UNIVERSITY OF PADUA

School of Engineering
Department of Information Engineering

Master of Science in
ICT for Internet and Multimedia

A DEEP LEARNING-BASED APPROACH FOR DEFECT CLASSIFICATION WITH CONTEXT INFORMATION IN SEMICONDUCTOR MANUFACTURING

Master Graduation Thesis by: SIMONE ARENA

Student Id: 1175697

Supervisor: GIAN ANTONIO SUSTO

Academic Year 2019-2020

24/02/2020

Avevamo studiato per l'aldilà
un fischio, un segno di riconoscimento.
Mi provo a modularlo nella speranza
che tutti siamo già morti senza saperlo.

— Eugenio Montale

ABSTRACT

In semiconductor manufacturing, machine learning and deep learning techniques have already become crucial for many relevant tasks such as anomaly detection, virtual metrology, predictive maintenance, fault detection, and classification. Moreover, thanks to the availability of large-scale image datasets and high-performance computing systems, deep learning models have recently achieved great successes in almost any large-scale image recognition task.
This thesis presents some methodological and experimental contributions to a deep learning-based approach for the automatic classification of microscopic defects in silicon wafers with context information.
Although the classification of defect patterns in wafers has been hugely studied in literature, the automatic categorization of microscopic defects has not been adequately addressed yet.
Furthermore, canonical deep learning-based image classification approaches have the limitation of utilizing only the information contained in the images. This work overcomes this limitation by using some context information about the defects, like the position of the defects in the wafer and in the die, to improve the current automatic classification system.
Of course there can be several strategies to embed context information with information extracted from images. In this work, we will analyse and experiment with some of these strategies and we will try to understand what are the most promising ones in the semiconductor manufacturing field.

SOMMARIO

Nella produzione di semiconduttori, le tecniche di machine learning e deep learning sono già diventate cruciali per molte attività rilevanti come l'individuazione di anomalie, la metrologia virtuale, la manutenzione predittiva, il rilevamento dei guasti e la classificazione. Inoltre, grazie alla disponibilità di dataset di immagini su larga scala e sistemi di elaborazione ad alte prestazioni, i modelli di deep learning hanno recentemente ottenuto grandi successi in quasi tutte le attività di riconoscimento delle immagini.
Questa tesi presenta alcuni contributi teorici e sperimentali ad un approccio basato sul deep learning per la classificazione automatica dei difetti microscopici nei wafer di silicio con informazioni di contesto.
Sebbene la classificazione di pattern di difetti nei wafer sia stata ampiamente studiata in letteratura, la categorizzazione automatica dei difetti microscopici non è stata ancora adeguatamente affrontata.
Inoltre, gli approcci di deep learning canonici per classificazione delle immagini hanno la limitazione di utilizzare solo le informazioni contenute nelle immagini. Questo lavoro supera questo limite attraverso l'utilizzo di alcune informazioni di contesto sui difetti, come la posizione dei difetti nel wafer e nel die, per migliorare l'attuale sistema di classificazione automatica.
Naturalmente ci possono essere diverse strategie per incorporare le informazioni di contesto con le informazioni estratte dalle immagini. In questo lavoro analizzeremo e sperimenteremo alcune di queste strategie e cercheremo di capire quali sono le più promettenti nel campo della produzione di semiconduttori.

ACKNOWLEDGMENTS

I would like to thank Prof. Gian Antonio Susto and Mattia Carletti for supporting me throughout the duration of my thesis work. I will bring with me pleasant memories of our "brainstormings".
My acknowledgments are addressed also to the Infineon team that guided me in this use case and from which I learnt a lot.
Finally, I would like to thank the University of Padua for allowing me to deepen my knowledge of my fields of interest and for introducing me to other equally interesting topics.

RINGRAZIAMENTI

Ringrazio i miei genitori. Dal momento in cui metto piede all'aeroporto, vi bastano pochi secondi per fare in modo che il tempo passato fuori non fosse mai trascorso. Due cose sono certe: l'infinitezza dell'universo e le vostre solite frasi a tavola, ma riguardo l'universo ho ancora dei dubbi.
Ringrazio Adriele. Se non sei qui significa che non hai ancora dato meccanica quantistica avanzata... o forse no... o forse entrambe.
Ringrazio il resto della famiglia, ormai tutti avanti con l'età (lol). Non vi preoccupate dell'assenza, Emanuele e Damiano saranno i messaggeri della corte Felina.
Ringrazio gli amici di sempre, da oltre 10 anni la mia seconda famiglia. Probabilmente non sarete qua, ma ben mi sta dato che ho saltato la maggior parte delle vostre lauree.
Ringrazio il TeamBallodiRiso, saggio oroscopo da consultare nei momenti di incertezza.
Ringrazio i miei amici disagiati e "cuttigghiari" dell'Ederle. Non si può dire che non abbiate messo alla prova la mia pazienza. Ma più che altro, ho messo a dura prova la vostra. E adesso ho qualche fratello e sorella in più.
Ringrazio gli altri amici non meno disagiati della Carli. Siete riusciti a non farmi rimpiangere i due anni precedenti. Mi dispiace lasciarvi prematuramente. Spero che capiate presto come fare la spesa per le feste.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

CP circuit probe

IC integrated circuit

WBM Wafer Bin Map

ML machine learning

ART1 adaptive resonance theory network

SVM Support Vector Machine

ANN Artificial Neural Network

CNN Convolutional Neural Network

WMSR Wafer map similarity ranking

SOM Self Organizing Map

OPTICS Ordering Point To Identify the Cluster Structure

JLNDA Joint Local and Nonlocal Discriminant Analysis

SVE Soft Voting Ensemble

SEM Scanning Electron Microscopes

ACS American Community Survey

CDML Cost-sensitive Deep Metric Learning

Part I

Problem description

1

INTRODUCTION

In semiconductor manufacturing, a wafer is a thin slice of semiconductor used for the fabrication integrated circuits (ICs) and which serves as substrate for microelectronic devices. The process of wafer fabrication involves several chemical and mechanical steps to produce ICs on wafers. One of such steps is wafer dicing, during which the wafer is divided into many dies. After the wafer fabrication, the wafer dies are electrically tested by means of a circuit probe (CP) test to evaluate the correct functionality of the integrated circuits. The resulting spatial outcome of the CP tests on a wafer is called Wafer Bin Map (WBM). The WBM may consist of binary values representing the pass/fail outcome of the CP test in each die, or continuous values representing the electrical measurements.

Typically, two kinds of defects occur in WBMs: global defects and local defects. Global defects are distributed all over the wafer and do not usually say much about the root problem that may have caused them. Local defects, instead, are normally characterized by significant spatial patterns that may provide useful information on specific manufacturing issues. Typical spatial patterns are rings, scratches, centers, donuts, circles, and semicircles. Some of these patterns are associated with different manufacturing problems; for example, the linear scratch is caused by the machine handling, the edge ring is caused by problems related to the etching process, and the center usually arises from the thin film deposition [19] [31].

Traditionally, wafer map defect detection and pattern recognition were performed by experienced human operators. However, according to the study conducted by A. Drozda-Freeman [1] the defect detection accuracy achieved by human experts is less than 45%. Furthermore, the size of electronic dies has been decreasing over time and wafers are getting larger, therefore human expert evaluation is getting impractical and the need for automated systems is becoming crucial.

1.1 RELATED WORK

Recently, many unsupervised and supervised machine learning (ML) algorithms have been applied to defect pattern detection and recognition. Some important unsupervised learning-based techniques used for wafer map defect classification are adaptive resonance theory network (ART1) [10] [2], Self Organizing Maps (SOMs) [10], multi-step ART1

[11], K-means [3], fuzzy K-means [4], and particle swarm optimization [13]. Yuan and Kuo proposed a model-based clustering algorithm which models the distribution of defects on wafer surface [43]. Their model is able to detect defect clusters with linear patterns, curvilinear patterns and ellipsoidal patterns. Liu and Chien developed a WBM clustering approach which integrates spatial statistics test, cellular neural network, adaptive resonance theory neural network and moment invariant to cluster different patterns effectively [5]. Hsu proposed a clustering ensemble approach to facilitate WBM defect pattern extraction [13]. Firstly, the two-dimensional wafer maps are mapped to one-dimensional data. Secondly, K-means and particle swarm optimization clustering algorithms are used to generate various diversity partitions. Finally, an adaptive response theory neural network is used to aggregate the diversity partitions. In [30], SOMs are combined with K-means clustering to extract systematic data patterns from spatially oriented wafer maps. In their two-stage solution, data is first processed by a SOM, and then the reference vectors of the SOM are clustered using K-means.

