FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Machine Learning for Wafer Bin Map Defect Pattern Classification
Student:	Bc. Jan Šefčík
Supervisor:	Mgr. Alexander Kovalenko, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

The aim of the thesis is to study and develop a method for pattern defect classification on integrated circuits wafer bin maps. One of the main problems is data labeling as defect pattern/classification depends on the manufacturer. Moreover, wafers can have multiple types of defects on a single wafer. Therefore, an efficient tool to properly classify unlabeled or partially-labeled defect patterns on substrate wafers is highly desired by semiconductor device fabrication companies. In this regard the tasks can be defined as:

- Analyze the latest state-of-the-art approaches for self- and semi-supervised pattern recognition using deep neural networks,
- Define general and specific obstacles, constraints, and recommendations in the field of classifying partially labeled, unlabelled, and multi labeled data.
- Implement and test proposed application

Electronically approved by Ing. Karel Klouda, Ph.D. on 4 February 2022 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Machine Learning for Wafer Bin Map Defect Pattern Classification

Bc. Jan Šefčík

Department of Applied Mathematics
Supervisor: Mgr. Alexander Kovalenko, Ph.D.

May 4, 2022

Acknowledgements

I would like to thank my supervisor Mgr. Alexander Kovalenko, Ph.D. for giving me the opportunity to work on this interesting project but also for his support, guidance and input during the course of the project.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 4, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Šefčík, Jan. *Machine Learning for Wafer Bin Map Defect Pattern Classification*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstract

Automatic classification of defect patterns in wafer bin maps is a challenging problem for semiconductor manufacturers. Recently, progress with supervised approaches has been made, but labeled datasets are usually small and of poor quality. The creation of high-quality datasets is expensive and time-consuming, limiting early production. This work analyzes a self-supervised/semi-supervised learning approaches that use unlabeled data. Based on the resizing problem analysis, this thesis proposed a smaller model that focuses on improving defect classification performance with diverse-sized wafers. The substantial improvement was made with the minor classes, in particular, with Scratch class.

Keywords wafer bin map defect pattern classification, semiconductor manufacturing, convolutional neural networks, semi-supervised learning, self-supervised learning

Abstrakt

Automatizovaná klasifikace vzorů defektů na deskách je náročný úkol pro výrobce polovodičů. V rámci supervizovaného učení byl udělán velký pokrok. Problémem je zisk olabelovaných datasetů. Datasety jsou malé a nemají dostatečnou kvalitu. Jejich vytvoření je drahé a časově náročné. Kvůli těmto důvodům je složité jejich použití při rané produkci. Tato práce analyzuje nejnovější přístupy pro práci s neoznačenými daty. Představuje metody, které vylepšují stávající modely trénované pouze na olabelovaných datech. Na základě provedeného průzkumu navrhuji menší model, který se zaměřuje na řešení problému různorodosti velikostí jednotlivých desek. Významné vylepšení proběhlo u minoritních tříd, hlavně u třídy Scratch.

Klíčová slova klasifikace defektů na deskách, výroba polovodičů, konvoluční neuronové sítě, semi-supervizované učení, samo-supervizované učení

Contents

Introduction **1**

1 Integrated Circuit Manufacturing **3**
1.1 Production of Semiconductor Devices 3
1.1.1 Wafer Fabrication 4
1.1.2 Testing 6

2 State-of-the-art methods **9**
2.1 Model Architectures 9
2.1.1 AlexNet 9
2.1.2 VGG16 10
2.1.3 ResNet 10
2.1.4 Xception 10
2.1.5 RegNet 11
2.2 Supervised Learning 11
2.2.1 Non-CNN Solution 12
2.2.2 CNN Baseline 13
2.3 Semi-supervised Learning 13
2.3.1 Mean Teacher 13
2.3.2 Mixed-Type Defect Patterns Classification 14
2.4 Self-supervised Learning 15
2.4.1 Self-supervised Semi-supervised Learning 15
2.4.2 Self-Supervised Visual Representation Learning 17
2.4.3 Representation Learning for WBMs 18
2.4.4 GAN and WBM Data 19
2.4.5 Multi-label WBM from Single Labeled Data 20
2.5 Technology 22
2.5.1 TensorFlow 22
2.5.2 Keras 22

2.5.3 PyTorch 22

3 Analysis and Design **23**
3.1 Dataset WM-811K 23
3.1.1 Defect Classes 25
3.2 Preprocessing 27
3.2.1 Augmentation 29
3.2.2 Resizing 31
3.3 Transfer Learning 34
3.3.1 Configuration 35
3.3.2 Supervised Learning Results 35
3.3.3 Semi-supervised 36
3.4 Misclassification 37
3.4.1 Confusion matrix 37
3.4.2 Edge-Local vs. Local 38
3.4.3 Scratch Defects 38

4 Realization **41**
4.1 Technology 41
4.1.1 Keras and Pytorch 41
4.1.2 Data Analysis 42
4.2 Dataset 42
4.2.1 Preprocessing 43
4.2.2 Augmentation 43
4.3 Model 43
4.3.1 Development 44
4.3.2 Normalization 45
4.3.3 Semi-supervised Learning 45
4.3.4 Self-supervised learning 46

5 Results **49**
5.1 Supervised 49
5.1.1 Single vs. Multiple Input 49
5.2 Augmentation 50
5.2.1 Same Method for All Classes 51
5.2.2 Different Method for Each Class 52
5.2.3 Batch Generation 52
5.2.4 Label smoothing 53
5.2.5 Size of labeled data 53
5.3 Semi-supervised 54
5.3.1 Size of labeled data 55
5.4 Self-supervised 56
5.4.1 Size of labeled data 57

Conclusion **59**

Bibliography **61**

A Acronyms **65**

B Contents of Enclosed CD **67**

List of Figures

1.1 The fabrication process of the integrated circuit 3
1.2 Fabrication process of the photo mask 5
1.3 None-directional vs. directional etching. 5
1.4 Yield with different IC sizes 6

2.1 Architecture of ResNet vs. RegNet 12
2.2 Mean teacher architecture overview 14
2.3 Architecture of SS-CDGMM 15
2.4 Illustration of S4L Rotation 16
2.5 Scheme of the pretraining method (WaPIRL) 19
2.6 Multi-label defect from single labeled wafers 21

3.1 Histogram of the distribution of labeled classes 24
3.2 Histograms of the wafer area 25
3.3 Histograms of the area for each defect 26
3.4 Example of failure patterns 27
3.5 Distribution of wafer area in dataset 28
3.6 WBM data representation 28
3.7 Examples of the augmentation methods 30
3.8 Example of the resizing loss 32
3.9 Confusion matrix of Xception predictions 37
3.10 Misclassification of Local defect pattern 38
3.11 Misclassification of Scratch defect pattern 39

4.1 Histogram of pseudo-labels prediction confidence 46

5.1 Confusion matrix of predictions 50

List of Tables

3.1 Data samples 23
3.2 Class label distribution 24
3.3 Results of wafer representation test on VGG-16 29
3.4 Results of the different input size 32
3.5 Results of supervised learning approaches 35
3.6 Supervised learning with popular CNNs results 36

5.1 Results of model with single and multiple inputs 50
5.2 Results of model with and without normalization 51
5.3 Class label distribution in training set 52
5.4 Results of supervised learning with different types of augmentation 53
5.5 Results of data samplers test 54
5.6 Results of model with and without smooth labeling 54
5.7 Results of the size of labeled data 55
5.8 Results of semi-supervised relative improvement 56
5.9 Results of semi-supervised learning - pseudo labels confidence . . . 57
5.10 Results of self-supervised learning - WaPIRL 58

List of Source Code Listings

1 Creation of balanced batch. 33
2 Class weights calculation. 34
3 Update of import metrics 42
4 Architecture of a block single input network 44

Introduction

The Production of semiconductor devices is a complicated manufacturing process that includes hundreds or even thousands of successive steps to create integrated circuits (ICs), mainly on silicon wafers. Because this process is both challenging and expensive, the goal is to manufacture wafers with zero defects. The patterns of defects can occur during this complicated process at any time. Each defect is linked to a specific step in the production process. Thanks to the information that caused the fault, it will be easier to fix the source of the defect faster.

This work is focused on the improvement of single-label classification of defect patterns on wafer bin maps (WBMs), that should lead to a better yield of the wafers. The conventional practices that rely on an experienced naked eye are not optimal. Here the machine learning comes to play. Convolutional neural networks (CNNs) are one of the most successful solutions for the image classification problem. Nonetheless, their performance heavily depends on the quantity and quality of the labeled data. This field of chip fabrication produces very specific data in comparison to the traditional computer vision tasks. The other problem is the lack of free labeled WBM data. In general, creation of labeled datasets is very expensive and time-consuming.

The current state-of-the-art self-supervised and semi-supervised learning approaches perform well against the supervised approach in basic computer vision tasks. This work proposes methods to improve the imbalanced datasets for supervised learning which affects the semi-supervised and self-supervised approaches.

In this work, the dataset WM-811K with over 800,000 samples will be used, but only 21 % of the samples have labels. In the analysis chapter, constraints and recommendations for augmentation methods, resizing and use of transfer learning are described. Further, I tested multiple methods of dataset balancing to improve the supervised learning. Then, I tried to apply the same principles to the semi-supervised (pseudo-labels) approach. I also demonstrated the use of a smaller CNN model with self-supervised (WaPIRL) method.

Chapter 1

Integrated Circuit Manufacturing

This chapter briefly describes the manufacturing process of integrated circuits and how they are tested (process control monitoring, wafer bin map). For example, the unit probe test creates data in the form of wafer bin maps, which are the inputs for this work.

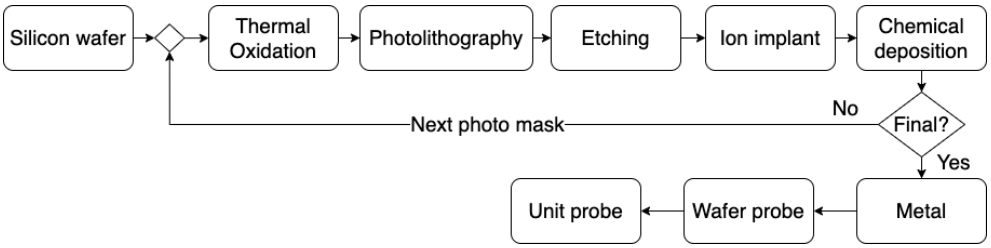

Figure 1.1: **The fabrication process of the integrated circuit.** The fabrication process of the integrated circuit starts with clean silicon wafers that are exposed to oxygen at height temperatures creating the insulator. The photolithography and etching creates the pattern for ion implantation which changes the properties of silicon. The chemical deposition creates thin film. Then, metal films and interconnections are done. The final step is the testing of each wafer unit.

1.1 Production of Semiconductor Devices

Fabrication of semiconductor devices is a multi-step sequence of photolithographic and chemical processing steps during which electronic circuits are slowly created on a wafer, made of pure semiconducting material. The key to

successful fabrication is the ability to change the properties of semiconducting material selectively. The following section describes the essential part of the fabrication process [1].

1.1.1 Wafer Fabrication

Silicon is the most common material for ICs fabrication. Manufacturers also use different materials for specific applications, such as GaN for LED diodes or SiC for high voltage power devices. The dominant IC material is silicon because it is inexpensive and pretty easy to work with, obtain and retain. ICs may be simple with a small number of parts or complex with millions of transistors [2].

1. **Silicon wafer**

   Silicon ingot is sawed up into wafers, those are then cleaned and polished. Their notch or flat defines the crystallographic orientation of silicon wafers. The integrated circuit in the form of the wafer segment is referred to as a die. The Group of wafers is called a lot, and it usually contains 25 wafers. Every wafer has its lot and wafer number.

2. **Thermal oxidation**

   The exposure of raw silicon to oxygen or water vapor at high temperatures results in the formation of silicon dioxide layer, see equation 1.1 (the coating is stable at high temperatures). This creates a perfect barrier which acts as an insulator.

$$Si + O_2 \rightarrow SiO_2 \tag{1.1}$$

3. **Photolithography**

   Photolithography defines the patterns that are used in a combination with etching, to pattern the deposited thin films. Combined with ion implantation, it can change the properties of silicon. Photolithography creates patterns in photoresist using ultraviolet light, and patterned reticle, see Figure 1.2.

4. **Etching**

   The current state-of-the-art etching method is dry etching, which uses halogen-containing gasses. Dry etching could be non-directional or directional. Wet etching usually uses acids and is primarily non-directional.

5. **Ion implantation** Elements such as boron (B), phosphorus (P), arsenic (Sb) or antimony (As) can be used to modify predictably the electrical properties of the silicon. An ion implantation is the most common method to introduce these dopant impurities into the wafer. Positively

charged ionized dopants are accelerated to have enough energy so that when they impact on a target wafer's surface they can penetrate to a certain depth.

6. **Chemical vapor deposition**

   Gases or chemical vapors react to form deposited films (at low pressure). The thickness of these films is in the range from a few micrometers to nanometers and are deposited/grown on a wafer surface. Reactions can be induced by heat, high-frequency energy, or light, depending on the enhancement type, e.g., plasma or photon.

7. **Metal**

   There are many metals used to create thin films to provide interconnections. Chemical Vapor Deposition (CVD) makes better films than sputter deposition, but CVD cannot deposit all metals. Because of evolution in development, more metal layers were needed to connect devices in large ICs. The key for multi-layer metal scheme was the development of chemical mechanical planarization (CMP), which creates a fully planar surface.

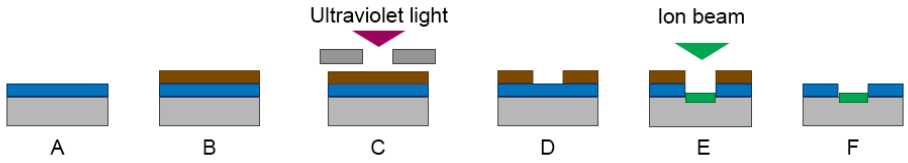

Figure 1.2: **Fabrication process of the photo mask**: **A)** Thin silicon dioxide layer, **B)** Coating with photoressit, **C)** Exposure of the photoresist with a patterned reticle, **D)** Developed photoresist, **E)** Ion implantation, **F)** Striped photoresist.

Figure 1.3: None-directional vs. directional etching.

1.1.2 Testing

A fabricated semiconductor wafer undergoes evaluative testing to ensure the integrated circuits are formed correctly and that they operate in a desired manner. These testing steps are called Process Control Monitoring and Wafer Sort.

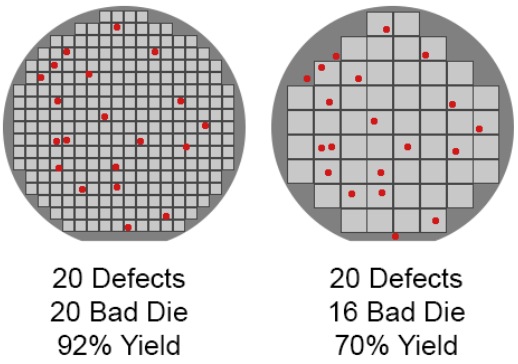

Figure 1.4: **Yield with different IC sizes** is different based on the die size and the defect density. The left wafer has 264 dice and the right wafer has just 54 dice.

