impurities, equipment malfunctions, and mishandling by the machine. Understanding these defect patterns is crucial for identifying and addressing manufacturing issues effectively [21].

Table 1: *Defect Patterns and Related Issues in Manufacturing.*

Defect Pattern	Related Issues
Center	It indicates issues with alignment or positioning during the manufacturing process.
Donut	It may be caused by problems with the etching process or contamination during manufacturing.

Edge-Loc	It could be caused by improper handling or alignment during manufacturing.
Edge-Ring	It associated with problems encountered during the etching process.
Loc	It could be caused by various factors such as contamination, uneven deposition, or material defects.
Near-full	It could be caused by problems with the deposition process or insufficient material.
Random	It could be caused by various factors such as impurities, equipment malfunctions, or process variations.
Scratch	It often indicates mishandling by the machine or improper contact during manufacturing.

In general, there are many methods for inspection processes, the most popular of which is the automated inspection machine test, which produces wafer maps, as shown in **Figure 5**. The use of wafer maps is to visualize abnormal locations on silicon wafers and other important information for tracking and manufacturing processes [22].

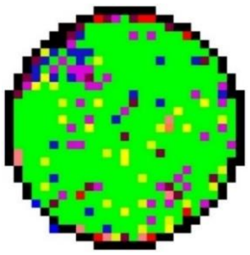


Figure 5: *A close-up view of a wafer map generated by wafer testing equipment*[3].

The wafer map indicates that each wafer dies if it passes or fails to meet performance standards [23]. Thus, a spatial pattern of wafer maps can be derived to classify whether the wafer production meets the performance standards and identify which wafer die contains a defect. During the inspection test phase, engineers can manually identify the causes of the defects in the wafer at each step, address them, improve the production lines, and reduce the production cost as much as possible. However, the accuracy of defect classification achieved by human experts is only

[3] Image was uploaded by Andre van de Geijn from English Wikipedia under public domain (source: https://en.wikipedia.org/wiki/Substrate_mapping#/media/File:Wafermap.jpg)

45%[24]. Furthermore, these methods are time-consuming, costly, complicated, and highly disciplinary.

In recent years, machine learning (ML), a subset of artificial intelligence (AI), has been gained significant traction in various research fields and has integrated into diverse domains, such as speech recognition and computer vision [25]. ML allows complex tasks to be solved without explicit programming by enabling machines to analyse data and uncover patterns similar to the human brain. Deep learning (DL) is a subset of neural networks within machine learning and can handle unstructured data in its raw form, including images and audio. Unlike traditional ML approaches that require human intervention for feature extraction determination, DL can automatically extract features from given data [26].

Various approaches exist for DL classification, including supervised, unsupervised, and partially supervised learning. In supervised learning; the model is trained using a dataset consisting of inputs (referred to as "features") and labelled outputs. This dataset is divided into three subsets: the training dataset used for model training, the validation dataset utilized to mitigate overfitting during training and enhance prediction accuracy and finally the testing dataset employed to validate overall model performance [27]. With the rapid advancement of DL techniques, automated defect detection and classification systems have emerged as promising solutions to overcome these limitations. The benefit of leveraging DL for wafer defect classification is improved performance, specifically, higher defect classification accuracy, minimized human error, and reduced time. The adaption and integration of DL for wafer defect classification is a heavily researched and developed application in which many studies have utilized wafer map images to deploy different pre-processing and learning strategies that are typically used to learn wafer map defect patterns [2].

This study introduced a silicon wafer defect classification system using deep learning techniques by adapting a deep convolutional neural network (CNN) as a model for wafer defect classification using wafer-map images as inputs. This approach relies on a series of data pre-processing and data augmentation operations to produce an effective method to advance the classification of single or mixed-type wafer map defects. In addition, this study introduced an automated pattern classification system for wafer defects called “DefectClassifierX.” The motivation behind this study was to develop an accurate and efficient system for classifying silicon wafer defects by leveraging DL techniques.

Research Problem and Questions

The production of semiconductor wafers involves complex procedures and strict quality-control measures. Defects in wafer manufacturing are common and necessitate thorough assessment, inspection, and analysis at each stage of production. Early defect classification plays a crucial role in addressing this issue effectively. Deep learning techniques have shown promise for achieving accurate and automated defect classification. However, a major challenge is the limited availability of public datasets specifically designed for wafer map defects [28].

The existing public ‘WM-811k' dataset suffers from class imbalance issues [29]. A new dataset called "Mixed-type Wafer Defect" has been derived from the 'WM-811k' dataset to overcome class imbalance issues. This new dataset focuses on capturing mixed types of wafer-map defect patterns, enabling researchers to develop more robust and accurate deep-learning models for defect classification [30]. In addition, researchers face access restrictions to electronic wafer maps and integrated circuit designs as they are proprietary to companies. These restrictions hinder the ability to conduct studies using real-world data for defect classification on wafers, thereby limiting the generalizability and applicability of the research findings.

Therefore, the research problem at hand revolves around addressing the challenges of limited public datasets, class imbalance issues, and restricted access to real-world data to develop effective deep learning models for early defect classification in semiconductor wafer production. Overcoming these challenges will contribute to improving quality control processes, reducing manufacturing costs, and enhancing overall productivity in the semiconductor industry [31].

The challenges of deep-learning-based classification models for wafer map defects can be summarized as follows [30], [32], [33], [34]:

1. Limited availability of datasets: One challenge in this research area is the scarcity of publicly available datasets for wafer map defects, which hampers the evaluation and comparison of deep-learning-based defect classification models on diverse datasets. Furthermore, owing to ownership rights and confidentiality concerns, electronic wafer designs are often inaccessible to researchers, making it difficult to obtain crucial resources for analysis.
2. Preparation of data: To ensure the accurate classification of defects, it is crucial to identify and implement appropriate methods for data preparation before the training and testing phases of the deep learning-based model.
3. Investigation of deep learning-based model for classification of mixed-type defects: It is important to assess the capability of the deep learning-based model

to accurately classify multiple defects within a single wafer, particularly when dealing with mixed types of wafer defect patterns.

4. Achieving high classification performance: It is essential to identify effective methods to achieve high classification performance using deep-learning-based models.
5. Overcoming performance problems: It is necessary to find solutions to address performance problems, such as memory limits during the pre-processing of data, training, and evaluation of the DL model.

These challenges were mapped using the following questions:

1. What are the appropriate methods for addressing class imbalance issues in an available dataset?
2. What are the appropriate methods for preparing data before the training and testing phases of a DL model?
3. How can mixed-type wafer defect patterns be generated? Can the DL model accurately classify multiple defects on a single wafer?
4. What are the methods for achieving high classification performance using deep learning-based classification models?
5. How can performance problems such as memory limits that may appear during the pre-processing of the data, training, and evaluation of the DL model be addressed?

Aims and Objectives of the Study

This study aims to revolutionize the classification of silicon wafer defects by developing an automated system that surpasses the limitations of traditional methods. A deep convolutional neural network architecture is employed as a model for classifying single and mixed types of wafer defects using wafer map images.

The approach involves data pre-processing and augmentation to enhance the accuracy of defect classification, reduce false positives and false negatives, and improve overall manufacturing efficiency. The implementation of this system can bring about significant implications for the semiconductor industry, including enhanced quality control, cost reduction, increased productivity, and improved customer satisfaction by achieving the objectives outlined below.

Therefore, this aim is achieved through the following objectives:

RO1: Produce a new, balanced wafer map defect dataset that includes various types of defects. Address the issues present in the current dataset, such as class imbalance, to ensure a more representative and balanced dataset for training and evaluation.

RO2: Publish the dataset to provide researchers with a representative and reliable resource for conducting in-depth studies and advancing the field of wafer map defect analysis.

RO3: Develop and deploy a customized DL model for accurate classification of wafer defects in an autonomous system that surpasses the performance of existing methods by achieving high precision and accuracy in categorizing different types of defects present on wafers.

RO4: Improve the DL model's capability to accurately classify mixed types of defect patterns in a single wafer map by developing techniques to address the complexity and variability associated with mixed-type defects, thereby achieving precise classification outcomes.

RO5: Enhance the processing speed and efficiency of the DL classification model and overcome performance limitations by employing parallel computation techniques such as Compute Unified Device Architecture (CUDA).

Motivation And Significance of The Study

This study aims to revolutionize wafer defect classification using deep learning techniques. Traditional methods have limitations in accuracy and efficiency, leading to quality control issues, increased costs, and decreased productivity. The available dataset also has issues such as class imbalance and limited labelled data, affecting classification accuracy and efficiency.

The proposed automated pattern classification system for wafer defects called "DefectClassifierX" utilizes deep learning, specifically a deep CNN architecture, to accurately classify both single and mixed types of wafer defects using wafer map images. The study employs modern techniques like data preprocessing and augmentation to overcome dataset problems, improve defect classification accuracy, reduce false positives and false negatives, and enhance manufacturing efficiency.

Implementing this system has significant implications for the semiconductor industry. It enhances quality control by accurately identifying and rectifying defects, increasing productivity through faster defect classification and allowing for timely interventions. Ultimately, it improves customer satisfaction by delivering high-quality electronic components.

The study's specific goals include producing a new and balanced wafer map defect dataset called "WM-300K+ wafer map [Single & Mixed]" for single and mixed types of defect patterns, publishing it to advance the field, developing and deploying a customized deep learning model for accurate classification, improving the model's capability to handle mixed-type defects, and enhancing process speed and efficiency through CUDA. This study has the potential to transform wafer defect classification, benefiting the semiconductor industry with enhanced quality control, cost reduction, increased productivity, and improved customer satisfaction.

Research Scope and Outlines

The research scope of this study is to create an accurate and effective system for classifying defects in silicon wafers through the application of DL techniques based on CNN that leverages wafer map images as input data. Various techniques for data preprocessing and augmentation are incorporated in the proposed methodology to enhance the classification of single or mixed types of wafer defect patterns. As a result, the baseline performance exceeds 90% accuracy. However, this research aims to achieve model performance with a maximum error rate of 10%. Therefore, we are introducing a production-ready software model solution. The "WM-811k" dataset is utilized, which includes single or mixed defect patterns and employs different strategies for data preprocessing to address imbalanced class issues. a new balanced wafer map defect dataset that includes various types of single and mixed defects will be published via the Kaggle website[4].

The study will present an application that takes input images of defects and outputs their respective defect classes. The application will be designed to be user-friendly, efficient, and reliable for seamless integration into the manufacturing process.

[4] The Kaggle website is an online platform for data science and machine learning that hosts a wide range of datasets.

Thesis Organization

The rest of this thesis is organized into several chapters. Chapter 2 discusses key theories and concepts related to digital images and convolutional neural networks (CNNs).

Chapter 3 reviews the existing literature related to the research topic, identifies gaps or limitations in the existing literature, highlights areas where further research is needed.

Chapter 4 describes the dataset used in the research, analyzes its characteristics, explains the data preprocessing techniques applied, presents data augmentation techniques for increasing defect pattern frequencies and generating mixed defect patterns, and discusses the categorical encoding technique used for labels or classes.

Chapter 5 presents the proposed methodology for solving the research problem, explains the GoogLeNet model and its components, discusses the implementation details of the classification framework and application development, discusses hyperparameter tuning and optimization techniques, and discusses evaluation metrics for assessing model performance.

Chapter 6 lists the development tools and system requirements used in the research project, describes the experimental setup, presents the results obtained from training experiments, evaluates model performance, discusses memory limit issues encountered during experiments and their solutions, analyzes and discusses results in comparison with previous research findings.

Chapter 7 summarizes the contributions made by the research study, discusses limitations faced during the research project, proposes potential future research directions or extensions, and provides a concise conclusion summarizing key findings and outcomes of the study.

Preliminaries

In this chapter, an overview of key theories and concepts related to the topic are discussed, including digital images and convolutional neural networks.

Digital Images

Digital images play a vital role in the digital information system and contemporary communication because of their capacity to visually depict and graphically convey information. Within computer vision, a digital image is formed through the amalgamation of an illumination source with the reflection of light rays from said source onto the scene being recorded [35]. A digital image can be described mathematically as a function $f\ (x, y)$, of coordinates (x, y). The value assigned to each coordinate represents the intensity or amplitude of the image at that point. Each element in the image is referred to as a pixel and holds information regarding color or intensity as shown in **Figure 6**.

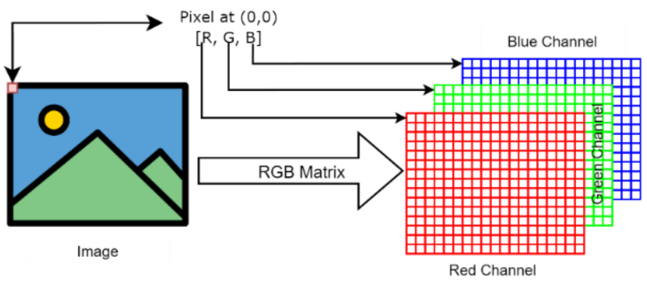


Figure 6: *The RGB representation of digital image, explaining the color channels used in pre-processing raw images for CNN-based defect classification.*

Typically, color information is represented using color models such as RGB, CIELAB, XYZ or CMYK [36]. RGB color space is an 8-bit depth where each pixel is represented as a tuple as demonstrated in the following equation [37]:

$$\langle R, G, B \rangle \in \{0 \dots 255\}^3 \tag{1}$$

Where R, G and B present the red, green and blue channel values for the pixel respectively. The values range for each channel is from 0 to 255.

Convolutional Neural Network

Artificial Neural Network (ANN), inspired by the structure and function of biological neurons in the human brain serves as the foundation for ANN. The human nervous system, which consists of specialized cells known as "neurons," is responsible for processing sensory information [38]. In ANN, each neuron receives input multiplied by a specific weight, influencing the computation performed by that particular unit. An instance of a neuron in an ANN is referred to as a "perceptron" [27].

The perceptron, a basic ANN developed by Frank Rosenblatt in 1958, is composed of multiple input nodes and one output node, as depicted in **Figure 7** [39].

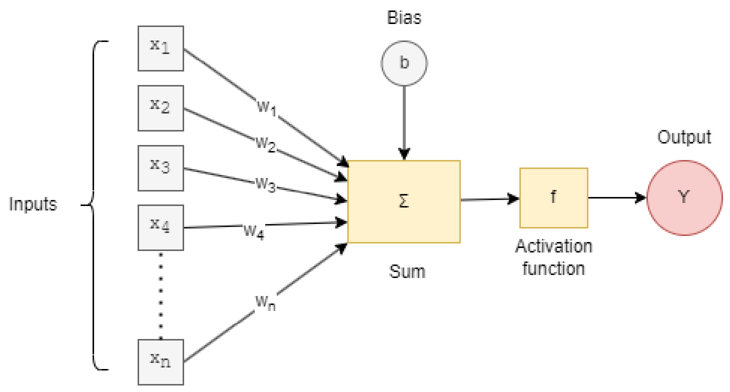


Figure 7: *The architecture of a single perceptron of ANN, illustrates the fundamental building block of neural network.*

ANN commonly consist of several layers, namely the input layer, hidden layers, and output layer. Each of these layers is composed of multiple perceptrons as depicted in **Figure 8**.

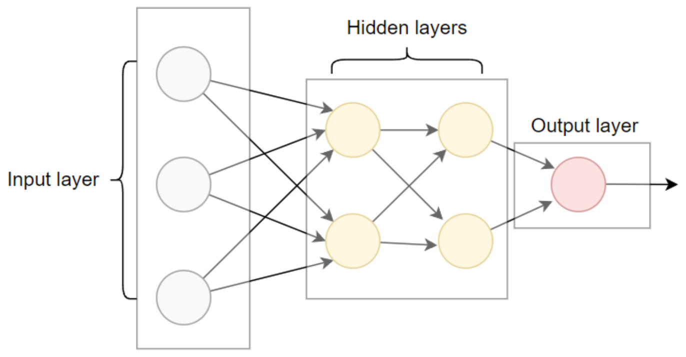


Figure 8: *The Architecture of ANN showcasing an input layer, two hidden layers, and an output layer.*

The input data flow through the perceptron from the input layer to the output layer (forward direction) is called "Feedforward propagation". Feedforward propagation refers to the process of passing input data through the network's layers in a forward direction to generate predictions or output. During feedforward propagation, the input data is fed into the first layer of the network, and the computations are performed sequentially layer by layer until the output layer is reached. Each layer applies a transformation to the input data using its weights and biases, followed by an activation function. The output of one layer serves as the input to the next layer until the final output is obtained. During the feedforward propagation phase, random initialization of weights and bias can lead to errors in achieving the desired output.

A mechanism known as "Backpropagation" is employed to address this issue. The backpropagation algorithm works by computing the gradient of the loss function concerning the network parameters (weights and biases) and then updating the parameters in the opposite direction of the gradient. This process is repeated iteratively until the loss is minimized and the network produces satisfactory results. During backpropagation, the error is first calculated at the output layer and then propagated backwards through the network. The error is used to update the weights and biases in each layer, with larger updates for layers that contribute more to the error. This process is repeated for each training example in the dataset until convergence is reached [39].

Back to CNN which is a class of DL was first invented by Yann LeCun and others in 1998. When ANN contains more than three layers, it is referred to as a deep learning algorithm. CNN is typically used in supervised learning tasks. In supervised learning, the CNN is trained using labelled data, where each input sample is associated with a corresponding target or output label. The goal of the CNN is to learn a mapping between the input data and the corresponding output labels. CNN can handle data in the form of arrays like RGB images. CNN can extract input information (features extraction) automatically by-passing inputs to its layers [40]. The term convolutional in CNN refers to the usage of convolutional layers to extract spatial features from images, making them more efficient and effective for image-related tasks than ANN.

The CNN typically consists of an input layer, hidden layers and an output layer. The hidden layers include one or more layers that perform convolutions as shown in **Figure 9.**

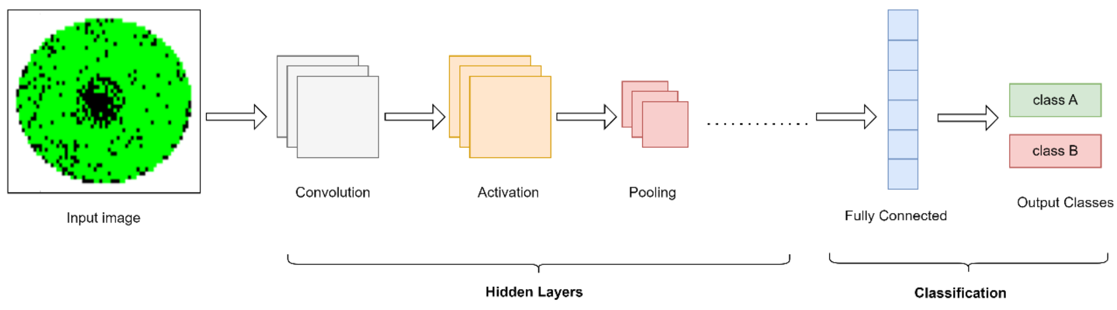


***Figure 9**: The architecture of CNN.*

Feedforward propagation in CNN is the same as in ANN however, there are some differences in the specific operations performed in each layer of a CNN compared to an ANN, but the core principles of feedforward propagation remain the same. It involves passing the input data through various layers, such as convolutional layers, pooling layers, and fully connected layers while applying activation functions and weight parameters. Also, the backpropagation phase in CNN follows the same general steps as in Artificial Neural Networks (ANNs) [41].

The steps of backpropagation in CNNs are as follows [41]:

1. Forward Pass: Perform a forward pass through the network by feeding the input data and calculating the output of each neuron layer by layer by performing specific operations, such as convolution, activation, and pooling, to generate intermediate outputs.
2. Loss Calculation: Compute the loss or error between the predicted output and the actual target output using a suitable loss function, such as categorical cross-entropy (CCE) for classification. The CCE is calculated by the following Equation:

   $$CCE = -\sum_{i}^{C} t_i \log(s_i) \tag{2}$$

   Where C is the number of classes, t_i is the target prediction for class i and s_i is the probability for i class.
3. Backward Pass: The backward pass is where the actual backpropagation happens. It involves calculating the gradients of the loss concerning the parameters (weights and biases) of the CNN.
4. Gradient Calculation and Weight Update: Propagate the gradients backwards through the layers, calculating the gradients of the loss concerning the weights

and biases of each neuron. Update the weights and biases using an optimization algorithm to minimize the loss.

5. Repeat steps 1-4: The forward pass, loss calculation, backward pass, and weight update steps are repeated iteratively for a certain number of epochs or until convergence is achieved.

Each layer in CNN performs specific operations on the input data, extracting relevant features and transforming the information [40]. These layers have their functionality as follows:

1. **Convolutional Layer:**

   It consists of a collection of learnable filters called "kernels". The inputs passe through these filters as tensors[5] with a shape of (number of inputs) × (input height) × (input width) × (input channels) to generate the output feature maps by performing a dot product between the kernels and the inputs with the same size as the kernels. This operation covers all elements in the input by sliding the filter over the element of the input with a fixed value known as "stride". The dimension of filters depends on the number of channels in input. For example, if the input is an RGB image (has 3 dimensions reflecting red, green, and blue channels) then kernels will have a depth of three. When the kernel size does not cover the whole image, padding can be added to extend the processing area and cover all pixels in the input image [42]. The convolution operation for a given RGB image I with three channels $C = 3$, a set of kernels K of a dimension $k_1 \times k_2$ and biases b is calculated by the following equation:

[5] A tensor is a concept in machine learning to organize and represent data such as multidimensional array.

$$(I \otimes K)_{i,j} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \sum_{c=1}^{C} K_{m,n,c} \cdot I_{i+m,j+n,c} + b \tag{3}$$

The output size of the convolution is calculated by the following Equation:

$$\mathit{The\ output\ size\ of\ Convolution} = \frac{W - F + 2P}{S} + 1 \tag{4}$$

Where W represent the width of a square input image, F represent the spatial size of the kernel, P represent the padding amount and S represents the sliding size. **Figure 10** shows an example of converting $4\ by\ 4$ image to $2\ by\ 2$ image by applying dot product for the input image with filter $2\ by\ 2$ with zero-padding and sliding size of 2.