When class labels are available, supervised learning techniques can yield better results than unsupervised learning methods. Support Vector Machines (SVMs) [33], Artificial Neural Networks (ANNs), general regression neural networks, back-propagation networks are usually applied for wafer map defect classification. In [33], Wu et al. pointed out that most of the previously developed wafer map failure pattern recognition systems used raw wafer maps as input data, which is not feasible for large-scale datasets. Moreover, wafer-based clustering does not preserve the rotation-invariance property; namely, two identical failure patterns with different orientation might be classified as different failure patterns. For these reasons, they proposed a set of rotation-invariant and scale-invariant features for producing a reduced representation of wafer maps. In their workflow, Radon-based and geometry-based features were first extracted from wafer maps and then combined; consequently, an SVM classifier was used to recognize the failure pattern. The extracted features were also used for similarity ranking. Wafer map similarity ranking (WMSR) is the task of retrieving all the wafer maps which present similar failure patterns to a queried wafer map. WMSR is motivated by the fact that similar failure patterns may have identical failure causes. Wu et al. performed WMSR in two stages. In the first stage, given a queried wafer map, the top-n similar wafer maps were selected based on the Euclidian distance among the extracted features. In the second stage, the top-n wafer maps extracted in the previous step are ranked according to the 2D normalized correlation coefficient. They also built the WM-811K dataset, which comprises 811457 real-world wafer maps collected from 46293 lots in real-world fabrication and divided into nine classes: Cen-

ter, Donut, Edge-local, Edge-ring, Local, Near-full, Random, Scratch, and Nonpattern.

The main limitation of [33] is that no more than one failure pattern can be detected in a wafer map. Fan et al. overcame this issue by combining the Ordering Point To Identify the Cluster Structure (OPTICS) clustering method with SVM classifier [29]. OPTICS is a density-based clustering technique which can detect arbitrarily shaped clusters without fixing the number of clusters a priori. Their method is divided into three steps: clustering, feature extraction and pattern recognition. During the training phase, salient clusters of wafer maps are derived through OPTICS and then density-based and geometry-based feature are extracted; afterwards, failure patterns are detected by means of SVM classifier. In the testing phase, instead, a test wafer map is labeled as Nonpattern if no cluster is detected. Otherwise, if one or more clusters are detected, features are extracted from each pattern and SVMs are used to classify each cluster. Each wafer is finally labeled accordingly to the classification result of each cluster.

Generally, the original feature set extracted from wafer maps is high-dimensional. Yu and Lu proposed a new manifold learning algorithm called Joint Local and Nonlocal Discriminant Analysis (JLNDA) to reduce the dimensionality of the feature set [19]. JLNDA tries both to maximize the inter-class separability by maximizing the distance of the projected samples of different clusters, and to preserve the intra-class local geometric structure. In this way both local and nonlocal information are retained. In their workflow, wafer maps are firstly denoised by means of a median filter. Secondly, geometrical features, gray features, texture features, and projection features are extracted from wafer maps. Then, local and nonlocal preserving projection-based and JLNDA-based methods are used for dimensionality reduction.

Piao et al. proposed a decision tree ensemble-based approach to aggregate and strengthen the contribution of different features sets in the wafer map failure pattern recognition [31]. Differently from the previous works, they only used Radon transform feature sets. After denoising by means of a median filter, Radon transform is applied to wafer maps and the maximum, minimum, average, and standard deviation of the projections are computed. Then a tree committee is built to aggregate the contribution of such features.

Each ML algorithm suffers from some limitations and the problem of finding the most suitable algorithm for defect classification in wafer maps is anything but trivial. Moreover, some classifiers may be specialized in detecting some defect classes, while other classifiers can have great expertise in discriminating other classes. Motivated by this,

Saqlain et al. proposed a Soft Voting Ensemble (SVE) classifier with multi-types features [32]. They first extracted density-, geometry-, and radon-based features from wafer maps, and then applied an ensemble of four classifiers: logistic regression , random forests, gradient boosting machine, and ANNs.

Recently, deep learning models, and especially Convolutional Neural Networks (CNNs), have become a de facto standard for any pattern recognition and image classification problem. Differently from other ML techniques, CNNs often don't need the preprocessing and feature extraction steps because they are able to learn abstract features which otherwise should be derived manually. Moreover, CNNs are robust to random noise and enjoy the equivariance property, therefore they are able to detect defect patterns regardless of the specific position and orientation. In [41], 28600 wafer map images for 22 defect classes were artificially generated and a CNN was employed for defect pattern classification. Moreover, they generated binary codes for wafer maps from the fully connected layer of the CNN and they used them for wafer maps retrieval tasks. Kyeong and Kim proposed a new approach for classifying mixed-type defect patterns using CNNs [24]. They built an individual CNN classifier for each defect class. They considered four classes, namely Circle, Scratch, Ring, and Zone, and they used both real and simulated data for training. If two defect patterns coexist, two classification models are expected to detect them, while the other two models won't notice them. Tello et al. used CNNs to improve classification accuracy on wafers which present multiple defect patterns [12]. Their approach consists of three phases. In the first phase a spatial filter is used to reduce random noise. In the second phase, 21 different features are extracted from wafer maps and a splitter based on information gain theory utilizes such features to build rules capable of splitting wafers as single-pattern or mixed-pattern. In the third phase, if the wafer has been labeled as single-pattern, then a randomized general regression network is used to classify it, otherwise, if the wafer has been labeled as mixed-pattern, a deep structured convolutional network carries out the prediction. Furthermore, in [42] a CNN and extreme gradient boosting are employed for wafer map retrieval tasks and defect pattern classification.

While the problem of classifying defects on wafer-level images has been widely studied in literature, the classification of microscopic defects at chip-level on silicon wafers has not been adequately addressed. This thesis work deals with the almost unexplored world of microscopic defects classification.

In the semiconductor fabrication process, ICs are made by linking many circuit structures on many layers of a wafer. Each circuit layer

is realised through the following steps: photolithography, etching, deposition, ion implantation, diffusion, and chemical and mechanical polarization. To fabricate high-density ICs, the wafer surface must be extremely clean and the circuit layers should be aligned to each other [36]. In order to inspect if there are any particles, spots, scratches, or irregular connections caused by misaligned circuits on stacked layers, Scanning Electron Microscopes (SEM) images of the wafer surface are acquired after the completion of each layer (specially between the etching and deposition steps). Such images can be used to detect microscopic defects and classify them as repairable or unrepairable defects. Repairable defects are limited to particle-type defects, which can be reworked by cleaning the surface with an air blower [36]. Cheon et al. proposed a CNN-based automatic defect classification system for classifying various types of wafer surface damages [36]. Moreover, they applied a k-NN anomaly detection algorithm to the feature set extracted by the CNN to identify unseen classes during training. Their CNN architecture consists of one input layer, two blocks of convolutional-convolutional-pooling layers, one fully-connected layer, and one output layer.

1.2 HANDCRAFTED FEATURES

This section overviews the manually-extracted features which have been used so far for WMSR and defect pattern recognition before the advent of deep learning techniques.

1.2.1 *Density-based features*

Density-based features are obtained by dividing the wafer maps into n regions and by computing the failure density on each region. These features turned out to be discriminative among classes since different defect classes have different defect density distribution in each region. [29] and [32] employed such kind of features.

1.2.2 *Geometrical features*

Geometry-based features are the most commonly employed features for wafer map pattern recognition and can be extracted by computing the linear and regional attributes. The number of lines detected through the Hough transform is often used as linear attribute. As regards regional attributes, a region-labeling algorithm is firstly applied to identify multiple defect regions in each wafer map. Then the most salient region, namely the region with the maximal area, is selected. Finally, some properties of the maximal region, such as its area, perimeter, eccentricity, compactness, and rectangular degree, are

chosen as regional attributes.