1. **Process Control Monitoring** (PCM) [3] measures special structures that are used to monitor technology-specific parameters. These special structures are fundamental parts of the final IC. The test structures are distributed across the wafer surface, either in die sites or in the scribe lines between dice. PCM data can be used to predict potential process issues.

2. **Wafer Sort** (WS) The PCM measurement is followed by comprehensive tests performed on each IC die, commonly called Wafer Sort or Unit Probe [4]. The die may either pass or fail the testing procedure. The information about passing/failed die is stored in an electronic file named Wafer Bin Map (WBM). The wafer map also specifies excluded dice as edge dice around the wafer circumference, test structures and alignment dice. So, after cutting the wafer, failed and excluded dice can be separated from the good dice.

   In some cases, the die may not pass a test due to a problem with the probecard setup. For example, if the probe needle is a little out, it cannot measure the die correctly and a probe defect will occur.

Yield is a quantitative measure of the quality of a semiconductor process. The yield at WS is calculated as the ratio of good die to all die on an individual wafer that entered into the WS.

Correctly classified defect patterns in wafer maps can help identify the root cause of the issue and increase semiconductor productivity or even yield. There are many types of possible defect patterns. Here, I will focus on defects from dataset WM-811K[1].

[1] http://mirlab.org/dataSet/public/

Chapter 2

State-of-the-art methods

This chapter describes learning methods derived from the supervised and unsupervised approaches and their basic principles. Supervised learning, approach where all the data have labels, is commonly used for classification, regression, etc.; unsupervised learning is used for clustering, outlier detection, etc. For each method, state-of-the-art models and their performance in various benchmarks are shown. The popular state-of-the-art deep learning frameworks TensorFlow, Keras, and Pytorch are described below.

Convolutional neural networks (CNNs) have achieved massive success in many computer vision tasks. A brief history of the evolution is presented in this chapter, from AlexNet [5] through Residual blocks of ResNet [6] to the recently published RegNet [7]. No transformers are presented here since, for this field, those are considered as inappropriate networks, because of the lack of large datasets needed to train the transformers.

2.1 Model Architectures

Popular CNN models, their development history and the progress that has been made since AlexNet are described in this section. All of them are used in solutions presented by the papers described below (except RegNet and Xception).

2.1.1 AlexNet

AlexNet was published in 2012 in the paper named "ImageNet Classification with Deep Convolutional Neural Networks" [5]. The network architecture has eight layers, five of them are convolutional, and three are fully connected. The novelty that AlexNet paper proposed was using Rectified Linear Unit (ReLU) [8] instead of *tanh* function (standard at the time). Thanks to the multi-GPU training capability the training time was reduce. Authors also introduce overlapping neurons for pooling which are harder to overfit.

Because of 60 million parameters, overfitting proved to be an issue. They have used dropout layers and data augmentation. The dropout layer is 'turning off' neuron sets by the probability threshold. This behavior forces the neurons to work with more robust features. On the other hand, dropouts increase training time.

2.1.2 VGG16

VGG16 was developed in 2014 and presented in the paper called "Very Deep Convolutional Networks for Large-Scale Image Recognition" [9]. VGG stands for Visual Geometry Group, which is the research group's name, and the number 16 refers to the number of layers.

It has improved over AlexNet [5] by replacing the 11×11 and 5×5 kernels with multiple 3×3 kernels, one after another. That increased number of ReLU [8] units and the decision function is more discriminative. It has fewer parameters than AlexNet [5] (27 per channel instead of 49 per channel).

VGG16 [9] uses 1×1 kernels to make the decision function more non-linear without any change in receptive fields. The small-size kernels allow the network to have many weight layers, which leads to better performance.

2.1.3 ResNet

ResNet stands for Residual Network, and it was published in 2015 in the paper called "Deep Residual Learning for Image Recognition" [6]. The number after the name ResNet stands for a number of networks layers. Because of the back-propagation and vanishing gradient problem, there were not any deeper VGG networks than VGG19 [9] back in the day. Since the publishing of ResNet, there have been proposed many variants of it.

The vanishing gradient problem occurs with deeper CNN because of the back-propagating values to the input layer, and the layers become less and less significant. They resolved the issue by by placing the skip connections between the residual blocks, see Figure 2.1, which are repetitively used through the network.

Two types of mapping were presented: the Identity connection and the Projection connection. The network learns the mapping using $x \rightarrow F(x) + G(x)$ where $G(x) = x$ is the identity function, and the shortcut is called Identity connection. In the case of different dimensions (stride > 1), the Projection connection is used. The function $G(x)$ changes the dimensions of the input x to output $F(x)$.

2.1.4 Xception

Xception stands for "Extreme Inception," it was developed by Google researcher Francois Chollet and proposed in the paper with title "Xception:

Deep Learning with Depthwise Separable Convolution" [10]. It is an interpretation of the Inception model presented in the paper named "Going Deeper with Convolutions" [11].

Xception uses shortcuts between convolutional blocks as ResNet does and uses the depth-wise separable convolution. The architecture is divided into three parts: Entry flow, Middle flow, and Exit flow. The input data goes through the Entry flow, then it repetitively goes through the Middle flow, and then it goes through the Exit flow.

Depthwise Separable Convolution is a type of convolution which is done separately for each channel. For example, three separate kernels are used for each channel with RGB images.

The other difference from the Inception network is that Xception has no non-linear function between the depth-wise and point-wise operations. They have proven that Xception achieved better performance with no intermediate activation function between these two types of convolutions.

2.1.5 RegNet

RegNet stands for RNN-Regulated Residual Networks, and it was submitted in 2021 as a paper called "RegNet: Self-Regulated Network for Image classification" [7].

The problem with ResNet skip connections is that each block focuses on learning its residual output. If the information learned inside the block cannot be reused again, it tends to be forgotten. See the Figure 2.1 for visualization of this problem.

They proposed a regulatory mechanism that is parallel to ResNet skip connections. Further, they used convolutional recursive neural networks (ConvRNNs) as the encoder of Spatio-temporal memory. The ConvRNNs are added to the ResNet building blocks and the "bottleneck" blocks. The detailed RegNet blocks architecture is on Figure 2.1.

Best performance by error rate was achieved by RegNet-20 with ConvLSTM (**7.28 %**) over ResNet-20 (8.38 %) on the CIFAR-10. This method can also be used with other variants of ResNet architectures.

2.2 Supervised Learning

Supervised learning is a type of machine learning approach to train models. The input data has to have a label that describes the output value. The aim of the learning process is to learn a mapping function used to get/predict labels of unseen data accurately. Supervised learning is appropriate for both classification and regression tasks. Labels work as a teacher who knows the correct answers, and wrong predictions are corrected during the iterative process of learning. The learning algorithm is stopped when the model achieves decent performance.

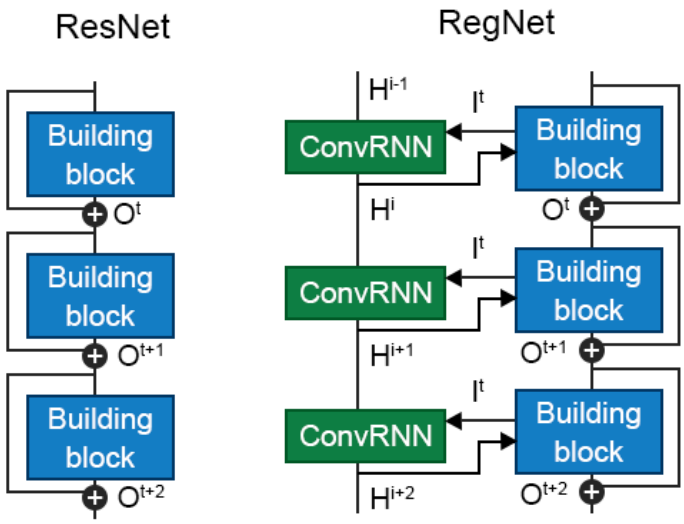

Figure 2.1: **Architecture of ResNet vs. RegNet**. The ResNet is build from residual blocks with skip connections. The RegNet is also using the ResNet's residual blocks and adds to it the convolutional recurrent network (ConvRNN). That should reduce the loss of information in each residual block.

2.2.1 Non-CNN Solution

Paper called "Wafer Map Failure Pattern Recognition and Similarity Ranking for Large-Scale Data Sets"[12] proposed a solution which uses rotation and scale invariant features.

In Jupyter Notebook[2] Ashish Patel implemented the feature extraction from the the paper[12] but only for the labeled pattern data (25 519 samples) from the WM-811K dataset.

1. **Density-based features:** The wafer is divided into 13 parts, and the defect density is calculated for each region. For example, the center defect has the most failures in the middle region.

2. **Radon-based features:** Radon transformation is used to get a 2D image representation of the wafer from multiple projections. Because of the difference in wafer size, he is extracting 40 features from radon transformation.

3. **Geometry-based features:** The most notable region of the wafer is extracted as a maximal component, and the features are the area, length, etc. of this part.

[2] `https://www.kaggle.com/code/ashishpatel26/wm-811k-wafermap/notebook`

The labeled part of the dataset was split into a training, test sets in default proportion 25 %. He achieved a training accuracy of 80.36 % and **79.04 %** on the testing dataset. The paper presented here as an option for a non-CNN solution, and because of the type of features he used, they do not have to use any re-scaling of the wafers.

2.2.2 CNN Baseline

The best results with supervised learning were published in the paper [13], where they used popular CNN models (AlexNet, VGG16, ResNet18, ResNet50) and trained them with a larger training set and only **10 %** sized test set. See the Section 3.3 for the results and more details. The best performance has model VGG16 with macro F1 score **0.781**.

2.3 Semi-supervised Learning

Humans can understand concepts after seeing just a few (labeled) examples. Semi-supervised learning [14] is based on this principle. Semi-supervised learning describes a group of algorithms that are learning from both labeled and unlabeled data (from the same distribution). The main difference between semi-supervised learning techniques is how information is getting from unlabeled data.

2.3.1 Mean Teacher

A Paper named "Mean are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results" [15] proposes a method that averages model weights instead of predictions. Their goal was to create a better method than Temporal Ensembling [16]. See Figure 2.2 for the architecture details.

Temporal Ensembling extends the Π-model that focuses on the consistency of outputs between the two versions of the same network (different dropout and augmentation) with the same input. Temporal ensembling takes into account the predictions from previous training epochs.

Classification cost is difference between of student softmax predictions with one-hot encoded labels. **Consistency cost** is difference between students and teachers softmax predictions.

Both models predict the input label with different noise inside (dropout layers). The student's prediction is compared with the correct label using classification cost and teacher output using consistency cost. In a case with unlabeled data, the consistency cost between student/teacher models is only calculated with different noise applied.

After the update of the student's weights (gradient descent), the teacher model uses the exponential moving average (EMA) of the student model's

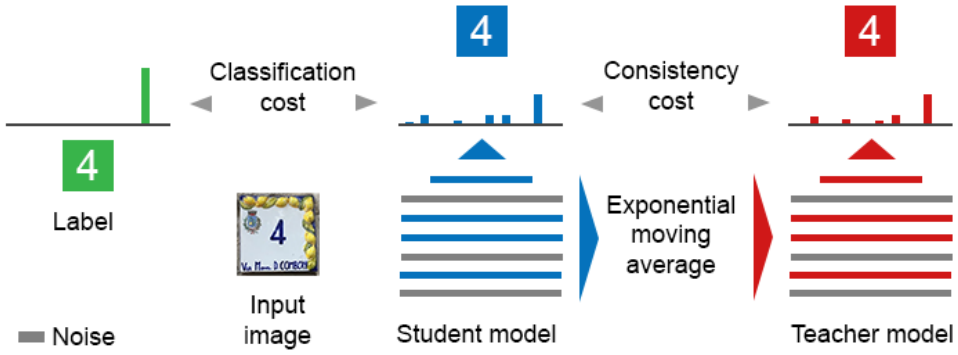

Figure 2.2: **The architecture of Mean teacher** consists of two networks: Student and Teacher. Both models predict the label for the same input but with different noise (Dropout layers). Student's predictions are compared with the correct label using classification cost and with teacher's prediction by consistency cost. In case of unlabeled data only the consistency cost is calculated. The student's weights are updated by gradient descent and the teacher's weights are exponential moving average of the student's weights.

weights instead of sharing them directly. With EMA, it can update its weight every step in the epoch. The weighted average also improves all the layer's outputs (it has a better intermediate representation). It is better for online learning with large datasets than temporal ensemblings. Both models can be used for prediction, but the teacher model performs better than the student.

2.3.2 Mixed-Type Defect Patterns Classification

The paper called "Semi-Supervised Multi-Label Learning for Classification of Wafer Bin Maps With Mixed-Type Defect Patterns" [17] proposed a semi-supervised solution for multi-label classification using a deep convolutional generative model. The classification of mixed-type defects is more complex than single defect classification because the classifier should classify each defect without knowing the number of defects on the wafer.

They were working just with four defects and their combinations (16 categories). The SS-DGM with a single discriminative network was having a problem learning features of all 16 classes. They proposed a solution that is an extension to the SS-DGM [18], they called it a semi-supervised convolutional deep generative multiple model (SS-CDGMM).

The structure of SS-DGM is extended with CNN to extract local invariant features, which together create a semi-supervised convolutional deep gener-

ative model (SS-CDGM). They propose multiple latent class variables (each for a different defect pattern) with multiple discriminative networks, unlike the SS-DGM, using only one. See Figure 2.3 for the scheme of SS-CDGMM. SS-CDGMM can be used for generating new WBMs. In the case of generating the WBM data from a labeled sample, the real labels are used.

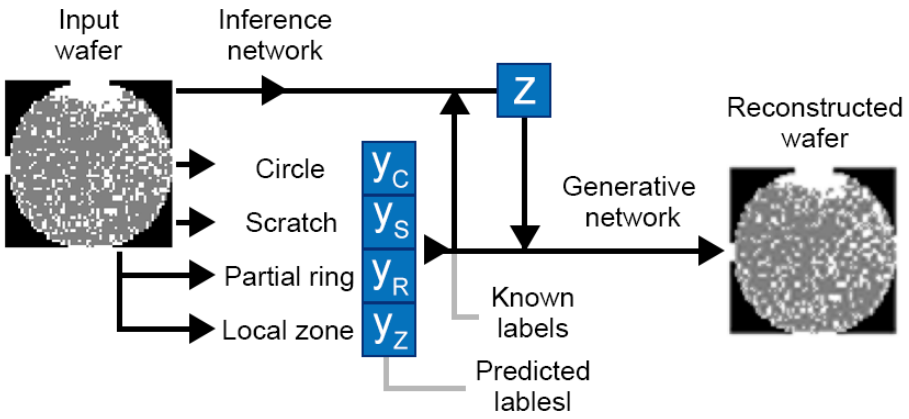

Figure 2.3: **Architecture of SS-CDGMM**. Each pattern has its own network to predict its label. The final labels are concatenated together to the inference network. Then the inference network's latent variable is used as an input for the generative network which reconstruct the WBM data. In case of labeled data, the real labels are used instead of the predicted labels.