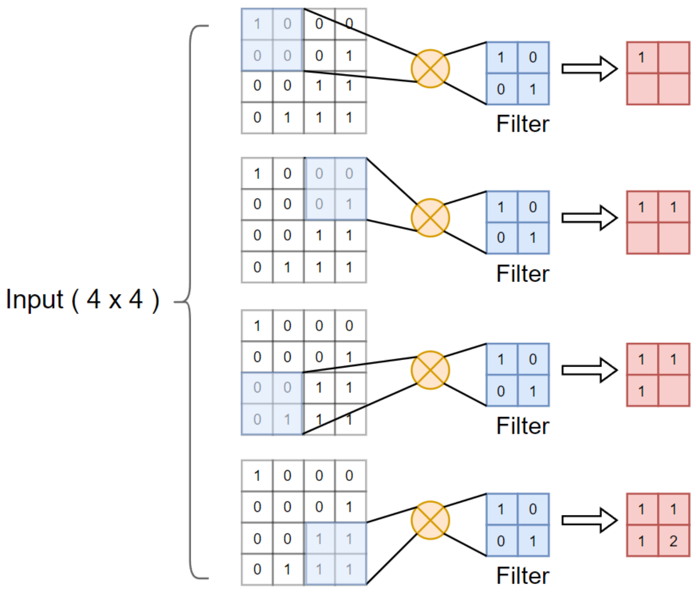


Figure 10: *An illustration of 4x4 to 2xrios image conversion using a 2x2 filter and zero-padding.*

2. **Pooling Layer:** It is used to sample/shrink the feature maps to extract important features by using pooling functions like Average Pooling, which takes the average pixel value for each patch[6], Min Pooling, which takes the minimum pixel value,

[6] Patch mean here the set of pixels that the pooling operation will apply on it and its always has a size less than feature map.

or Max Pooling, which takes the maximum pixel value [43]. **Figure 11** illustrates the operation of these functions. All three types of pooling operations are commonly used in CNN to reduce the spatial dimensions of feature maps and help prevent overfitting. By reducing the size of feature maps, pooling operations also help to decrease the computational complexity of subsequent layers in the network.

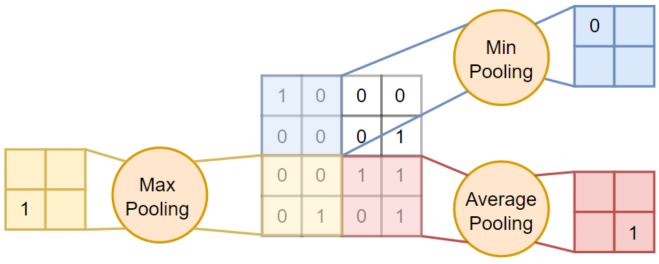


Figure 11: *An illustration of pooling operations types in CNN.*

3. **Activation layer:** The activation layer comes after all layers of CNN and consists of the activation function, which determines whether or not to activate the neuron. This operation allows CNN to adjust weights and biases, improving the learning phase (back-propagation) [44]. There is a common function associated with CNN called the rectified linear activation function (ReLU), as visualized in **Figure 12**.

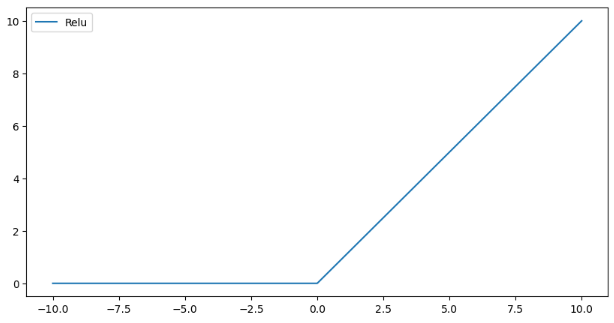


Figure 12: *Visualization of ReLU activation function.*

ReLU gained its popularity due to its phenomenal performance within CNN. The ReLU function converts all values to positive numbers as expressed in the following equation [45].

$$f(x) = \max(0, x) \tag{5}$$

4. **Fully connected layer**: This layer represents the classifier of CNN. it's located at the end of CNN where each node of this layer is connected to all nodes of the previous layer as shown in **Figure 13**. It is responsible for learning and mapping the high-level features extracted by the preceding convolutional and pooling layers to the desired output classes or predictions.

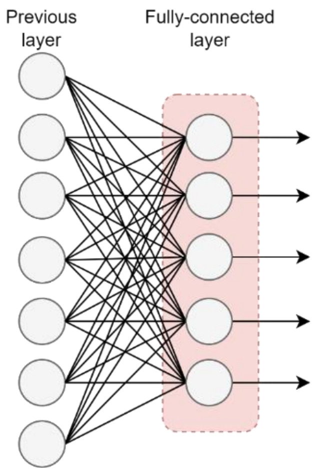


Figure 13: *The Fully-connected layer, an essential component in CNN for finalizing the classification of defects.*

The input of this layer is flatted into a one-dimensional array and then performs the calculation as in CNN feedforward propagation. The calculation is repeated for all layers and then an activation function called the "softmax activation function" is used to calculate the probability of the input to obtain the particular class. The softmax activation function formula is as follows [27]:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_j^k e^{z_j}} \tag{6}$$

Where the $\vec{z}$ represent the vector of input, the e^{z_i} represent the exponential of i element of the $\vec{z}$ and the k represents the number of classes.

5. **Dropout layer**: This layer is a regularization technique used to prevent overfitting by breaking the dependencies among neurons and encouraging the network to learn more robust representations that generalize better to new data. It works by randomly setting a fraction of the nodes in a layer to 0 during training, which creates a new and slightly modified network architecture for each run. The dropout probability determines how many nodes are set to 0, and the optimal probability depends on the layer type. For the input layer, a dropout probability close to one is optimal, while for hidden layers, a probability close to 50% leads to better results[46].

Chapter Two

Literature Reviews

In this chapter, an overview of relevant literature is presented. It explores existing research and identifies gaps in the literature that the study aims to address.

2.1 Overview of Relevant Literature

Several prior studies have been conducted on wafer defect classification, with each study incorporating different datasets, CNN architectures, and training techniques. In **Table 2**, a comprehensive overview of these studies is presented. It also includes details such as the utilized dataset, number of defect patterns considered in the analysis, whether single or mixed defects were classified, the specific CNN architecture employed in each study and its corresponding test accuracy for classification.

Table 2: *A comprehensive overview of the performance of different CNN models in defect classification.*

Ref	**Dataset used**	**#'s defects patterners**	**Single or Mixed defects classification**	**CNN architecture**	**Accuracy**
[47]	WM-811K	8	Single	Proposed architecture with 13 layers	93.25%
[48]	WM-811K and MixedWM38	38	Single and Mixed	(WM-PeleeNet) with 9 layers	93.6% for single and 97.5% for mixed
[49]	WM-811K	8	Single	(ShuffleNet-v2) with 7 layers	96.93%

[50]	WM-811K	9	Single	(CNN-WDI) with 16 layers	96.2%
[51]	Dataset-TT	12	Single	(ResNet) with (VGG-16)	96.2%
[29]	WM-811K	8	Single	(ResNet-18)	95.46%
[52]	WM-811K	9	Single	(YOLO) version 3, 3-tiny and 4	95.7%
[53]	MixedWM38	38	Single and Mixed	(U-Net)	95.8%
Our work	pre-processed WM-811K	36	Single and Mixed	(Modified GoogLeNet)	**99.9%**

Researchers in the field of wafer defect classification have made significant contributions to CNN architectures. While some researchers have designed their own CNN architectures, others have utilized existing ones. However, they face challenges such as imbalanced datasets and issues with overfitting and underfitting during model training. Researchers have employed data augmentation, regularization techniques, and attention mechanisms to tackle these problems. Additionally, investigations into the efficacy of current CNN architectures for accurately classifying various wafer defects are being conducted. Furthermore, semantic segmentation has been applied by some researchers to segment different defect patterns on wafer maps. As a result of these efforts from the research community improvements in accuracy and efficiency within wafer defect classification systems can be observed.

Researchers have developed custom CNN architectures for classifying wafer defects, aiming to improve performance and efficiency. In [47], they proposed using a 13-layer CNN to detect and classify eight known wafer map defects. The authors applied a median filter algorithm to remove noise from the wafer map images and resized them to 224x224 pixels. A Dropout method with a probability of 0.5 was employed to address the overfitting problem during training. The 'WM-811K' dataset which consists of 811,457 wafer map images including 172,950 labelled images and nine single defect patterns, served as the dataset for this study. The dataset was split into three subsets - training (60%), validation (15%), and testing (25%). The experiments conducted resulted in an impressive accuracy of 99.98% for detecting the presence of defect patterns on the wafer map. However, when it comes to classification, they achieved an average accuracy of 93.25%. The 'Donut' defect had a minimum average accuracy rate of 86%, while both the 'Edge-Ring' and 'Near-Full' defects achieved a maximum average accuracy rate of 100%.

Researchers have also utilized existing CNN architectures like 'PeleeNet', 'ShuffleNet-v2', and 'CNN-WDI' for the classification of wafer defects. By adapting these models specifically for this task, researchers intend to capitalize on their strengths and enhance performance within this specific domain. In [48], the researchers proposed a lightweight classifier for wafer defect classification based on a proposed CNN architecture called ‘’WM-PeleeNet” with nine layers derived from the 'PeleeNet' CNN architecture to achieve a good balance between accuracy and efficiency in wafer defect classification. The datasets used were the 'WM-811K' and 'MixedWM38' datasets. The 'MixedWM38' dataset contains 38,000 wafer maps with nine single defect patterns and 29 mixed defect patterns. A data augmentation approach was used based on convolutional autoencoder, GAN-based, and image transformation methods to address the unbalanced distribution of defect images in the datasets. The input images were resized to 224x224 pixels to standardize the dimensions. The experiments showed that an average accuracy of 93.6% was achieved for single wafer defect patterns, with the minimum average accuracy for classifying the 'Local' defect being 92.2%, and the maximum average accuracy for classifying the 'Edge-ring' defect being 97.6%. Also, for mixed types of wafer defects using the 'MixedWM38' dataset, the experiments showed an average accuracy of 97.5%, with the minimum average accuracy for classifying the 'Center with Edge-Ring with Scratch' defect being 93.4%, and the maximum average accuracy for classifying the 'Donut with Edge-Ring with Scratch' defect being 100%.

In [49], a silicon wafer defect identification and classification model was proposed by researchers. This model consists of a pre-trained deep transfer learning model called ShuffleNet-v2 with seven layers using CNN architecture. It achieves an overall accuracy of 96.93%, precision of 95.40%, recall of 96.26%, and F1-score of 95.75% in classifying the defects. The training and testing phase utilized the 'WM-811K' wafer dataset, with data augmentation performed using a six-layer convolutional autoencoder CNN model. The total number of images used after data augmentation was 19,707, representing nine different patterns for wafer defects. However, because it is focused on being lightweight, the ShuffleNet-v2 may sacrifice some accuracy to achieve faster inference time and lower computational cost.

Furthermore, in [50] the researchers proposed a deep learning-based CNN for automatic wafer defect identification with 16 layers. The model utilizes convolution layers to extract features and incorporates data augmentation and regularization techniques to enhance classification performance. Experimental results demonstrate that the model surpasses previous machine learning-based models, achieving an average classification accuracy of 96.2% on a wafer dataset 'WM-811K' consisting of nine wafer defect patterns. Methods such as image flipping, shifting, rotating, and zooming were applied during augmentation to address the class-imbalance problem in the dataset.

However, no mention was made regarding how they addressed the overfitting problem associated with their CNN model's high accuracy.

The researchers proposed an automatic defect classification system for wafer defect identification and classification. In, [51] the researchers present a system that utilizes DL techniques, specifically a ResNet101-based CNN model that was pre-trained on the ‘ImageNet’ dataset. For the sizing classification of two specific classes, the system employs a single-shot detection architecture based on VGG-16. The research utilized a dataset consisting of 8 defect classes, with 2 subclasses representing similar defects but varying in sizes. Data augmentation techniques were implemented to expand the dataset size. The obtained results showed a top-1 accuracy of 91.1% and a top-3 accuracy rate of 96.2%, with each class being correctly classified at least 69% of the time. The study acknowledges that the used dataset was relatively small and nonuniform. Future studies will explore employing advanced CNNs on larger datasets.

Some researchers have modified the existing CNN architecture by replacing certain layers with different algorithms to improve and increase the accuracy of defect classification in the system. In [29], researchers propose a DL method that leverages the attention mechanism and cosine normalization to learn robust knowledge from imbalanced datasets. They introduce an improved convolutional attention module called CBAM to enhance the representation capabilities of the CNN model. They use the ResNet-18 CNN model with an improved convolutional block attention module (CBAM). The proposed method achieves an average accuracy of 95.46% on the imbalanced ‘WM-811K’ dataset with nine wafer defect patterns. The researchers propose the use of a cosine normalization algorithm as a replacement for the fully connected layer to address quantity distribution imbalance. The fine-tuning of the classifier was achieved by minimal iterative training which decreased the quantitative distribution sensitivity. Resolving the issue of imbalanced datasets may facilitate the implementation of the algorithm in actual manufacturing.

Some researchers have investigated the performance of several current CNN architectures in wafer defect classification. This investigation aims to evaluate the effectiveness of these architectures in accurately classifying different types of wafer defects. In [52], researchers focus on using DL approaches for automated wafer defect detection in semiconductor manufacturing. This study emphasizes the importance of defect localization by evaluating the performance of ‘YOLOv3’ and ‘YOLOv4’ object detection models in accurately locating and classifying wafer defects. The dataset ‘WM-811K’ was used with nine wafer defect patterns and input wafer images were resized to 416x416 dimensions. The results show that these models achieve over 95.7% classification accuracy in real-time. Other architectures like ‘ResNet50’ and ‘DenseNet121’ were also evaluated for defect classification but lacked localization abilities. The study highlights the significance of defect detection for quality management and yield improvement in semiconductor manufacturing. It discusses the effectiveness of ‘YOLOv4’ in defect localization and classification, with an average F-score of 0.92. The challenges in training DL models for wafer defect detection and the trade-off between underfitting and overfitting are also addressed in their study.

Finally, other researchers have used semantic segmentation in the field of image processing and computer vision. This technique involves dividing an image into multiple parts or regions and assigning a semantic label or category to each region. This technique is typically implemented using CNN. in [53] the researchers proposed a new framework for segmenting different defect patterns on wafer maps using a semantic segmentation approach, particularly when multiple defect types are mixed on the same wafer. They used ‘U-NET’ CNN architecture which consists of an encoder path that captures the contextual information and a decoder path that recovers the spatial information. The proposed method works well on single and known/unknown mixed types of defects. The authors extract defects from the single-defect wafer map of the ‘MixedWM38’ dataset to generate single-defect pixel-level labels. They then generate a mixed defect pattern dataset suitable for semantic segmentation using single-defect wafer maps and labels. The proposed method achieves an average accuracy of over 97% on the test set of their synthetic dataset and 95.8% on the ‘MixedWM38’ dataset when using the trained model for testing. This study addresses the challenge of identifying and distinguishing complex defect patterns that arise when different defect types coexist on the same wafer. This can be a difficult task due to the complexity and variability of mixed-type defects.

2.2 Gaps In the Literature

In the current literature on wafer defect classification, there are notable gaps that require attention. One of the main challenges is dealing with multiple defect types on the same wafer which makes accurately classifying each defect difficult. The overlapping or mixed nature of different defect types can lead to ambiguity and confusion in the classification process. Developing effective algorithms and techniques to handle such complex scenarios is crucial for improving the accuracy of wafer defect classification. However, few studies have classified mixed types of wafer defects, indicating a need for further investigation.

Moreover, some studies have attempted to address this challenge by performing complex tasks for data augmentation. However, not all input wafer images require such complex methods for data augmentation. In some cases, the input wafer images may be relatively simple, and simpler data augmentation techniques may be more appropriate. Therefore, further research is required to identify the most effective data augmentation techniques for different types of wafer defect images.

There is a need to address the lack of information on the variability of defect patterns in the literature. Defects can have various characteristics and appearances which makes it difficult to create a reliable classification model. The differences in size, shape, texture, and intensity of defects make it even more challenging to classify them correctly. Advanced feature extraction methods should be explored along with innovative approaches that can accurately capture and represent the wide range of defect patterns to overcome this gap.

Furthermore, the absence of standardized datasets for assessing and comparing classification methods makes it challenging to evaluate objectively CNN model performance and determine the most effective methods. The creation of standardized datasets covering a wide range of defect types would significantly aid fair comparisons and advancements in wafer defect classification research. Addressing these gaps in the literature will contribute to the advancement of wafer defect classification techniques and pave the way for more reliable and efficient defect classification and quality control processes in semiconductor manufacturing.

2.3 Dataset and Data Pre-processing

This chapter focuses on the dataset used in the study. It provides a description and analysis of the dataset, highlighting its characteristics. The data pre-processing techniques employed are explained, including data cleaning and extracting, converting images to RGB format, and image resizing. Additionally, data augmentation techniques are discussed, such as increasing defect pattern frequencies and using a mixed defect patterns generator. The chapter concludes with a discussion on categorical encoding techniques.

2.3.1 Dataset Description and Analysis

In this study, a dataset called “WM-811k” [7] was used to derive a new balanced dataset. This dataset available under the public domain on the Kaggle website consists of nine distinct patterns of wafer defects [54]. **Table 3** provides detailed information about wafer maps, including their visual representation, size, identification details, usage labels, and the types of defects they may exhibit in this dataset.

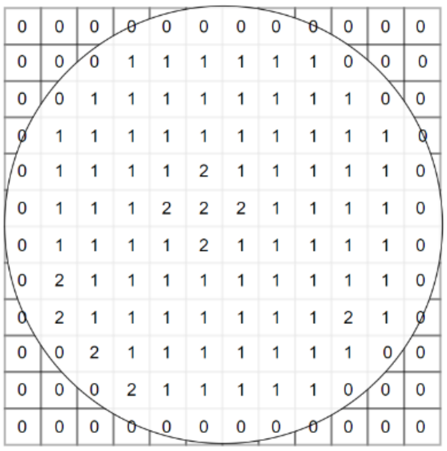
Table 3: *The description of columns in the “WM-811K” dataset.*

#	Column Name	Column Description	Column Type
1	waferMap	A two-dimensional array presentation for an 8-bit image for a wafer map.	2d array
2	dieSize	The size of the die in the wafer is in millimetres (die width * die height).	Float (64 bits)
3	lotName	Identification string for wafer fabrication process batch number. Many wafer maps can hold the same lotName.	string
4	waferIndex	Identification number for wafer map.	Float (64 bits)

[7] The dataset available under public domain on Kaggle website (source: https://www.kaggle.com/datasets/qingyi/wm811k-wafer-map)

5	trianTestLabel	Label to determine if the rows are used for training or testing processes. (Test, Training)	string
6	failureType	Wafer defects type. (Center, Donut, Edge-Loc, Edge-Ring, Local, Near-full, None, Random and Scratch)	1d array

Notably, the "waferMap" column in our dataset represents wafer map images. These images are stored as two-dimensional arrays, with each pixel being represented by an 8-bit unsigned integer, as shown in **Figure 14**.



0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	2	1	1	1	1	1	0
0	1	1	1	2	2	2	1	1	1	1	0
0	1	1	1	1	2	1	1	1	1	1	0
0	2	1	1	1	1	1	1	1	1	1	0
0	2	1	1	1	1	1	1	1	2	1	0
0	0	2	1	1	1	1	1	1	1	0	0
0	0	0	2	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure 14: *Visualization of wafer map representation with color map value from a dataset.*

The color map used for visualizing the wafer map image comprises three values according to **Equation 7** that helps in identifying the defect patterns and their distribution in the wafer map, which is crucial for accurate defect classification using CNN models.

$$pixel\ value = \begin{cases} 0 \rightarrow Non - Die\ Area\ (background\ or\ non - manufacturing\ region) \\ 1 \rightarrow Die\ Pass\ Test\ Successfully\ (no\ defects) \\ 2 \rightarrow Die\ Contain\ Defect\ (defective\ region) \end{cases} \tag{7}$$

The dataset contains a total of 811,457 wafer maps. However, it is worth noting that approximately 79% of these wafer maps do not have any pattern label for defect type or have “none” pattern label as shown in the histogram graph of failure types in **Figure 15**. A total of 25,519 wafer maps in the dataset have defect pattern labels.

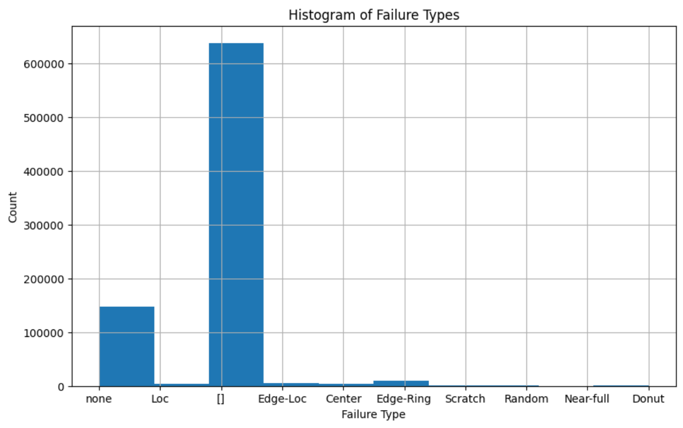
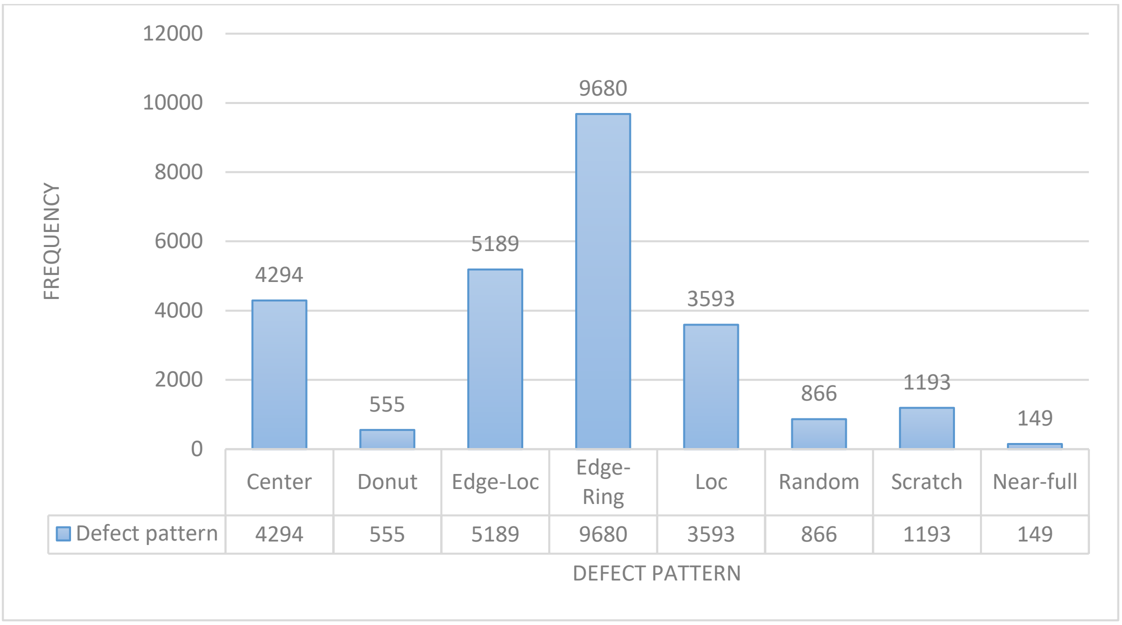


Figure 15: *Histogram of failure types, showcasing the distribution of different defect patterns in the dataset.*

Additionally, by examining the frequency distribution of defect patterns in the dataset for wafer maps with defect pattern labels as depicted in **Figure 16**, we can observe significant variations across different types of defect patterns. Moreover, it is important to highlight that the wafer maps arrays exhibit dimensions spanning across 632 unique values. Consequently, considering these variations in both defect pattern frequencies and wafer map dimensions becomes crucial during the data pre-processing phase to ensure accurate classification of defects using our proposed model.



