Wafer Defect Detection Using Convolutional Neural Networks: Network Evaluation and Model Optimization

Colin M. Perry

December 26, 2024

Abstract

Purpose - This report presents the development of a Convolutional Neural Network (CNN) designed to detect defects in wafer maps. By automating defect detection with deep learning, this project demonstrates significant potential for improving accuracy and efficiency. Additionally, this report will discuss extensions of this technology to broader manufacturing process control, highlighting its role in advancing smart manufacturing systems.

Methodology - This project utilizes the PyTorch machine learning library to build and train a CNN. The WM-811K Wafer Defect Dataset, a dataset of 811,457 sensor based wafer maps, was used to train the model. Data was filtered and selected as appropriate, and oversampling was applied on the data to improve model performance on minority classes.

Findings - The development of the model was successful, with an overall test accuracy of nearly 95This score shows great robustness of the model, and it is very encouraging given the limited resources used to train the model.

Value - Although the current demonstrated model is not entirely novel, it serves as a proof of concept for a lightweight wafer defect classification system. This system is readily extensible and can be adapted into larger smart manufacturing systems serving gross yield optimization through process control, process monitoring, tool dispatch scheduling, and system anomaly detection.

1 Introduction

In the era of Industry 4.0, advancements in artificial intelligence and machine learning have revolutionized manufacturing by enabling highly precise and scalable solutions for process development, anomaly detection, and equipment monitoring. Among these advancements, Convolutional Neural Networks have emerged as a powerful tool for image-based defect detection and classification. The ability to detect and classify defects while guiding root cause analysis has been a boon for industries where minor declines in yield or tool uptime can lead to serious financial losses. By leveraging CNNs, manufacturers can achieve higher yields, reduced downtimes, and more efficient process control, aligning closely with the goals of modern smart factories.

This report explores the development of a CNN model implemented in PyTorch tailored to predict wafer defects, a critical challenge in semiconductor device fabrication. Utilizing the WM-811k Wafer Defect Dataset, which comprises over 800,000 wafer maps collected from process sensor data, this project highlights the application of advanced machine learning techniques such as data augmentation and class balancing to improve model accuracy and generalizability.

A discussion of next steps, as well as how this technology can be expanded will also be discussed. Technological propositions regarding the implementation of advanced optical microscopy and SECS/GEM to improve network performance will be investigated.

2 Method

A CNN was chosen as the model for this project as CNNs are regarded as one of the most robust models for learning image data. The WM-811k dataset was chosen due to its large size (>800,000 samples), as well as its direct relationship to the semiconductor industry. The dataset was not immediately usable for machine learning, however, and various data wrangling functions needed to be implemented. Firstly, as my machine learning implementation was supervised, I dropped all data that was not labeled. This brought the amount of samples in the working dataset from

$n = 811,457$ to $n = 172,950$. The wafer maps provided by the dataset were also of non-uniform size. My first approach at normalizing my dataset to a single wafer map shape was to simply pad each wafer map with empty space. Although this ultimately allowed my network to run, it lead to low overall performance and awful performance in certain defect classes. This lead me to investigate interpolation as a solution to the data size mismatch issue. Ultimately, I decided to utilize the PyTorch *interpolation* function to normalize all wafer maps to a $120x120$ image.