2.4 Self-supervised Learning

Self-supervised learning [19] is a general learning mechanism that uses replacing/pretext tasks. These tasks are designed in such a way that solving them requires learning helpful representation. This technique has a broad range of applications beyond the scope of image processing. Self-supervised learning is a sub-class of unsupervised learning.

2.4.1 Self-supervised Semi-supervised Learning

A Paper name "Self-supervised Semi-supervised learning" [20] proposed a method that combines both learning approaches. They developed a framework of self-supervised semi-supervised learning (S^4L). The main idea of their work is to train the network only with 10 % of labeled data (in the case of a completely labeled dataset). Next step is using one of the proposed methods. For the final fine-tuning, they used again 10 % or 1 % of labeled data from the ILSVRC-2012 dataset.

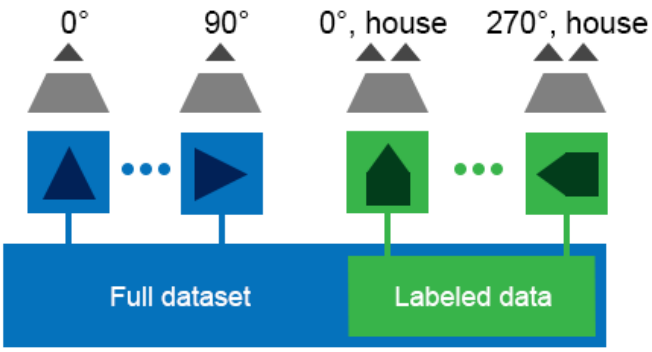

Figure 2.4: **Illustration of S^4L Rotation**. The model is using both labeled and unlabeled data. For each image, new input images are created by rotation (0°, 90°, 180°, 270°). The model is predicting the label and the angle of rotation. In case of unlabeled data only the angle is predicted.

1. S^4L**-Rotation** prediction is used as a pretext task with possible degrees being $[0°, 90°, 180°, 270°]$. The network predicts the rotation angle with unlabeled data and additionally the label for labeled data.

2. S^4L**-Exemplar** uses cropping, random horizontal mirroring, and HSV color space randomization to create eight different instances of one image in a batch.

Self-supervised methods are evaluated by their learned representation and how useful that is. They achieve this by treating the model as a fixed feature extractor with a linear logistic regression model on top of it. The regression model is trained on a different dataset.

They have shown that the size of the validation set is not that important, validation sets with size of 1000, 5000, and over 50000 images from ILSVRC-2012 were tested. The small validation set selected the same model as the best model and the large validation set selected the same model. The evaluation of the model works well even with a small-sized validation set.

Best results were achieved with S^4L-Rotation with the improvement of **3.4 %** over the supervised baseline (all methods were using the ResNet50V2 architecture). Proposed methods are complementary to each other with pseudo samples [21], virtual adversarial training (VAT), and VAT with entropy minimization.

They have proposed a Mix Of All Models (MOAM) which combines all methods in three steps. MOAM has achieved a **0.5 %** better top-5 score than the Mean teacher[15].

2.4.2 Self-Supervised Visual Representation Learning

The Paper called "Revisiting Self-Supervised Visual Representation Learning" [22] is an overview of CNN models and pretext tasks that can be used in visual representation methods, focusing on the architecture design and the quality of representation.

Most pretext tasks are inspired by self-supervised methods from natural language processing which uses missing word prediction based on the context.

Unsupervised image-based pretext tasks:

1. **Patch based**

   It predicts the relative position of the patch. The input is two image patches, one is the anchor, and the other is the query patch. The network has to predict the relative position of the query patch with respect to the anchor patch in 8-neighbors positions(eight possible positions).

2. **Jig-saw**

   The network is learning the image representation by solving the Jig-saw puzzle. Part of the image is split into nine tiles that are shuffled, and the network has to put them back onto the right positions. The idea is that network will learn high-level features/relations like the position of the mouth with respect to the eyes without their low-level features like color, texture, etc.

3. **Fill the blank**

   The part of the image is hidden, and the network tries to predict the missing part. The idea is that the network will learn repetitive structures of the domain like buildings with windows and doors.

4. **Rotation**

   Prediction of the image rotation is the most popular pretext task. The network predicts which rotation from pool $[0°, 90°, 180°, 270°]$. It does not make any sense to predict rotation, but it has been proven that rotation works. The idea is that the network will understand the orientation, e.g., the sky is on the top of the image, and the grass is on the bottom.

5. **Colorization**

   This pretext task is about the prediction of the colors from a grayscale image. For the object with multiple color variants, it has several right solutions. The intuition is that the network will have the same understating about the colors, e.g., the sky is blue, or the grass is green.

   There is more pretext task-specific to video, natural language processing, and sound. Also, combinations of the mentioned pretext task are commonly used, e.g., Relative position with colorization.

6. **Their combination** The jigsaw puzzle is a combination of jigsaw and colorization. Jigsaw++ is a combination of the Jigsaw puzzle with clustering-based pseudo labels [21].

They have tested six CNN architectures (variations of ResNet50 and VGG) and four self-supervised approaches mentioned above. They have shown that pretext tasks should be compared with connection to the architecture.

2.4.3 Representation Learning for WBMs

The paper called "Self-Supervised Representation Learning for Wafer Bin Map Defect Pattern Classification" [13] proposed a solution that consists of two stages:

1. **Self-supervised pretraining** contains training of the CNN encoder to capture high-level features in WBMs.
2. **Supervised fine-tuning** transfers weights of the CNN from pretraining to the prediction of WBM pattern failures with labeled data.

The proposed framework is called wafer-oriented pretext-invariant representation learning (WaPIRL), see Figure 2.5, and its architecture consists of five components:

1. **Data augmentation operator** $t(\cdot)$ produce pairs of WBMs for the self-supervised learning
2. **Encoder network** $f_{\Theta}(\cdot)$ (VGG, ResNEt, AlexNet)
3. **Projection head** $g_{\Phi}(\cdot)$
4. **Memory bank** M
5. **Self-supervised contrastive loss function** L_{SSCL}

They have experimented with five different options for WBM augmentation:

1. **Cropping** - random selection of the rectangular area $[0.5, 1.0]$ and resizing it to the original resolution.
2. **Cutout** - maximum of four regions were selected and removed (replaced by zero values).
3. **Noise addition** - add random noise using Bernoulli distribution with probability 0.005.
4. **Rotation** - random angle from range $[0°, 360°]$

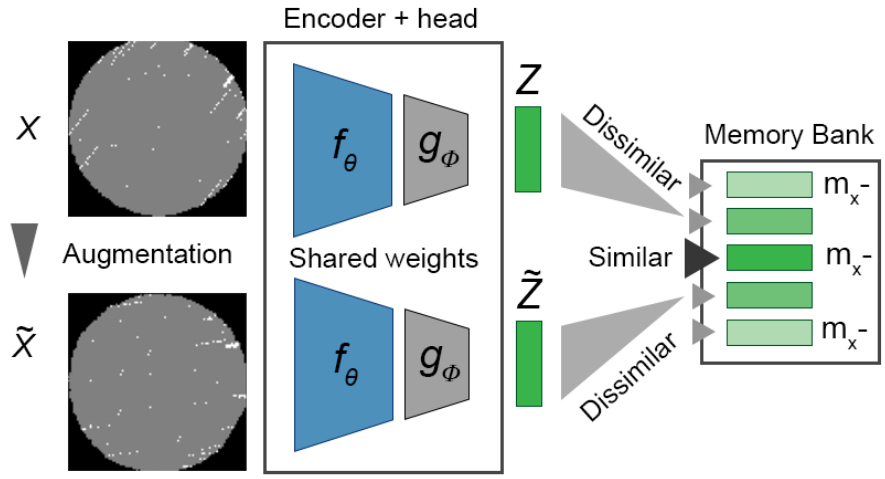

Figure 2.5: **The self-supervised pretraining in the framework WaPIRL** has five main components: Data augmentation, encoder, projection head, memory bank and contrastive loss function. The original wafer and its augmentation are trained using contrasting self-supervised learning. The embeddings from projection heads are compared with each other. The heads are replaced with softmax layer for the classification part.

5. **Shifting** - random translation in horizontal and vertical direction $[-0.25, 0.25]$

See Figure 3.7 for visual examples of wafer augmentation. They have tested its performance on the WM-811K dataset on popular CNN architectures (AlexNet, VGG16 and versions of ResNet). The best performing architecture was VGG16 with crop pretext task with 5 % of labeled data and more. Test with 1 % labeled data using the rotation as pretext task with VGG16 was the only one which was better. They were using their custom training, test set split where the test set was only 10 % of the labeled dataset size.

2.4.4 GAN and WBM Data

The Paper named "Using GAN to Improve CNN Performance of Wafer Map Defect Type Classification" [23] uses the WM-811K dataset with horizontal stripe (labeled WBMs only and exclusion of high/low resolutions). For more about this dataset, see chapter Analysis and design section 3.1 Dataset. Classes are significantly unbalanced, and many classes do not have enough samples. They proposed a balancing by using generative adversarial networks (GANs).

They have used architecture with three convolutional layers with max-pooling and one dense layer before the last layer with softmax. GAN consists of two neural networks (Generator, Discriminator) that are trained together. The role of the Discriminator is to tell if the input sample is real or fake.

Repetition of this process teaches Generator to generate better and better samples that Discriminator has to distinguish. There has been massive progress in GAN, and one of the improved versions is Deep Convolutional GAN (DCGAN) [23]. They use the DCGAN structure to get more WBMs.

They have created custom smaller datasets for testing the augmentation methods from the WM-811K dataset. The training set has 7 560 samples, and the testing set has 1000 samples. Their datasets were more balanced than the original dataset. They have generated only minor classes (Location, Edge location, Center, Scratch, Random, Near full, and Donut) with DCGAN.

1. **Original dataset (DS0)**
2. **DCGAN dataset (DS1)** - DS0 + generated minor classes using CGAN
3. **Classic augmentation dataset (DS2)** - axis flip and rotation of data from DS0
4. **DCGAN with Classic augmentation (DS3, DS4)** - DS0 + CGAN data from DS2 (DS3 - all classes, DS4 - low accuracy classes only)

The most problematic classes were Location, Edge Location, and Scratch. The accuracy of the scratch class with the DS0 dataset was only **71.7 %**, and after the augmentation, it was **88.3 %**. There was no larger improvement in other classes. For example, there was no improvement in accuracy for Edge local defect.

2.4.5 Multi-label WBM from Single Labeled Data

The paper called "Mixup-based classification of mixed-type defect patterns in wafer bin maps" [24] focuses on the classification of mixed defect patterns. Neural networks made big progress in single defect pattern classification, but the mixed-type defects did not get much attention. The reason is the lack of multi-labeled data, having enough data is an essential requirement for model training. They propose a single pattern defects method for training CNN to classify mixed-type defects. Their method generates the multi-labeled data on the fly.

Consider single labeled WBMs, which are resized to a fixed dimension, and all test bins are replaced with a value of one, while non-defective and outer regions are assigned a value of zero.

Original Mixup [25] creates virtual training instances that take an average of two training instances. They proposed **Summation Mixup** which is the sum of input instances with an upper bound.

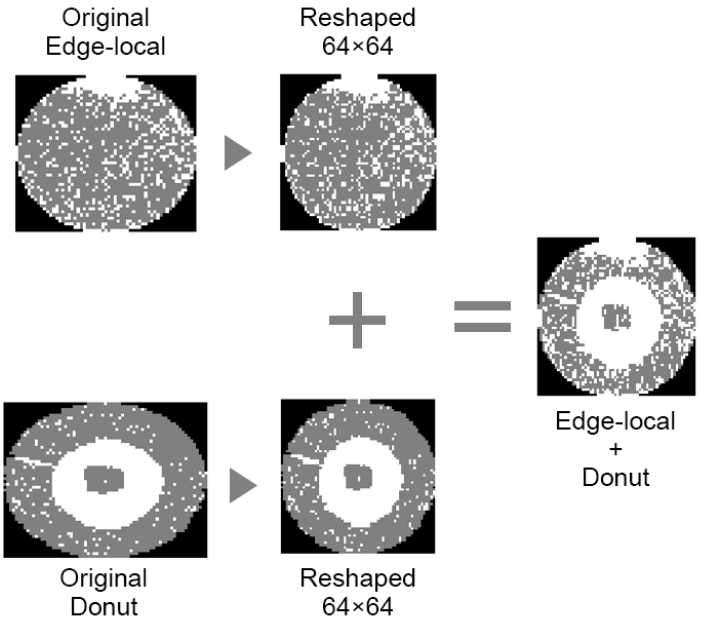

Figure 2.6: **Multi-label defect from single labeled wafers**. Adding two single-labeled wafers together creates multi-labeled wafer. This method could be used only for suitable patterns (non-intersecting). They have also tested this method on triple-labeled data.

They were using the dataset WM-811K for experiments. For more about this dataset, see chapter Analysis and design section Dataset3.1. This method needs mixed-type data only for the evaluation of final performance. See Figure 2.6 for the generation process of mixed-type defect data.

The creation of two datasets (combination of two and three defects) was done by random sampling of two (three) WBMs from one of the classes (Center, Donut, Edge-Loc, Edge-Ring, Loc, and Scratch). They have not used Random and Near-Full classes because they cover a big area of WBM. Domain experts manually sorted out the non-realistic WBMs. This process results in a datasets consisting of 1500 and 6000 WBMs with two and three different types of defects, respectively.

The method have been tested using five CNN architectures (AlexNet, VGG-16, ResNet-18, ResNet-34, ResNet-50). They have created their own test sets from single labeled WBMs (100 and 300 samples). The summation method shows improvement across all tested architectures.

2.5 Technology

In this section, the most popular frameworks (TensorFlow, Keras, and PyTorch) for deep learning are briefly described, compared and contrasted [26].

2.5.1 TensorFlow

TensorFlow presented in the paper named "TensorFlow: A system for large-scale machine learning" [27] is an open-source deep learning framework with multiple levels of abstraction and support for any platform, e.g., Android.

It has the Serving framework for easier deployment of trained models to production. Keras has been integrated into TensorFlow, and both of them have high-level APIs. It is possible to define a model using Keras interface, and the rest can be implemented in TensorFlow. Tensorflow is usually used for working with large datasets where there is a need for high performance.

2.5.2 Keras

Keras [28] is an open-source library written in Python, and it is focused on fast experiments with neural networks. It is a high-level API for TensorFlow, The Microsoft Cognitive Toolkit (CNTK)[3], and Theano[4].

Keras cannot be used for low-level computing and it was integrated into TensorFlow in 2017, but it is still able to work independently. Keras is the best for prototyping with small datasets, and it has support for multiple back-ends.