	Center	Donut	Edge-Loc	Edge-Ring	Loc	Random	Scratch	Near-full
▦ Defect pattern	4294	555	5189	9680	3593	866	1193	149

Figure 16: *Visualization of Defect Patterns and their frequency in the "WM-811K" dataset.*

2.3.2 Data Pre-processing

Data pre-processing is essential to prepare the dataset for analysis and modelling, ensuring more accurate and reliable results. This becomes even more important in the context of a dataset that has variations in defect pattern frequencies and wafer map dimensions. The pre-processing methods include data cleaning, extraction, converting wafer map images to RGB format, and resizing these images to $56 \, x \, 56$ dimensions. The data pre-processing phase includes several steps, as demonstrated in **Figure 17**. Additionally, Figure 17 illustrates the other steps to prepare data for the training and testing phase.

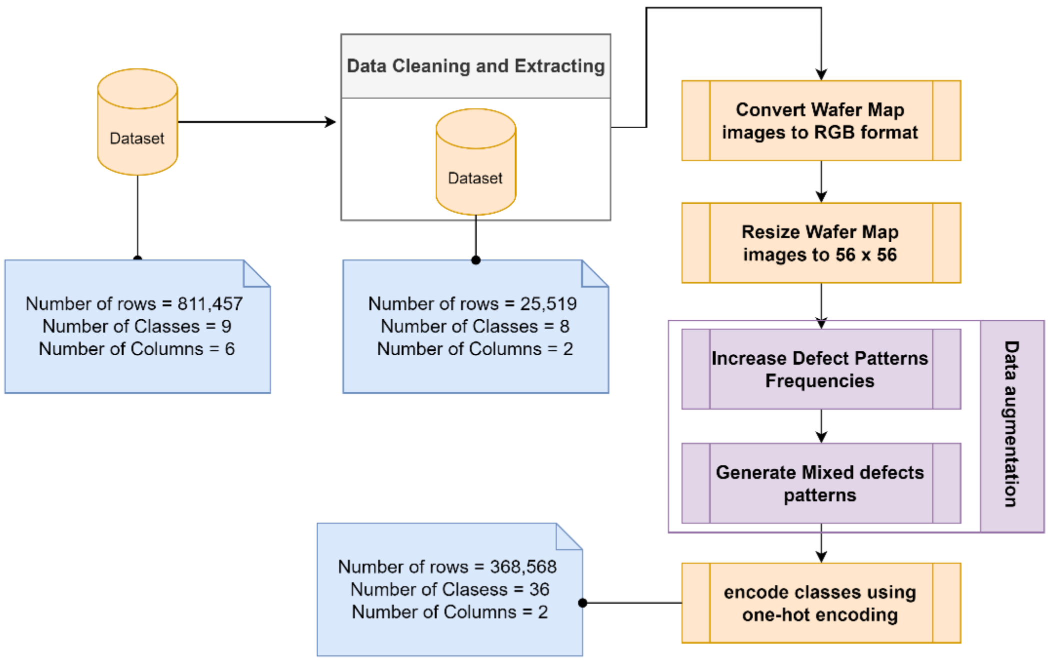


Figure 17: *Illustration of the main steps in data preprocessing, data augmentation, and encoding class labels, outlining the essential preprocessing steps required for training a CNN model for defect classification.*

2.3.3 Data Cleaning and Extracting

During the dataset preparation process for our deep learning model, a critical stage involved addressing classes that either had no defects or exhibited inaccurate defect patterns (failure type). These classes contained instances where the wafers showed no defects or where the captured defect patterns were not accurately represented.

A decision was made to exclude these classes from further analysis to ensure the quality and relevance of the dataset. The primary objective was to prioritize classes that represented authentic defect patterns and avoid introducing any noise or misclassification during the training process. By removing these classes, we successfully refined the dataset to include only relevant defect patterns. This refinement was crucial in facilitating more accurate and effective training for our deep learning model. As a result, a total of 785,938 rows were removed from the dataset, while 25,519 rows representing genuine defect patterns were retained.

This meticulous approach in dataset preparation ensures that our deep learning model focuses on authentic defect patterns, ultimately enhancing its ability to detect and classify defects with higher precision and reliability.

2.3.4 Converting Images to RGB Format

In this stage, the wafer map images in the dataset, which are represented in grayscale format, were mapped to three specific values: 0, 127, or 255. This mapping was done according to **Equation 8**, where these values corresponded to the minimum, median, and maximum pixel intensities, respectively.

$$pixel\ value = \begin{cases} 0 \ \rightarrow \ 0 \\ 1 \ \rightarrow 127 \\ 2 \ \rightarrow 255 \end{cases} \tag{8}$$

The reason behind this mapping is that the original wafer map images had pixel values of 0, 1, and 2. Since these values are relatively low, the resulting images would appear predominantly black. By mapping the pixel values to 0, 127, and 255, the images are adjusted to have a wider range of intensities, allowing for better visual representation and analysis where 0 would correspond to pure black, while a value of 255 would represent pure white. The value of 127 would represent a mid-grey tone, which is halfway between black and white.

In the next stage of the process, each grayscale wafer map image was converted to an RGB image by replicating the same intensity value across all three-color channels (red, green, and blue) of the RGB image. The reason for this is that in grayscale images, the intensity value represents the brightness of the pixel, which can be interpreted as the amount of light in the red, green, and blue channels combined. By replicating this value across all three channels in the RGB image, it appears similar to the original grayscale image. This conversion to RGB format allows for better visualization and analysis of the wafer map images, which can aid in identifying and classifying defects.

2.3.5 Image Resizing

The CNN typically require input images to be of a fixed size, and variations in image dimensions can cause issues with model training and performance. The dataset contains variations in wafer map image dimensions up to 632 different dimensions. One approach to dealing with variations in wafer map dimensions is to resize the images to a fixed size before feeding them into the CNN. This can be achieved using image processing techniques such as Bicubic interpolation, which involves using a weighted average of 16 neighbouring pixels to determine the value of each pixel in the resized image. Bicubic interpolation is based on a mathematical algorithm that uses cubic convolution to calculate the new pixel values based on the surrounding pixels [55], [56]. In this stage, each wafer map image is resized to $56 \ x \ 56$. **Figure 18** shows a sample of a resized RGB wafer map image.

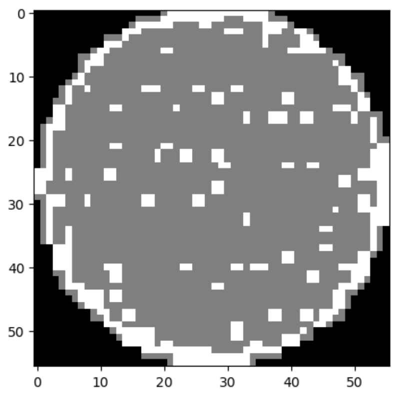


Figure 18: *A close-up view of a resized wafer map.*

2.3.5.1 Data Augmentation Techniques

Data augmentation techniques play a crucial role in machine learning and computer vision tasks by artificially expanding the size and diversity of training datasets. These techniques involve applying diverse transformations and modifications to existing data, resulting in the generation of new augmented samples. The primary objective is to enhance the generalization and performance of CNN models by exposing them to a wider range of variations and scenarios. This, in turn, strengthens their ability to handle unseen data while mitigating the risk of overfitting [57].

The utilization of data augmentation techniques offers several benefits in addressing common challenges such as limited training data, class imbalance, and overfitting [58]. By introducing variations in the input data, these techniques enable models to learn from a more comprehensive set of examples, making them more robust and adaptable to different scenarios. Consequently, the models become better equipped to handle real-world data with diverse characteristics, leading to improved performance and accuracy [59], [60].

In this study, two approaches of data augmentation techniques were utilized to enhance the performance and accuracy of the classification model [61]. The first approach involved increasing the frequency of defect patterns by applying various transformations such as random rotation, random horizontal flip, and random vertical flip. This was done to address the class imbalance in existing defect patterns and improve the model's ability to generalize and perform well on unseen data. By generating additional samples through these transformations, the model was exposed to a wider range of variations and scenarios, enhancing its robustness and reducing the risk of overfitting.

The second approach involved combining different defect patterns to generate mixed types of wafer defects that inherit characteristics from each original pattern or introduce additional features not found individually in any of them. This approach allowed for the creation of new samples that could not be generated through simple transformations, resulting in a more diverse and representative dataset. By training the model on this augmented dataset, it was able to learn more complex patterns and generalize better to real-world data.

Overall, these two approaches of data augmentation techniques proved to be effective in improving the performance and accuracy of the classification model. By increasing the size and diversity of the training dataset, the model was better equipped to handle unseen data and achieve higher accuracy rates [59], [61].

2.3.6 Increase Defect Patterns Frequencies

2.3.6.1 Image Random Rotation

It is a technique that refers to computer vision that involves randomly rotating an image by a certain degree within a specified range. The main steps used to rotate a pixel of an RGB image around its center by a specific radian angle ϑ (each color channel independently) are as follows [62], [63]:

1. Determine the center of the image by the following equations:

$$cx = \frac{the\ width\ of\ orginal\ image}{2}, cy = \frac{the\ height\ of\ orginal\ image}{2} \tag{9}$$

   Where cx and cy represent the x and y coordinates of the image's center, respectively.

2. Convert the rotation angle from degree to radian by the following equations:

$$\vartheta = \vartheta^{\circ} * (\frac{\pi}{180}) \tag{10}$$

   Where ϑ represents the equivalent radian angle of the degree angle ϑ°.

3. Translate the coordinates so that the center of the image is at the origin of the transformation by the following equation:

$$tx = x - cx, ty = y - cy \tag{11}$$

   Where tx and ty represent the x and y of translated coordinates, respectively.

4. Apply the rotation transformation to the translated coordinates by the following equations:

$$rx = tx \times cos(\vartheta) - ty \times sin(\vartheta) \tag{12}$$

$$ry = tx \times sin(\vartheta) + ty \times cos(\vartheta) \tag{13}$$

   Where rx and ry represent the x and y of the rotation coordinates, respectively.

5. Translate the coordinates back to their original position:

$$x' = rx + cx, y' = ry + cy \tag{14}$$

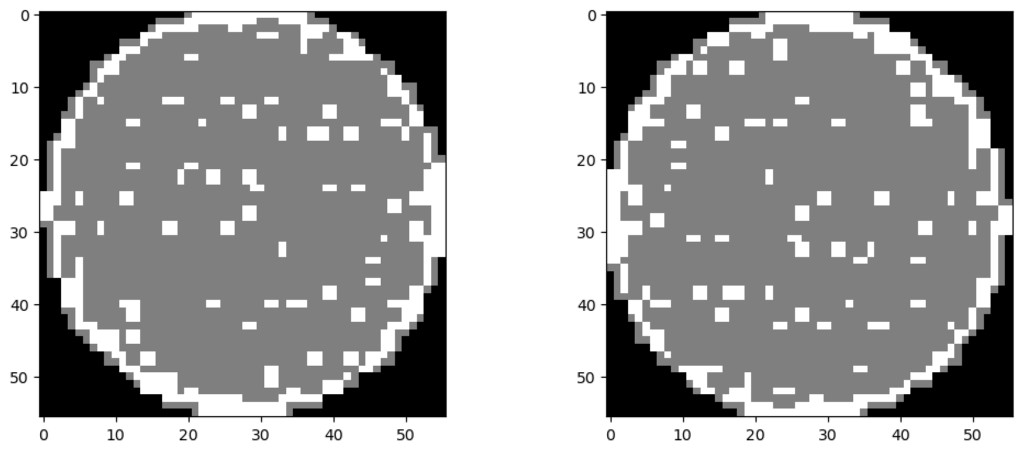
   Where x', y' represents the rotated coordinates of the original position, respectively.

Based on the steps mentioned earlier, **Pseudocode 1** provides a clear outline of the main procedure for rotating an RGB image by a specific angle around its center. The process starts by loading the input image and extracting its pixel values, which are then stored in a 2D array. Each point in the image is represented by coordinates (x, y), indicating the pixel value at that position in the form of an RGB channel vector.

A series of calculations are performed to determine the new position for each pixel to carry out the rotation. This involves several key steps. First, the center of the image is determined. Next, the rotation angle is converted from degrees to radians. The coordinates are then translated to align the center with the origin, which simplifies the subsequent transformation equations. The rotation transformation equations are applied to obtain the new rotated coordinates for each pixel. Finally, the coordinates are translated back to their original position. By following this pseudocode, the RGB image can be effectively rotated around its center by the specified angle. These calculations ensure that each pixel is correctly positioned in the rotated image, allowing for accurate and precise transformations. **Figure 19** shows a sample of the RGB wafer map image and the obtained rotated image by an angle of π.

Pseudocode 1 Pseudocode of RGB image rotation

```
Input: original image $I$ , rotation angle in degrees $\vartheta^{\circ}$
1:  Load the original image $I$ as a 2D array.
2:  Determine the center of $I$ $(cx, cy)$ by the Equation 9.
3:  Convert the rotation angle into a degree $\vartheta^{\circ}$ to radian by the Equation 10.
4:  Create a new blank image $\bar{I}$ to hold the rotated image pixels with the same dimensions
    of $I$.
5:  For x in the range of [0, $I\ width - 1$]:
6:      For y in the range of [0, $I\ height - 1$]:
7:          Translate the coordinates $x, y$ so that the center of the image is at the
            origin by Equation 11 to obtain $tx$ and $ty$.
8:          Apply the rotation transformation by Equations 12 and 13 to the translated
            coordinates $tx, ty$ to obtain $rx$ and $ry$.
9:          Translate the coordinates back to their original position by Equation 14 to
            obtain $x'$ and $y'$.
10:         Set the pixel value of $\bar{I}$ at $(x, y)$ by the pixel value of $I$ at $(x', y')$
11:     End For
12: End For
13: Return $\bar{I}$
```



(A) (B)

***Figure 19**: A comparison of the original RGB wafer image (A) and the rotated RGB image by π (B).*

2.3.6.2 Image Random Horizontal and Vertical Flip

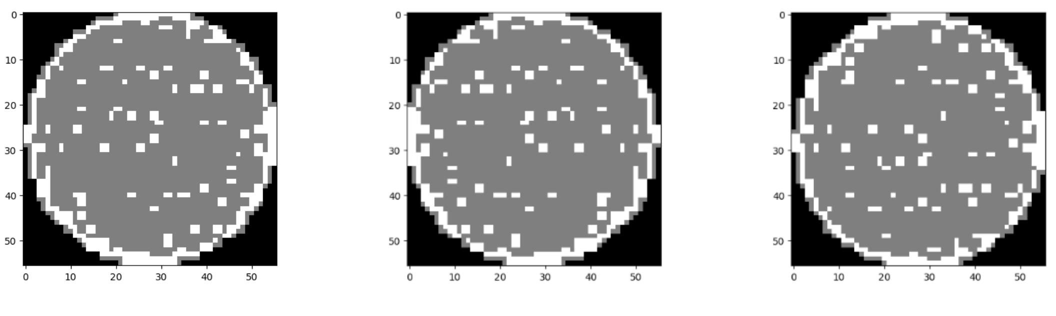
During the data augmentation process, random horizontal and vertical flip transformations were applied to the images. These transformations are applied independently to each pixel in the image, resulting in a horizontally or vertically flipped version of the original image [64]. This approach was used to generate additional training data and increase the diversity of the dataset, which is essential for training deep learning models.

Pseudocode **2** outlines the implementation to flip an RGB image. The pseudocode takes three inputs: the original image, the probability, and the flip type. The algorithm allows for flipping an RGB image horizontally or vertically based on the specified flip type and probability. The resulting flipped image will have the same dimensions as the original image. The probability is used to introduce randomness and variability into the output, which can be useful in generating diverse and realistic results.

Pseudocode 2 Pseudocode of RGB image flipped

```
Input: original image $I$ , probability $p$, Flip type $f_{type}$
1:  Load the original image $I$ as a 2D array.
2:  Create a new blank image $\overline{I}$ to hold the flipped image pixels with the same dimensions
    of $I$.
3:  Generate a random number $r$ in the range of [0,1].
4:  If $r$ is greater than $p$ , then exit, otherwise continue.
5:  If $f_{type}$ equal to Horizontal, then:
6:      For x in the range of [0, $I\ width - 1$]:
7:          For y in the range of [0, $I\ height - 1$]:
8:              Maps the pixel at coordinates (x, y) in $I$ to the pixel at coordinates (x,
                $width$ - y - 1) in $\overline{I}$.
9:          End For
10:     End For
11: If $f_{type}$ equal to Vertical, then:
12:     For x in the range of [0, $I\ width - 1$]:
13:         For y in the range of [0, $I\ height - 1$]:
14:             Maps the pixel at coordinates (x, y) in $I$ to the pixel at coordinates ($height$
                - x – 1, y) in $\overline{I}$.
15:         End For
16:     End For
17: Return $\overline{I}$
```

Figure 20 shows a sample of an RGB wafer map image and the obtained flipped image vertically and horizontally. As can be seen, flipping the image horizontally or vertically can result in a significant change in the appearance of the wafer map. This transformation can help to increase the robustness of the model by exposing it to different variations of the same image.



(A) (B) (C)

***Figure 20**: A comparison of the original RGB wafer image (A) and its vertically (B) and horizontally (C) flipped versions.*

2.3.7 Mixed Defect Patterns Generator

Incorporating multiple defects' patterns is a useful technique for improving the performance and accuracy of deep learning models in real-world scenarios. By combining existing patterns in novel ways, a comprehensive collection of training data is created, enhancing the representation of the various possible patterns that a model may face during practical applications. This can help to improve the model's ability to generalize and perform well on unseen data [45].

The maximum pixel value from all input images at the same position is taken since the pixel value that represents a defect has a maximum value of 255. **Pseudocode 3** presents the pseudocode for a mixed defect patterns generator algorithm. This algorithm takes a list of input images as input and generates a mixed image by setting each pixel value of the mixed image to the maximum pixel values of all input images at each corresponding location. The resulting mixed images will have the same dimensions as the input images and will contain the highest pixel values from each input image.

Pseudocode 3 Pseudocode of mixed defect patterns generator algorithm.

```
Input: input images ($input_{images}$)
1:  Create a new blank image $M$ to hold the mixed image pixels with the same
    dimensions as the first image of $input_{images}$
2:  For x in the range of [0, $M\ width - 1$]:
3:      For y in the range of [0, $M\ height - 1$]:
4:          Obtain the maximum pixel value $pixel_{\max value}$ at $(x, y)$ for all $input_{images}$.
5:          Set $M[x, y]$ to $pixel_{\max value}$
6:      End For
7:  End For
8:  Return $M$
```

By using this approach, a diverse and representative dataset can be created, which is essential for training deep learning models effectively. The resulting mixed defect pattern images can help expose the model to a wider range of variations and scenarios, enhancing its robustness and reducing the risk of overfitting.

This approach allows for the exploration and generation of novel defect patterns that may occur in real-life situations. For instance, by combining two types of defects, new defect patterns can be created, as indicated in **Table 5**. Similarly, the combination of three defects yields additional new defect patterns, as shown in **Table 6**. Furthermore, when four defects are combined, it leads to the creation of even more new defect patterns, as illustrated in **Table 7**.

***Table 4:** Description of mixed defect patterns in level two, detailing the combination of two types of defects found in different regions of the wafer.*

#	Mixed Defect Pattern Name and Symbol	Mixed Defect Pattern Description
1	Center with Edge-Loc (C+EL)	This defect pattern combines defects found in the center region of the wafer with defects located along the edge but not in the very outer rim. It represents defects that occur both at the wafer's center and middle edge.
2	Center with Edge-Ring (C+ER)	This pattern merges defects from the center with those along the very outer rim or edge of the wafer. It indicates issues affecting both the central area and the wafer's perimeter.
3	Center with Loc (C+L)	This combines defects seen in the wafer's center with those located elsewhere but not near the edge. It shows problems in the middle as well as other scattered locations.
4	Center with Scratch (C+S)	This pattern unites defects in the center with scratches or abrasions found elsewhere on the wafer. It points to flaws at the core along with scratch-type defects randomly distributed.
5	Loc with Scratch (L+S)	This merges defects located elsewhere on the wafer with scratches. It represents randomly positioned defects accompanied by scratches in various locations.
6	Donut with Scratch (D+S)	This combines donut-shaped defects with scratches anywhere on the wafer. It signifies flaws forming a donut pattern plus additional scratch.
7	Donut with Edge-Loc (D+EL)	This brings together donut defects with those along the edge but not the outer rim. It shows donut issues as well as defects on the middle edge.
8	Donut with Edge-Ring (D+ER)	This merges donut defects with those along the very outer rim. It points to donut flaws accompanied by problems at the wafer's perimeter.
9	Donut with Loc (D+L)	This combines donut defects with those found elsewhere on the wafer but not near the edge. It represents donut issues together with randomly located defects.
10	Edge-Loc with Loc (EL+L)	This merges defects located along the edge but not the rim with those elsewhere on the wafer. It signifies flaws on the middle edge together with randomly positioned defects.
11	Edge-Loc with Scratch (EL+S)	This combines defects along the edge but not the rim with scratches anywhere on the wafer. It shows flaws on the middle edge co-occurring with scratches.
12	Edge-Ring with Loc (ER+L)	This merges defects along the very outer rim with those elsewhere on the wafer. It represents problems at the perimeter along with randomly located defects.
13	Edge-Ring with Scratch (ER+S)	This combines defects along the outer rim with scratches anywhere on the wafer. It signifies flaws at the edge accompanied by scratches.