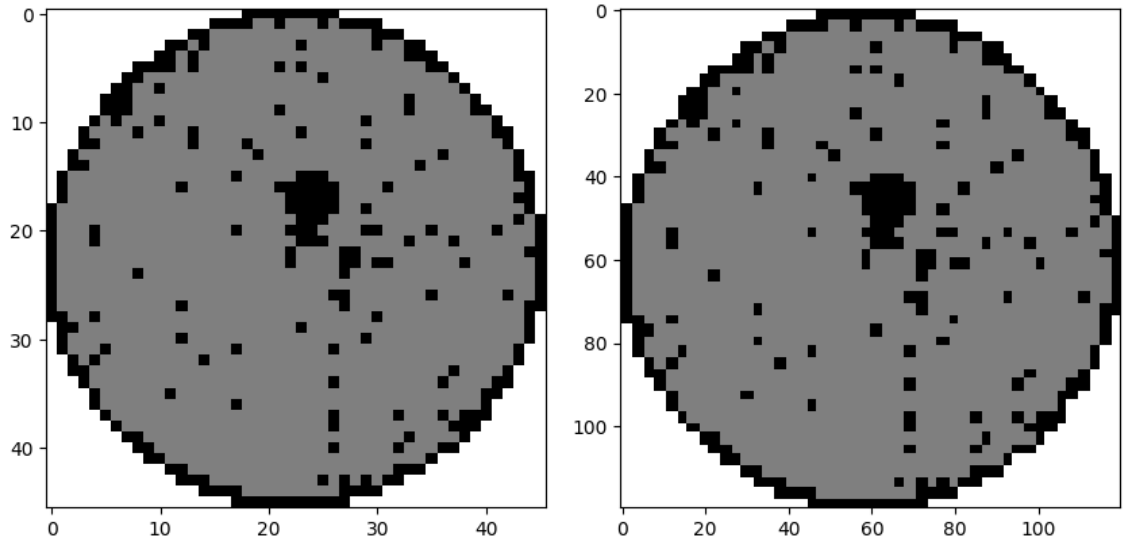

Figure 1: Original 45x48 wafer map (left) and 120x120 interpolated image (right)

Once the wafer map sizes were normalized, I sought to increase the number of data points in the minority class through oversampling. Before oversampling, the majority class "*none*" had almost 1,000 times the amount of samples as the most infrequent minority class, *Near-Full*. As the data set was a collection of sensor data from arbitrary processes, I found it appropriate to oversample through rotational and reflective image transformation. The five transformations I applied were 90˚ rotation, 180˚ rotation, 270˚ rotation, reflection across the x axis, and reflection across the y axis. After this oversampling, The majority class "*none*" had only about 165 times the amount of samples as the most infrequent minority class, *Near-Full*.

Table I: Defect Class Sample Count

Class	**Count (Original)**	**Class**	**Count (Oversampled)**
Center	4294	Center	25764
Donut	555	Donut	3330
Edge-Loc	5189	Edge-Loc	31134
Edge-Ring	9680	Edge-Ring	58080
Loc	3593	Loc	21558
Near-full	149	Near-full	894
Random	866	Random	5196
Scratch	1193	Scratch	7158
none	147431	none	147431

After oversampling, the total used sample count increased from $n = 172,950$ to $n = 300,545$, with a much better class balance. The data set was then randomly split 80/10/10 between training data, validation data, and testing data. Each time there was a change in the model (ie. hyperparameter tuning), the data would be randomly split again. This ensured that the model would not be trained on any implicit bias based on the way the various sets were split during hyperparameter tuning.

The CNN was implemented using the PyTorch machine learning library, and computing was done in the Google Colab notebook environment. This environment was chosen to allow for economical access to large compute power, especially GPUs for GPU acceleration. My CNN ultimately had two convolution layers with three fully connected layers. The exact architecture,

as well as the hyperparameters that the model was trained with (batch size, epoch count, learning rate, etc) can be found in the Data section below.

During the model creation, hyperparameters were tuned using validation scores returned during training. These scores allowed me to make notes of how performant the model was at identifying the various defect classes, how quickly the model converged, and how well the model performed after convergence. I had initially intended to employ Response Surface Methodology (RSM) in order to get a statistically derived prediction for the optimal hyperparameters. Unfortunately, given the limitations of the hardware I had access to, I instead had to opt for a limited single factorial design of experiment.