1.2.3 *Gray features*

The gray histogram characterizes the pixel distribution at different grayscale in wafer maps. The mean, variance, skewness, peak, energy, and entropy of the gray histogram of the most salient region can be employed as gray features, as done by [19].

1.2.4 *Texture features*

The gray level co-occurrence matrix of an image is the distribution of co-occurring grayscale pixel values at a given offset. In other words, it tells us how often different combination of pixel gray levels occur in a wafer map. Some typical statistics of this matrix, like the energy, contract, correlations, entropy, and uniformity, can be utilized as texture features, like shown by [19].

1.2.5 *Radon-based features*

The Radon transform is the projection of an image along a radial line oriented at a certain angle. Through several projections, the Radon transform can be used to generate a bidimensional representation of a wafer map. Therefore, a wafer map ca be described by a matrix G where in each entry of G the radon transform computed at a certain position and orientation is stored. Then the raw mean G_m and raw standard deviation G_s are computed from G. Afterwards G_m and G_s are resampled, for example by using cubic interpolation, to obtain the final Radon-based features [33].

1.3 CLASSIFICATION WITH CONTEXT

Both classical ML and deep learning approaches have the limitation of utilizing only the information contained in the image to classify the defect. Differently, human experts also consider some domain knowledge about the context in which the sample defects were acquired. For instance, when performing defect classification by hand, engineers take into account that some defect types can only appear in the memory section of a chip.

Actually, the problem of extracting useful information from context metadata do not pertain only to the semiconductor manufacturing world, but it's a more general problem. The work by [21] deals with object recognition and visual positioning in urban environments through the use of geo-services on mobile devices. In their work, geo-information was used in combination with visual features to constrain the search to a local context. In [35], a set of priors acquired from Geographical Information System databases was used to improve object detection. The priors were extracted from the exact positions of traffic signals, road signs, fire hydrants, and street lights in urban areas. Divvala et al. classified contextual information of real-world pictures into different categories: local pixel, 2D scene gist, 3D geometric, semantic, photogrammetric, illumination, weather, geographic, temporal, and cultural context [37]. Their main contribution was to develop a standardized setting on which evaluating different types of context. The work of [18] exploited context information extracted from the season and rough location in which pictures were taken to improve the performance of object region recognition and annotation. Tang et al., in collaboration with Stanford University and with Facebook AI Research group, tackled the problem of performing image classification with location context [26]. By exploiting the GPS coordinates of the images, they were able to use geographic datasets and surveys collected by different institution and agencies to improve classification performances. Their work mainly consists of two steps:

1. Constructing effective location features from GPS coordinates.
2. Incorporating such location features into the CNN architecture.

In the first step they extracted four types of features, namely geographic map, American Community Survey (ACS), hashtag context, and visual context features.
Geographic map features were extracted by using 10 different types of maps from Google Maps. Each map contains information about the location in the form of a colored map, where different colors stand for different features. For each image, they took the normalized pixel color values in a 17x17 patch around the image coordinates and used these values as geographic map features. Intuitively, features extracted from precipitation, temperature or elevation maps may tell us how likely is to see an umbrella, a coat or some snow in a picture.
They extracted ACS features by exploiting the ongoing ACS survey which provides statistical data about the age, sex, income, health insurance, work status and other living conditions, arranged by zip code. Statistics like age and income may convey information about the probability of finding a toy or an expensive car in a picture.
A lot of context information lies directly on the internet. For this reason, they employed the distribution of Instagram hashtags in a neighborhood of the images as hashtag context features. Similarly,

visual context features were extracted by exploiting the visual signal around the GPS coordinates of the images.
In the second step, they tried to concatenate the previously extracted features in the CNN architecture at different depths.
In [17], context metadata was used to improve the classification accuracy on plankton images. In their work, they incorporated graphic, geo-temporal, and geometric metadata to boost the performance of CNN classifiers. Similarly to [26], they tried to combine the features extracted from context metadata at different depth of the CNN architecture.

This work aims at using context information about the defects to improve the current deep learning-based automatic classification system. As we will see in Chapter 2, the contextual information available for our industrial case study is quite different from the context information employed in the aforementioned works. However, the methodology and some ideas can be extended to our use case as well.

1.4 OVERVIEW

This section outlines the general structure of the thesis. Chapter 2 presents a description of the data cleaning pipeline and of the available context information, and an overview of the data pre-processing strategy. Chapter 3 mainly analyses how different defect classes are distributed over the wafer and over the die. Moreover, a metric to measure distance between classes is formally defined. In Chapter 4 the methodology of the work is described. Firstly, some metrics to evaluate the effectiveness of the model are defined. Secondly, two different strategies to embed information from context attributes into the network architecture are analysed. Particularly, a probabilistic framework is formalised and applied to simple examples. Chapter 5 reviews the results of the several experiments. Chapter 6 sums up the potentialities and limitations of current work and attempts to give some guidelines for future improvements. In Appendix A some of the most famous state-of-the-art architectures based on Convolutional Networks are described.

Part II

Industrial case study

2

DATASET

All the data described in this section has been provided us by Infineon Technology. Specifically, the available data for the Defect Image Classification use cases consists of a collection of 10 datasets with 2.5 million images. Images are taken by SEMs. A SEM is a type of electron microscope which produces images of a specimen by scanning its surface with a focused beam of electrons. The electrons, by interacting with atoms in the specimen, produce several informative signals about the surface topography and composition of the specimen. Secondary electrons emitted by the specimen's atoms excited by the electron beam can be detected by using an in-lens detector or an external detector. Depending on the type of detector, SEM images can be therefore divided into in-lens detector images and external detector images. We will refer to in-lens detector images as '000' images and to external detector images as '001' images. SEM images can have resolution higher than one nanometer.

Besides SEM images, some context information about defects is available. Table 1 overviews some of the context attributes.

2.1 DATA CLEANING PIPELINE

All datasets were cleaned according the following steps:

1. Remove missing, broken and duplicate images.
2. Remove all images whose resolution is not 480x480.
3. Remove all images whose image number is not '000' or '001'
4. Remove all defects which don't have both '000' and '001' images.
5. Remove all defects whose manual labels are invalid or don't exist in the defect catalog.

ATTRIBUTES	EXPLANATION
Technology	self-explanatory
Product	self-explanatory
Lot	uniquely identifies a collection of 25 wafers, which are processed together
Wafer	uniquely identifies a wafer within a lot
Step	operation number of the defect inspection step
defect id	uniquely identifies a defect
Equipment	measurement tool which detected the defect
xsize[μm], ysize[μm]	defect size as determined by the existing defect detection software
dsize[μm], area[μm^2]	die size as determined by the existing defect detection software
x(wafer)[μm], y(wafer)[μm]	xy coordinates of the defect on the wafer
x(die)[μm], y(die)[μm]	xy coordinates of the defect within the chip on the wafer
die x, die y	position of the chip on the wafer
Timestamp	self-explanatory

Table 1: Overview of context information.

2.2 DATASET SELECTION AND PREPROCESSING

Among all the available datasets, we decided to pick the biggest and most tested one. Such dataset, which has been code-named as 'kiel' dataset, contains both '000' and '001' SEM images taken at the second, third, and fourth metal layers. The dataset has been cleaned according to the procedure described in section 2.1. Moreover, for this work, only '000' images have been used.
The clean Kiel dataset consists of about 320000 images from 52 classes; however, only around 84% of the images have a corresponding data entry in the context information table. Kiel dataset and its context information have been joined by defect Id; after the join operation we end up with about 268000 samples. Based on class distribution, 15 out of 52 classes have been selected. Such classes cover 90% of the volume. All the other classes have been grouped into a single class, which we will name as '404' class. The distribution of the classes is imbalanced. Figure 1 shows an histogram representing the number of samples for each class.