Table 5: *Description of mixed defect patterns in level three, detailing the combination of three types of defects found in different regions of the wafer.*

#	Mixed Defect Pattern Name	Mixed Defect Pattern Description
1	Center with Edge-Loc with Scratch (C+EL+S)	This pattern merges all four defect types - defects in the center, along the middle edge, and scratches anywhere on the wafer. It points to issues affecting the core, edge and random locations with scratches.
2	Center with Edge-Ring with Scratch (C+ER+S)	This combines defects in the center, along the outer rim, and scratches anywhere. It represents flaws at the core and perimeter together with scratches across the wafer surface.
3	Center with Edge-Loc with Loc (C+EL+L)	This pattern combines defects in the center region of the wafer with those along the edge and at the location. It represents issues affecting the core, middle edge, and the location.
4	Center with Edge-Ring with Loc (C+ER+L)	This pattern merges defects from the center with those along the outer rim and at the location. It indicates issues affecting the central area, perimeter, and the location.
5	Center with Loc with Scratch (C+L+S)	This pattern unites defects in the center with those at the location and scratches. It points to flaws at the core, the location, and scratch-type defects randomly distributed.
6	Donut with Edge-Loc with Scratch (D+EL+S)	This merges donut defects with those along the edge, at the location, and scratches. It signifies flaws forming a donut pattern, problems at the edge, the location, and additional scratches.
7	Donut with Edge-Ring with Scratch (D+ER+S)	This merges donut defects with those along the outer rim, at the location, and scratches. It represents problems forming a donut pattern, the perimeter, the location, and scratches.
8	Donut with Edge-Loc with Loc (D+EL+L)	This combines donut defects with those along the edge and at the location. It shows donut issues as well as defects on the edge and the location.
9	Donut with Edge-Ring with Loc (D+ER+L)	This merges donut defects with those along the outer rim and at the location. It points to donut flaws accompanied by problems at the edge and the location.
10	Donut with Loc with Scratch (D+L+S)	This combines donut defects with those at the location and scratches. It represents donut issues, the location, and scratches.
11	Edge-Loc with Loc with Scratch (EL+L+S)	This merges defects along the edge, at the location, and scratches. It signifies flaws on the middle edge, the location, and scratches.

Table 6: *Description of Mixed Defect Patterns in Level four, detailing the combination of four types of defects found in different regions of the wafer.*

#	Mixed Defect Pattern Name	Mixed Defect Pattern Description
1	Center with Loc with Edge-Loc with Scratch (C+L+EL+S)	This pattern combines defects in the center region of the wafer with those at the location, along the edge, and scratches. It represents issues affecting the core, the location, the middle edge, and scratches.
2	Center with Loc with Edge-Ring with Scratch (C+L+ER+S)	This pattern merges defects from the center with those at the location, along the outer rim, and scratches. It indicates issues affecting the central area, the location, the perimeter, and scratches.
3	Donut with Loc with Edge-Loc with Scratch (D+L+EL+S)	This merges donut defects with those at the location, along the edge, and scratches. It signifies flaws forming a donut pattern, the location, the middle edge, and additional scratches.
4	Donut with Loc with Edge-Ring with Scratch (D+L+ER+S)	This merges donut defects with those at the location, along the outer rim, and scratches. It represents problems forming a donut pattern, the location, the perimeter, and scratches.

In total, a collection of 28 new mixed defect patterns can be generated, in addition to the 9 single defect patterns. These mixed defect patterns provide a broader representation of the possible defect variations that may be encountered in practical applications. By incorporating these mixed defect patterns into the dataset, the model can be trained to recognize and classify a wider range of defect types and combinations. **Figure 21** showcases samples of these new mixed defect patterns, visually demonstrating the diverse and unique nature of these combined defects. These visual examples help to illustrate the effectiveness of combining different defect types to generate new and realistic defect patterns.

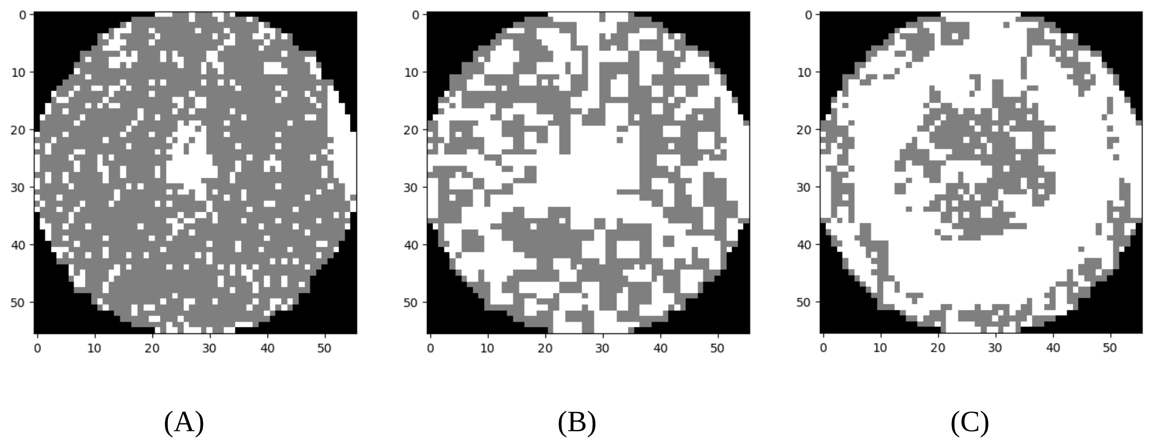


Figure 21: *Examples of Mixed Defect Patterns, including (A) Center with Edge-Loc, (B) Center with Edge-Loc with Scratch, and (C) Donut with Loc with Edge-Loc with Scratch.*

2.3.7 Categorical Encoding Technique

Categorical encoding is a crucial process in CNN for representing categorical data as numerical values. One common method is one-hot encoding, where each category is represented by a binary vector with values of 1 or 0 based on its presence [65]. The dataset contains various defect patterns (classes) such as Loc, Edge-Loc, Center, and others which have been encoded using the one-hot encoding technique outlined in **Pseudocode 4**.

Pseudocode 4 Pseudocode of one-hot encoding algorithm

```
Input: list of unique defects patterns ($classes$)
  1:  Create a new list $L$ that holds binary representation for each class.
  2:  For index equal to 0 and index less than the length of $classes$:
  3:        Create a new vector $V$ of length $classes$ with a value of zeros.
  4:        Set the element at the index of $V$ to 1.
  5:        Append $V$ to $L$.
  6:  End For
  7:  Return $L$
```

After encoding classes using one-hot encoding, each class is represented by a binary vector of length 36. In this representation, the index corresponding to the class has a value of one and all other indices have values of zero. For example: The first class 'Loc' is represented as [1,0,0,...,0,0] and the last class 'ER+S' is represented as [0,0,0,...,0,1].

Chapter Three

Methodology

The proposed methodology is presented in this chapter. It outlines the overall approach taken in the study. The GoogLeNet model is introduced, including its inception modules and architecture. The implementation of the classification framework is explained, along with the application development and deployment process. Hyperparameter tuning and optimization techniques are discussed, as well as the evaluation metrics used.

3.1 The Proposed Methodology

To address the gaps in the literature discussed in Section 2.2 and based on the study aims and objectives discussed in Section 1.3, an automatic wafer defects classification based on the CNN deep learning model is presented. After performing data preprocessing and data augmentation techniques to address data imbalance issues and create new mixed defect patterns for further study. The methodology involves generating a new dataset and exporting it for use by researchers.

Figure **22** demonstrates the process of selecting optimal variables using Optuna to train the proposed model. Various evaluation metrics are employed to assess the performance of the model in both training and testing, aiming to achieve accurate defect type predictions with minimal error. After that, the pre-trained model is deployed and integrated with a proposed automated pattern classification System for wafer defects called “DefectClassifierX”.

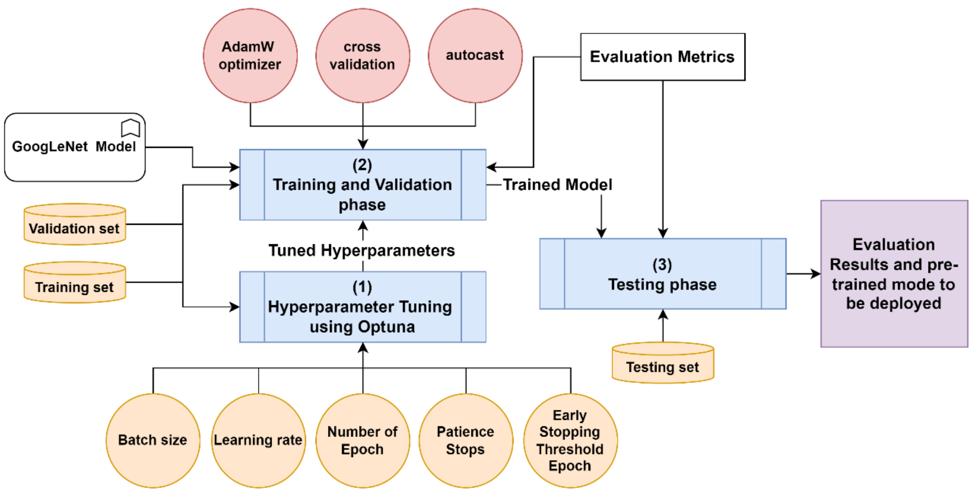


Figure 22: *The Proposed Classification Framework, detailing the structure and workflow of a CNN-based defect classification model, from hyperparameter tuning to final output.*

3.2 GoogLeNet Model

GoogLeNet, developed by Google researchers in 2015, is a deep convolutional neural network architecture specifically designed for image classification [66]. With its original complex structure comprising 22 layers and 9 inception modules, GoogLeNet utilizes multiple convolutional layers of varying filter sizes and pooling operations within these modules. This enables the effective extraction of features at different scales, making it highly effective for image classification tasks.

In addition to its unique design, GoogLeNet also includes auxiliary classifiers as intermediate layers, providing additional supervision during training while addressing the issue of vanishing gradients. This approach helps to improve the accuracy of the model by reducing overfitting and improving generalization [67].

One of the key benefits of using GoogLeNet is its computational efficiency. This is achieved through the use of inception modules, which allow for efficient feature extraction at different scales. Additionally, GoogLeNet can combat the vanishing gradient problem during training, which is a common issue in deep neural networks [68].

However, it's worth noting that the original GoogLeNet architecture was designed to handle images with dimensions of 224 x 224 and 1024 classes. To modify it for use with images with dimensions of 56 x 56 and 36 classes, a new fully connected layer was added at the end of the GoogLeNet layers that maps the 1024 classes to 36 classes. This modification allows GoogLeNet to be used effectively for our scenario.

3.2 Inception Modules

The size of important elements in the image can vary significantly. This variability poses a challenge when selecting an appropriate kernel size for convolution operations. A larger kernel is required to extract information from widely distributed objects in the image, whereas a smaller kernel is preferable for capturing details of less dispersed elements. Expanding the size of neural networks, both in terms of depth and dimensions, is a common approach to improve their efficiency [69].

However, larger network sizes come with risks such as overfitting. Moreover, increasing network size requires more computational resources. GoogLeNet solves these issues by performing convolutions on input from the previous layer with different kernel sizes including 1x1, 3x3 and 5x5 instead of one kernel [70]. For example, for an RGB image with dimensions of 56x56, assuming no padding, the number of operations required to apply a kernel size of 5x5 is $(56 - 5 + 1) \times (56 - 5 + 1) \times (5 \times 5 \times 3) \times (3) = 22{,}702{,}400$ operations. But if a kernel size of 1x1 is applied, the total number of operations for that filter is $(56 \times 56) \times (3 \times 16) = 2{,}985{,}984$. After applying the output of 1x1 kernel to 5x5 kernel, the total number of operations is $2{,}985{,}984 + \left((52x52)x\ (16\ x\ 5\ x\ 5)\right) = 13{,}722{,}624.$ There is a large amount of reduction in computation. **Pseudocode 5** outlines the pseudocode of the inception module implementation where the algorithm takes input parameters as illiterates in **Table 7** and the definition of sub-modules called "branches" are demonstrated in **Table 8**.

Table 7: *Description of Parameters in the Inception Module.*

#	Parameter Name	Parameter Description
1	in_channels	Number of input channels.
2	branch1x1	Number of output channels for the 1x1 convolution in the first branch.
3	branch3x3reduce	Number of output channels for the 1x1 convolution in the second branch (reduction).
4	branch3x3	Number of output channels for the 3x3 convolution in the second branch.
5	branch5x5reduce	Number of output channels for the 1x1 convolution in the third branch (reduction).
6	branch5x5	Number of output channels for the 3x3 convolution in the third branch.
7	branch_pool	Number of output channels for the 1x1 convolution in the fourth branch.

Table 8: *Description of Inception Sub-Modules, detailing the name and functionality.*

#	Branch Name	Branch Definition
1	first branch (branch1)	1x1 convolutional layer that takes the in_channels as input and produces branch1x1 output channels.
2	second branch (branch2)	It is a sequential module that consists of two convolutional layers: 1. A 1x1 convolutional layer with branch3x3reduce output channels 2. A 3x3 convolutional layer with branch3x3 output channels and padding.
3	third branch (branch3)	It is a sequential module similar to branch2, with different channel sizes.
4	fourth branch (branch4)	It is a sequential module that consists of two layers: 1. A max-pooling layer with a kernel size of 3x3 and stride 1, along with padding to maintain the spatial dimensions. 2. A 1x1 convolutional layer with branch_pool output channels.

Pseudocode 5 Pseudocode of Inception module algorithm

Input: input x, in_channels, branch1x1, branch3x3reduce, branch3x3, branch5x5reduce, branch5x5 and branch_pool

```
1: Define branches as demonstrated in Table 2.
2: Pass $x$ through branch1 to obtain branch1 output.
3: Pass $x$ through branch2 to obtain branch2 output.
4: Pass $x$ through branch3 to obtain branch3 output.
5: Pass $x$ through branch4 to obtain branch4 output.
6: Concatenate the outputs of the four branches (branch1, branch2, branch3, and branch4)
   along the channel dimension (dimension 1) to obtain $output$
7: Return $output$
```

3.3 The GoogLeNet Architecture

The GoogLeNet architecture has been modified that consist of 19 layers, including convolutional and max-pooling layers as visualized in **Figure 23**. It also incorporates inception modules, an average pooling layer, a dropout layer, and a linear layer for the final output. The "BasicConv2d" layer is responsible for extracting features through convolutional operations from the input data. Additionally, there's the "MaxPool2d" to reduce spatial dimensions and multi-branch convolutional blocks known as Inception modules that aid in capturing different scales and types of features. Lastly, there's an "AvgPool2d" layer to apply average pooling to the input feature maps before using a dropout preceding the fully connected last layer designed to map input features to output classes while preventing overfitting [46].

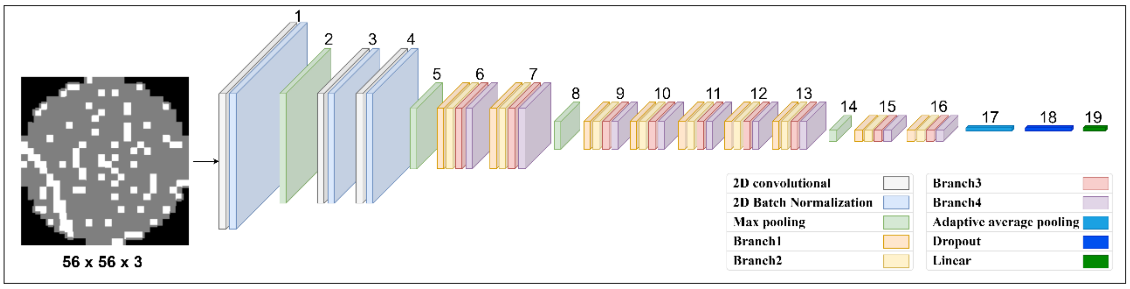


Figure 23: *The visualization of the modified GoogLeNet model architecture.*

The architecture is designed to take an input of shape [512, 3, 56, 56] and produce an output of shape [512, 36] as illustrated in **Table 9**. It's notable that the number 512 here presents the batch size, the number 3 presents the number of input image channels and the numbers 56,56 represent the input image dimensions. All the convolutions, including the convolutions inside the inception module, use rectified linear activation.

Table 9: *Description of layers in the modified GoogLeNet model with input and output shapes.*

#	Layer Name	Input Shape	Output Shape
1	BasicConv2d	[512, 3, 56, 56]	[512, 64, 28, 28]
2	MaxPool2d	[512, 64, 28, 28]	[512, 64, 14, 14]
3	BasicConv2d	[512, 64, 14, 14]	[512, 64, 14, 14]
4	BasicConv2d	[512, 64, 14, 14]	[512, 192, 14, 14]
5	MaxPool2d	[512, 192, 14, 14]	[512, 192, 7, 7]
6	Inception 1	[512, 192, 7, 7]	[512, 256, 7, 7]
7	Inception 2	[512, 256, 7, 7]	[512, 480, 7, 7]
8	MaxPool2d	[512, 480, 7, 7]	[512, 480, 3, 3]
9	Inception 3	[512, 480, 3, 3]	[512, 512, 3, 3]
10	Inception 4	[512, 512, 3, 3]	[512, 512, 3, 3]
11	Inception 5	[512, 512, 3, 3]	[512, 512, 3, 3]
12	Inception 6	[512, 512, 3, 3]	[512, 528, 3, 3]
13	Inception 7	[512, 528, 3, 3]	[512, 832, 3, 3]
14	MaxPool2d	[512, 832, 3, 3]	[512, 832, 2, 2]
15	Inception 8	[512, 832, 2, 2]	[512, 832, 2, 2]
16	Inception 9	[512, 832, 2, 2]	[512, 1024, 2, 2]
17	AvgPool2d	[512, 1024, 2, 2]	[512, 1024, 1, 1]
18	Dropout	[512, 1024]	[512, 1024]
19	Fully connected	[512, 1024]	[512, 36]

The weights of the modified GoogLeNet model will be initialized via loading the pre-trained weights of the original GoogLeNet model that have already captured relevant information from the training data from a dataset called “ImageNet”. The utilization of a pretrained GoogLeNet model and weight initialization from it provides a strong foundation for our deep learning architecture. It leverages the prior knowledge acquired by the pretraining process and allows us to focus on fine-tuning the model to suit our specific task. By incorporating this approach, we aim to enhance the performance and efficiency of our model while reducing the need for extensive training on dataset [71].

3.4 Classification Framework Implementation

The Classification Framework Implementation relies on the PyTorch library for deep learning. PyTorch is an open-source python framework developed by Facebook's AI researchers for machine learning based on the Torch library using Lua language, and it has C++ and Python interfaces. It possesses pretty efficient memory usage, and it is very popular among researchers. PyTorch implements two high-level features: tensor computing with graphical processing unit (GPU) acceleration and deep neural networks based on an automatic differentiation type-based system [72].

PyTorch provides a flexible and efficient way to build and train CNN. The new dataset obtained after data pre-processing and data augmentation is then used to train and validate the model using k-fold cross-validation. The data loader is an important component in the framework as it handles the loading and batching of the dataset for training and validation in each fold. By using data loaders, the framework can efficiently load and pre-process data in parallel, making it easier to train models on large datasets. The data loader also provides other useful functionalities such as shuffling the data, dropping the last batch if it's incomplete, and loading data in a background thread while the model is training [73].

The classification framework consists of three parts: the train function, the valid function, and the main code. These parts use a common variable as illiterates in **Table 10**.

Table 10: *Description of Common Variables in the Classification Framework.*

#	Variable Name and Symbol	Variable Description
1	classes number ($CLASS_{NUM}$)	It defines the number of defect pattern classes
2	batch size ($BATCH_{SIZE}$)	It defines the number of samples that are processed in each iteration during training
3	Device (DEVICE)	It defines which device to use (CPU or GPU).
4	Optimizer (O)	It defines the optimizer used for updating the model's parameters during training.
5	loss function (F)	It defines the loss function used for the training model
6	number of folds (n_{splits})	It defines the number of subsets into which a dataset is divided during cross-validation

The train function, also known as *train_epoch*, is a crucial component of the deep learning model training process. Its main responsibility is to train the model for one epoch, which involves iterating through the dataset and updating the model's parameters based on the computed loss. The *train_epoch* function takes in several arguments, including the model, *dataloader*, loss function, and optimizer. Once these arguments are passed in, the function sets the model to training mode and begins iterating through the *dataloader*. For each batch of images and labels, the optimizer gradients are zeroed out to prevent any interference from previous iterations. The output from the model is then computed, and the loss is calculated using the specified loss function. To perform backpropagation with mixed precision training, PyTorch 'autocast ()' is used. This approach allows for faster training times and more efficient use of GPU resources [74].

Finally, the function returns the total training loss and the number of correct predictions. These metrics are crucial for evaluating the performance of the model during training and making any necessary adjustments to improve accuracy. The pseudocode for the *train_epoch* function is shown in **Pseudocode 6**.

Pseudocode 6 Pseudocode of *train_epoch* function

```
Input: model $M$, data loader $D$, classes numbers $CLASS_{NUM}$, batch size $BATCH_{SIZE}$,
$DEVICE$, optimizer $O$ and loss function $F$.
1:  Initialize $train_{loss}$ and $train_{correct}$ variables to keep track of the training loss
    and the number of correct predictions.
2:  Set $M$ to training mode.
3:  For each input images and labels in $D$, do the following:
3:      Move images and labels to $DEVICE$ memory.
4:      Reshape the labels to shape of $[BATCH_{SIZE}, CLASS_{NUM}]$
5:      set all the gradients of the optimizer to zero.
6:      Use autocast to perform automatic mixed precision training on the device
        specified by $DEVICE$.
7:      Pass the input images through the $M$ to obtain the $output$ predictions.
8:      Convert the labels to class indices using argmax function.
9:      Compute the loss between the predictions and the labels using $F$.
10:     Scale the loss value using scaler to take advantage of mixed precision
        training.
11:     Update the $O$ parameters using the gradients computed during the backward
        pass.
12:     Compute the batch loss by multiplying the loss value with the number of
        images in the batch and add it to $train_{loss}$.
13:     Compute the predicted class labels by finding the indices of the maximum
        values in each output prediction.
14:     Compute the number of correct predictions by comparing the predicted
        class labels with the labels and summing up the correct matches.
15:     Update the scaler's state to maintain the correct scaling factor for future
        backward passes.
16: End For each
17: Return $train_{loss}$ and $train_{correct}$
```

The valid function, also known as *valid_epoch*, plays a crucial role in evaluating the performance of the trained model on a validation dataset for one epoch. It takes in the model, *dataloader*, and loss function as arguments. Inside the *valid_epoch* function, the model is set to evaluation mode to ensure that no gradients are computed during inference. It then iterates through the *dataloader*, processing each batch of images and labels. For each batch, the output from the model is computed by passing the images through the model's forward pass. The loss is then calculated using the specified loss function, comparing the predicted output with the ground truth labels.