2.5.3 PyTorch

Pytorch [29] was developed by Facebook's AI researchers and became open-sourced in 2017. It is a framework for machine learning based on the Torch library using Lua language, and it has C++ and Python interfaces. It posses 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.

Compared to TensorFlow, Pytorch has limited visualization, and worse model deployment, due to the lack to the Serving framework or other alternative. In comparison with Keras, Pytorch is faster with better debugging options.

[3] `https://github.com/microsoft/CNTK`
[4] `https://github.com/Theano/Theano`

Chapter 3

Analysis and Design

This chapter describes the detailed analysis of the dataset, augmentation and preprocessing of WBM data. This chapter defines constrains and recommendations for classification with partially labeled, unlabeled and heavily imbalanced data. In the section Transfer Learning are tested popular CNNs with the analysis of misclassificaitons.

3.1 Dataset WM-811K

The dataset used in this work was provided by MIR lab[5], and it is a recommended dataset for wafer map classification task. The dataset contains nine classes (Center, Donut, Edge-Local, Edge-Ring, Local, Random, Scratch, Near-full, None), see Figure 3.4 for visual examples of each defect. The wafers have multiple sizes and resolutions (not every wafer has a square shape, which is defined by the size of the IC). Dataset has information about die size, lot number, index in the lot, defect type, and if the sample belongs to the training, test set.

Table 3.1: Data samples

	waferMap	dieSize	lotName	waferIndex	trainTest	failureType
0	[[0,...	1683	lot1	1	[[Training]]	[none]
1	[[0,...	600	lot2	24	[[Test]]	[Edge-Loc]
2	[[0,...	600	lot3	4	[]	[]

The dataset contains **811 457** wafer bin maps from 47 543 lots (not every lot has all 25 wafers). Only **3.1 %** (25,519) wafers have pattern labels, **18.2 %** have non-pattern labels and the rest **78.7 %** is unlabeled. The distribution of

[5] `http://mirlab.org/dataSet/public/`

WBMs between pattern labels is unbalanced; see Figure 3.1 for the histogram of classes.

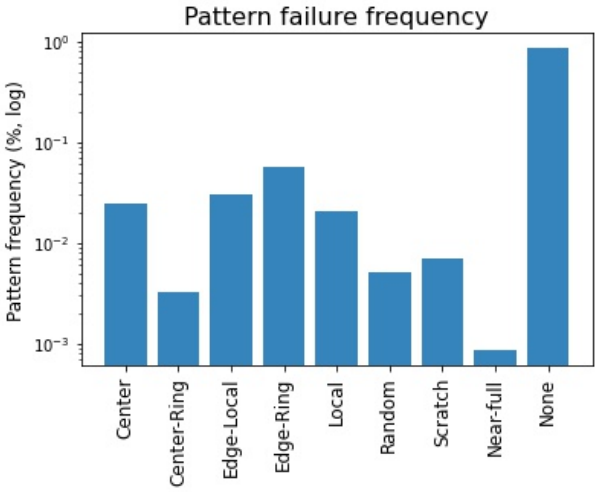

Figure 3.1: **Histogram of the distribution of labeled classes in the dataset**. The y-axis is logarithmic.

In total, there is **172,950** labeled samples (including the none defect), see Table 3.2 for the class label distribution. The number of samples differs from other papers (I assume that the dataset has been updated). Some papers were using the same dataset, but they have one more class (Horizontal Stripe). Others were using the dataset with the same classes, but they have created their own dataset or have not used all defect patterns.

Table 3.2: Class label distribution

Class	Sample counts
Center	4,294
Donut	555
Edge-Local	5,189
Edge-Ring	9,680
Local	3,593
Random	886
Scratch	1,193
Near-full	149
None	147,431

Initially, I thought that the train and test flag were switched because the test set was larger than the training set. After the inspection of each set, I have done the split between the training, test, and validation sets by myself.

The distributions of wafers were very uneven; see Figure 3.2 for the histogram of the area of the wafer in the original test and training set.

I have split the dataset so that the test set is 30 % (54,885 samples) of the dataset size, and the validation set is 15 % (18,160 samples), the rest is the training set (10,2905 samples).

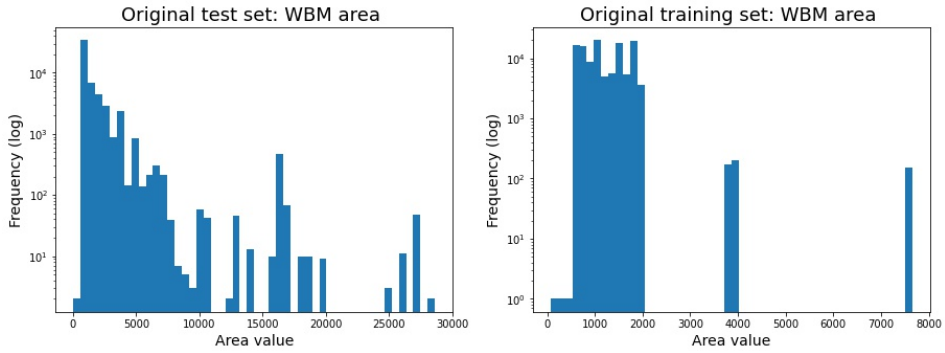

Figure 3.2: **Histograms of the area distribution in original test and training sets.** Because of the different distributions, I have split the dataset by myself.

3.1.1 Defect Classes

To see a typical defect pattern except for none, go to the Figure 3.4. Some defects are more common than others. For example, thanks to progress made in wafer fabrication, the scratch defect is less and less frequent. It is hard to obtain new samples of scratch defects to enlarge the dataset.

- **Local** is a coherent block of defective dice next to each other. This defect can be caused by the wrong position of the stamp.
- **Center** is a special case of a local defect that is in the middle of the wafer.
- **Donut** is a ring in the middle of the wafer with no dead dice in the center.
- **Edge-Local** is type of a local defect but it only occurs on the edge of the wafer.
- **Edge-Ring** is similar to a Donut defect pattern, but the defective dice are located on the edge or near it and the middle of the wafer has mostly good dice.
- **Random** defects are dead dice that do not form any pattern or shape, which could be caused by the fabrication process.

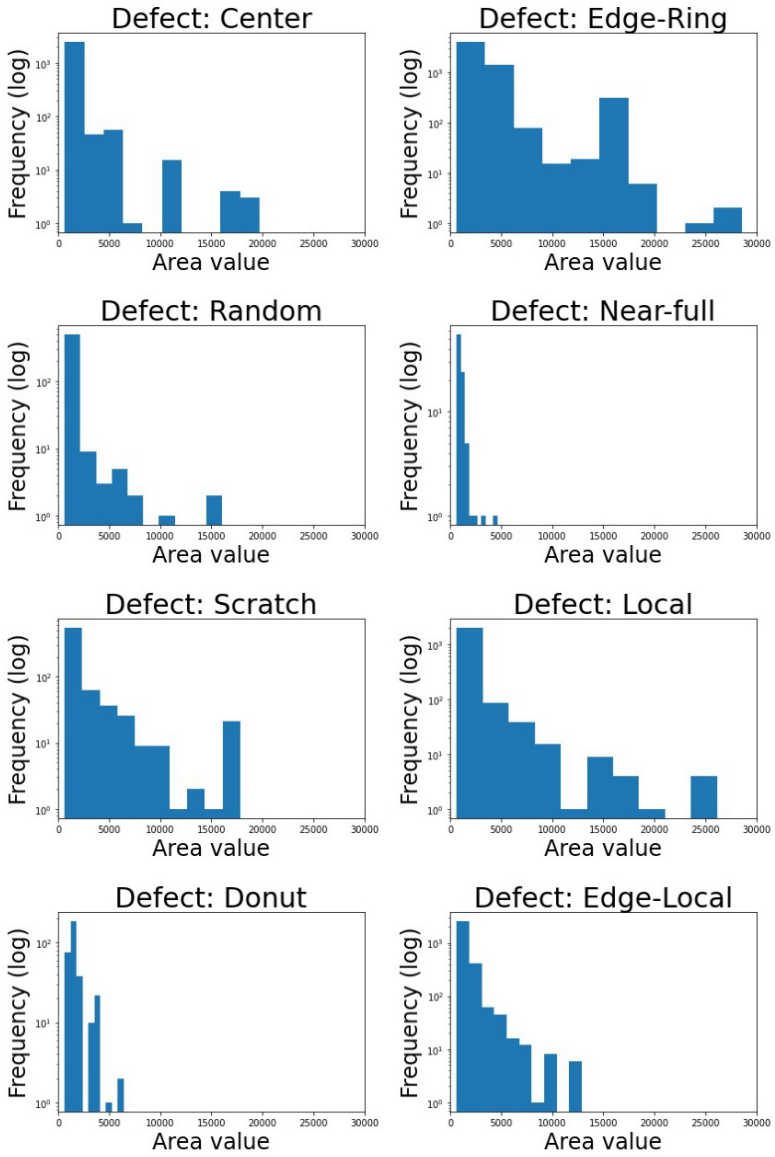

Figure 3.3: **Histograms of the area for each defect** in the training set. The near-full defect has only samples with maximal area of 3000, but other defects has almost ten time larger area.

- **Scratch** could be caused by manual handling. Wafers in one lot can bump into each other or the defect can by caused by some particle. Scratch are bad dice in the form of line. It could be straight line, round shape or squiggly.

- **Near-full** is a type of defect which covers almost all dice, and just a few of them pass all tests. In the dataset are wafers that has no good dies and they fall into this class.

- **None** is a class where all other defects or clear wafers are collected. It is quite hard to tell where the boundary is between None class and Random or even scratch. There have to be set rules that are used during the labeling. For example, the scratch could be defined as a line with a minimum length of five dice (based on the wafer resolution).

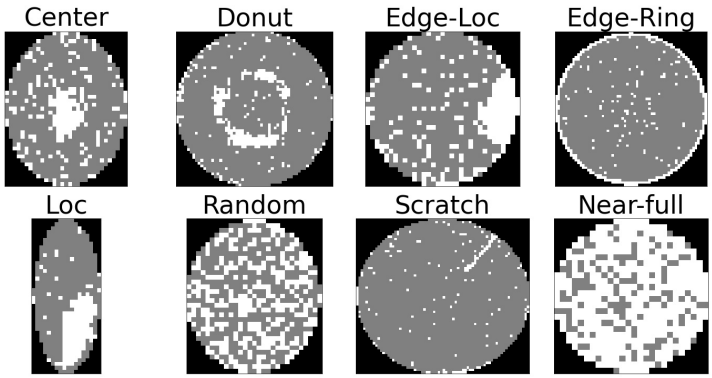

Figure 3.4: **Example of failure patterns from each class**.

The wafers from the dataset does not have the same dimensions, see Figure 3.5 for the histogram of wafer area. There are **632** types of wafer dimensions and **547** different wafer areas. The smallest wafer is 15×3 with the area of 45 and the largest wafers are 300×202 with the area of 60600. Because the CNNs have limitations in the terms of the input shape, the WBMs have to be reshaped to the same dimensions.

3.2 Preprocessing

The main task in preprocessing is the data preparation and augmentation to obtain more data. Ideally, the data are converted to numerical representation. Dataset WM-811K has columns lotName, trainTestLabel, and failureType, which should be converted to numerical form. The lotName values will be replaced with numbers instead of a combination of the word lot and its number. See Figure 3.6 for the WBM data visualization.

TrainTestLabel should be converted into binary values, and the empty rows will have the value -1. The last column, failureType, will be categorized into values from 0 to 9, where the number represents the defect category or it could be converted to one-hot encoding.

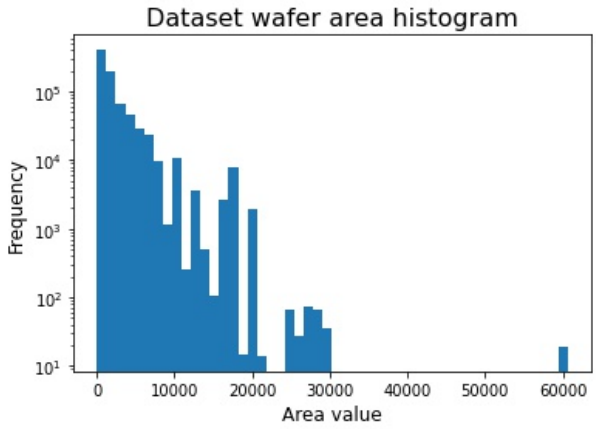

Figure 3.5: **Distribution of wafer area in dataset** with the logarithmic y-axis.

The wafers are stored in the dataset as a 1-dimensional array with discrete values 0,1,2 where:

- **0** represents the padding area around the wafer,
- **1** stands for the die (passed all tests),
- **2** means the damaged die (have not passed all test).

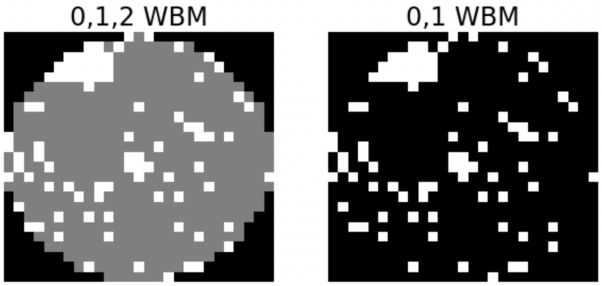

Figure 3.6: **WBM data representation**.

In some papers, they were using just binary values to represent the wafer. The zeros represent dice that passed all tests or empty areas around the wafer, and the ones stand for the defects.

I have tested, if the representation of the wafer has any impact on the model performance. I have tested both variants where the wafer is represented only with ones and zeros and with zero, one, and two for the defective dice. I choose the VGG16 as the model to test this because it has been used in papers where they have used both of these representations.

See Table 3.3 for the results of the test. There is some decrease in classification performance in defect classes Near-full, Scratch, Random, and little worsening in Edge-local and Local. My interpretation is that the information about the edge of the wafer is important for the model. For this test, I have been using only axis-flip for augmentation. I suppose that information about the edge of the wafer would be more important for wafers augmented by zoom and shift method.

While testing smaller custom architectures, I have found out that without a setting of `padding = same` in convolutional layers, the model has a problem with the classification of defects on the edge of the wafer. It adds zeros as a padding around the data and it will have the same output dimensions as output.

Table 3.3: Results of wafer representation test on VGG-16

	0	1	2	3	4	5	6	7	8
{0,1}	0.83	0.83	0.70	0.97	0.63	0.76	0.52	0.85	0.97
{0,1,2}	0.86	0.86	0.77	0.97	0.67	0.84	0.72	0.92	0.98

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. The results represents F1 score for each class.

Almost all solutions on Kaggle[6] used only part of the wafers of the same size. They have not performed any re-scaling on the wafers, and their model performance cannot be compared with solution which uses all wafers.

The input of my models are 2-dimensional arrays because the wafers have only two dimensions with three possible values. In case of the transfer learning with CNNs trained on the ImageNet [30], the CNNs require 3-dimensional input. The paper named "An Approach to Run Pre-Trained Deep Learning Models on Grayscale Images" [31] change the input shape of VGG16 architecture with pretrained wights by averaging the weights in the input layer. There was very little decrease in performance but the main benefit was the smaller input size.