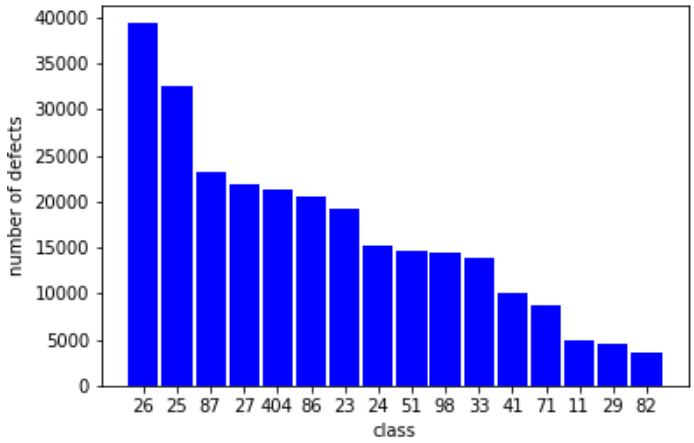


Figure 1: Distribution of the classes

The 'wafer' attribute shown in table 1 is an integer number between 1 and 25 which uniquely identifies a wafer within a lot; however, such attribute is not unique in absolute. That's why we needed to create new wafer identifiers. Such identifiers can be created by simply concatenating the 'lot' attribute with the 'wafer' attribute.

Figure 2 shows some clean SEM images of defects belonging to different classes. In figure 3, instead, we can find an example of defect annotations in three different wafers. Notice that defects of different classes are represented with different colours.

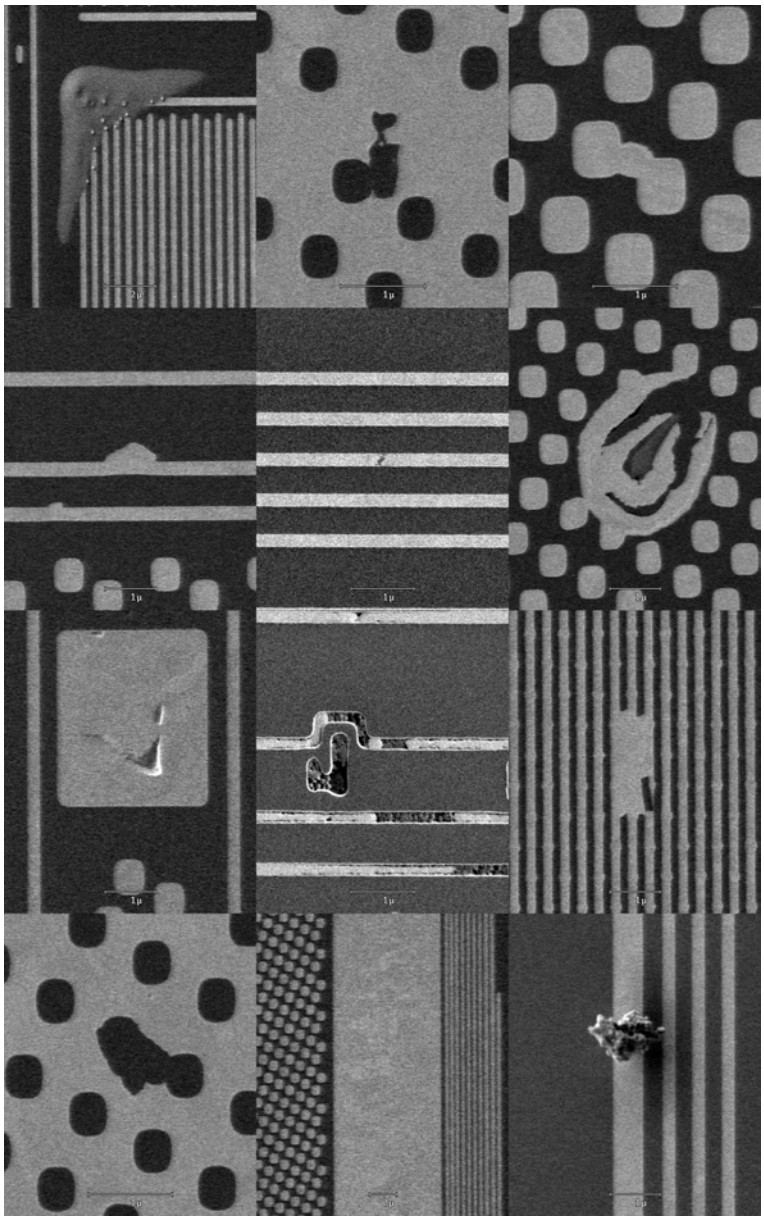
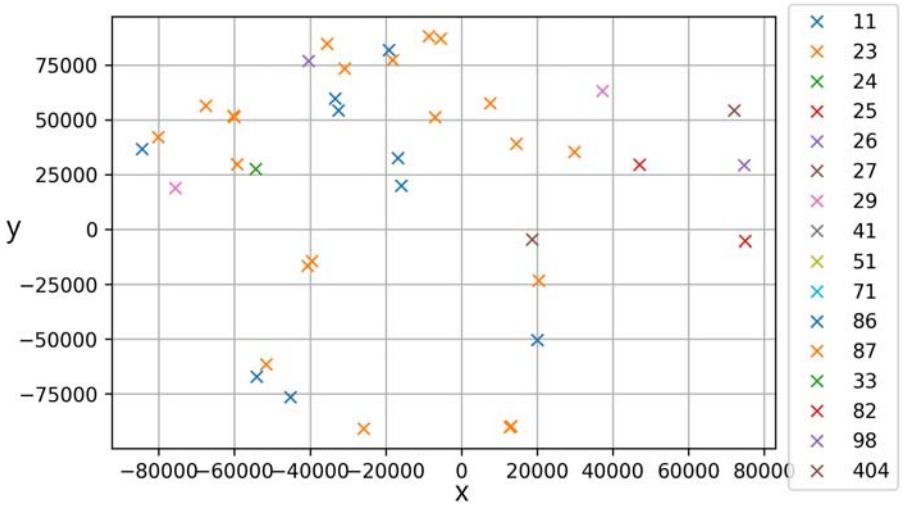
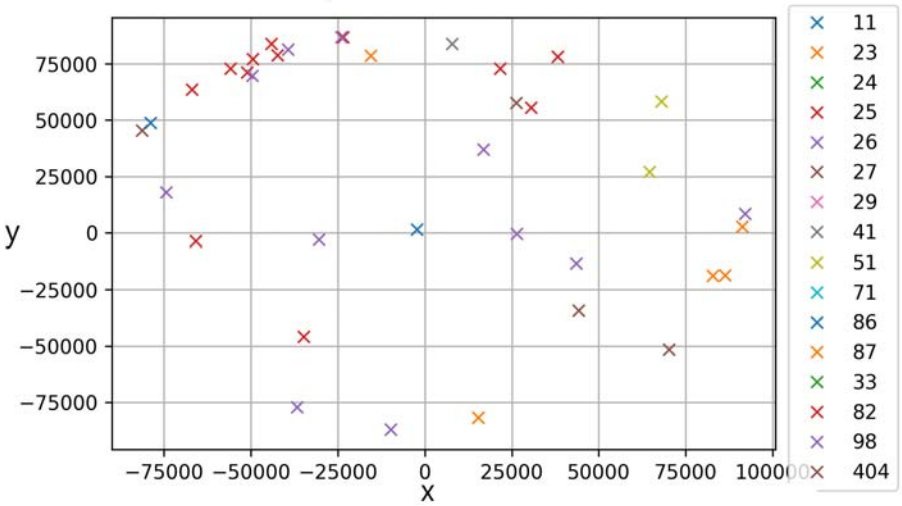