The function keeps track of the total validation loss and the number of correct predictions made by the model. These metrics are important for evaluating the model's performance on the validation dataset. After iterating through all the batches in the data loader, the *valid_epoch* function returns the total validation loss and the number of correct predictions. The pseudocode for the *valid_epoch* function is illustrated in **Pseudocode 7**.

Pseudocode 7 Pseudocode of valid_epoch function

```
Input: model $M$, data loader $D$, classes numbers $CLASS_{NUM}$, batch size $BATCH_{SIZE}$,
$DEVICE$, optimizer $O$ and loss function $F$.
  1:  Initialize $valid\_loss$ and $valid_{correct}$ variables to keep track of the validation
      loss and the number of correct predictions.
  2:  Set $M$ to evaluation mode.
  3:  For each input images and labels in $D$, do the following:
  3:      Move images and labels to $DEVICE$ memory.
  4:      Reshape the labels to shape of $[BATCH_{SIZE}, CLASS_{NUM}]$
  5:      Pass the input images through the $M$ to obtain the $output$ predictions.
  6:      Convert the labels to class indices using argmax function.
  7:      Compute the loss between the predictions and the labels using $F$.
  8:      Compute the batch loss by multiplying the loss value with the number of
          images in the batch and add it to $valid_{loss}$.
  9:      Compute the predicted class labels by finding the indices of the maximum
          values in each output prediction.
 10:      Compute the number of correct predictions by comparing the predicted
          class labels with the labels and summing up the correct matches.
 11:  End For each
 12:  Return $valid_{loss}$ and $valid_{correct}$
```

The main code for model training performs k-fold cross-validation using a loop that iterates through each fold. For each fold, a train sampler and test sampler are created using *SubsetRandomSampler* to split the dataset into training and validation sets. Then, two data loaders are created for training and validation using these samplers. A dictionary called history is initialized to keep track of the training and validation loss and accuracy for each epoch. This dictionary stores the metrics for each epoch, allowing for easy tracking of model performance over time. Early stopping is implemented in the form of a patience parameter to prevent overfitting. This parameter determines the number of epochs to wait before stopping training if the validation loss does not improve. This approach helps to prevent the model from overfitting to the training data and improves generalization [75].

The function then trains and validates the model for a specified number of epochs using the train and valid functions as described above respectively. Finally, the performance of each fold is stored in a dictionary called "foldperf". This dictionary contains the performance metrics for each fold, allowing for easy comparison of model performance across different folds. The pseudocode for the main code function is illustrated in **Pseudocode 8**.

Pseudocode 8 Pseudocode of main code

```
Input: number of folds to split $n_{splits}$, data loader $D$, classes numbers $CLASS_{NUM}$,
batch size $BATCH_{SIZE}$, $DEVICE$, optimizer $O$ and loss function $F$.
1:  Initializes an $KFold$ object with a number of folds to split of $n_{splits}$, randomly
    shuffling the data before splitting and a random seed of 42.
2:  For each fold $K$ in $KFold$, do the following:
3:      Obtain $train_{idx}$ and $valid_{idx}$ from $K$ which contains the indices of
        splitting data for training and validation.
4:      Create a new sampler[8] $train_{sampler}$ for training data using $train_{idx}$
        indices.
5:      Create a new data loader $train_{loader}$ for training data using $train_{sampler}$
        and $BATCH_{SIZE}$.
6:      Create a new sampler $valid_{sampler}$ for validation data using $valid_{idx}$
        indices.
7:      Create a new data loader $valid_{loader}$ for validation data using $valid_{sampler}$
        and $BATCH_{SIZE}$.
8:      Initialize an empty dictionary $history$ to store the training and validation
        losses and accuracies.
9:      Initialize variables $best_{loss}$, $patience$, and $counter$ for early stopping.
3:      Create a new instance of GoogLeNet model $M$, AdamW optimizer $O$ with a
        learning rate of $LEARNING_{RATE}$, loss function $L$ as cross-entropy loss and
        grad scaler $S$.
4:      For each epoch $E$ in the range of $NUM_{EPOCH}$, do the following:
7:          Call train_epoch function with $train_{sampler}$ to obtain $train_{loss}$,
            $train_{correct}$.
```

[8] A sampler is an object that specifies the strategy for sampling data from a dataset during training or evaluation. It determines the order in which the samples are accessed and fed into the model.

```
            Call valid_epoch function with $valid_{sampler}$ to obtain $valid_{loss}$, $valid_{correct}$.
            Compute the epoch training loss $E_{train\_loss}$ by $\frac{train_{loss}}{lenght\ of\ train_{sampler}}$.
8:          Compute the epoch validation loss $E_{valid\_loss}$ by $\frac{valid_{loss}}{lenght\ of\ valid_{sampler}}$.
9:          Compute the epoch training accuracy $E_{train\_acc}$ by $\frac{train_{correct}}{lenght\ of\ train_{sampler}} * 100$.
12:         Compute the epoch validation accuracy $E_{valid\_acc}$ by $\frac{valid_{correct}}{lenght\ of\ valid_{sampler}} * 100$.
13:         Append $E_{train\_loss}$, $E_{valid_{loss}}$, $E_{train\_acc}$ and $E_{valid\_acc}$ to $history$.
14:         If the number of $E \geq num\_epochs_{threshold}$, then:
                If the $valid_{loss} < best_{loss}$then:
                    Set $best_{loss}$ to $valid_{loss}$ and reset $counter$ to zero.
                    Otherwise increment $counter$ by one.
                End if
            End if
            If the $counter \geq patience$ then break.
        Append history to foldperf.
16: End For each
17: **Return** foldperf.
```

3.5 Application Development and Deployment

Overall, the deployment of a CNN model enables the practical application of deep learning techniques in various domains, providing real-time predictions, automation, scalability, and improved decision-making capabilities. The deployment process involves integrating the CNN model into an application or system where it can be utilized to perform tasks such as wafer defect patterns classification or any other relevant task for which the model was designed. An automated pattern classification System for wafer defects called “DefectClassifierX” is present which is a cross-platform application for automating wafer defect pattern classification. DefectClassifierX interface includes a range of elements and features that facilitate the classification process and enhance the user experience whereas **Figure 24** shows the “welcome” screen of the proposed application. Some of these elements may include:

1. **Intuitive interface:** The user interface is easy to use and provides a seamless experience for users. It included simple and clean designs, organizing elements logically and clearly.
2. **Classification tool:** The application provides a user-friendly tool for defect classification. These tools may include buttons or selection tools to specify the type of defect.
3. **Information presentation:** The application displays detailed information about the classification process, such as the types of defects and any relevant statistics.
4. **Reporting functionality:** The application offers the capability to generate a report summarizing the classification process and the ability to download it.

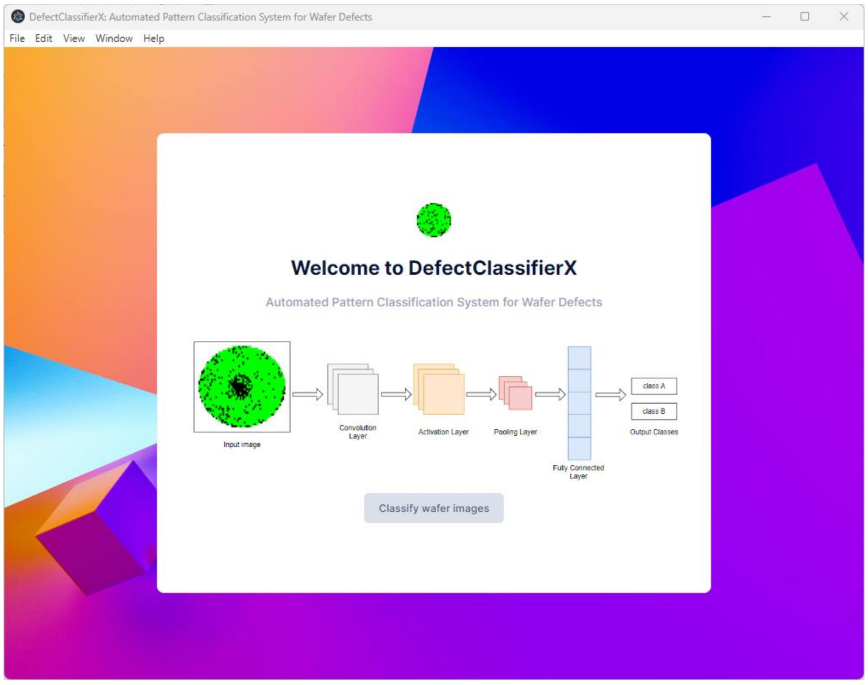

DefectClassifierX: Automated Pattern Classification System for Wafer Defects
File Edit View Window Help
Welcome to DefectClassifierX
Automated Pattern Classification System for Wafer Defects
Input image
Convolution Layer
Activation Layer
Pooling Layer
Fully Connected Layer
class A
class B
Output Classes
Classify wafer images

Figure 24: *"Welcome" screen of the DefectClassifierX application.*

The DefectClassifierX is implemented on top of a JavaScript framework called “Electronjs” used for developing a cross-platform desktop application that can run on Windows, macOS, and Linux operating systems using web technologies such as HTML, CSS, and JavaScript [76]. It combines the Chromium[9] rendering engine and Node.js runtime to provide a platform for creating native-like desktop applications.

3.6 DefectClassifierX Components

The DefectClassifierX consists of five built-in modules and two modules that run on a separate python server as illustrated in **Figure 25.**

The built-in modules in DefectClassifierX are as follows:

1. **The configuration module:** it reads the configuration file provided by the administrator which contains application parameters, such as the predefined classes, restful port, connection protocol, and all Uniform Resource Locators (URLs) that serve all the functionality of the application in a JavaScript Object Notation (JSON) format. This service is used by all other services to perform predefined tasks and provide the user with the ability to update the configuration parameters.
2. **The inputs handler module:** it **is** responsible for handling user input images via the native NodeJS file services to read files and extract the data from them.
3. **The JSON formator module: is** responsible for converting data representation for JSON format to be handled via the Python server.
4. **The report generator module:** it responsible for generating a report in XLS format for the classification result.
5. **The communication module:** its responsible for mange the communication between the DefectClassifierX and the python server by making a Hypertext Transfer Protocol (HTTP) request to send or fetch data from the server.

[9] Chromium is an open-source web browser project that serves as the foundation for many popular browsers, including Google Chrome.

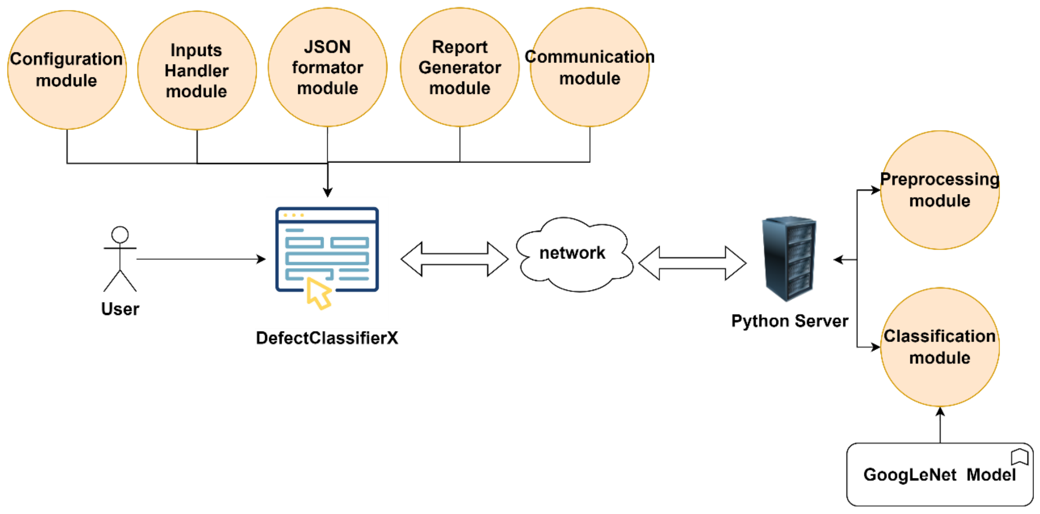


Figure 25: *The main components of DefectClassifierX.*

The python server is implemented using a "Flask" framework which provides the necessary tools for building web applications and application programming interfaces (APIs). The python server consists of two modules as follows:

1. **The Classification module:** it is responsible for making classification using the input wafer map images via a trained CNN model.
2. **The Pre-processing module:** it is responsible for making pre-processing using the input wafer map images to prepare the data for classification.

Typically, Flask uses the hypertext transfer protocol (HTTP) to allow clients to communicate with a server and request data or perform actions. The logic beyond Flask is to define routes using the decorator "@app.route()" which defines the URLs that the server will respond to. Within each route, there is a view function that handles the incoming requests related to that route [77]. The server contains two main routes as illustrated in **Table 11**. The pre-processing route which takes the wafer map images as input, performs the pre-processing steps as explained in **Section 3.2,** saves the output of pre-processing steps and returns these pre-processing images as response in JSON format. The classifying route takes the output of the pre-processing route as input and then performs a classification using the trained CNN model and returns the result as a list of input images name with its defect class in JSON format.

Table 11: *DefectClassifierX Python server APIs.*

#	URL	HTTP request method	Request data	Response data
1	pre-processing	POST	Wafer map images	Wafer map images after pre-processing
2	classifying	GET	none	Classification results

2.3.8 DefectClassifierX Workflow

All modules in the DefectClassifierX application work harmoniously. **Figure 26** illustrates the flow of the starting point of DefectClassifierX to perform a classification task.

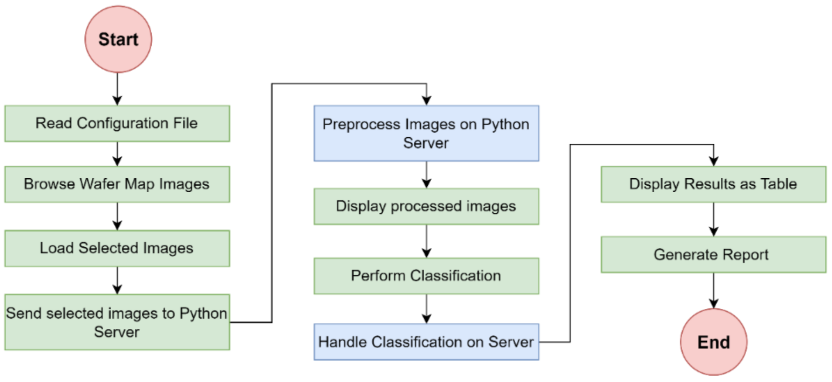


Figure 26: *The flow chart of DefectClassifierX Workflow.*

The main steps that the user must perform to classify defect patterns in wafer map images are as follows:

1. **Read Configuration File:** At the beginning of the DefectClassifierX, the DefectClassifierX reads the configuration file to retrieve its parameters via the Configuration module such as routes URLs for server, server IP address and port.
2. **Browse and load Wafer Map Images:** The DefectClassifierX provide the user interface where the user can browse and select wafer map images from their local computer via the Inputs Handler module as shown in **Figure 27**. This module relies on the “FileReader” API in JavaScript that allows the user to select image files. When

the user clicks the "load images" button and files are selected, the change event is triggered, and the selected files are accessed through “event.target.files”. Then these files' content is extracted such as the number of selected images, extensions, images dimensions and display a sample from these selected files [78].

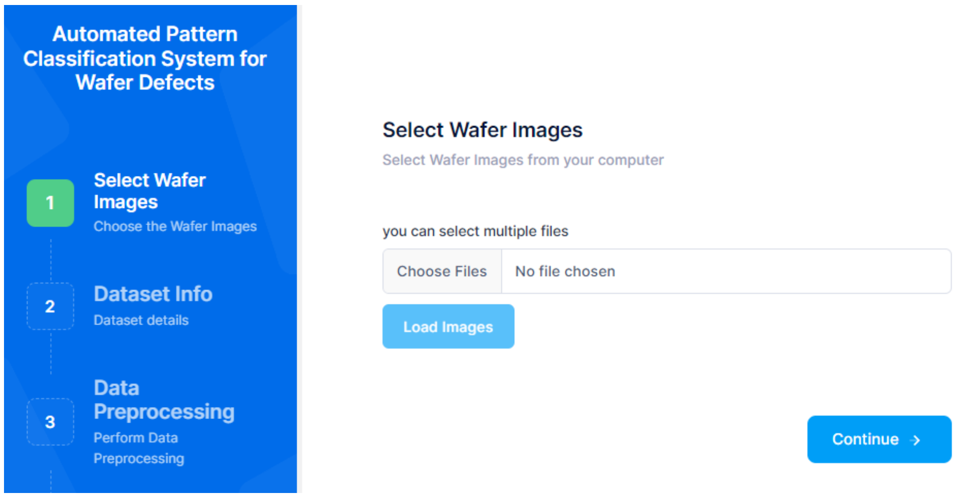


Figure 27: *The “select wafer images” page in DefectClassifierX.*

3. **Send selected images to Python Server:** when the user clicks the “Perform Data Preprocessing” button, the DefectClassifierX send a POST request to the Python server, including the selected images as the request payload via the Communication module in JSON format.
4. **Preprocess Images on Python Server:** The Python server receives the POST request and preprocesses the wafer map images as required via the Preprocessing module. After preprocessing, the server saves these processed images and sends them back as a response from the Python server as URLs pointing to the location of the processed images on the server.
5. **Display processed images:** After retrieving a response from the server, the DefectClassifierX displays samples of processed images as shown in **Figure 28**.

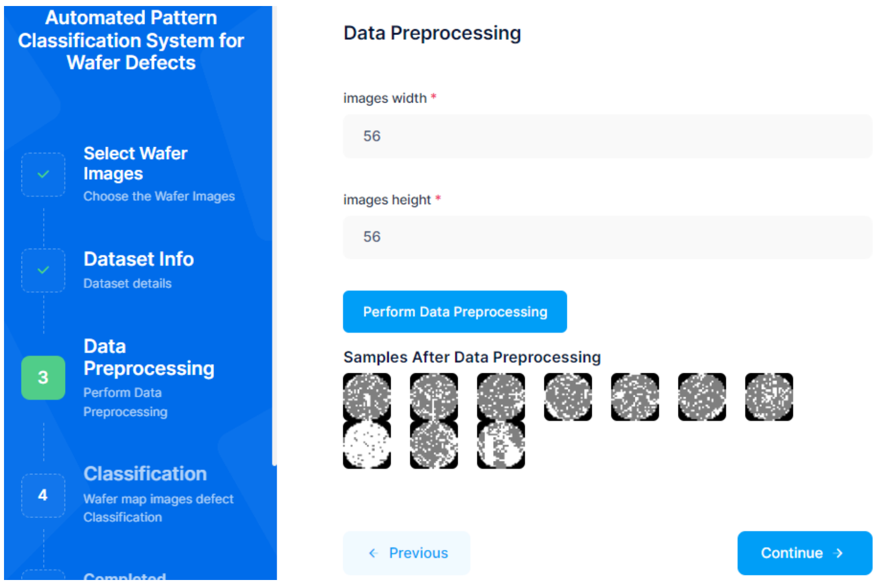

Automated Pattern Classification System for Wafer Defects
Select Wafer Images
Choose the Wafer Images
Dataset Info
Dataset details
3
Data Preprocessing
Perform Data Preprocessing
4
Classification
Wafer map images defect Classification
Completed
Data Preprocessing
images width *
56
images height *
56
Perform Data Preprocessing
Samples After Data Preprocessing
Previous
Continue

Figure 28: *The "data preprocessing" page in DefectClassifierX.*

6. **Perform Classification:** When the user clicks the "Perform Classification" button in the application as shown in **Figure 29**, the DefectClassifierX send a GET request to the server, indicating that a classification task should be performed.

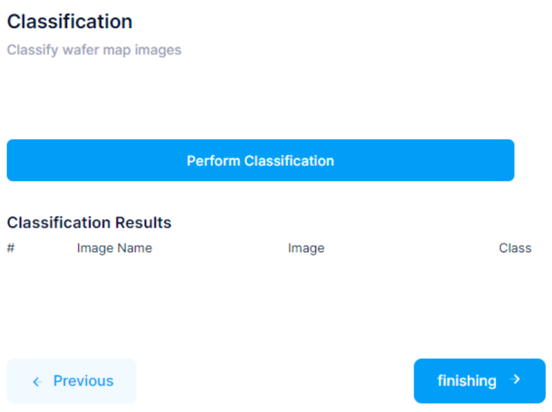
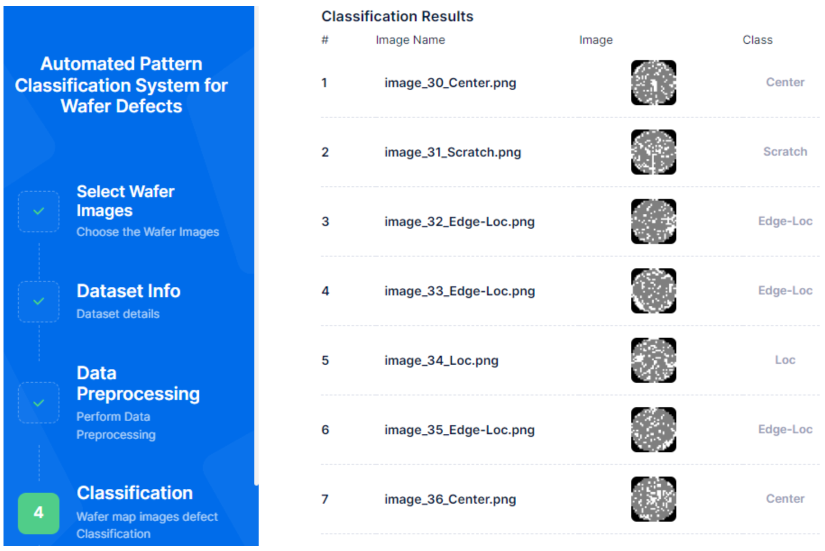


Figure 29: *The "classification" page in DefectClassifierX.*

7. **Handle Classification on Server:** On the server, receive the GET request for classification and perform the necessary on the preprocessed wafer map images and send the classification results back to the DefectClassifierX as a response from the server.