3.2.1 Augmentation

Dataset is very unbalanced due to significant differences in representation of classes (see Figure 3.1 for the histogram of labeled data), so it is necessary to balance it. The major class None has **147 431** samples, and the smallest class Random has **149** samples. The easiest method for data balancing can be using class weights during training, under-sampling, over-sampling [32] or wafer augmentation. For example the rotation or patch augmentation can be also used as pretext task for semi-supervised or self-supervised learning.

[6] `https://www.kaggle.com/datasets/qingyi/wm811k-wafer-map/code`

The paper [23] proposed a solution that involved augmenting the dataset using GAN to enhance the minor classes in the dataset. They were working with the data from WM-811K with one new class Horizontal Stripe, and they have created their custom dataset with 7,560 samples in the training set and 1000 samples in the test set. The defect distribution was not the same as the WM-811K dataset has.

The improvement in performance with GAN augmentation was not that significant (only for the scratch defect) because of the small custom dataset. I decided not to use it and used only classical augmentation like rotations, axis flip, zoom, etc. The augmentation by wafer rotation or axis flip is moving the wafer's base/facet. This is not a problem, especially with these types of defects from dataset WM-811K. It could be a problem with other types of defects that are defined by their position to the wafer facet. In case of the WM-811K dataset, the model will probably learn some position invariant to the facet location.

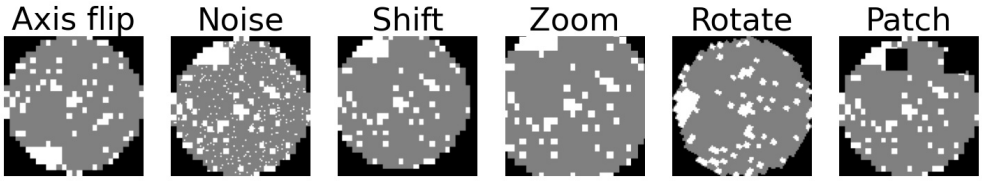

Figure 3.7: **Examples of all augmentation methods on one wafer**.

- **Rotation**

  The wafer is randomly rotated in the range $\langle 0°,\ 360° \rangle$ using a uniform distribution. Rotation can also be used as a pretext task for self-supervised learning where the model is predicting the angle of rotation.

- **Flip**

  The wafer is flipped by horizontal, vertical, and diagonal axis. In case of different defect type, for example the horizontal stripe used in the paper [23] would no be horizontal (parallel to the facet) after the diagonal axis flip. It could be misclassified as vertical scratch or probe defect. There are types of defects that have specific positions to the wafers facet (parallel or perpendicular).

- **Shift**

  The wafer is randomly moved in 2D space. The direction and length are picked from a uniform distribution with range $\langle -n, n \rangle$ where n is a fraction of the size of the wafer. Because of the random direction it could move the defect pattern on the edge of wafer out of the frame.

- **Zoom**

  A part of the wafer is zoomed-in using a randomly selected coefficient from uniform distribution in range $\langle 1.1, 1.5 \rangle$. The wafer is enlarged and shifted, then the wafer is center-cropped to its original size. Zoom has same issue as the shift augmentation and it could be more saturated by the zoom factor.

- **Patches**

  Random patches are generated onto the wafer. The number of patches is random, and the size is fixed and computed from the size of the wafers. The maximum number of patches is 3 per wafer. These patches are often used as pretext tasks in self-supervised learning, where the model predicts the content of the patch.

- **Noise**

  The noise from the Bernoulli distribution is added to the wafer. The Bernoulli distribution was selected because it fits perfectly for this task. The wafer cannot have a large portion of noise because of the clarity of the defect, and with big noise, there could be created new defects that would cause misclassifications.

- **Combination of multiple wafers**

  The paper [24] proposed a solution of combining wafers together to create multiple defects on one wafer. I think, this approach could be used even for the augmentation of single defect wafers, with a combination of wafers with the same defect, or with a clear wafer (only a few damaged dice). Not all defects could be augmented this way and this poses a problem. Another problem could involve the difference in sizes. Despite these problems, I think that this method could be used for the augmentation of scratch (the most problematic class) defects in the WM-811K dataset.

3.2.2 Resizing

Because of the nature of CNNs, the input vector must have the same shape for all data (wafers in this case). WBM data has a wide range of resolutions, from really small to large, this could turn to be problematic.

The wafers from the dataset have over 600 different sizes. The smallest one is 15×3 and the largest one is 300×202. As you can see from the Figure 3.5 of the wafer area histogram, the range is quite wide, and smaller wafers are more frequent. There are 19 large wafers that I have removed from the dataset because they are not significant in this dataset.

There is a resizing problem due to the interpolation that can create "new" defects types or change their characteristics, as you can see in the Figure 3.8. In this dataset, I have came across a problem regarding the downsizing of the

scratch defect and class random, which can become more like a near-full class. Other defects do not get that bad, but average performance is slightly worse. There might be more possible issues with different WBM defects (different classes).

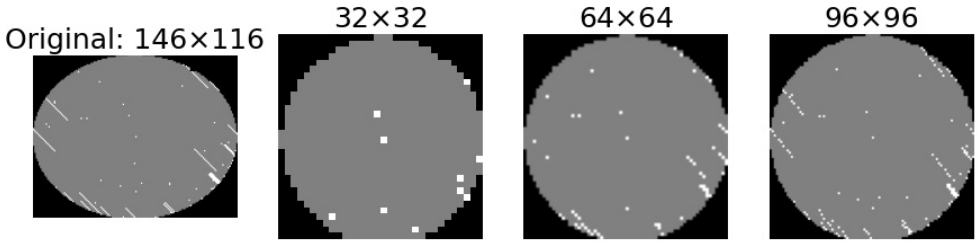

Figure 3.8: **Example of the resizing loss** During the up-scaling, there is no loss of information, same as with the down-scaling. The nearest neighbor interpolation is used for resizing.

Papers and notebooks mentioned in the State-of-the-art methods2 chapter proposed different reshaping sizes. Some of them use just part of the dataset with almost the same sizes, e.g., $48 \times 50, 50 \times 49, 51 \times 50$, and reshape them into similar dimensions such as 50×50 or use only wafers with dimension 24×24. Other papers reshaped all wafers to the same dimension, e.g., $64 \times 64, 96 \times 96$. This results in classification problems, especially with small wafers (mentioned sizes are larger than most of the wafers).

Table 3.4: Results of the different input size

	0	1	2	3	4	5	6	7	8
64×64	0.83	0.86	0.71	0.95	0.66	0.84	0.53	0.94	0.97
96×96	0.86	0.83	0.72	0.96	0.66	0.84	0.64	0.92	0.98

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. Results represents the F1 score for each defect class using VGG16 architecture with labeled data from the WM-811K dataset.

I have tried working with larger wafers than 96×96, and the training time turned out to be very long. The results were slightly better, but the training time was too long. I have been using smaller custom CNN for reshaping the wafers to 32×32, and it performed poorly because of the large downsized wafers that were misclassified. Some users on Kaggle[7] were designing networks

[7] `https://www.kaggle.com/code/kcs93023/keras-wafer-classification-cnn2d-with-augmentation`

only for specific shape size, e.g., 26×26 with no resizing and achieved really good score.

For best result, there would be a set of networks for each wafer size, but this is not the reasonable solution. I wanted to avoid having multiple independent models.

In Table 3.4, there are results of testing VGG16 with different input shapes. Because of the imbalanced character of the data, where the major class has over **85 %** of samples, I have tried four balancing techniques:

- **Under-sampling and Over-sampling**

  Major class None makes 85 % of the dataset, and it is under-sampled for the sake of dataset balance. The rest of the classes are still imbalanced. I have tried image augmentation methods described above instead of basic over-sampling (repeating the sample from minor class).

- **Augmentation**

  The augmentation mostly helped the minor classes. In some cases, it even decreased the performance for some classes, see section3.2.1 for more information about augmentation and chapter5 for the test results of augmentation methods.

- **Batch balancing**

  For batch balancing, I have used the data generators, which enabled me to add a number of samples from each class to the batch. The rest of the batch is filled up randomly from all classes.

  I implemented the Python class `DataGenerator` that is used for real-time data that are fed into the model. The return value of the `__data_generation` function are balanced batch sized data.

  My custom function is called `__data_generation` and it generates samples from all classes in the same ratio, and the rest of the batch is randomly filled up. See pseudo code 1 for more details. The result is the batch with a balanced number of samples from each class.

```
batch = []
n_samples = int(batch_size/n_classes)
rest = batch_size%n_classes
for class_i in range(class_indicies):
    batch.append(class_i.sample(n_samples))

batch.append(X[random()].sample(rest))
```

Listing 1: Creation of balanced batch.

- **Sample weights**

  In the case of an imbalanced dataset, there is an option to use weights to apply a different level of importance to each class. Sample weights work like a weighted mean compared to a simple mean. Sample weights can be used in the loss function to force the model to learn minor classes.

  I have implemented function for the weights computation, see the following pseudo code 2 for details. It is based on the inverse frequency of each class in the whole dataset. I have found many different versions of the normalization of these weights. I have chosen that the sum of the weights equals one.

```
def class_weights(labels):
    _, frequency = np.unique(labels, return_counts=True)
    weights = 1. / frequency
    weights /= weights.sum()  # normalize
    return weights
```

Listing 2: Class weights calculation.

3.3 Transfer Learning

Transfer learning is a method of learning that uses an already trained model together with a new data. The learned weights can be frozen and then only the classifier is trained for the new classification task. The CNNs, that I am using in this thesis, are all pretrained on the ImageNet dataset [30]. To use them for a different problem, they need just the replacement of the input and output layers. These large networks are not appropriate for this specific task. The input data are not classic images with three dimensions with pixel values ranging from 0 to 255. Nevertheless, they are used in papers where WBMs are used.

I have been using transfer learning for first testing and analyzing of the augmentation methods and wafer resizing. It is hard to compare the results of all the papers described in the chapter2, when they have used only part of the dataset or different sizes of training, validation and test sets. For example, paper [13] which presets the WaPIRL framework was using only 10 % of the labeled data as test set which is not ideal.

In the paper [13], they have compared self-supervised learning versus supervised learning with different types of pretext augmentation. The best performing model was VGG16 with supervised training, which achieved macro F1 score of **0.871** and **0.897** macro F1 score with self-supervised learning with crop augmentation.

For better comparison, I have tested these popular architectures by myself. I have tested these CNNs: VGG16 [9], ResNet50v2 [6] (were used in the

paper [13]) and the Xception [10]. My priority is to make the F1 score for each class more balanced. Many papers used the macro F1 score as a comparison metric, but using mean of the F1 scores is a problem since it not always represents the best result. The minor class could still have bad result, but if larger classes achieve really good results, the macro F1 score is better, but the model still struggles with the classification of the minor classes.

3.3.1 Configuration

The architectures of the popular CNNs require a 3-dimensional input of the same size. All wafers are converted to the 3D image just by stacking the 2-dimensional wafers on top of each other. All the wafers were reshaped to $96 \times 96 \times 3$. I have been using the configuration of the optimizer, learning rate, etc., all inspired by the paper [13] described above.

Each model was tested with the same training, valid, and test sets. The major class from dataset has been under-sampled, and minor classes have been augmented with axis flip to create a more balanced dataset. The following tests were performed with a batch size of 128 samples.

To save the best model, I have used the early stopping function, which selects the best performing model based on the validation loss. I also reduced the learning rate when the model had not improved in the last seven epochs.

I have implemented the transfer learning in Keras because it was the fastest option for me to start first tests. Based on the first results with sample weights, I have the added augmentation and under-sampling of training set. Next step was adding the batch balancing, which was done by implementing the data generator class.

3.3.2 Supervised Learning Results

In the paper [13], they tested AlexNet, VGG16, ResNet-18, and ResNet-50. VGG16 achieved the best **0.897** macro F1 score with a self-supervised approach. All models were trained with same configuration. The only variables that were not fixed were random selection in under-sampling and batch balancing.

Table 3.5: Results of supervised learning approaches

	0	1	2	3	4	5	6	7	8
VGG16	0.92	0.88	0.81	0.98	0.74	0.88	0.74	0.86	0.99
ResNet50	0.92	0.88	0.81	0.98	0.74	0.88	0.74	0.86	0.99
Xception	0.93	0.90	0.83	0.98	**0.78**	0.89	**0.79**	0.97	0.99

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. The results represents average of five runs.

The best model with a supervised learning approach was Xception which achieved **0.894** macro F1 score (with a smaller training set and larger test set than paper [13]). Xception showed significant improvement with classification of the Scratch and the Edgle-Local defect patterns.

Table 3.6: Supervised learning with popular CNNs results (macro F1 score)

Models	Paper [13]	Mine
AlexNet	0.840	-
VGG16	0.871	0.870
ResNet-18	0.858	-
ResNet-50	0.870	0.876
Xception	-	0.894

My results are an average of five runs, and the result from the paper is the average of ten runs. My test set is 3-times larger then the one used in paper [13]

See Table 3.6 for results of all models with comparison of the supervised learning results from the paper [13] and mine. I have not tested all models because it is not the primary goal of this thesis. During the training, the epoch of the ResNet50 and VGG16 have very similar training times, around one minute per epoch (185ms/step), but Xception's epoch lasted three times longer (590ms/step).

3.3.3 Semi-supervised

I have tested the semi-supervised method with pseudo-labels as a one of the last test in this thesis. I wanted to try if there is any room for improvement with larger CNNs without any special settings.

The results were kind of disappointing, the Xception model have approved only in few runs and mostly have not improve at all. After the closer inspection of the prediction of pseudo labels I have found that the model was too much confident with its predictions. The use of sample smoothing have not flatten the distribution of confidence (it was only shifted to the left).

The paper named "Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks" [33] address this problem and propose a solution for over confident networks[8].

[8] I did not have time to test this solution due to the lack of time

3.4 Misclassification

In this section are analyzed the misclassifications of the Xception[10] model on the WM-811K dataset. The Xception model was the best performing model from the popular CNNs with supervised training.

From the 51885 test samples Xception misclassified only 1235 samples which is 97.6 % of accuracy. The accuracy metric is not that important especially with the imbalanced data.

3.4.1 Confusion matrix

See Figure 3.9 for the confusion matrix. The most misclassified defects are from the scratch class. The most common mistake is the classification as the None class which is the biggest in the dataset.