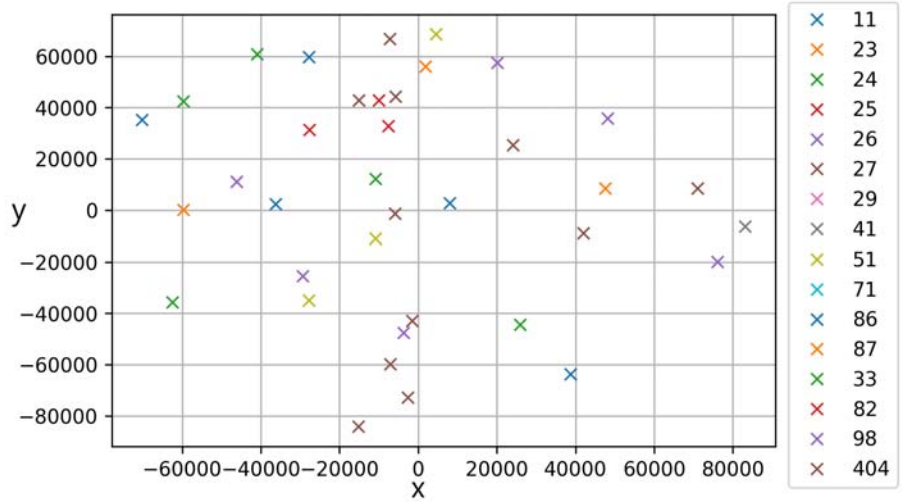
Figure 2: Clean SEM images belonging to different defect classes



(a) Annotation on wafer AB572058-08



(b) Annotation on wafer AB340943-25



(c) Annotation on wafer AB886432-15

Figure 3: Defect annotations on 3 different wafers.

3 DATA ANALYSIS

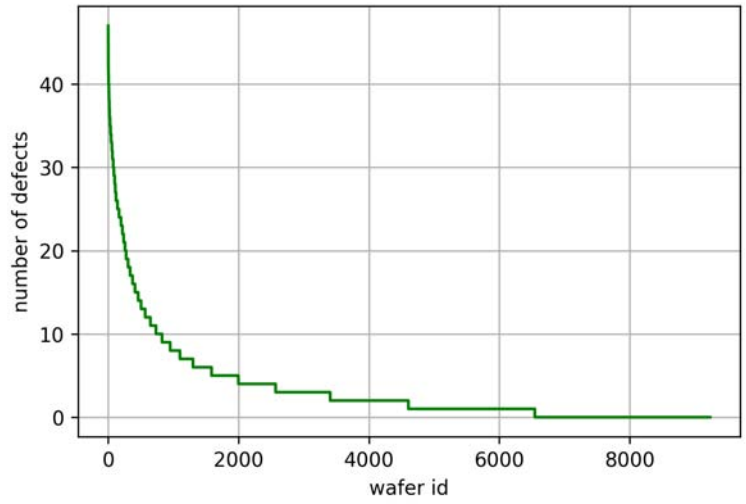
The data analysis described in this section has been performed on a subset of 190000 samples e.g. the training set. However, similar results have been observed also for the validation and test sets. The details about the train/validation/test split are described in Chapter 4.
The analysis mainly focuses on understanding how different defect classes are distributed over the wafer and over the die.

3.1 DATA ANALYSIS: WAFER LEVEL

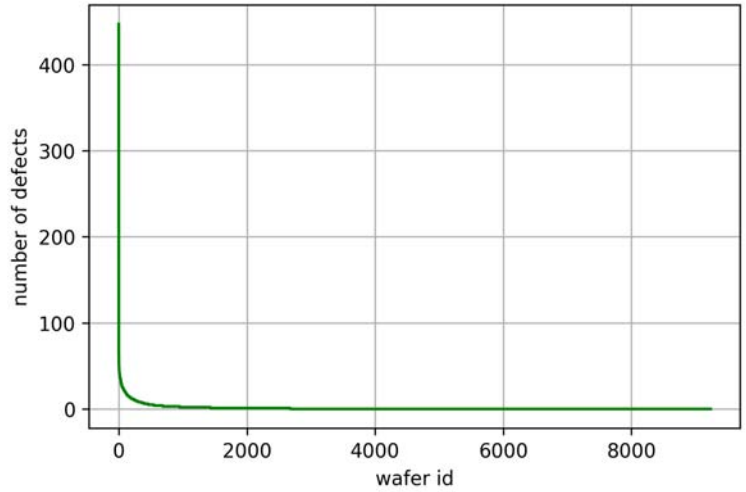
Defects on the training set are taken from 9236 wafers. Figure 4 shows the tail distribution of the fraction of defects from each wafer for classes '26', '33', and '71'. All the other classes have tail distributions almost identical to the distribution of class '26'. On the other hand, classes '33' and '71' have peculiar tail distributions which look more skewed than the others.
The x and y coordinates of the defects in the wafer ranges from -100000 to +100000. The first step of the analysis is to build density matrices for each defect class. In order to do so we divided the wafer space into a square grid containing 400 "little" squares of dimension 10000x10000. Then, for each class, we computed the number of defects which lie on the same "little" square in the grid. Afterwards, we normalized the obtained counts by dividing by the total number of defects per each class. The heatmaps represented in figure 5 show the results we obtained. It is possible to notice that some classes present specific patterns on the wafer. For example, class '51' presents a ring pattern near the edge of the wafer, and classes '25', '26', and '27' present a high defect density in the bottom right edge of the wafer. Moreover, we can notice that there are classes which present small regions with high defect density. For instance, in classes '33' and '404' the defects are mainly distributed on the small white squares shown in figure 5. Such small regions with high defect density may be due to:

1. bugs in defects' coordinates,
2. the fact that defects for a certain class are mostly taken from a specific wafer.

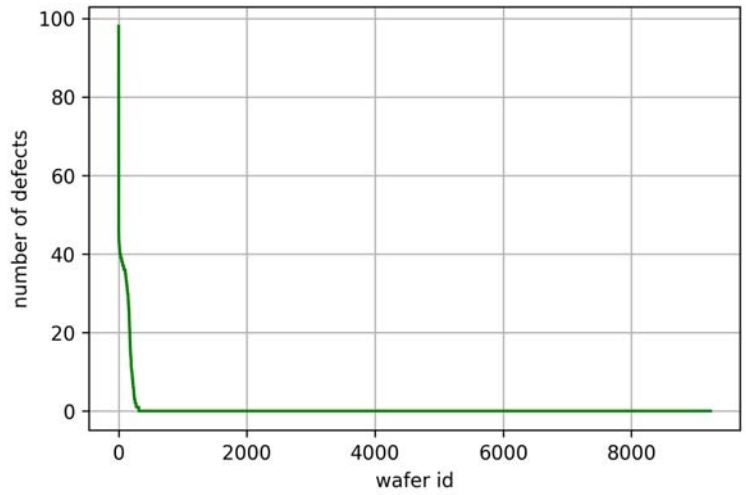
We made further investigations on this issue and we discovered that these two events do not actually occur. Therefore, we have reasons to believe that high defect density regions are indeed a class property.



(a) Class '26'.



(b) Class '33'.



(c) Class '71'.

Figure 4: Tail distributions of the fraction of defects taken from each wafer for three classes.

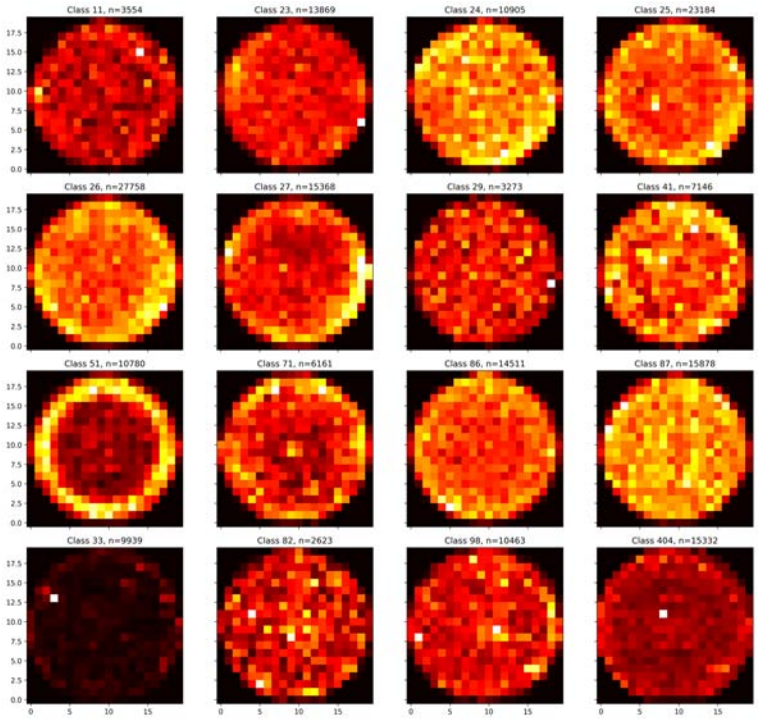


Figure 5: Heatmaps which represent how defects are distributed on the wafer for each class.