8. **Display Results as Table:** In the application, parse and process the classification results received from the server. Display them in a table format on the user interface for easy viewing and analysis as shown in **Figure 30**.

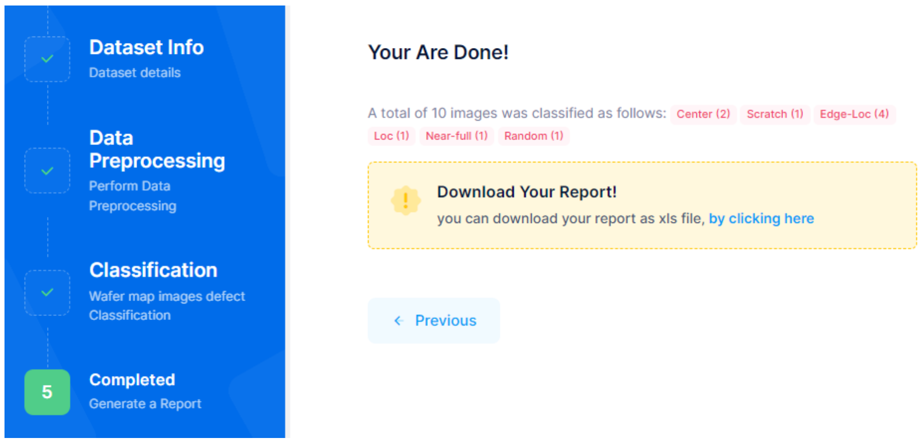

Automated Pattern Classification System for Wafer Defects
Select Wafer Images
Choose the Wafer Images
Dataset Info
Dataset details
Data Preprocessing
Perform Data Preprocessing
4
Classification
Wafer map images defect Classification

Classification Results

#	Image Name	Image	Class
1	image_30_Center.png		Center
2	image_31_Scratch.png		Scratch
3	image_32_Edge-Loc.png		Edge-Loc
4	image_33_Edge-Loc.png		Edge-Loc
5	image_34_Loc.png		Loc
6	image_35_Edge-Loc.png		Edge-Loc
7	image_36_Center.png		Center

Figure 30: *The "classification results" page in DefectClassifierX.*

9. **Generate Report and Download as XLS File**: Provide an option for the user to generate a report based on the classification results and download the generated XLS file as shown in **Figure 31**.



Your Are Done!

A total of 10 images was classified as follows: Center (2) Scratch (1) Edge-Loc (4) Loc (1) Near-full (1) Random (1)

Download Your Report!
you can download your report as xls file, by clicking here

Figure 31: *The "Report generator" page in DefectClassifierX.*

Hyperparameter Tuning and Optimization Technique

Hyperparameters are parameters that govern the learning process and dictate the values of model parameters acquired by a learning algorithm. The use of the prefix 'hyper_' indicates their significance as overarching parameters in determining both the learning process and resulting model parameters [79]. Hyperparameters are settings or configurations that are not learned from the data but are set before training the model such as batch size, learning rate, number of epochs and patience stop as illustrated in **Table 12**. Optuna which is an open-source hyperparameter optimization framework is selected to find the optimal hyperparameters which have a significant impact on the model's performance[80].

Table 12: *The selected hyperparameters for tunning with their descriptions.*

#	Hyper Parameter Name	Hyper Parameter Symbol	Hyper Parameter Description
1	Batch size	batch_size	The batch size refers to the number of training examples that are used in both the forward and backward passes of a neural network during its training phase.
2	Learning rate	learning_rate	Determines the step size at which the optimizer adjusts the weights of the model during training.
3	Number of Epochs	num_epochs	The epoch refers to one complete pass through the entire training dataset during the training process.
4	Patience Stops	patience	Number of epochs without improvements to wait before early stopping
5	Early Stopping Threshold Epoch	num_epochs_threshold	The minimum number of epochs that must be completed before early stopping is checked.

The Optuna employs the Bayesian Optimization Algorithm to search for the global maximum or minimum solution of a scalar objective function within a bounded domain. It works by modelling and specifying the distribution of the objective function. Optuna is based on the Bayesian Optimization Algorithm which is used to find the global maximum or minimum solution of a scaler objective function in a bounded domain ($f: \mathbb{R}^d \rightarrow \mathbb{R}$) [81]. It aims to model the objective function f to specify its distribution. For a set of points $x \in \mathbb{R}^d$, the evaluation of membership of the objective function f is calculated by the following equation [82]:

$$x_{new} = max_{x \in \mathbb{R}^d} f(x) \tag{12}$$

The search space for the hyperparameters as listed in Table 12 including their corresponding suggestion methods and values are listed in **Table 13** [80], [83].

Table 13: *The search space for the selected hyperparameters.*

#	Hyper Parameter Name	Hyper Parameter Method	Hyper Parameter Values
1	Batch size	List of categorical	[16, 32, 64,128, 256, 512]
2	Learning rate	Log-uniform distribution	[1e-4, 1e-1]
3	Number of Epochs	Integer values within a specified range	[10, 100]
4	Patience Stops	Integer values within a specified range	[5, 20]
5	Early Stopping Threshold Epoch	Integer values within a specified range	[10, 100]

Optimizers play a crucial role in adjusting the weights and learning rate of a model to minimize the error function or maximize production efficiency. One specific type is Gradient Descent, an iterative technique that modifies parameters to reduce a given convex function [84]. It achieves this by moving the step determined by the learning rate in the opposite direction of the steepest ascent, utilizing derivatives to locate minima [85]. Optimizers rely on model-specific parameters such as weights and biases. An example optimizer is Adam with Weight Decay Regularization (AdamW), which was introduced in 2019 as a modification of the Adaptive Moment Estimation (Adam) algorithm that aims to improve the weight decay behavior of Adam. AdamW incorporates adaptive learning rates from its predecessor and adds weight decay regularization into its framework. Ultimately, employing the AdamW optimizer aims at minimizing the cost function value as much as possible [86].

In the Adam optimization algorithm, weight decay is commonly implemented by introducing a penalization factor in the loss function to encourage smaller weights. Nevertheless, this approach may result in suboptimal outcomes as the penalty affects both weights and adaptive learning rates. To address this limitation, AdamW integrates weight decay directly into the optimization process. Rather than adding a penalty term to the loss function, AdamW applies a decay term exclusively to the gradients during each training iteration [68]. This selective application of decay only impacts weights and not adaptive learning rates, resulting in enhanced performance for generalization and improved convergence properties. The weights θ decay is calculated by the following equation [86]:

$$\theta_{t+1} = (1-\lambda)\theta_t - \alpha\nabla f_t(\theta_t) \tag{13}$$

Where λ presents the rate of the weight decay/step, α presents the learning rate and $\nabla f_t(\theta_t)$ present the batch gradient of t.

Evaluation Metrics

Various evolutionary metrics were employed to assess the effectiveness of the proposed approach, encompassing confusion matrix, accuracy, F1 score, precision, recall and Receiver Operating Characteristic (ROC).

The utilization of the confusion matrix aids in evaluating the classification performance of the CNN model by contrasting true and predicted values. Within this matrix, rows correspond to true values while columns represent predicted values. The outcomes derived from this assessment yield four possibilities: true positive, false positive, true negative and false negative [87]. **Table 14** provides a list of four key terms and their descriptions.

Table 14: *Key Terms for Classification Model Evaluation: Definitions and Descriptions of True Positive, False Positive, True Negative, and False Negative.*

#	Name	Description
1	True Positive (TP)	The model correctly predicted the positive class.
2	False Positive (FP)	The model incorrectly predicted the positive class.
3	True Negative (TN)	The model correctly predicted the negative class.

4	False Negative (FN)	The model incorrectly predicted the negative class.

Accuracy refers to the extent to which the model effectively categorizes all instances within a dataset. It is computed by dividing the total number of correct predictions by the overall number of predictions made [88]. The accuracy is calculated by the following equation:

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{14}$$

In this context, TP represents true positives, FP represents false positives, TN represents true negatives, and FN represents false negatives.

The F1 score is a quantitative indication of the equilibrium between precision and recall. It is determined by computing the harmonic mean of precision and recall. Precision measures the proportion of correctly predicted positive instances out of all predicted positive instances. This measure is calculated by dividing true positives by the sum of true positives and false positives. On the other hand, recall gauges how many actual positive instances are accurately identified as positive. It can be obtained by dividing true positives by the sum of true positives and false negatives [88]. The F1 score is calculated by the following equation:

$$F1\,score = 2 * \frac{(precision * recall)}{(precision + recall)} \tag{15}$$

The Receiver Operating Characteristic (ROC) curve is a metric used to evaluate the performance of a CNN model in class discrimination. It measures the ability of the model to accurately classify different classes by analysing true positive rate (TPR) and false positive rate (FPR) across various threshold values for classification. It plots the TPR against the FPR at various threshold settings [89].

The formulas for TPR and FPR are as follows:

$$TPR = \frac{TP}{(TP + FN)} \tag{16}$$

$$FPR = \frac{FP}{(FP + FN)} \tag{17}$$

The shape and position of the ROC curve can give insights into the model's performance [90]:

1. The closer the ROC curve is to the top-left corner of the plot, the better the model's performance. This indicates a higher TPR and lower FPR, meaning the model can accurately classify positive cases while minimizing false positives.
2. A ROC curve that is close to the diagonal line (connecting the bottom-left to the top-right corners) suggests that the model's performance is no better than random guessing.
3. If the ROC curve falls below the diagonal line, it suggests that the model's performance is worse than random guessing. This indicates the model is performing poorly in distinguishing between positive and negative cases.

Chapter Four

Findings and Discussions

This chapter focuses on the findings of the study and provides a detailed discussion. It begins by describing the development tools and system requirements used. The experimental setup is explained, followed by the presentation of experimental training results. Model evaluation is conducted, and any memory limit issues encountered during experiments are addressed. Finally, a discussion of the results is presented, comparing them with past research.

4.1 Development Tools and System Requirements

The proposed CNN model for classifying different types of defects across the wafer map was implemented using a Python environment. There are many Python libraries used for the development of this solution as illustrated in **Table 15**.

Table 15: *Python Libraries and Frameworks used in classification framework: Description and Version Information.*

#	Library Name	Library Description	Library-Version
	Pandas	An open-source Python library is utilized to analyse and manipulate data, specifically tabular data, particularly data frames, which are similar to spreadsheets in MS Excel [91].	2.1.1
1	Numpy	An open-source Python library for numerical computation, it offers functionalities that are both efficient and convenient for performing mathematical and logical operations on data arrays [92].	1.24.3
2	Pillow	It's an open-source python library that is used to manipulate matrices. it provides all the necessary functions for matrix processing [93]	9.4.0
3	Matplotlib	It's a cross-platform python library that is used to visualise data in graphical form and provide an interactive visualization in a python environment [94]	3.7.1
4	Torch	It's an open-source python framework that is used in scientific computation and provides all necessary algorithms for deep learning. It was developed by Ronan Collobert, Samy Bengio and Johnny Mariéthoz [95]	2.0.1
5	PyTorch	It's an open-source python library that is used to develop deep learning models based on neural networks introduced by Facebook. It allows to performs of dynamic computation on the central processing unit (CPU) and graphic processing unit (GPU). PyTorch is based on the Torch framework [72]	2.0.1

6	CUDA	It's a parallel computing environment developed by NVIDIA that is used to perform different types of computations on a graphic processing unit (GPU) [96], [97]	11.8
7	Torchvision	Torchvision is a component of the PyTorch library that offers various utilities and datasets for computer vision. It provides multiple functionalities, including image transformation, data loading, and pre-trained models for tasks such as image classification, object detection, and segmentation.[98]	0.15.2

Also, the proposed DefectClassifierX application was implemented using python and NodeJS environments. **Table 16** showcases key libraries and their respective descriptions and versions. It includes Node.js, Electronjs, Flask, and xlsx.js.

Table 16: *Libraries and Frameworks used in DefectClassifierX development: Description and Version Information.*

#	Library Name	Library Description	Library-Version
1	Node.js[10]	Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to run JavaScript code outside of a web browser. It is built on the V8 JavaScript engine used by Google Chrome and provides an event-driven, non-blocking I/O model that makes it lightweight and efficient.	18.16.1
2	Electronjs[11]	is an open-source framework that allows developers to build cross-platform desktop applications using web technologies such as HTML, CSS, and JavaScript.	26.2.4
3	Flask[12]	Flask is a lightweight and flexible web framework for building web applications using the Python programming language.	2.2.2
4	xlsx.js[13]	xlsx.js is a JavaScript library for reading and writing Microsoft Excel files in the xlsx file format. It allows developers to create, modify, and parse Excel files directly from their web applications using JavaScript.	0.18.5

We carried out all experiments including development, training and testing on an NVIDIA GeForce RTX 3080 GPU with AMD Ryzen 9 5900HX CPU, 32 GB RAM and windows 11 operating system.

[10] Official website: https://nodejs.org/en/
[11] Official website: https://www.electronjs.org/
[12] Official website: https://flask.palletsprojects.com/en/2.1.x/
[13] Official website: https://oss.sheetjs.com/js-xlsx/

4.2 Experimental Setup

The raw data set underwent cleaning, pre-processing, and augmentation by the theoretical principles outlined in **Sections 3.2** and **3.3** to ready it for training of the CNN model. The quantity of wafer map images was expanded to a total of 368,568 images. Each defect pattern constitutes approximately 3% of the dataset's entirety out of these images, encompassing a combined total of 36 unique defect patterns for both single and mixed types. Furthermore, an approach utilizing one-hot encoding was employed to encode the defect patterns as discussed in **Section 3.4.**

Through these efforts, we successfully created a new dataset called "WM-300K+ wafer map [Single & Mixed]"[14] for single and mixed types of defect patterns, ranging up to four levels and published it via the Kaggle website. This enriched dataset will be published on Kaggle, providing researchers with a valuable resource for further research and study in the field of wafer defect pattern classification. By making this dataset available, we hope to foster collaboration, encourage new insights, and advance the state-of-the-art in this domain.

The tuned hyperparameters including batch size, learning rate, number of epochs, patience stops and early stopping threshold epoch were selected using Optuna as discussed in **Section 4.4**. **Table 17** demonstrates the values of the optimal hyperparameters that were used during the training phase.

Table 17: *Tunned Hyperparameter optimal values.*

#	Hyper Parameter Name	Hyper Parameter Value
1	Batch size	512
2	Learning rate	0.0008
3	Number of Epochs	100

[14] The "WM-300K+ wafer map [Single & Mixed]" dataset can be accessed via this link. The dataset contains up to 368,568 images for 36 (single and mixed) defect patterns.

4	Patience Stops	5
5	Early Stopping Threshold Epoch	15

The dataset was split into 80% for training and 20% for testing. Subsequently, the training set was further divided into 70% for training and 30% for validation. This division allowed for training the model on a subset of the data while validating its performance on another subset as shown in **Table 18**.

Table 18: *Distribution of Wafer Maps in the Dataset, detailing the number of wafer maps used for training, validation, and testing.*

#	Dataset	Number of wafer maps
1	Total	368,568
2	Training set	206,397
3	Validation set	88,457
4	Testing set	73,714

The loss function employed for training the model is the cross-entropy loss. This loss function measures the dissimilarity between the predicted probability distribution and the true distribution of the target classes. The AdamW optimization algorithm was utilized to update the model's parameters during training. This algorithm adjusts the model's parameters based on the gradients computed from the loss function. For cross-validation, 6 splits were created using a random seed of 42. These splits were employed to divide the dataset into training and validation sets during each fold of the training process.

4.3 Experimental Training Results

The proposed model was trained and validated using the training and validation dataset as demonstrated in Section 4.5. The hyperparameters were used as illustrated in Table 3. This study applies a stratified 6-fold cross-validation in which of them have the same proportion of class distribution. **Table 19** shows the average accuracy and loss value on both the training and validation sets for the 6 folds. The average training accuracy across all folds is 99.40%, indicating that the model performs well on the training data. The average validation accuracy is 97.80%, suggesting that the model generalizes reasonably well to unseen data. Also, the average training loss is 0.013, which indicates that the model's predictions are close to the actual values during training and the average validation loss is 0.044, indicating that the model's predictions are slightly less accurate on the validation data compared to the training data. Overall, these results suggest that the proposed CNN model is effective in classifying wafer defects.

Table 19: *Cross-Validation Results, detailing the average training and validation accuracy, as well as training and validation loss for each fold in a CNN-based defect classification model.*

#	Fold Number	Average Training Accuracy	Average Validation Accuracy	Average Training Loss	Average Validation Loss
1	Fold 1	98.35	95.17	0.044	0.153
2	Fold 2	99.47	97.88	0.011	0.038
3	Fold 3	99.60	98.26	0.007	0.025
4	Fold 4	99.64	98.45	0.005	0.018
5	Fold 5	99.67	98.47	0.004	0.017
6	Fold 6	99.68	98.56	0.004	0.014
Total		99.40	97.80	0.013	0.044

Figures 32 and **33** visualize the average accuracy percentage and loss values across the 6 folds per Epoch.

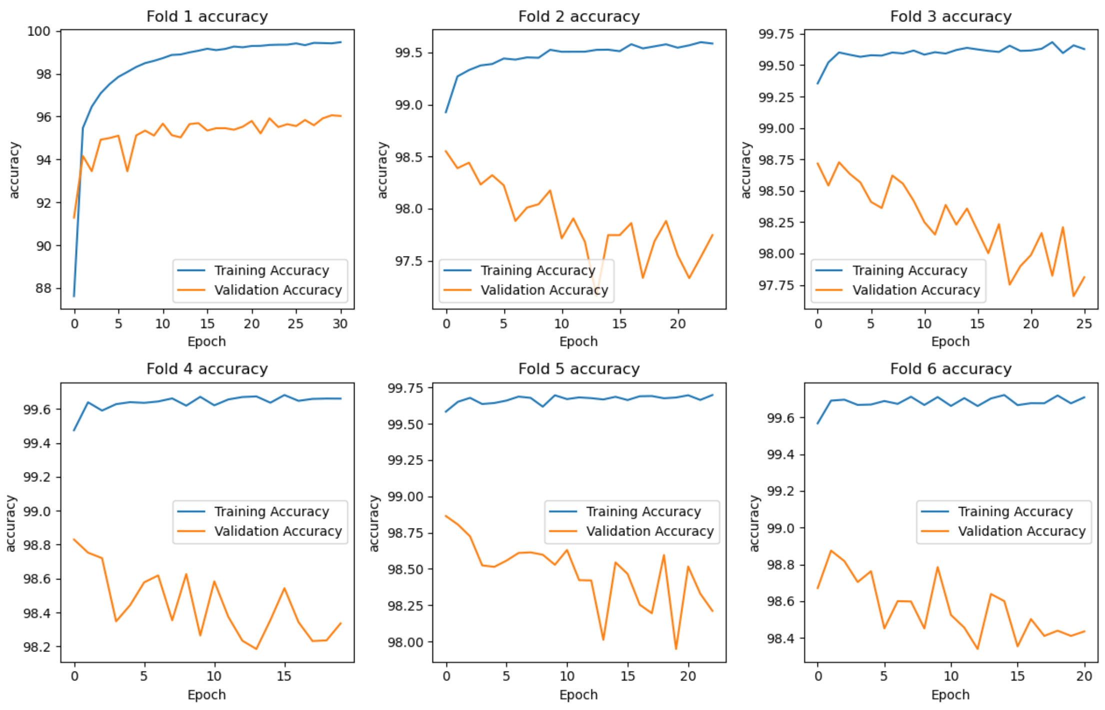


Figure 32: *Line plot of Average Accuracy per Epoch, showing the trend of model performance across the 6 folds during training.*

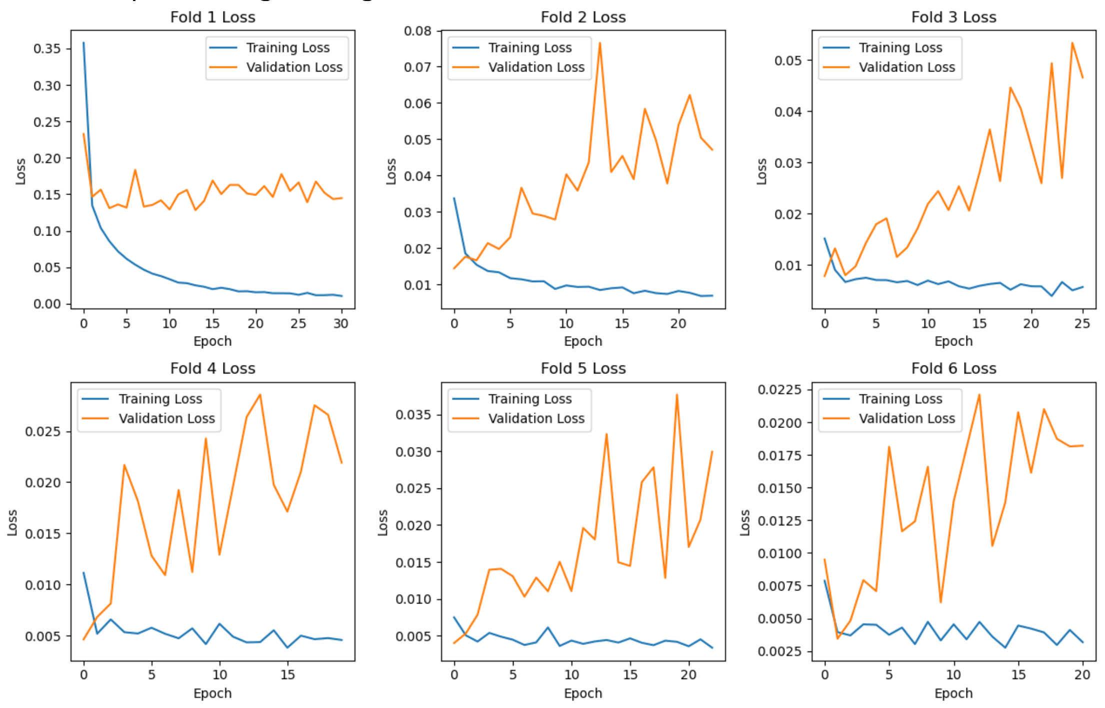


Figure 33: *Line plot of Average Loss per Epoch, showing the trend of model performance across the 6 folds during training.*

Also, the new dataset is used to train and validate two other CNN model namely ShuffleNetV2 and ResNet-50. From the **Table 20**, we can observe the performance of these models in terms of accuracy and loss metrics. Modified GoogLeNet achieved the highest average training accuracy of 99.40% and a relatively high average validation accuracy of 97.80%. It also had the lowest average training loss of 0.013 and a moderate average validation loss of 0.044. ShuffleNetV2 performed slightly lower than Modified GoogLeNet with an average training accuracy of 98.55% and an average validation accuracy of 96.52%. It had a higher average training loss of 0.036 and a higher average validation loss of 0.077. ResNet-50 showed similar performance to Modified GoogLeNet with an average training accuracy of 99.37% and an average validation accuracy of 97.83%. It had a slightly higher average training loss of 0.014 and a slightly lower average validation loss of 0.041. Overall, Modified GoogLeNet demonstrated the highest training accuracy and relatively good generalization performance on the validation set, as indicated by its high validation accuracy and low validation loss. ShuffleNetV2 and ResNet-50 also performed well but showed slightly lower accuracy and higher loss compared to Modified GoogLeNet.