Upon completion of the training of my model, I tested its performance using the testing data set. This gave me the macro accuracy score:

$$\text{Accuracy} = \frac{CorrectPredictions}{TotalPredictions} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives

I was also able to identify the relative accuracy of each class using the following formula:

$$\text{Relative Class Accuracy for Arbitrary Class } i = \frac{TP_i}{TP_i + FN_i}$$

Where:

- TP_i = True Positives for arbitrary class i
- FN_i = False Negatives for arbitrary class i

In addition to these accuracy scores, it is also important to measure the *ROC-AUC* score, or Receiver Operating Characteristic Area Under the Curve. This score provides a comparison of a models True Positive Rate (TPR) against its False Positive Rate (FPR). This difference is meaningful when there is a large disparity in the majority and minority classes (such as with the data set I am using) because macro accuracy is primarily dependent on the model's ability to predict the majority class. Eg. a model can achieve 99% accuracy if it always predicts the majority class where the majority class makes up 99% of inputs.

3 Results

After testing the model on the testing set, I was given the following accuracy score:

Class	**Test Accuracy (%)**
Overall Test Accuracy	94.823
Center	96.430
Donut	91.536
Edge-Loc	85.589
Edge-Ring	98.819
Loc	82.012
Near-full	91.667
Random	94.301
Scratch	57.022
none	98.693

Table II: Test Accuracy for Each Class

After calculating the *ROC-AUC* score, I was able to generate the following graph:

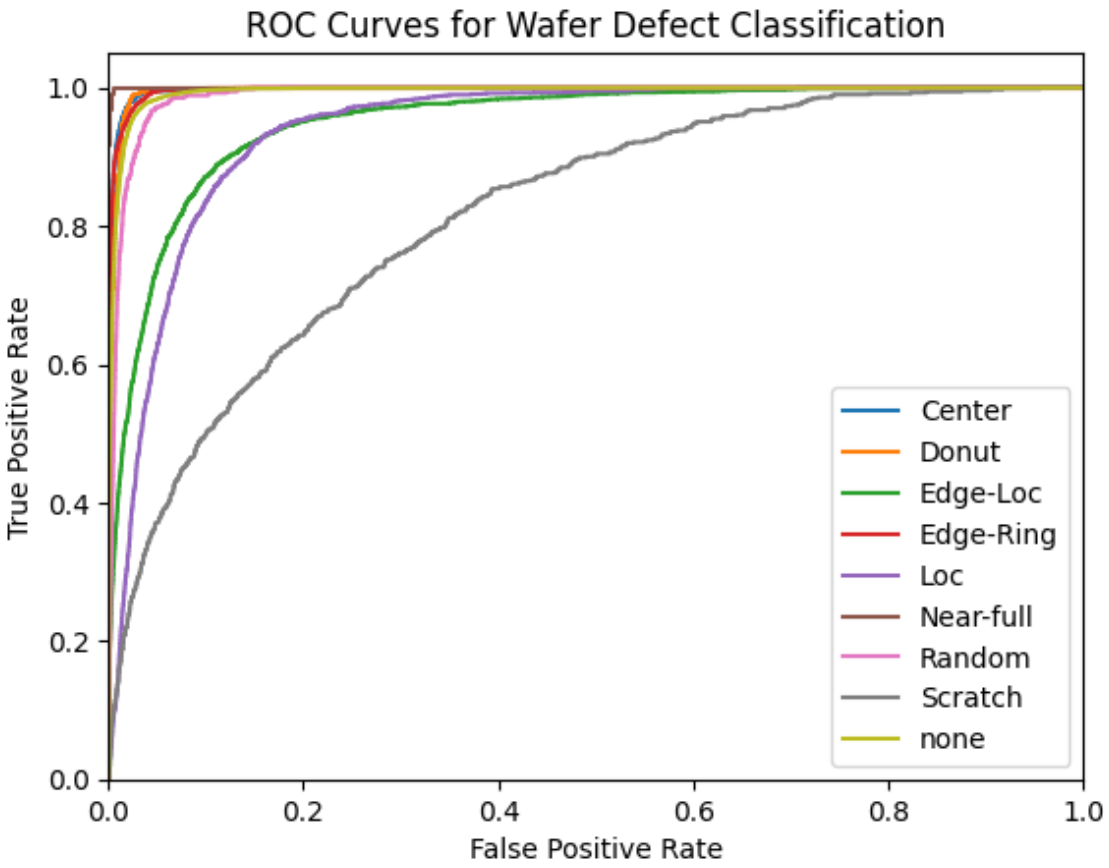

Figure 2: ROC Curve with Macro Score of 0.965

It is clear from these results that the model generally performs well. The high test accuracy of the model is a good indicator of how robust the model is. Many important defect classes (such as "$Edge - Ring$") were also correctly identified, which would be meaningful for semiconductor fabs for identifying issues with wafer clamps or other edge-sensitive processes. However, one may note that certain defect classes (such as "$Scratch$") are not well identified by the model. The "$Scratch$" identification accuracy was just over 57%, which is still significantly better than guessing, but this performance still lags behind all of the other defect classes.

4 Conclusion

Designing and training Convolutional Neural Networks for the identification of defects and anomalies is an incredibly important problem for efficient manufacturing. The ability to automate the discovery of defects and signal processing anomalies could lead to incredible financial savings. This project allowed me to train my own model for defect detection on sensor data displayed as a wafer map. Processing the data in the dataset and optimizing the architecture of the model has been incredibly insightful towards the hurdles of a problem like this.

Ultimately, I am happy with how my model performed. In the future, though, I would like to address why the performance of classes like "$Scratch$" were so low. I posit that there could be modifications in the architecture of the network that would improve the networks ability to identify the implicit patterns in the wafer map that lead to a defect having a true clssification of "$Scratch$". Similarly, I would like to do statistical analysis of the hyperparameters in the form of RSM to optimize the convergence time and overall performance of the model. Finally, I would like to do a slight redesign of the model so that it can classify multiple defects for a single defect (ie. multi-label classification).

5 Discussion

5.1 Anomaly Detection via Advanced Microscopy

Of course, the ability to identify defects using visual data and region maps is extremely useful, as this data is generally very cheap to collect through advanced microscopy and other optical methods. Additionally, the equipment used to collect this data is relatively cheap and easy to maintain, especially compared to methods such as electron beam microscopy and x-ray microdiffraction.

Unfortunately, few solutions exist on the market for large scale anomaly detection via optical methods. Currently, leaders in wafer defect detection tooling such as KLA rely primarily on optical imaging and phase- contrast imaging (such as Differential Interference Contrast Microscopy). These methods have historically been effective at obtaining a sufficient amount of information about the inspected system to classify defects with high accuracy. However, as device designs become ever smaller and device geometries become more and more complex, it will be necessary to design newer systems to overcome these arising issues.

In the burgeoning field of Quantum Information Science, device heterostructures may have layers as thin as 7nm with features in the sub-10nm range. These processes are long, sensitive, and extremely costly. As such, these researchers would benefit greatly from non-destructive and high throughput detection systems.

To solve this problem, I propose an optical detection device using interferometry. Optical profilometry tool manufacturers have already proven that interferometry is viable for non-destructive generation of 3D maps of a device surface. Wyko, in particular, has achieved vertical resolutions of 3Å. I believe that an ML model trained on these 3D maps could be highly performant and satisfy a previously unfilled production niche.

5.2 Process Control and Process Monitoring

At the factory level, detections and classification of defects in lots can also be used for process control and process monitoring. Leveraging the SECS/GEM protocol that many WFE tools are equipped with, it would be trivial to track processing information about each lot.

Using total wafer defects as a metric, a model could be trained to identify troublesome tools, process and process conditions. This would allow for the automatic flagging of tools that require maintainance or servicing. Additionally, this information can be used by process and process integration engineers to design optimal processes.

A model designed for process control could also easily be extended to develop smart dispatch schedules.

Acknowlegements