Based on the heatmaps shown above, we defined a metric to measure similarity among classes. Consider the sample heatmap shown below.

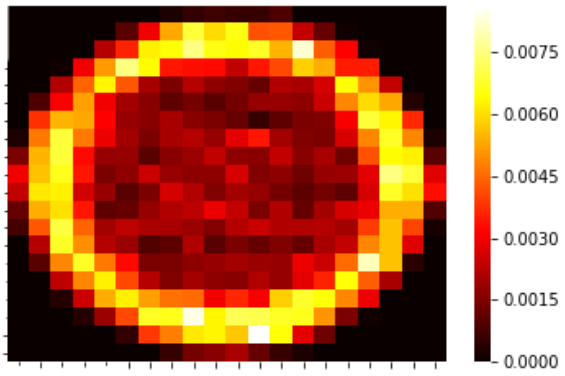


Figure 6: Example of heatmap.

Recall that, given a class C, each little square in the grid contains the fraction of defects in that area. The distance between two classes is defined as the sum of the pairwise differences between the values stored in each little square of the grid.

Formally, given two classes C_1 and C_2 and their respective density matrices D^{c_1} and D^{c_2} , the distance between C_1 and C_2 is defined as:

$$dist(C_1, C_2) = \sum_{i=1}^{20} \sum_{j=1}^{20} [D_{ij}^{c_1} - D_{ij}^{c_2}] \tag{1}$$

where $D_{ij}^{c_k}$ is the i, j entry of D^{c_k} for any class k.
Consequently, the similarity between C_1 and C_2 is defined as:

$$sim(C_1, C_2) = 1 - dist(C_1, C_2) \tag{2}$$

We can easily notice that if $C_1 = C_2$, and therefore $D^{c_1} = D^{c_2}$, then $dist(C_1, C_2) = 0$ and $sim(C_1, C_2) = 1$. Instead, if C_1 and C_2 present defects in disjoint sets of "little" squares, then $dist(C_1, C_2) = 1$ and $sim(C_1, C_2) = 0$

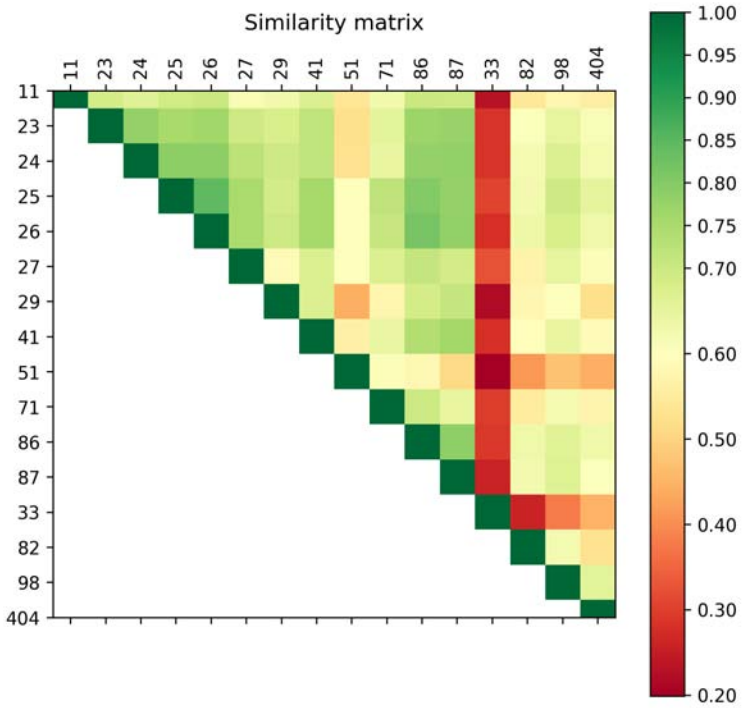


Figure 7: Similarity matrix at wafer level.

Based on this definition, class '33' turns out to be very different from all the other classes, as we can notice from figure 7. Class '51' looks quite different from the other classes as well. Moreover, it can be noticed that classes '23', '24', '25', '26', '86', '87' are quite close to each other.

3.2 DATA ANALYSIS: DIE LEVEL

Analogously to what has been done at wafer level, we derived the density matrices which represent how defects of each class are distributed over the die. However, in this case the process was not straightforward because the dies have different dimension depending on their basic type as shown in figures 8 and 9. Therefore, for each basic type we estimated the die dimension by taking the maximum x and y values. Then, we normalized all defects' coordinates between 0 and 1 by dividing the original coordinates by their respective die size.

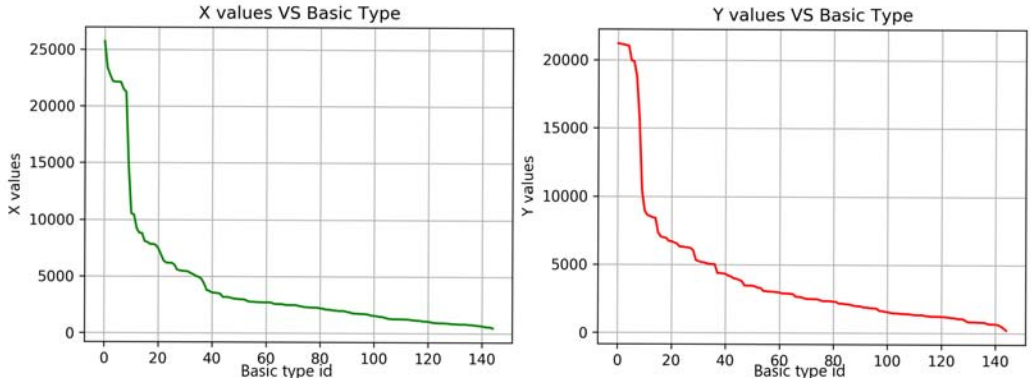


Figure 8: Tail distribution of the estimated width and length of dies for each basic type.

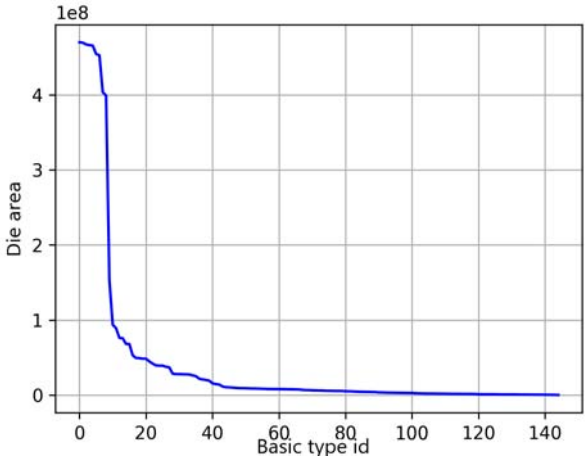


Figure 9: Tail distribution of the estimated area of dies for each basic type.

These plots show the dies' sizes as function of the basic type. We can observe that there are few basic types (about 10%) for which the die dimension is noticeably bigger than the others.
Now that the coordinates of the defects in the die have the same scale, it is possible to plot the heatmaps likewise to what was done for

defects' coordinates in the wafer. Also in this case, we divided the die space into a square grid of 400 "little" squares and made normalized counts of defects in each "little" square. The result is shown in figure 10.