Table 20: *Evaluation results of three deep learning models, Modified GoogLeNet, ShuffleNetV2 and ResNet-50 in terms of their average training and validation accuracy, as well as their average training and validation loss.*

#	Model	Average Training Accuracy	Average Validation Accuracy	Average Training Loss	Average Validation Loss
1	Modified GoogLeNet	99.40	97.80	0.013	0.044
2	ShuffleNetV2	98.55	96.52	0.036	0.077
3	ResNet-50	99.37	97.83	0.014	0.041

4.4 Model Evaluation

The confusion matrix in **Figure 34** highlights the strong performance of several classes, including "C+EL," "C+EL+L," "C+EL+S," and "C+ER." These classes demonstrate high numbers of true positives (TP) and true negatives (TN), indicating that the model accurately classifies instances belonging to these classes. Additionally, the low values for false positives (FP) and false negatives (FN) further support the model's effectiveness in these cases. However, there are certain classes, such as "Edge-Loc" and "Loc," where a higher number of false negatives can be observed compared to true positives. This suggests that the model struggles with accurately classifying instances belonging to these classes, potentially leading to missed detections of defects. On a positive note, instances in the "Near-full" class show very few occurrences of both false positives and false negatives. This indicates that the model performs well in accurately identifying instances within this class, demonstrating its effectiveness in detecting near-full defects. Overall, while the model shows strong performance in some classes, further improvements may be needed to enhance its accuracy in correctly classifying instances for classes like "Edge-Loc" and "Loc."

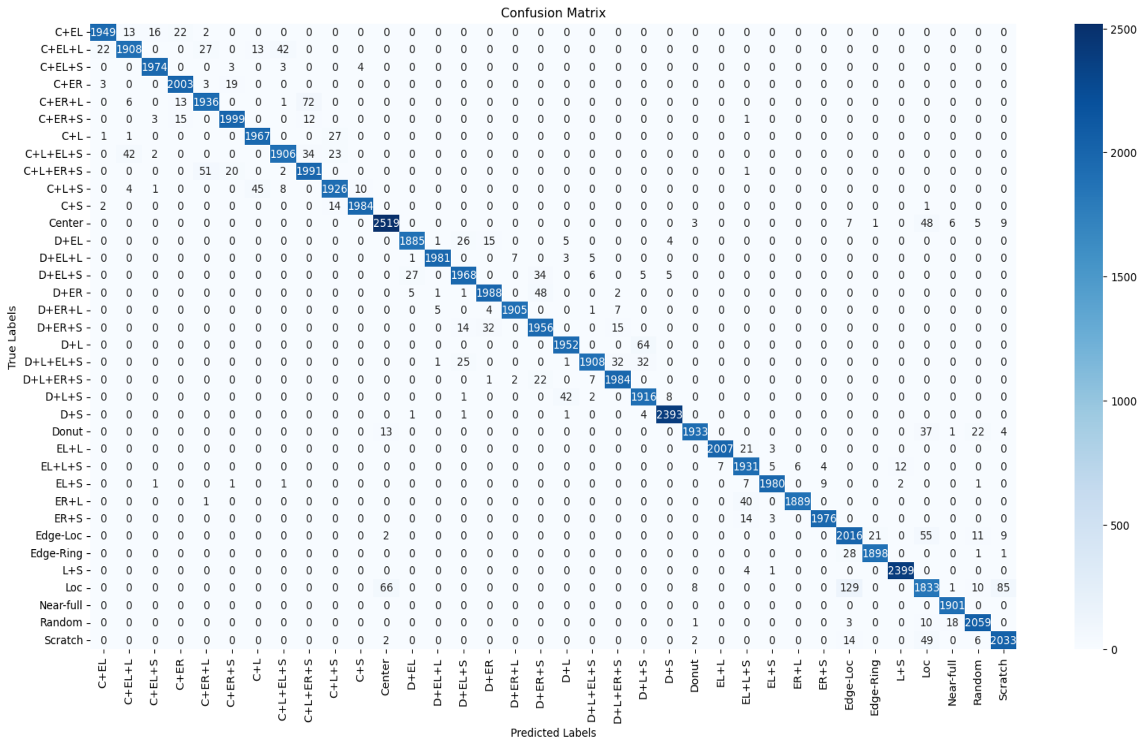


Figure 34: Confusion Matrix, detailing the true positives, false positives, true negatives, and false negatives for each class in a CNN-based defect classification model

Table 21 shows the accuracy, precision, recall, and F1 score of a classification algorithm for 36 wafer defect patterns. The average accuracy, precision, recall, and F1 score for all classes are 99.9%, 97%, 97% and 97% respectively. The algorithm correctly classifies 99.9% of the instances across all classes, indicating a high level of overall correctness in its predictions. The algorithm achieves a precision of 97%, which means that out of all the instances it predicts as positive, 97% are true positives. This indicates a low rate of false positive predictions. The algorithm achieves a recall of 98% which means that it identifies 97% of the actual positive instances correctly. This indicates a low rate of false negatives, as the algorithm captures a high proportion of the positive instances. The F1 score combines both precision and recall into a single metric with an F1 score of 97%. The algorithm demonstrates a good balance between precision and recall, indicating overall robust performance. These high values for accuracy, precision, recall, and F1 score suggest that the classification algorithm is effective, and accurate in identifying and classifying instances across all classes.

Table 21: *Performance Metrics for Each Defect Pattern Class in a CNN-based Defect Classification Model, detailing the accuracy, precision, recall, and F1 score for each class.*

#	Defect Pattern Name	Accuracy (%)	Precision	Recall	F1 Score
1	C+EL	100%	0.99	0.97	0.98
2	C+EL+L	100%	0.97	0.95	0.96
3	C+EL+S	100%	0.99	0.99	0.99
4	C+ER	100%	0.98	0.99	0.98
5	C+ER+L	100%	0.96	0.95	0.96
6	C+ER+S	100%	0.98	0.98	0.98
7	C+L	100%	0.97	0.99	0.98
8	C+L+EL+S	100%	0.97	0.95	0.96
9	C+L+ER+S	100%	0.94	0.96	0.95
10	C+L+S	100%	0.97	0.97	0.97
11	C+S	100%	0.99	0.99	0.99
12	Center	100%	0.97	0.97	0.97
13	D+EL	100%	0.98	0.97	0.98

14	D+EL+L	100%	1.00	0.99	0.99
15	D+EL+S	100%	0.97	0.96	0.96
16	D+ER	100%	0.97	0.97	0.97
17	D+ER+L	100%	1.00	0.99	0.99
18	D+ER+S	100%	0.95	0.97	0.96
19	D+L	100%	0.97	0.97	0.97
20	D+L+EL+S	100%	0.99	0.95	0.97
21	D+L+ER+S	100%	0.97	0.98	0.98
22	D+L+S	100%	0.95	0.97	0.96
23	D+S	100%	0.99	1.00	1.00
24	Donut	100%	0.99	0.96	0.98
25	EL+L	100%	1.00	0.99	0.99
26	EL+L+S	100%	0.96	0.98	0.97
27	EL+S	100%	0.99	0.99	0.99
28	ER+L	100%	1.00	0.98	0.99
29	ER+S	100%	0.99	0.99	0.99
30	Edge-Loc	100%	0.92	0.95	0.94
31	Edge-Ring	100%	0.99	0.98	0.99
32	L+S	100%	0.99	1.00	1.00
33	Loc	99%	0.90	0.86	0.88
34	Near-full	100%	0.99	1.00	0.99
35	Random	100%	0.97	0.98	0.98
36	Scratch	100%	0.95	0.97	0.96
	Total	99.9%	0.97	0.97	0.97

The TPR and FPR for 36 classes were calculated to compute the ROC curve for each class individually. As shown in **Figure 35**, the ROC curve for all classes is closer to the top-left corner of the plot which indicates that the model can accurately classify positive cases while minimizing false positives. the majority of the classes exhibit a high true positive rate, with many surpassing 0.98 and a low false positive rate (3.9045e-04). This indicates that the model is effective at correctly identifying positive cases while minimizing false positives and performing effectively and accurately in distinguishing between these classes. Nevertheless, the "Loc" class have a lower true positive rate of approximately 0.8598, suggesting that there may be challenges for the algorithm to differentiate them accurately, potentially necessitating further fine-tunning or optimization efforts.

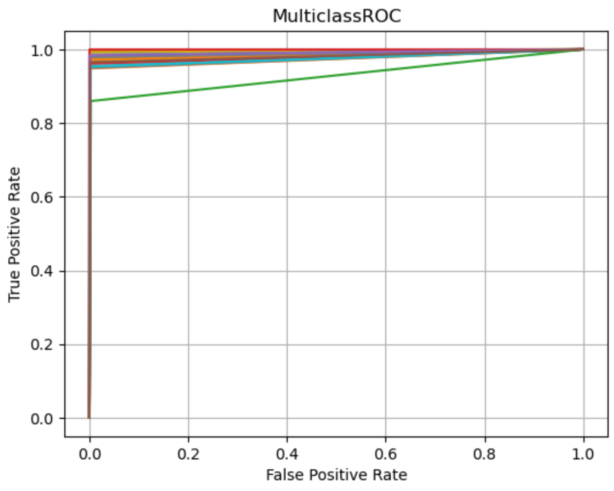


Figure 35: *ROC Curves for all Defect classes, detailing the trade-off between true positive rate and false positive rate for each class.*

Also, the other trained models are evaluated using the same testing set. **Table 22** provides insights into the models' performance in terms of accuracy, precision, recall, and F1 score. All three models achieved high average testing accuracy, with Modified GoogLeNet and ResNet-50 both achieving 99.9% accuracy, and ShuffleNetV2 achieving 99.8% accuracy. In terms of precision, recall, and F1 score, all models performed consistently well, with an average precision, recall, and F1 score of 0.97 for each model. This indicates that the models were successful in accurately classifying positive cases while minimizing false positives (precision), capturing true positive cases (recall), and achieving a balanced trade-off between precision and recall (F1 score).

Table 22: *Evaluation results of three deep learning models, Modified GoogLeNet, ShuffleNetV2 and ResNet-50 in terms of their average accuracy, precision, recall, and F1 score.*

#	Model	Average Testing Accuracy	Average Precision	Average Recall	Average F1_score
1	Modified GoogLeNet	99.9%	0.97	0.97	0.97
2	ShuffleNetV2	99.8%	0.96	0.96	0.96
3	ResNet-50	99.9%	0.97	0.97	0.97

4.5 Addressing Memory Limit Issues During Experiments

Memory limit errors were encountered during the data preprocessing and model training stages due to inadequate memory for handling the data or computations. Our study focused on optimizing memory usage during data preprocessing and model training in machine learning tasks. We encountered memory limit errors due to inadequate memory resources, which hindered our progress.

Various strategies were employed such as using generators and data loaders, freeing up unnecessary variables, incorporating memory-efficient data structures, implementing batch processing, leveraging the auto-cast feature, and harnessing the CUDA parallel computing platform to overcome these errors. Our findings show that these memory optimization strategies were effective in addressing memory limit errors and improving the overall performance of our proposed CNN model.

By using generators and data loaders, we were able to efficiently handle large datasets within limited memory resources. Freeing up unnecessary variables and employing memory-efficient data structures further contributed to efficient memory utilization. Batch processing allowed for efficient training, while the auto-cast feature reduced computational resource requirements and enabled the use of larger batch sizes. Leveraging the CUDA parallel computing platform provided an opportunity to harness the power of NVIDIA GPUs for general-purpose computing tasks, further optimizing resource allocation. The PyTorch allows to use of CUDA devices with simple APIs to transfer data to GPU memory and perform operations on GPU. Also, the training and validation processes are performed on GPU. **Table 23** demonstrate a performance analysis for using CUDA for training and validation phase of Modified GoogLeNet. The Modified GoogLeNet model achieved an average CPU usage of 35.17% and a maximum memory usage of 19469.69 MB during execution on the CPU. When executed on the GPU, the model achieved an average GPU usage of 81.30% and a maximum CUDA memory usage of 6812.47 MB. The execution time on the GPU was significantly faster, taking only 4,641.36 seconds. The speedup achieved by using the GPU instead of the CPU is approximately 38.78x, indicating a significant performance improvement.

Table 23: *Performance Analysis of Modified GoogLeNet for traning phase.*

	CPU			GPU			
Model	**Average CPU usage (%)**	**Maximum Memory usage (MB)**	**Execution time (s)**	**Average GPU usage (%)**	**Maximum CUDA Memory Usage (MB)**	**Execution time (s)**	**Speedup**
Modified GoogLeNet	35.17%	19469.69 MB	180,000 s	81.30%	6812.47 MB	4,641.36 s	38.78

4.6 Discussion of Results Comparing with Past Research

In this investigation, we examined the effectiveness of our proposed CNN model for classifying single and mixed types of wafer defect patterns. Our results indicate that our algorithm attained an accuracy of 99.9%, outperforming the state-of-the-art works as demonstrated in Section 2.1 by 2.4%. Moreover, neither of the previous studies specifically addresses the use of parallel programming techniques like CUDA to improve performance in wafer defect classification. Based on previous studies, many researchers who deal with 'WM-811K' dataset have resized images to 224x224 or 416x416, even though the wafer image dimensions of 27x25 have the maximum count in the original dataset. However, this resizing process has some drawbacks such as loss of fine-grained details present in the original image and increased computational resources like memory and CPU consumption - impacting model performance and training/testing times. To reduce these issues and improve efficiency, we choose 56x56 standard dimensions for all wafer images using Bicubic interpolation for smoother results.

Our findings are consistent with previous research for single and mixed defect pattern classification. For example, in [48] the researchers use sophisticated methods such as convolutional autoencoder in a GAN-based architecture to perform data augmentations which is computationally expensive and time-consuming. However, in our finding, we use a simple data augmentation as illustrated in **Section 3.3** which is relatively easy to implement and computationally efficient. In addition, their experiments showed an average accuracy of 97.5% for mixed types of wafer defects using the 'MixedWM38' dataset with a minimum average accuracy of 93.4% for classifying the 'Center with Edge-Ring with Scratch' defect and a maximum average accuracy of 100% for classifying the 'Donut with Edge-Ring with Scratch' defect. Our findings demonstrate that our proposed model exceeds their work in terms of accuracy, reaching up to 99.9% while in the class 'Center with Edge-Ring with Scratch,' we achieved an accuracy of 100%. similarly, the researchers in [53] use the 'MixedWM38' dataset with a semantic segmentation approach to generate multiple defect types. They achieved an average accuracy of 95.8%. however, our findings demonstrate that our proposed model exceeds their work in terms of accuracy, reaching up to 99.9%.

In [47], the researchers just address the overfitting issue by utilizing the dropout method with a probability of 0.5. however, they didn’t address the unbalanced classes issue that exists in the ‘WM-811K’ dataset. They achieved an average accuracy of 93.25% while the 'Donut' defect had a minimum average accuracy rate of 86%. Our finding solves the overfitting issue and unbalanced classes issue that exists in the ‘WM-811K’ dataset by utilizing data augmentation methods and the “stop early” method by stopping the training process before the model has fully converged. Also, our findings demonstrate that our proposed model exceeds their work in terms of accuracy, reaching up to 99.9% while the class Donut achieved an accuracy of 100%.

Also, our finding shows that our proposed classification framework outperforms the work in [49] that use the ShuffleNet-v2 model in term of accuracy, precision, recall and F1-score in single wafer defect patterns classification. In [50] the researcher uses simple data augmentation methods such as zooming and shifting, however, these methods can indeed result in the loss of some information in an image. They didn’t explain how they controlled shifts and zooms while preserving the integrity of the image information. they achieved an average classification accuracy of 96.2%. our proposed classification framework outperformed their work by 3.7% for single defect patterns classification.
In [51], the researchers achieved a top-3 accuracy rate of 96.2% which indicates that the correct label is included in the top three predictions for 96.2% of the cases. Therefore, in terms of overall accuracy, our proposed model with a 99.9% accuracy outperforms the model with a top-3 accuracy rate of 96.2%. It should be noted here that a dataset containing real images of wafer defects was used in their work, which is a positive thing. In [29], the researchers leverage the attention mechanism and cosine normalization to solve the imbalanced WM-811K dataset and they use fine-tuning methods for minimal iterative training. The attention mechanism and cosine normalization can be computationally expensive and the interpretation of attention weights can be challenging. Therefore, in terms of overall accuracy, our proposed model with a 99.9% accuracy outperforms the model with an accuracy of 95.46% by 4.44%.

In [52] the researchers evaluate the performance of YOLOv3, YOLOv4, ResNet50, and DenseNet121 in wafer defect patterns classification. This work provides robustness and diversity to identify which models are more robust and perform consistently across the WM-811K dataset. However, the YOLOv4 model achieved an accuracy of 95.7%, our proposed model with a 99.9% accuracy outperforms it by 4.2%. also, their work using the YOLOv4 model achieved an average F-score of 0.92 but our proposed model achieved an average F-score of 0.97. Overall, our proposed CNN model has outperformed the state-of-the-art works in terms of accuracy for single and mixed defect patterns classification.

Overall, our proposed work has superior performance compared to all other works, achieving the highest accuracies across three pretrained CNN models due to the methodology that involves data preprocessing, simple data augmentation and different tools and mechanisms to enhance training process.

Chapter Five

Conclusions

This final chapter concludes the thesis. It highlights the research contributions made by the study and discusses any limitations encountered. Future works are suggested for further exploration in the field. The chapter concludes with a summary of the main findings and a concise conclusion.

5.1 Research Contributions

Overall, our work contributes a novel framework, leveraging the GoogLeNet architecture, for single and mixed wafer defect patterns classification. We provide a cleaned balanced dataset called “WM-300K+ wafer map [Single & Mixed]” and achieve high accuracy, addressing the challenges associated with complex mixed-type defects. These contributions advance the field of wafer defect pattern classification and pave the way for improved wafer defect classification in semiconductor manufacturing processes. The key research contributions of our study are:

1. **Proposed CNN Model Based on GoogLeNet**: Our research introduces a novel CNN model based on the GoogLeNet architecture for wafer defect pattern classification. The proposed GoogLeNet model provides a strong foundation for accurate and robust classification of wafer defect patterns.
2. **Proposed the DefectClassifierX application**: an automated pattern classification system for wafer defects called "DefectClassifierX" is present. This system utilizes the proposed CNN model based on the GoogLeNet architecture to accurately and efficiently classify wafer defect patterns.
3. **Classification of Single and Mixed Wafer Defect Patterns**: Our proposed CNN model is designed to classify both single and mixed types of wafer defect patterns.

This allows for comprehensive and reliable classification, even when multiple defect types are present on the same wafer.

4. **Generalization to New Defect Types:** Our proposed CNN model demonstrates promising generalization capabilities to new or unseen defect types. While training on a specific set of defect patterns, the model's underlying architecture and learned features enable it to potentially classify previously unseen defect types with reasonable accuracy.
5. **Achievement of High Accuracy:** Through extensive experimentation and evaluation, our proposed CNN model achieves an impressive average accuracy of 99.9% in wafer defect pattern classification. This high accuracy demonstrates the effectiveness and reliability of our approach in accurately identifying and classifying different types of wafer defects.
6. **Publication of WM-300K+ Wafer Map Dataset:** In addition to proposing a novel CNN model, we also created a new dataset called "WM-300K+ wafer map [Single & Mixed]." This dataset is noteworthy as it is cleaned, and balanced, and consists of more than 300,000 wafer map images with 36 classes. We publish this dataset on Kaggle, providing a valuable resource for the research community and enabling further advancements in wafer map analysis and classification.
7. **Utilization of CUDA for Enhanced Training and Testing Speed:** To speed up the training and testing process of our proposed CNN model, we utilize CUDA, a parallel computing platform that enables significant performance improvements when training deep neural networks.

5.2 Limitations

While our research makes significant contributions to the field of wafer defect pattern classification, certain limitations should be acknowledged:

1. **Dependency on Dataset Quality:** The effectiveness of our approach is highly dependent on the quality and diversity of the training dataset. If the dataset contains inaccuracies, noise, or biases, it may impact the model's performance and generalizability. Therefore, ensuring a high-quality and representative dataset is crucial for obtaining reliable results.
2. **Computational Resource Requirements:** Our proposed CNN model, particularly when using the GoogLeNet architecture and CUDA for enhanced speed, may require significant computational resources during training and testing. This includes high-performance GPUs and sufficient memory capacity. Researchers with limited access to such resources may face challenges in replicating our experiments or applying our approach in resource-constrained environments.
3. **Data Augmentation Limitations:** While data augmentation can help address dataset issues such as class imbalance, the effectiveness of this technique may vary depending on the specific characteristics of the dataset. In some cases, data augmentation may not fully mitigate the challenges associated with imbalanced data, leading to potential biases or limitations in the model's performance.
4. **Lack of Evaluation on Real Wafer Map Images**: Our proposed model has not been evaluated on real wafer map images. Although we achieved high accuracy using our dataset, the model's performance on real-world wafer map images may differ because of variations in image quality, noise, and other factors specific to real

production environments. Further evaluation and validation of real wafer map images are necessary to assess the model’s practical applicability.