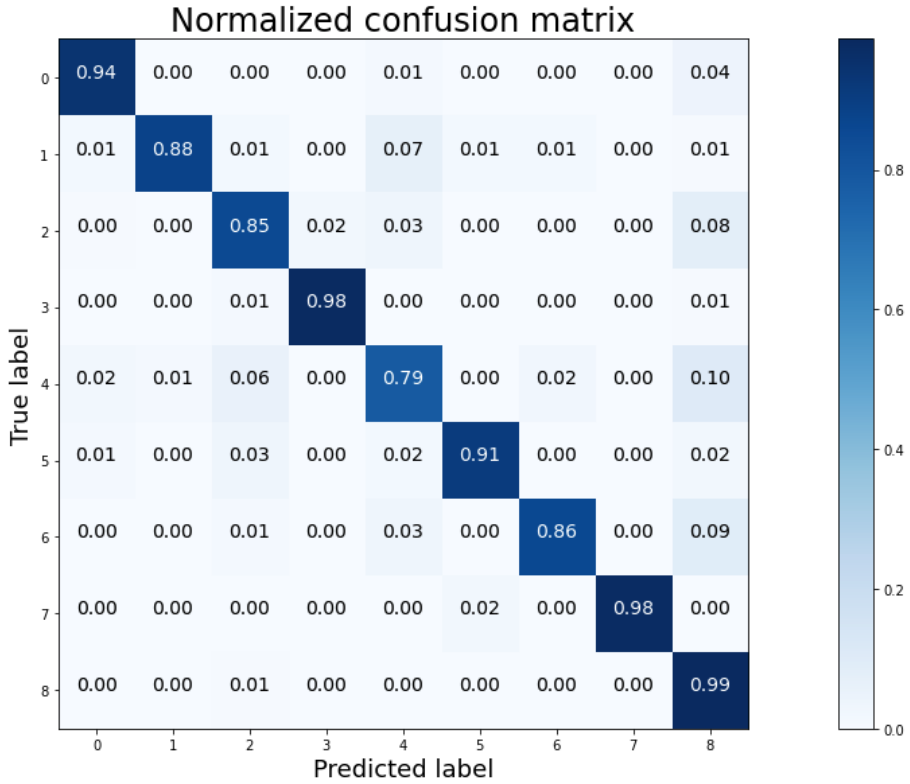

Figure 3.9: **Confusion matrix of Xception predictions.** This is the results of single run that achieved 0.898 macro F1 score. 0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None.

There are also visible misclassifications between classes Edge-Local (2) and Local (4) where only difference is the position on the wafer.

Even with strong under-sampling of the major class there are many misclassifications. I think that is because of the None class represents all defects that do not fit in any other class. For that reason is hard to separate this class from the other classes.

3.4.2 Edge-Local vs. Local

The Local (4) defect pattern was the most problematic to classify for the Xception. I have done analysis of the misclassified patterns for these two classes. See Figure 3.10 for good example of the edge case between these two classes.

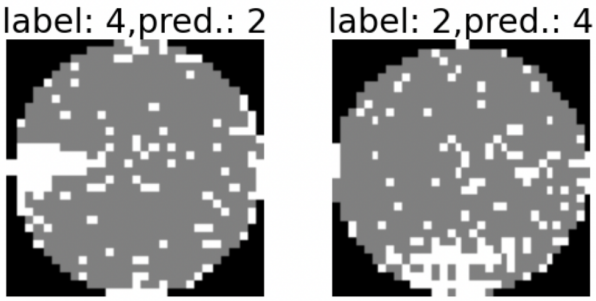

Figure 3.10: **Misclassification of Local defect pattern** It is hard to distinguish what pattern falls into which category. 2:Edge-Local, 4:Local.

Based on my analysis of the misclassified wafers by my naked eye. I assume that in the most cases the difference between Local and Edge-Local is that the bad dies are closer to the middle of the wafer and there are some good dies along the edge in case of the Local defect pattern. But with WBM data of smaller resolution there are not many of good dies, Figure 3.10 shows that.

Other source of the misclassifications could be the interpolation in resizing. In case of the smaller wafer the one bad die is interpolated into the multiple bad dies which can change the specification of the defect pattern.

3.4.3 Scratch Defects

See Figure 3.11 for examples of misclassification of the Scratch class. The problem with scratch defect is that it could be anywhere in the wafer and it can be thin or thick. During the labeling there have to be set rules on how many bad dies in a row it takes to form a scratch. This rule should changes for different resolutions.

Because of the improvements in the fabrications process the scratch defect is less and less common. It is hard to obtain new samples of the scratch defects to enlarge the number of samples in the dataset.

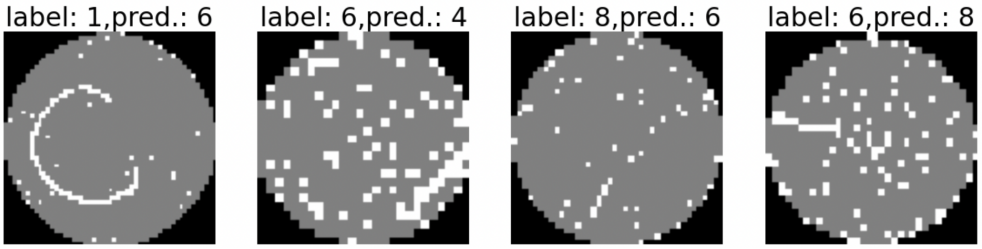

Figure 3.11: **Misclassification of Scratch defect pattern** The borders between the classes are unclear in some cases. 1:Donut, 4:Local, 6:Scratch, 8:None.

Chapter 4

Realization

In this section, the proposed implementation of the custom model will be described. The goal was to create a more memory-efficient and faster model than popular CNNs because they are not appropriate for wafer classification task. They were proposed to classify photo like images not binary data.

4.1 Technology

I have written most of the code in `Jupyter Lab` notebooks[9] and all models have been trained on Nvidia Tesla P100 with 16BG of memory and 64GB of RAM. The access to the server was provided by company Inference tech[10]

For testing the WaPIRL[11], I have been using their python scripts that need updates of some libraries to make it all work with newer versions of libraries. See 3for updated imports. I have implemented smaller single input model into the WaPIRL framework.

4.1.1 Keras and Pytorch

Primarily, I have been using the Keras framework and PyTorch with popular data science, computer vision python libraries: `pandas`[12], `NumPy`[13], `matplotlib`[14], and `openCv`[15].

After the implementation of transfer learning with popular CNN models with simple augmentation and batch balancing, I have not been able to create a data generator for multiple input model in Keras, so I switch to PyTorch.

[9] `https://jupyter.org/`
[10] `https://inferencetech.com/`
[11] `https://github.com/hgkahng/WaPIRL`
[12] `https://pandas.pydata.org/`
[13] `https://numpy.org/`
[14] `https://matplotlib.org/`
[15] `https://opencv.org/`

```
# file ./utils/metrics.py
# Before:
from pytorch_lightning.metrics import (
    MulticlassROC, MulticlassPrecisionRecall
)
from pytorch_lightning.metrics.functional import (
    auc, precision, recall
)
# After:
from torchmetrics.functional import (
    auc, precision, recall
)
from torchmetrics.classification import (
    ROC as MulticlassROC
)
from torchmetrics.functional import (
    precision_recall as MulticlassPrecisionRecall
)
```

Listing 3: Update of import metrics

The other reason was the WaPIRL framework which was implemented in PyTorch, therefore, I would have to learn PyTorch at some point.

4.1.2 Data Analysis

I have been using the `pandas` library for working with the dataset and basic analysis. The original dataset was a pickled file, but I have been using the `Feather format`[16] for exporting the training, validation and test sets, and other data frames.

4.2 Dataset

The dataset WM-811K is already split into the train and test sets with labeled patterns, but their sample distribution was not good, so I have to split the dataset by myself. The test set is 30 % of the original dataset size, and the training set is then split to create the validation set, which is 15 % the size of the training set. To keep the same distribution in all sets I have used the parameter `stratify=True` in function `train_test_split` from `scikit-learn`[17] library.

[16] `https://github.com/apache/arrow`
[17] `https://scikit-learn.org/stable/`

I created new columns for wafer dimension and wafer area, then I transformed other columns into numeric values for better usage. Because of the distribution of the wafers area, I have created bins to convert them to categorical data. The area was split into three normalized bins (0, 0.5, 1) with roughly same number of samples. The number of bins is low because of

The wafer is stored as a 1-dimensional array, and the dimension information is needed while resizing it to the 2D array. The validation set is obtained as 15 % of the training set with the same class distribution. All sets are exported in feather format.

4.2.1 Preprocessing

The WBM data are stored as a 1-dimensional array, and then they are reshaped to a 2-dimensional array with original dimensions. Before the model input, the wafers are reshaped into the same size, e.g., 64×64. The augmentation of the training set is performed before the final reshape.

In case of the dual input CNN model the wafer are resized to the dimensions (28×28 and 96×96). With single input model only the 96×96 wafer shape is used. To balance the training set, I have used multiple augmentation techniques such as axis flip, rotation, noise addition, shift, zoom and patches.

Other balancing method was balancing the samples in batches. Because of the imbalanced data there was big chance that even with large sized batch it will not contain many different classes. To solve this issue I implemented the batch balancing 1.

4.2.2 Augmentation

I implemented described augmentation methods in section 3.2.1. They were used in paper [13] as a part of the self-supervised pretraining. The coefficients in the augmentation methods are relative to the original wafer size, but there is a room for improvement to find better coefficients values.

4.3 Model

All papers using the CNN approach had issues with re-scaling the input wafers onto the same shape. Because of the wide range of shapes in the dataset, models perform worse with smaller wafers when re-scaling to larger shapes and vice versa. In case of down-scaling the loss in performance is more significant because of the loss of information caused by the interpolation.

First, I wanted to proposed a model with multiple input shapes and input for wafer area/shape. The model should learn the features on the same wafers in different resolutions, and the information about the original size should help choose a better representation. For example, with two input sizes and a smaller wafer, the model will put more emphasis on the features from the

CNN with a smaller input when the results are not clear. But my analysis showed the area distribution and the information about the wafer size would add bias to the model, so I left this idea.

After a closer inspection of the dataset, I have found that some defects occur only on smaller wafers, see Figure 3.3 for the distribution of area size for each defect. This means, the model could have some area bias. The result could be that the model would not classify any Near-full defect for larger wafers because the maximum wafer area for this defect is only 3000 pixels.

4.3.1 Development

At first, I started with simple 3-layered CNN architecture with two dense layers as classifiers. This model was inspired by paper [23], where they were using the 64×64 and binary preprocessing. Based on my analysis, I have found out that this setup is worse than 96×96 wafers with bins from $\{0, 1, 2\}$. Because of the larger wafer size, I have to make my model deeper because the network has not been able to extract suitable features for classification.

The next step was to build a deeper CNN inspired by architecture from the VGG network with a dual convolutional layer and one max pooling layer. To this network, I added a smaller one for smaller inputs and input for the wafer area. The first idea was to improve performance with smaller wafers that suffer from re-scaling to larger sizes.