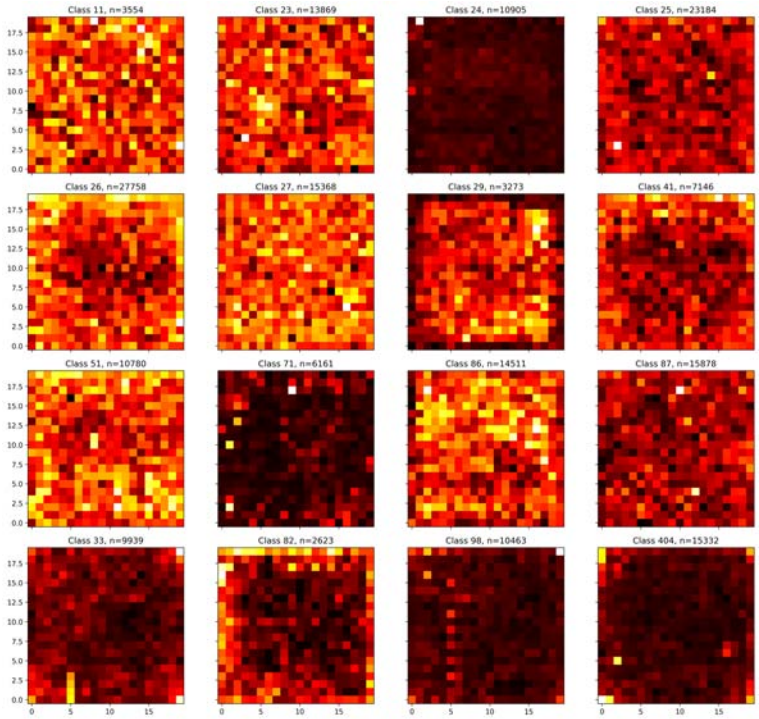


Figure 10: Heatmaps which represent how defects are distributed on the die for each class

From the previous figure it is possible to notice that some classes present specific patterns in the die; for example for class '29' the majority of defects do not lie in the border of the die, while for class '82' most of defects lie in the border.
Subsequently, we used the same similarity metric adopted before to measure distances among classes at die level. The similarity matrix is shown in figure 11.
In this case, we can notice that classes '71' and '82' are very different from the other classes; class '29' is considerably different from the other classes as well.

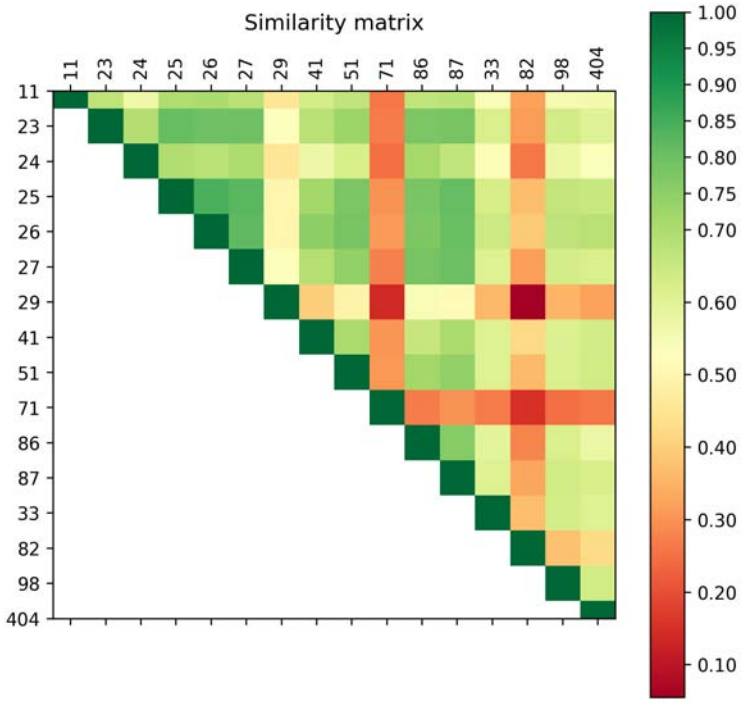


Figure 11: Similarity matrix at die level.

4 CLASSIFICATION FRAMEWORK

Figure 12 overviews the classification framework we exploited for our experiments. Given an input image to the model, it outputs a prediction X with a certain confidence. If X is a focus class (not a '404' class) and the confidence is high enough the prediction is taken as good, otherwise it is manually reviewed. Specifically, a softmax threshold is used to filter out uncertain predictions.
Therefore, besides classification accuracy, precision, and recall, it makes sense to evaluate the effectiveness of the model also in terms of:

1. Remaining effort: number of images that need to be manually reviewed divided by the total number of images. This metric states how much work is still to be done after the model is introduced.
2. Effort reduction: 1 - remaining effort. It states how much effort can be saved if the model is introduced.
3. Productivity gain: inverse of the remaining effort. This measures how much more total volume an operator supported by a deep learning model could handle.

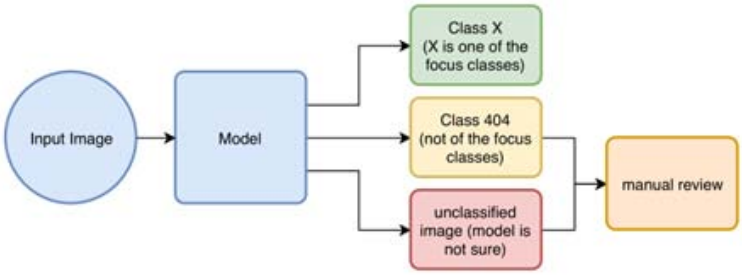


Figure 12: Classification framework (taken from Infineon).

4.1 PREVIOUS WORK

In the previous work, after data cleaning, the data was split into a training and test set. The training and test sets consist of 80% and 20% of the data respectively. The partition was done by using a stratified 5-fold split where all images were randomly assigned to one of the 5 folds while maintaining class balance within each fold.

After that, some transfer learning strategies have been applied to well-known architectures (like VGG-16 [25], ResNet [22] [23], Inception [7] [6], and Xception [8]) trained on the ImageNet database [16]. For a survey of transfer learning refer to [27]. Instead, a description of the aforementioned network architectures is given in Appendix A.
Different hyper-parameter settings for such architectures were investigated and the models were evaluated on actual production data.
As we saw in Chapter 2, the dataset is imbalanced. To tackle this issue, the costs of the different classes were adjusted such that under-represented classes were given more importance during training and vice versa. Note that this is only a possible way to deal with class imbalance problem; for example, Wang et al. proposed a novel loss function for training deep neural networks on imbalanced datasets [39]. Since the problem of class imbalance is out of the scope of this work, we refer the curious reader to [28], [34], and [38].

4.2 NEW WORK

From previous work, we picked the best performing architecture, that is Xception, with the best performing hyper-parameter settings (learning rate, optimizer, momentum, and so on) and we used it as a baseline.

As pointed out on Chapter 2, context information is available only for 84% of the images, so we needed to work on a restricted dataset. Moreover a different training/validation/test split was proposed. In the new split, the wafers have been randomly divided into 7 folders. Then, the defects of the wafers belonging to the first five folders were used as training set, and the defects of the wafers belonging to the sixth and seventh folders were used as validation and test sets respectively. In this way defects of the same wafer cannot belong to different sets. This novel split is motivated by the fact that defects belonging to the same wafer may be correlated, as visually proved in Chapter 3.
Another reason for choosing this split is that some hyper-parameters tuning can be safely performed on the validation set without the risk of overfitting the test data. For example, in this use case, we selected the "best weights" from the epoch in which we obtained the highest validation accuracy, and then utilized such weights for the predictions on the test set.
However, this use case does not aim at optimizing the hyper-parameter configuration of the architecture, but rather at improving the defect classification accuracy by using some context information. Among the available context information, we decided to use only the XY coordinates of the defects on the wafer and on the die.
We utilized this location information mainly in two ways:

1. By combining in different ways the coordinates (both Cartesian and polar) of the defects with the features extracted by the network, as schematically shown in figures 13, 14, 15, 16. In these figures we proposed four architectures. In the first and fourth ones context features are directly concatenated with the features extracted by Xception. Particularly, in the fourth architecture a fully connected layer is added after the concatenation. In the second and third architectures context features go through two fully-connected layers before being concatenated with the image feature vectors. Specifically, in the second architecture a fully connected layer is added before the classification layer.