6. *Future Works*

Several areas warrant further investigation and exploration. Future work should focus on addressing the following aspects:

1. Expand the dataset to include a wider range of defect types and variations, enabling the model to handle a broader array of real-world scenarios.
2. Optimize computational resource requirements by developing more efficient architectures or exploring alternative hardware configurations that can achieve comparable performance with reduced resource demands.
3. Further evaluation and validation of real wafer map images are necessary by conducting extensive evaluations to understand the model’s performance under these realistic conditions and identify any necessary adaptations or improvements.

7. *Conclusion*

In our study, we proposed a novel CNN model based on the GoogLeNet architecture for wafer defect pattern classification, which achieved an impressive average accuracy of 99.9%. We also developed an automated pattern classification system for wafer defects called "DefectClassifierX" that utilizes the proposed CNN model to accurately and efficiently classify wafer defect patterns. Our contributions include the creation of a new dataset called "WM-300K+ wafer map [Single & Mixed]" and the use of CUDA for enhanced training and testing speed. Notably, our approach surpasses existing benchmarks and achieves state-of-the-art results in wafer defect pattern classification, showcasing the advancements made in our proposed CNN model and its potential to significantly enhance the accuracy and reliability of wafer defect analysis in semiconductor manufacturing.

However, our research also highlights several limitations, including dependence on dataset quality, computational resource requirements, and data augmentation limitations. Although our proposed model achieves high accuracy using our dataset, it has not been evaluated on real wafer map images, indicating a need for further evaluation and validation to assess its practical applicability. Future work should prioritize addressing these limitations by improving dataset quality, optimizing computational resource requirements, enhancing data augmentation techniques, and evaluating the proposed model on real wafer map images. By doing so, we can continue to advance the field of wafer defect pattern classification, improving the reliability, generalizability, and practical applicability of our approach for real-world scenarios.

Overall, our study contributes a powerful CNN model capable of accurately classifying both single and mixed wafer defect patterns, surpassing previous studies in accuracy. With further research and development, our approach holds significant promise in enhancing the efficiency and effectiveness of wafer defect analysis in semiconductor manufacturing.

References

Y. Q. Chen, B. Zhou, M. Zhang, and C. M. Chen, “Using IoT technology for computer-integrated manufacturing systems in the semiconductor industry,” *Applied Soft Computing Journal*, vol. 89, 2020, doi: 10.1016/j.asoc.2020.106065.

A. A. R. M. A. Ebayyeh and A. Mousavi, “A Review and Analysis of Automatic Optical Inspection and Quality Monitoring Methods in Electronics Industry,” *IEEE Access*, vol. 8. 2020. doi: 10.1109/ACCESS.2020.3029127.

Fortune Business Insights, “Consumer Electronics Market Size to Hit USD 989.37 Billion.” Accessed: Aug. 12, 2023. [Online]. Available: https://www.globenewswire.com/news-release/2023/02/28/2616731/0/en/Consumer-Electronics-Market-Size-to-Hit-USD-989-37-Billion-by-2027-At-5-3-CAGR.html

X. Guo, V. Verma, P. Gonzalez-Guerrero, S. Mosanu, and M. R. Stan, “Back to the future: Digital circuit design in the FinFET Era,” *J Low Power Electron*, vol. 13, no. 3, 2017, doi: 10.1166/jolpe.2017.1489.

K. T. Turner and S. M. Spearing, “Mechanics of direct wafer bonding,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 462, no. 2065, 2006, doi: 10.1098/rspa.2005.1571.

K. A. Jackson and W. Schroter, *Handbook of semiconductor technology set*. 2008. doi: 10.1002/9783527619290.

J. N. D. Gupta, R. Ruiz, J. W. Fowler, and S. J. Mason, “Operational planning and control of semiconductor wafer production,” *Production Planning and Control*, vol. 17, no. 7, 2006, doi: 10.1080/09537280600900733.

A. Ciftja, T. A. Engh, and M. Tangstad, “Refining and Recycling of Silicon : A Review,” *World*, no. February, 2008.

W. Kern, “Evolution of silicon wafer cleaning technology,” in *Proceedings - The Electrochemical Society*, 1990. doi: 10.1149/1.2086825.

M. Itano, F. W. Kern, M. Miyashita, and T. Ohmi, “Particle Removal from Silicon Wafer Surface in Wet Cleaning Process,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 6, no. 3, 1993, doi: 10.1109/66.238174.

A. Khairnar, “Chapter 2 Thin Film Deposition and Characterization Techniques,” *Thin Film Growth and Characterizaiton Techniques*, no. 2014, 2010.

T. Workman *et al.*, “Die to Wafer Hybrid Bonding and Fine Pitch Considerations,” in *Proceedings - Electronic Components and Technology Conference*, 2021. doi: 10.1109/ECTC32696.2021.00326.

W. R. Mann, “Wafer test methods to improve semiconductor die reliability,” *IEEE Design and Test of Computers*, vol. 25, no. 6, 2008, doi: 10.1109/MDT.2008.174.

I. S. Amiri, M. M. Ariannejad, D. Vigneswaran, C. S. Lim, and P. Yupapin, "Performances and procedures modules in micro electro mechanical system packaging technologies," *Results Phys*, vol. 11, 2018, doi: 10.1016/j.rinp.2018.09.008.

K. Kyeong and H. Kim, "Classification of Mixed-Type Defect Patterns in Wafer Bin Maps Using Convolutional Neural Networks," *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 3, 2018, doi: 10.1109/TSM.2018.2841416.

Semiconductor Industry Association, "Chipmakers Are Ramping Up Production to Address Semiconductor Shortage. Here's Why that Takes Time," Feb. 2021, Accessed: Aug. 12, 2023. [Online]. Available: https://www.semiconductors.org/chipmakers-are-ramping-up-production-to-address-semiconductor-shortage-heres-why-that-takes-time/

R. Desineni and E. Tuv, "High-Value AI in Intel's Semiconductor Manufacturing Environment," *Intel*. Intel, 2018. Accessed: Oct. 26, 2023. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/ai-in-semiconductor-manufacturing-paper.pdf

T. Yuan, S. Z. Ramadan, and S. J. Bae, "Yield prediction for integrated circuits manufacturing through hierarchical bayesian modeling of spatial defects," *IEEE Trans Reliab*, vol. 60, no. 4, 2011, doi: 10.1109/TR.2011.2161698.

U. Kaempf, “The Binomial Test: A Simple Tool to Identify Process Problems,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 8, no. 2, 1995, doi: 10.1109/66.382280.

J. Yu and X. Lu, “Wafer Map Defect Detection and Recognition Using Joint Local and Nonlocal Linear Discriminant Analysis,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 29, no. 1, 2016, doi: 10.1109/TSM.2015.2497264.

M. Piao, C. H. Jin, J. Y. Lee, and J. Y. Byun, “Decision tree ensemble-based wafer map failure pattern recognition based on radon transform-based features,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 2, 2018, doi: 10.1109/TSM.2018.2806931.

C. K. Hansen and P. Thyregod, “Use of wafer maps in integrated circuit manufacturing,” *Microelectronics Reliability*, vol. 38, no. 6–8, 1998, doi: 10.1016/S0026-2714(98)00127-9.

S. Parrish, “A Study of Defects in High Reliability Die Sort Applications,” *International Symposium on Microelectronics*, vol. 2019, no. 1, 2019, doi: 10.4071/2380-4505-2019.1.000463.

M. Liukkonen and Y. Hiltunen, “Recognition of Systematic Spatial Patterns in Silicon Wafers Based on SOM and K-means,” 2018. doi: 10.1016/j.ifacol.2018.03.075.

T. Tiwari, T. Tiwari, and S. Tiwari, “How Artificial Intelligence, Machine Learning and Deep Learning are Radically Different?,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 8, no. 2, 2018, doi: 10.23956/ijarcsse.v8i2.569.

L. Deng and D. Yu, “Deep Learning: Methods and Applications Foundations and Trends R in Signal Processing,” *Signal Processing*, vol. 7, no. 2013, 2013.

J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*, vol. 61. 2015. doi: 10.1016/j.neunet.2014.09.003.

F. López de la Rosa, J. L. Gómez-Sirvent, R. Morales, R. Sánchez-Reolid, and A. Fernández-Caballero, “A deep residual neural network for semiconductor defect classification in imbalanced scanning electron microscope datasets,” *Appl Soft Comput*, vol. 131, 2022, doi: 10.1016/j.asoc.2022.109743.

Q. Xu, N. Yu, and F. Essaf, “Improved Wafer Map Inspection Using Attention Mechanism and Cosine Normalization,” *Machines*, vol. 10, no. 2, 2022, doi: 10.3390/machines10020146.

R. E. Sarpietro *et al.*, “Explainable Deep Learning System for Advanced Silicon and Silicon Carbide Electrical Wafer Defect Map Assessment,” *IEEE Access*, vol. 10, 2022, doi: 10.1109/ACCESS.2022.3204278.

X. Ran, “Research on the Optimization of Defect Detection Based on Convolutional Neural Network Architecture,” *SHS Web of Conferences*, vol. 144, 2022, doi: 10.1051/shsconf/202214402018.

J. Ma *et al.*, “Review of Wafer Surface Defect Detection Methods,” *Electronics (Switzerland)*, vol. 12, no. 8, 2023, doi: 10.3390/electronics12081787.

M. B. Alawieh, D. Boning, and D. Z. Pan, “Wafer map defect patterns classification using deep selective learning,” in *Proceedings - Design Automation Conference*, 2020. doi: 10.1109/DAC18072.2020.9218580.

T. Ishida, I. Nitta, D. Fukuda, and Y. Kanazawa, “Deep Learning-Based Wafer-Map Failure Pattern Recognition Framework,” in *Proceedings - International Symposium on Quality Electronic Design, ISQED*, 2019. doi: 10.1109/ISQED.2019.8697407.

A. Vyas, S. Yu, and J. Paik, “Fundamentals of digital image processing,” in *Signals and Communication Technology*, 2018. doi: 10.1007/978-981-10-7272-7_1.

V. K. Mishra, S. Kumar, and N. Shukla, “Image Acquisition and Techniques to Perform Image Acquisition,” *SAMRIDDHI : A Journal of Physical Sciences, Engineering and Technology*, vol. 9, no. 01, 2017, doi: 10.18090/samriddhi.v9i01.8333.

S. Süsstrunk, R. Buckley, and S. Swen, “Standard RGB color spaces,” in *Final Program and Proceedings - IS and T/SID Color Imaging Conference,* 1999. doi: 10.2352/cic.1999.7.1.art00024.

K. Zakka, “CS231n Convolutional Neural Networks for Visual Recognition,” *Stanford University,* 2021.

R. E. Neapolitan and X. Jiang, “Neural Networks and Deep Learning,” in *Artificial Intelligence,* 2018. doi: 10.1201/b22400-15.

R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights into Imaging,* vol. 9, no. 4. 2018. doi: 10.1007/s13244-018-0639-9.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature,* vol. 323, no. 6088, 1986, doi: 10.1038/323533a0.

S. Indolia, A. K. Goswami, S. P. Mishra, and P. Asopa, “Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach,” in *Procedia Computer Science,* 2018. doi: 10.1016/j.procs.2018.05.069.

D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, “Flexible, high performance convolutional neural networks for image classification,” in *IJCAI International Joint Conference on Artificial Intelligence*, 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-210.

V. Romanuke, “Appropriate Number and Allocation of RELUS in Convolutional Neural Networks,” *Research Bulletin of the National Technical University of Ukraine “Kyiv Politechnic Institute,”* vol. 0, no. 1, 2017, doi: 10.20535/1810-0546.2017.1.88156.

A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun ACM*, vol. 60, no. 6, 2017, doi: 10.1145/3065386.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, 2014.

N. Yu, Q. Xu, and H. Wang, “Wafer defect pattern recognition and analysis based on convolutional neural network,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, 2019, doi: 10.1109/TSM.2019.2937793.

N. Yu, H. Chen, Q. Xu, M. M. Hasan, and O. Sie, “Wafer map defect patterns classification based on a lightweight network and data augmentation,” *CAAI Trans Intell Technol*, 2022, doi: 10.1049/cit2.12126.

R. Doss, J. Ramakrishnan, S. Kavitha, S. Ramkumar, G. Charlyn Pushpa Latha, and K. Ramaswamy, “Classification of Silicon (Si) Wafer Material Defects in Semiconductor Choosers using a Deep Learning ShuffleNet-v2-CNN Model,” *Advances in Materials Science and Engineering*, vol. 2022, 2022, doi: 10.1155/2022/1829792.

M. Saqlain, Q. Abbas, and J. Y. Lee, “A Deep Convolutional Neural Network for Wafer Defect Identification on an Imbalanced Dataset in Semiconductor Manufacturing Processes,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 3, 2020, doi: 10.1109/TSM.2020.2994357.

C. Phua and L. B. Theng, “Semiconductor wafer surface: Automatic defect classification with deep CNN,” in *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, 2020. doi: 10.1109/TENCON50793.2020.9293715.

P. P. Shinde, P. P. Pai, and S. P. Adiga, “Wafer Defect Localization and Classification Using Deep Learning Techniques,” *IEEE Access*, vol. 10, 2022, doi: 10.1109/ACCESS.2022.3166512.

J. Yan, Y. Sheng, and M. Piao, “Semantic Segmentation Based Wafer Map Mixed-Type Defect Pattern Recognition,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023, doi: 10.1109/TCAD.2023.3274958.

M.-J. Wu, J.-S. R. Jang, and J.-L. Chen, “Wafer Map Failure Pattern Recognition and Similarity Ranking for Large-Scale Data Sets,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 28, no. 1, pp. 1–12, 2015, doi: 10.1109/TSM.2014.2364237.

C. Suresh, S. Singh, R. Saini, and A. K. Saini, “A Comparative Analysis of Image Scaling Algorithms,” *International Journal of Image, Graphics and Signal Processing*, vol. 5, no. 5, 2013, doi: 10.5815/ijigsp.2013.05.07.

D. D. Muresan and T. W. Parks, “Adaptively Quadratic (AQua) image interpolation,” *IEEE Transactions on Image Processing*, vol. 13, no. 5, 2004, doi: 10.1109/TIP.2004.826097.

A. Bansal, R. Sharma, and M. Kathuria, “A Systematic Review on Data Scarcity Problem in Deep Learning: Solution and Applications,” *ACM Comput Surv*, vol. 54, no. 10, 2022, doi: 10.1145/3502287.

Y. Fujimoto, K. Fukushima, and K. Murase, “Extensive studies of the neutron star equation of state from the deep learning inference with the observational data augmentation,” *Journal of High Energy Physics*, vol. 2021, no. 3, 2021, doi: 10.1007/JHEP03(2021)273.

D. Hirahara, E. Takaya, T. Takahara, and T. Ueda, “Effects of data count and image scaling on Deep Learning training,” *PeerJ Comput Sci*, vol. 6, 2020, doi: 10.7717/peerj-cs.312.

E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “AutoAugment: Learning Augmentation Policies from Data,” *Cvpr 2019*, no. Section 3, 2018.

S. Wang, Z. Zhong, Y. Zhao, and L. Zuo, “A Variational Autoencoder Enhanced Deep Learning Model for Wafer Defect Imbalanced Classification,” *IEEE Trans Compon Packaging Manuf Technol*, vol. 11, no. 12, 2021, doi: 10.1109/TCPMT.2021.3126083.

E. R. Davies, *Computer Vision: Principles, Algorithms, Applications, Learning: Fifth Edition*. 2017.

R. Szeliski, *Texts in Computer Science: Computer Vision Algorithms and Applications Second Edition*, no. January. 2022.

J.-H. Park, “Spatial Transformations of Shapes,” 2022. doi: 10.1007/978-3-031-08946-6_1.

C. Seger, “An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing,” *Degree Project Technology*, 2018.

C. Szegedy *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015. doi: 10.1109/CVPR.2015.7298594.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016. doi: 10.1109/CVPR.2016.308.

I. Loshchilov and F. Hutter, “Fixing Weight Decay Regularization in Adam,” 2018.

P. Aswathy, Siddhartha, and D. Mishra, “Deep GoogLeNet Features for Visual Object Tracking,” in *2018 13th International Conference on Industrial and Information Systems, ICIIS 2018 - Proceedings*, 2018. doi: 10.1109/ICIINFS.2018.8721317.

X. Ding, X. Zhang, J. Han, and G. Ding, “Scaling Up Your Kernels to 31×31: Revisiting Large Kernel Design in CNNs,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2022. doi: 10.1109/CVPR52688.2022.01166.

M. V. Narkhede, P. P. Bartakke, and M. S. Sutaone, “A review on weight initialization strategies for neural networks,” *Artif Intell Rev*, vol. 55, no. 1, 2022, doi: 10.1007/s10462-021-10033-z.

A. Paszke *et al.*, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, 2019.

PyTorch, “Datasets & DataLoaders.” Accessed: Nov. 17, 2023. [Online]. Available: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

S. Narang *et al.*, “Mixed precision training,” in *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

Z. Ji, J. D. Li, and M. Telgarsky, “Early-stopped neural networks are consistent,” in *Advances in Neural Information Processing Systems*, 2021.

P. Maria, “JavaScript Beyond the Browser,” *Open Repository Theseus, University of Turku*, vol. 1, 2018.

Devndra Ghimire, “Comparative study on Python web frameworks: Flask and Django,” *Metropolia University of Applied Sciences*, 2020.

MDN contributors, “FileReader.” Accessed: Nov. 18, 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/FileReader?locale=en

Y. Bengio, “Practical recommendations for gradient-based training of deep architectures,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7700 LECTURE NO, 2012, doi: 10.1007/978-3-642-35289-8_26.

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. doi: 10.1145/3292500.3330701.

M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, “BOA : The Bayesian Optimization Algorithm 1 Introduction,” *Proceedings of the genetic and evolutionary computation conference GECCO-99*, vol. 1, no. 99003, 1999.

E. Brochu, T. Brochu, and N. Freitas, “A Bayesian interactive optimization approach to procedural animation design,” in *Computer Animation 2010 - ACM SIGGRAPH / Eurographics Symposium Proceedings, SCA 2010*, 2010.

S. Falkner, A. Klein, and F. Hutter, "Practical hyperparameter optimization for deep learning," in *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*, 2018.

H. Jiang and X. Li, "Parameter estimation of statistical models using convex optimization," in *IEEE Signal Processing Magazine*, 2010. doi: 10.1109/MSP.2010.936018.

A. D. Jagtap, K. Kawaguchi, and G. Em Karniadakis, "Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 476, no. 2239, 2020, doi: 10.1098/rspa.2020.0334.

I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

D. Silwal, "Confusion Matrix, Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures," Linkedin.

R. Joshi, "Accuracy, Precision, Recall &amp; F1 Score: Interpretation of Performance Measures - Exsilio Blog," 2016.

J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve," *Radiology*, vol. 143, no. 1, 1982, doi: 10.1148/radiology.143.1.7063747.

C. Marzban, “The ROC curve and the area under it as performance measures,” *Weather Forecast*, vol. 19, no. 6, 2004, doi: 10.1175/825.1.

The Pandas Develompent Team, “Pandas-dev/pandas: Pandas,” *Zenodo*, vol. 21, no. 1He. 2020.

C. R. Harris *et al.*, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825. 2020. doi: 10.1038/s41586-020-2649-2.

J. Zook, M. Ossmann, and G. Tingwald, “Pillow,” in *The Covert Life of Hospital Architecture*, 2022. doi: 10.2307/j.ctv20pxz7f.8.

J. D. Hunter, “Matplotlib: A 2D graphics environment,” *Comput Sci Eng*, vol. 9, no. 3, 2007, doi: 10.1109/MCSE.2007.55.

R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A matlab-like environment for machine learning,” 2011.

NVIDIA, “Cuda C Best Practices Guide,” *Nvidia Corporation*, vol. 7.5, no. September, 2015.

Nvidia, “NVIDIA CUDA C programming guide,” *NVIDIA Corporation*, no. February, 2011.

TorchVision maintainers and contributors, “TorchVision: PyTorch’s Computer Vision library.” GitHub, 2016. Accessed: Nov. 19, 2023. [Online]. Available: https://github.com/pytorch/vision

الملخص

يعد تصنيع رقائق أشباه الموصلات عملية معقدة معرضة للعيوب. في هذه الدراسة، نقدم تطبيق DefectClassifierX، وهو نظام آلي لتصنيف أنماط العيوب من خلال توظيف نموذج شبكة عصبية تلافيفية يعتمد على بنية GoogLeNet بالاضافة الى استخدام معمارية الحوسبة المتوازية CUDA من أجل تسريع عملية التدريب والاختبار. نهدف في هذه الدراسة إلى تحسين تصنيف العيوب في عملية تصنيع أشباه الموصلات من خلال التصنيف الدقيق لأنماط عيب الرقاقة المفردة والمختلطة. من أجل التحقق من النهج المقترح، تم اجراء تجارب شاملة باستخدام مجموعة بيانات جديدة تسمى "WM-300K+ wafer map [single and mixed]," حيث تتكون من 36 نمط من العيوب المفردة والمختلطة. أظهرت هذه نتائج التجارب أنه قيمة كل من precision, recall و F1-score خلال عملية الاختبار للنموذج كانت 0.97 وهو ما يشير الى أن أداء النموذج المقترح ممتاز. كما وأظهرت مستوى ملحوظ من الدقة، بمتوسط دقة تصنيف 99.9% لكل من أنواع العيوب المفردة والمختلطة. يتفوق نهجنا في الأداء على الدراسات السابقة في تصنيف أنماط عيوب الرقاقة ولديه القدرة على تحسين كفاءة وفعالية تحليل عيوب الرقاقة بشكل كبير في تصنيع أشباه الموصلات. بالإضافة إلى ذلك، استخدمنا ضبط متغيرات التعلم hyperparameter باستخدام Optuna ونفذنا آلية إيقاف التعلم المبكر لتحسين التقارب. علاوة على ذلك، قمنا بتوظيف محسن AdamW لزيادة تعزيز أداء النموذج. يتوافق DefectClassifierX مع أنظمة التشغيل المتعددة، مما يضمن إمكانية الوصول لقاعدة مستخدمين أوسع. في حين أن نتائجنا مشجعة، إلا أن هناك حاجة إلى مزيد من البحث لمعالجة القيود المتعلقة بجودة مجموعة البيانات ومتطلبات الموارد الحسابية وتقنيات زيادة البيانات. بالإضافة إلى ذلك، من المهم تقييم النموذج باستخدام صور حقيقة لخرائط الرقاقات لتقييم قابلية التطبيق العملي.