The final model is very similar to VGG16 architecture with batch normalization. See Listing 4 for the architecture of one block. The input of the network is $96 \times 96 \times 1$ and the number of kernels in each convolutional layer are increasing multiples of two. Final model has six convolutional layers.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-9          [-1, 256, 24, 24]         295,168
              ReLU-10          [-1, 256, 24, 24]               0
      BatchNorm2d-11          [-1, 256, 24, 24]             512
           Conv2d-12          [-1, 256, 24, 24]         590,080
             ReLU-13          [-1, 256, 24, 24]               0
        MaxPool2d-14          [-1, 256, 12, 12]               0
      BatchNorm2d-15          [-1, 256, 12, 12]             512
----------------------------------------------------------------
```

Listing 4: Architecture a block of single input network used for self-supervised learning.

4.3.2 Normalization

I have tested the **Batch Normalization** presented in paper named "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" [34]. It calculates the mean and the standard deviation of each input variable to a layer per batch. Batch normalization has trainable parameters that provide the standardization of the layer inputs.

Layer normalization is inspired by batch normalization and it was introduced in the paper [35] named "Layer Normalization". It normalizes the inputs by the features not by batches. This is a significant improvement over the Batch normalization (removing the dependency on batches). I have used the implementation of the paper from Github[18].

Because of the strongly imbalanced dataset, I tried many balancing techniques and found a batch balancing gives me the best results. Even with the use of sample weights, I have not been able to make the result of the model stable. All models struggled with the scratch class (even though it is not the smallest class); it always has the lowest F1 score. For example, other classes have F1 scores between 0.7 to 0.99, and the F1 score of the scratch class was only 0.5.

Without any balancing, there would possibly be no samples from the minor classes. I have implemented batch balancing in Keras using the data generator; see 1 for the pseudo-code. The final result is that each batch contains samples from all classes, and it is almost perfectly balanced, see Chapter 5 for the tests and their results. Popular CNNs achieved the best-supervised performance with batch balancing, which I have also implemented it for my small model.

4.3.3 Semi-supervised Learning

I have implemented semi-supervised learning approach using pseudo samples from the paper [21]. The unlabeled samples are classified with a model trained on labeled data. The predicted labels are used as normal labels, and the model is trained with new labels and data. The final step is training the model with the labeled samples again. I have not tested the pseudo labels with popular CNNs because of the long training time.

Training with pseudo labels was stopped if the model did not approved the validation loss in last 10 epochs. The same early stopping was used in supervised pretraining.

After testing these variants, I noticed that every run has a different distributions of predicted labels. Some classes had suddenly over 100 000 samples more then in the previous runs. Based on this finding, I deduced that the model is not able to separate the classes well or there could be many edge cases in the unlabeled data. This unstable behavior was happening without

[18] `https://github.com/CyberZHG/torch-layer-normalization`

batch balancing. The distribution of the pseudo label predictions were more stable (more similar between each runs) with batch-balancing.

Because of the unstable classification of pseudo labels, I have tried to filter them by their probability values (the confidence of the prediction). In PyTorch, I have to use the softmax function on model predictions to get the values. In the next step, the pseudo samples are filtered by their confidence value. The result was the samples that were confidently predicted, see the Figure 4.1 for the confidence histogram of the pseudo labels.

The paper named "When Does Label Smoothing Help?" [36] shows that it improves the generalization and the model is less confident. My idea was to use label smoothing with classification of the pseudo labels to make the distribution of prediction more flat. It will be easier to remove less confident predictions.

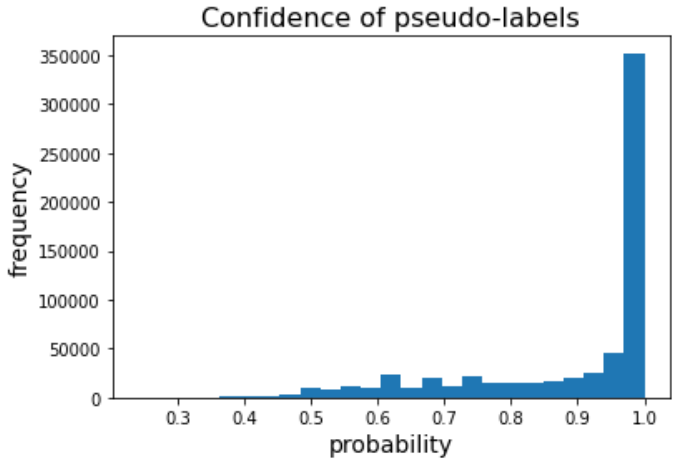

Figure 4.1: **Histogram of pseudo-labels prediction confidence.**

I have been using the confidence threshold with value of 90 %. After the filtering out the less confident predictions and under-sampling the major class there were about 50 % of samples left. The under-sampling threshold was set manually for each run based on the distribution of predicted labels.

During the training with all pseudo-labels the model sometimes performed worse then the supervised version, but after the filtering out the less confident predictions training step the performance was better. See chapter Results5 for details and results.

4.3.4 Self-supervised learning

I have been using the WaPIRL framework, proposed in the paper [13], to test it with smaller model for self-supervised learning. The code of the framework

is available on the author's GitHub page[19]. Before trying the framework with my custom model, I tested it with the ResNet-18 (it was the default option). The model training consists of two phases: self-supervised pretraining and supervised fine-tuning; see the Section2.4.3 for more information.

I implemented a smaller model with single input into the WaPIRL framework with 6 convolutional layers and batch normalization. The goal was to test it against the larger and more complex models. I changed the backbone of the AlexNet to my small architecture. For adding the new architectures to the framework, classes and configuration files and etc. have to be implemented. The class of new architecture inherits the interface from the class `BackboneBase`. There have to be set the configuration dictionaries for pretraining and fine-tuning.

The pretraining stage took more than a 24h of training time, and the training was for 100 epochs with multiprocessing. The batch size was 256 samples with wafer input shape 96×96.

[19] `https://github.com/hgkahng/WaPIRL`

Chapter 5

Results

In this chapter, the results of the proposed implementations will be described. The variations of the model were tested with different sizes of labeled and unlabeled data or different augmentation methods. Testing of the popular CNN models or the augmentation method is described above in the Analysis and Design 3 chapter.

5.1 Supervised

Analysis showed that my custom CNN model performs similarly to some popular CNNs. My focus was to improve the F1 score, especially for minor classes. The comparison is hard to do because they were usually presenting accuracies or means of F1 scores. I have tested these methods with my custom model because of the shorter training time, see Figure 5.1 for the confusion matrix of predictions.

5.1.1 Single vs. Multiple Input

I want to create model that has a comparable performance with the state-of-the-art models but it is smaller and faster. Based on the analysis the Xception was the best performing model, but it was really slow to train. The best model from the paper [13] was VGG16.

First idea was to solve the resizing problem by giving the model some information about the original shape of wafer. Dataset analysis showed that the range of fail pattern area differs between failures. The information about the area would add some bias to the model. The testing did not show that the information about the area has any significant effect on model performance, see Table 5.1 for the results.

The second test was about trying to add normalization into the network. I have tested two types: batch normalization [34] and layer normalization [35].

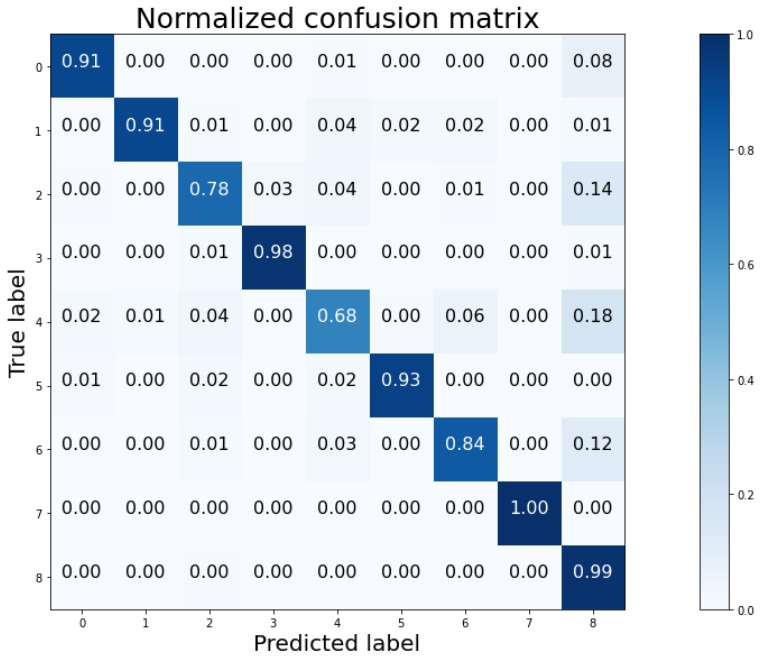

Figure 5.1: **Confusion matrix of predictions by supervised trained model with augmentation** The classes with the worst performance are: Edge-Local, Local and Scratch. The performance on scratch class were improved by augmentation. 0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None.

5.2 Augmentation

I wanted to test my hypothesis that using combination of augmentation methods for each class could improve the supervised learning training and performance. All papers I have read were using only one type of augmentation, except the paper [23] that proposed GAN augmentation. For example the shift augmentation is not the best way how to augment a failure around the

Table 5.1: Results of model with single and multiple inputs

	0	1	2	3	4	5	6	7	8
single	0.88	0.86	0.78	0.97	0.69	0.85	0.55	0.88	0.98
double	0.90	0.85	0.78	0.98	0.72	0.85	0.66	0.93	0.98
double + area	0.90	0.86	0.79	0.97	0.71	0.88	0.66	0.92	0.98

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. Results represents average of five runs and the models were trained with the same set of parameters

wafer edge, it could move the failure pattern out off the image.

I have done the augmentation for minor classes only three times (maximum), because the axis flip has three variants and I did not want to repeat same wafers. It will be better for comparison that all classes has the same number of sample frequency for all tests. The major class "None" was undersampled because it was 85 % of all labeled data and to speed up the training. All other classes were augmented using the methods mentioned above in section3.2.1. Only the class 3 (Edge-Ring) was augmented just once because it was the second largest class.

I used the model with dual inputs for two wafer shapes (28×28 and 96×96) and with batch normalization after every max pooling layer. The batch size was set to 128 because I have been training multiple models at once. I have tested the batch size 128 vs. 256 and there was no difference in results/performance. During the training, each batch was balanced in a way that every class has almost the same number of samples.

5.2.1 Same Method for All Classes

See Table 5.4 for the results. I have tested following augmentation methods: axis flip, noise addition, wafer shift, rotation, zoom and patch. The none row in the Table 5.4 shows the results for data with no augmentation (only under sampling).

The most problematic class is scratch. Overall best result were achieved with patch augmentation, where the patch sizes were set to the 10 % of the wafer length. The results show that there is no single augmentation method that is better than the others. In most cases, the result are same for all variants.

The underlined values in Table 5.4 mark the inappropriate augmentation methods for those failure patterns. The bold values are the methods that achieved significant improvement over the other methods.

See Figure 5.3 for training dataset sample counts per class.

Due to the number of samples in each class, it shows that classification of a single wafer in the smaller classes has a huge impact on the final result.

Table 5.2: Results of model with and without normalization

norm.	0	1	2	3	4	5	6	7	8
none	0.84	0.78	0.70	0.96	0.66	0.73	0.21	0.92	0.95
batch	0.90	0.**85**	0.78	0.98	0.72	0.85	**0.66**	0.93	0.98
layer	0.90	0.78	0.76	0.97	0.69	0.84	0.56	0.94	0.98

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. Results represents average of five runs and the models were trained with the same set of parameters

Based on the results of testing, the most problematic classes are: Edge-Local, Local are Scratch.

The best augmentation method in paper [13] was crop/zoom augmentation. In my testing the zoom augmentation showed a significant improvement in the most problematic class (scratch), but it under-performed in other three classes. They were using augmentation as a part of the self-supervised pre-training rather than as a method to balance the dataset. This result may be caused by imperfect parameter configuration for the zoom augmentation.

Based on my testing the overall best single augmentation technique was the patch/cutout. It achieved the best score in multiple classes and most importantly in the scratch class. The macro F1 score with patch augmentation was 0.856, better than the none augmented version with 0.824 F1 score.

5.2.2 Different Method for Each Class

Based on my augmentation, I have tried to improve the results with simple augmentation techniques by augmenting each defect with appropriate method. See Table 5.4 for the results. Some classes have not shown any improvement at all or slight decrease. I have tried the combination of patch and rotation augmentation which achieved one of the top scores in almost all categories. The Near-full class is the smallest class with only 45 wafers in the test set. This means that one misclassified near-full pattern has significant influence on the metrics.

5.2.3 Batch Generation

Because of the imbalanced dataset all models have problems with classification of the minor classes especially the class scratch. My idea was to use sampler to make sure that every batch contains sample from all classes to improve the training.

Table 5.3: Class label distribution in training set (without augmentation)

Class	Sample counts
Center	4,294
Donut	331
Edge-Local	3,087
Edge-Ring	5,760
Local	2,138
Random	515
Scratch	710
Near-full	88
None	(under-sampling)

See Table 5.5 for test results. The batch size was set to 128 samples and sample smoothing to 0.1. The best score was achieved with batch sampler that generates balanced batches with the same number of samples from each class.

5.2.4 Label smoothing

I have tested different values of label smoothing described in the paper [36].

See the Table 5.6 for the test results. The label smoothing improves the performance mostly in scratch and random classes. The test set has approx. 45 samples and it is hard to make any conclusions. But with scratch class there is visible improvement in the F1 score.

When I tried the label smoothing set to 0.15 and higher, the performance in supervised learning dropped significantly. In confusion matrix, more misclassifications were visible between the problematic classes and the None class.

5.2.5 Size of labeled data

I have tested different sizes of training set and how it affects the final predictions. See Table 5.7 for the results of supervised learning. The results proofs that the size of the dataset is very important especially for the classes that are hard to separate.

The data were split from the training set with the same distribution. The original training set has 102,905 samples (100 %). For the training I have used

Table 5.4: Results of supervised learning with different types of augmentation

Aug. type	0	1	2	3	4	5	6	7	8
none	0.88	0.86	0.73	0.97	0.67	0.85	0.54	0.94	0.98
axis flip	0.90	0.85	0.78	0.98	0.72	0.85	0.66	0.93	0.98
noise	0.91	0.86	0.78	0.97	0.70	**0.87**	0.60	0.93	0.98
shift	0.91	0.84	0.78	0.97	0.70	0.85	0.66	0.90	0.99
rotation	0.90	**0.87**	0.78	0.96	0.71	0.82	0.63	0.94	0.99
zoom	0.90	0.84	0.75	0.97	0.69	0.84	**0.69**	0.90	0.98
patch	0.91	0.84	0.79	0.97	0.71	**0.87**	**0.69**	0.94	0.99
combination	0.91	0.88	0.79	0.97	0.72	0.86	0.71	0.93	0.99

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. Results are the average of five runs and the models were trained with the same set of parameters. The augmented classes were: 0, 1, 2, 4, 5, 6, 7, 8, 9. Underlined values are indicate bad results for the augmentation technique for specific defect. Bold values are marking the good/improved results for single augmentation methods.

the batch sampler and label smoothing. With less data the model performance was unbalanced and it differs a lot for run to run.

5.3 Semi-supervised

I have implemented the pseudo-labels semi-supervised method presented in [21]. Model was trained on labeled training set and in the next step the training set and unlabeled data (with pseudo labels) were merged together and the model was trained on this data.

Many papers [15, 22] that proposed semi-supervised or self-supervised learning were using already labeled datasets and were working with the data without the labels. I have been using only the labeled data which are around 21 % of all the data and only 3 % are the labeled patterns which is very little data.

After the classification of unlabeled data, the None class has still over 50-60 % of samples (the exact number depends on the model's predictions). The under-sampling was performed again on the pseudo labels for the major class because of the imbalanced data.

Using the sample smoothing for supervised and unsupervised training, the result in the Table 5.8 shows that label smoothing improves the classification performance for the scratch class. The scratch class is the most problematic class in this dataset.

Table 5.5: Results of data samplers test

smooth.	0	1	2	3	4	5	6	7	8
none	0.92	0.66	0.81	0.97	0.74	0.90	0.59	0.96	0.99
weight	0.91	0.87	0.80	0.97	0.72	0.86	0.61	0.94	0.99
batch	0.92	0.86	0.80	0.97	0.72	0.89	**0.67**	0.97	0.99

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. Results are the average of five runs and the models were trained with the same set of parameters.

Table 5.6: Results of model with and without smooth labeling

label smooth.	0	1	2	3	4	5	6	7	8
0	0.92	0.86	0.79	0.97	0.71	0.86	0.63	0.91	0.99
0.05	0.91	0.86	0.79	0.97	0.73	0.89	0.64	0.95	0.99
0.10	0.92	0.86	0.80	0.97	0.72	0.89	**0.67**	0.97	0.99

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. Results represents average of five runs and the models were trained with the same set of parameters.

The relative average improvement in scratch class is **5 %** and other classes have not improved or worsened which is good result. The average of macro F1 scores of five runs was **0.869** with 0.1 sample smoothing. For this testing, I was using only pseudo labels with more then 90 % of confidence. The value of the threshold has a big influence on how many of the pseudo samples will be used in the semi-supervised training and its duration.

Next, I have tested the value of the confidence threshold. If the confidence of the prediction of the unlabeled sample is larger then the threshold, it is added to the training set for semi-supervised learning. The original labeled training set is enriched with the new pseudo-samples.

Based on the testing of the sample smoothing I have set the label smoothing to 0.1 and in the next step I will test the probability threshold value, see Table 5.9 for the results. I have focused on the final F1 score not the improvement over the supervised learning.

The results showed that after removing the less confident pseudo-labels the performance got better only for the scratch class. The are no bigger improvements or declines in other classes. In theory, it is better for model to train with the more data than less.

5.3.1 Size of labeled data

I have tested different sizes of training set with supervised learning and how it affects the final predictions. I have use these models to predict the pseudo labels and test if it is still possible to improve the performance with worse performing models.

With 75 % sized dataset the pseudo labels improve the final result in 4 out of 5 runs with 1 % of improvement in macro F1 score. With smaller dataset the improvement was less frequent and less significant on average. In many cases there was significant drop in performance around 1-2 % of macro F1 score.

Table 5.7: Results of supervised learning with different the size of labeled data

DS size	0	1	2	3	4	5	6	7	8
25 %	0.90	0.82	0.75	0.97	0.64	0.87	0.19	0.91	0.98
50 %	0.91	0.84	0.79	0.97	0.70	0.87	0.53	0.92	0.98
75 %	0.92	0.85	0.80	0.97	0.73	0.88	0.58	0.94	0.99
100 %	0.92	0.86	0.80	0.97	0.72	0.89	**0.67**	0.97	0.99

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. Results represents average of five runs and the models were trained with the same set of parameters.

It was visible how the performance of the pseudo labels method are strongly affected by the supervised model and suitable samples. I think the pseudo samples are method that can be useful in many cases. For example with enlarging the existing dataset. It could be used in iterative manner to speed up the data labeling process.

5.4 Self-supervised

I have tested smaller single input model with the WaPIRL framework to train it using self-supervision approach. The single input architecture achieved macro F1 score of **0.826** with the supervised learning and with self-supervised approach using WaPIRL framework achieved **0.862**.

The authors have used the batch size 256 samples, but then with 3-dimensional inputs it filled up almost whole GPU memory and I was not able to train other models.

I have pre-trained the ResNet-18 with 50 epochs and 128 batch size, see Table 5.10 for the results. The pretraining step took over 12h and final fine-tuning took around 2h (50 epochs). I have tested the two versions of fine-tuning with 50 and 100 epochs. The supervised baseline for the ResNet-18 was **0.858** macro F1 score. The results are from the best model based on the validation loss.

The ResNet-18 after 50 epochs achieved **0.792** F1 score and after 100 epochs it was **0.878**. I have used the different optimizer (weak Adam) to the one they used in their testing. The average of ten runs presented in the original paper was **0.895** with 100 epochs for pretraining and fine-tuning.

The single input model achieved similar results as ResNet-18 fine-tuned on 50 epochs and it performed slightly worse on 100 epochs. Improvement of 1 % was gained with 100 epochs of pretraining and 100 epochs of classification.

The result shows that the model learned some information during the pretraining, but still the supervised fine-tuning, which relies on the labeled data, has a huge impact on the final results.

Table 5.8: Results of semi-supervised relative improvement with different label smoothing and same confidence threshold (90 %)

smooth.	0	1	2	3	4	5	6	7	8
0	-0.01	0.00	0.00	0.00	0.01	0.00	0.01	0.01	-0.01
0.05	0.01	-0.01	0.00	0.00	0.00	-0.02	**0.05**	-0.01	0.00
0.10	0.00	0.01	0.00	0.00	0.00	-0.01	**0.05**	-0.01	0.00

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. Results are the average of five runs and the models were trained with the same set of parameters

5.4.1 Size of labeled data

Based on my testing and research the self-supervised learning is using the information from the unlabeled data better than the semi-supervised pseudo labels method which strongly relies on the quality of supervised trained model.

Table 5.9: Results of semi-supervised learning - pseudo labels confidence threshold

smooth.	0	1	2	3	4	5	6	7	8
0	0.89	0.86	0.77	0.97	0.69	0.85	0.59	0.94	0.98
0.50	0.91	0.87	0.75	0.97	0.72	0.86	0.67	0.93	0.98
0.70	0.92	0.86	0.80	0.97	0.72	0.86	0.68	0.92	0.99
0.90	0.92	0.87	0.80	0.97	0.72	0.88	**0.72**	0.96	0.99

0:Center, 1:Donut, 2:Edge-Local, 3:Edge-Ring, 4:Local, 5:Random, 6:Scratch, 7:Near-full, 8:None. Results represents average of five runs and the models were trained with the same set of parameters

Table 5.10: Results of self-supervised learning - WaPIRL

	Pretraining	Fine tuning	macro F1 score
ResNet-18	50	50	0.792
		100	0.878
	100	100	*0.895*
Simple model	50	50	0.807
		100	0.851
	100	50	0.796
		100	**0.862**

The numbers in the columns Pretraining and Fine-tuning are number of epochs. The results of the ResNet-18 with 100 pretraining and 100 fine-tuning epochs (*0.895*) is from the paper [13].

Conclusion

This work aims to analyze the state-of-the-art methods and propose a solution to improve the classification of defects on wafer bin maps on the WM-811K dataset. These methods can be applied on any other WBM dataset, bearing in mind the defect pattern types. This, in turn, means that the yield of the wafers will be higher which leads to better efficiency and use of resources. Due to the global shortage of chips, this might turn to be beneficial.

First, I tested the state-of-the-art networks using transfer learning. The best performance was achieved with Xception in supervised learning approach. For other testing, I have developed a smaller model that uses only 2-dimensional wafer input instead of 3-dimensional as the state-of-the-art models.

Based on my research, I implemented a semi-supervised approach using the pseudo-labels to improve the classification with utilization of unlabeled data. Another tested method was the self-supervised framework WaPIRL, where I added a smaller model as the backbone of the framework and tested its performance against the larger network.

In the third chapter, the analysis of the WM-811K dataset and pattern defects is described. I tested variants of WBM data preprocessing, augmentation and resizing. Because of the fact that WM-811K is heavily imbalanced as any other real world WBM dataset, I have implemented methods of data balancing.

The testing of semi-supervised and self-supervised methods showed, that the labeled data are essential. They are still used in the models to obtain the pseudo-labels or in fine-tuning after the self-pretraining. My model has achieved a competitive performance and the model itself is more suitable for the WBM classification task than the state-of-the-art models.

There are so many possible methods that need to be tried in the integrated circuits fabrication domain, which is very specific due to its data. Next, the testing of WaPIRL framework would need more time to test and implement new methods, that have potential to improve the model performance.

In my opinion the outcomes of this thesis were met, but there is still many methods that can be explored. Future steps might involve a search for the optimal small model or a usage of augmentation methods for data balancing or for a pretext tasks.

Bibliography

1. REINHARDT, K.; KERN, W. *Handbook of Silicon Wafer Cleaning Technology, 2nd Edition*. Elsevier Science, 2008. Materials science and process technology series. ISBN 9780815517733. Available also from: https://books.google.cz/books?id=5ORtHBrWyuQC.
2. JONES, Scotten W. *Introduction to Integrated Circuit Technology*. [N.d.]. Accessed: 20-3-2022.
3. SNOWDEN, Laird R. Process Control Monitor to Device Data Correlation. In: *45th ARFTG Conference Digest*. 1995, vol. 27, pp. 74–82. Available from DOI: 10.1109/ARFTG.1995.327108.
4. MANN, W.R.; TABER, F.L.; SEITZER, P.W.; BROZ, J.J. The leading edge of production wafer probe test technology. In: *2004 International Conference on Test*. 2004, pp. 1168–1195. Available from DOI: 10.1109/TEST.2004.1387391.
5. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM*. 2017, vol. 60, no. 6, pp. 84–90. ISSN 0001-0782. Available from DOI: 10.1145/3065386.
6. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. *Deep Residual Learning for Image Recognition*. arXiv, 2015. Available from DOI: 10.48550/ARXIV.1512.03385.
7. XU, Jing; PAN, Yu; PAN, Xinglin; HOI, Steven; YI, Zhang; XU, Zenglin. *RegNet: Self-Regulated Network for Image Classification*. arXiv, 2021. Available from DOI: 10.48550/ARXIV.2101.00590.
8. NAIR, Vinod; HINTON, Geoffrey. Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair. In: 2010, vol. 27, pp. 807–814.
9. SIMONYAN, Karen; ZISSERMAN, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv, 2014. Available from DOI: 10.48550/ARXIV.1409.1556.

10. CHOLLET, François. *Xception: Deep Learning with Depthwise Separable Convolutions*. arXiv, 2016. Available from DOI: 10.48550/ARXIV.1610.02357.

11. SZEGEDY, Christian; LIU, Wei; JIA, Yangqing; SERMANET, Pierre; REED, Scott; ANGUELOV, Dragomir; ERHAN, Dumitru; VANHOUCKE, Vincent; RABINOVICH, Andrew. *Going Deeper with Convolutions*. arXiv, 2014. Available from DOI: 10.48550/ARXIV.1409.4842.

12. WU, Ming-Ju; JANG, Jyh-Shing R.; CHEN, Jui-Long. Wafer Map Failure Pattern Recognition and Similarity Ranking for Large-Scale Data Sets. *IEEE Transactions on Semiconductor Manufacturing*. 2015, vol. 28, no. 1, pp. 1–12. Available from DOI: 10.1109/TSM.2014.2364237.

13. KAHNG, Hyungu; KIM, Seoung Bum. Self-Supervised Representation Learning for Wafer Bin Map Defect Pattern Classification. *IEEE Transactions on Semiconductor Manufacturing*. 2021, vol. 34, no. 1, pp. 74–86. Available from DOI: 10.1109/TSM.2020.3038165.

14. CHAPELLE, Olivier; SCHLKOPF, Bernhard; ZIEN, Alexander. Semi-Supervised Learning. *IEEE Transactions on Neural Networks*. 2006, vol. 20.

15. TARVAINEN, Antti; VALPOLA, Harri. *Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results*. arXiv, 2017. Available from DOI: 10.48550/ARXIV.1703.01780.

16. LAINE, Samuli; AILA, Timo. *Temporal Ensembling for Semi-Supervised Learning*. arXiv, 2016. Available from DOI: 10.48550/ARXIV.1610.02242.

17. LEE, Hyuck; KIM, Heeyoung. Semi-Supervised Multi-Label Learning for Classification of Wafer Bin Maps With Mixed-Type Defect Patterns. *IEEE Transactions on Semiconductor Manufacturing*. 2020, vol. 33, no. 4, pp. 653–662. Available from DOI: 10.1109/TSM.2020.3027431.

18. KINGMA, Diederik P.; REZENDE, Danilo J.; MOHAMED, Shakir; WELLING, Max. *Semi-Supervised Learning with Deep Generative Models*. arXiv, 2014. Available from DOI: 10.48550/ARXIV.1406.5298.

19. ZHU, Xiaojin; GOLDBERG, Andrew. *Introduction to Semi-Supervised Learning*. 2009. Available from DOI: 10.2200/S00196ED1V01Y200906AIM006.

20. ZHAI, Xiaohua; OLIVER, Avital; KOLESNIKOV, Alexander; BEYER, Lucas. *S4L: Self-Supervised Semi-Supervised Learning*. arXiv, 2019. Available from DOI: 10.48550/ARXIV.1905.03670.

21. LEE, Dong-Hyun. Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*. 2013.

22. KOLESNIKOV, Alexander; ZHAI, Xiaohua; BEYER, Lucas. *Revisiting Self-Supervised Visual Representation Learning*. arXiv, 2019. Available from DOI: 10.48550/ARXIV.1901.09005.

23. JI, YongSung; LEE, Jee-Hyong. Using GAN to Improve CNN Performance of Wafer Map Defect Type Classification : Yield Enhancement. In: *2020 31st Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*. 2020, pp. 1–6. Available from DOI: 10.1109/ASMC49169.2020.9185193.

24. SHIN, Wooksoo; KAHNG, Hyungu; KIM, Seoung Bum. Mixup-based classification of mixed-type defect patterns in wafer bin maps. *Computers & Industrial Engineering*. 2022, vol. 167, p. 107996. ISSN 0360-8352. Available from DOI: https://doi.org/10.1016/j.cie.2022.107996.

25. ZHANG, Hongyi; CISSE, Moustapha; DAUPHIN, Yann N.; LOPEZ-PAZ, David. *mixup: Beyond Empirical Risk Minimization*. arXiv, 2017. Available from DOI: 10.48550/ARXIV.1710.09412.

26. TERRA, John. *CKeras vs Tensorflow vs Pytorch: Key Differences Among the Deep Learning Framework* [online]. 2022 [visited on 2022-04-16]. Available from: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article.

27. ABADI, Martin; BARHAM, Paul; CHEN, Jianmin; CHEN, Zhifeng; DAVIS, Andy; DEAN, Jeffrey; DEVIN, Matthieu; GHEMAWAT, Sanjay; IRVING, Geoffrey; ISARD, Michael; KUDLUR, Manjunath; LEVENBERG, Josh; MONGA, Rajat; MOORE, Sherry; MURRAY, Derek G.; STEINER, Benoit; TUCKER, Paul; VASUDEVAN, Vijay; WARDEN, Pete; WICKE, Martin; YU, Yuan; ZHENG, Xiaoqiang. TensorFlow: A system for large-scale machine learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. Available also from: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

28. CHOLLET, François et al. *Keras* [https://keras.io]. 2015.

29. PASZKE, Adam; GROSS, Sam; MASSA, Francisco; LERER, Adam; BRADBURY, James; CHANAN, Gregory; KILLEEN, Trevor; LIN, Zeming; GIMELSHEIN, Natalia; ANTIGA, Luca; DESMAISON, Alban; KÖPF, Andreas; YANG, Edward; DEVITO, Zach; RAISON, Martin; TEJANI, Alykhan; CHILAMKURTHY, Sasank; STEINER, Benoit; FANG, Lu; BAI, Junjie; CHINTALA, Soumith. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv, 2019. Available from DOI: 10.48550/ARXIV.1912.01703.

30. DENG, Jia; DONG, Wei; SOCHER, Richard; LI, Li-Jia; LI, Kai; FEI-FEI, Li. Imagenet: A large-scale hierarchical image database. In: *2009 IEEE conference on computer vision and pattern recognition*. 2009, pp. 248–255.

31. AHMAD, Ijaz; SHIN, Seokjoo. An Approach to Run Pre-Trained Deep Learning Models on Grayscale Images. In: *2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. 2021, pp. 177–180. Available from DOI: 10.1109/ICAIIC51459.2021.9415275.

32. BATISTA, Gustavo E. A. P. A.; PRATI, Ronaldo C.; MONARD, Maria Carolina. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. *SIGKDD Explor. Newsl.* 2004, vol. 6, no. 1, pp. 20–29. ISSN 1931-0145. Available from DOI: 10.1145/1007730.1007735.

33. KRISTIADI, Agustinus; HEIN, Matthias; HENNIG, Philipp. *Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks*. arXiv, 2020. Available from DOI: 10.48550/ARXIV.2002.10118.

34. IOFFE, Sergey; SZEGEDY, Christian. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv, 2015. Available from DOI: 10.48550/ARXIV.1502.03167.

35. BA, Jimmy Lei; KIROS, Jamie Ryan; HINTON, Geoffrey E. *Layer Normalization*. arXiv, 2016. Available from DOI: 10.48550/ARXIV.1607.06450.

36. MÜLLER, Rafael; KORNBLITH, Simon; HINTON, Geoffrey. *When Does Label Smoothing Help?* arXiv, 2019. Available from DOI: 10.48550/ARXIV.1906.02629.

Appendix A

Acronyms

WBM Wafer Bin Map

IC Integrated Circuit

CVD Chemical Vapor Deposition

CMP Chemical Mechanical Planarization

CNN Convolutional Neural Network

LED Light-Emitting Diode

PCM Process Control Monitoring

WS Wafer Sort

ReLU Rectified Linear Unit

GPU Graphics Processing Unit

ResNet Residual Neural Network

RGB Red, Green, Blue

RNN Regulated Residual Network

ConvLSTM Convolutional Long Short-Term Memory

ConvRNN Convolutional Regulated Residual Network

RegNet Self-Regulated Network

EMA Exponential Moving Average

SS-DGM Semi-supervised Deep Generative Model

SS-CDGMM Semi-supervised Convolutional Deep Generative Multiple Model

SS-CDGM Semi-supervised Convolutional Deep Generative Model

HSV Hue, Saturation, Value

ILSVRC2012 ImageNet Large Scale Visual Recognition Challenge 2012

VAT Virtual Adversarial Training

MOAM Mix Of All Models

WaPIRL Wafer-oriented Pretext-invariant Representation Learning

GAN Generative Adversarial Network

DCGAN Deep Convolutional Generative Adversarial Network

DS Dataset

CNTK Microsoft Cognitive Toolkit

RAM Random Access Memory

Appendix B

Contents of Enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `data` ..... dataset, training, validation, test sets
  - `notebooks` ..... Jupyter notebooks and python modules
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format