2. By using the XY coordinates of the defects to estimate some prior probability distributions. Indeed, as we will see in detail in section 4.4, the distribution of the K closest neighbors of a sample defect can be used to approximate the probability that the sample defect belong to a certain class given its Cartesian coordinates. Such probability estimates can be combined with the predictions of the network to produce a novel vector with new predictions.

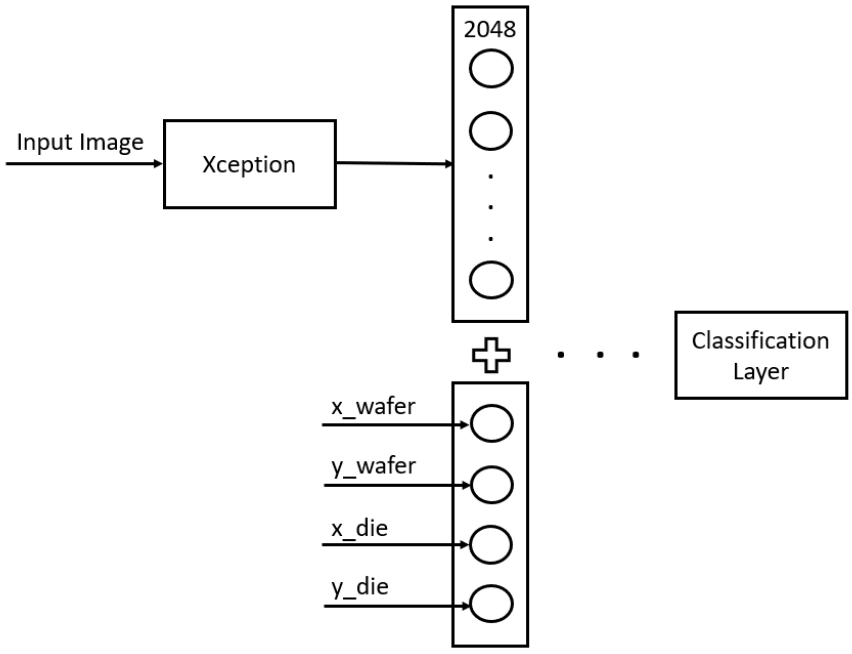


Figure 13: Model1; context features are directly concatenated with the features extracted by Xception.

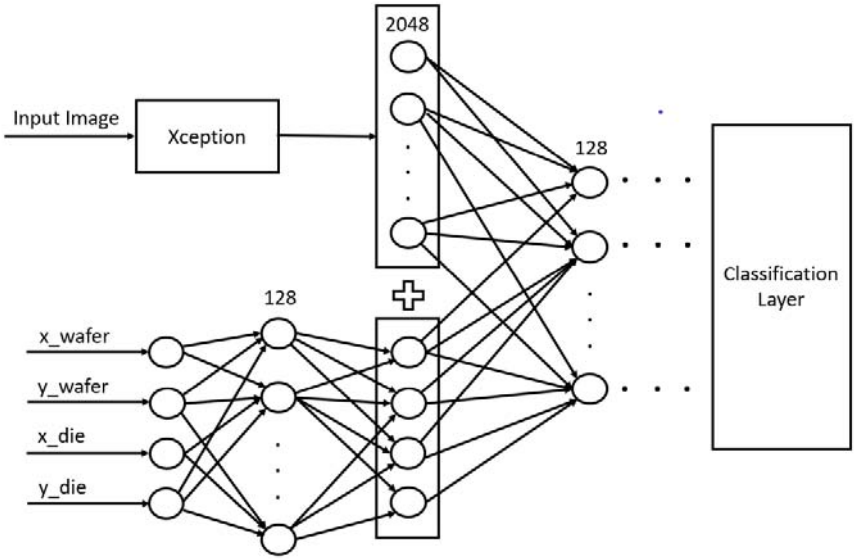


Figure 14: Model2; context features go through two fully-connected layers before being concatenated with the features extracted by Xception. A further fully-connected layer is added before the classification layer.

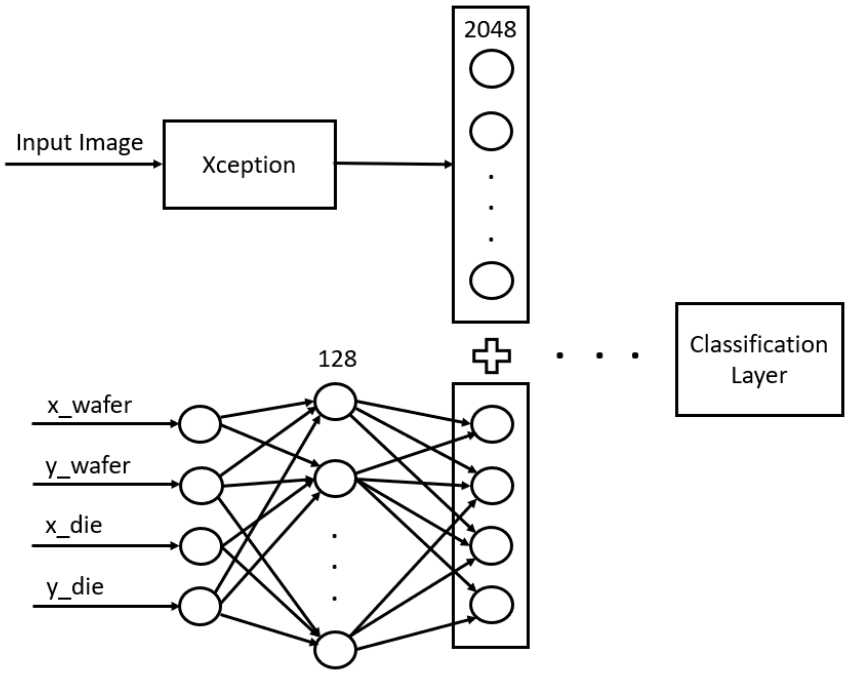


Figure 15: Model3; context features go through two fully-connected layers before being concatenated with the features extracted by Xception.

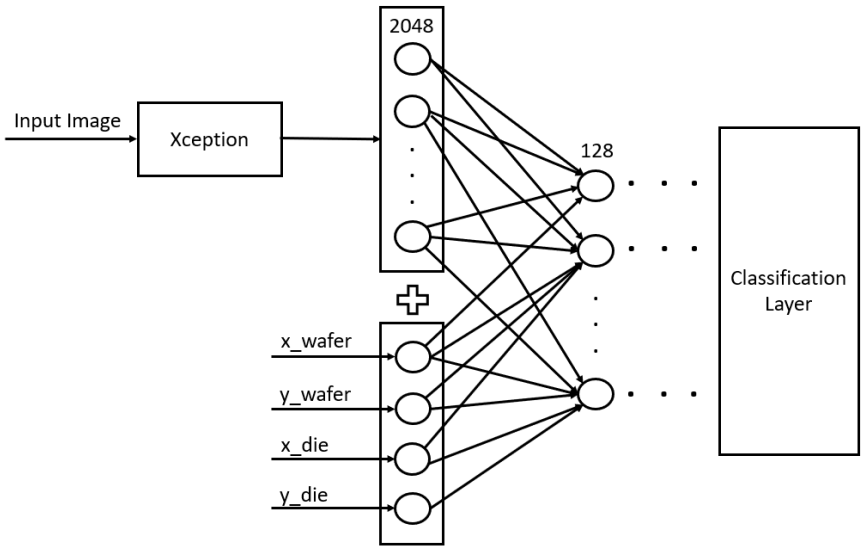


Figure 16: Model4; context features are directly concatenated with the features extracted by Xception. A further fully-connected layer is added before the classification layer.

4.3 TRAINING MULTI-STREAM NETWORKS IN KERAS

When training a deep learning model, loading a big dataset directly into a machine is infeasible. For this reason, data generators are used to generate real-time data and feed deep learning architectures with such data in fixed-length batches.
The class *DataGenerator* of Keras provides some built-in data generators; however, they can't be used for feeding user-defined multi-stream models. The next lines of code present the data generator that we needed to use to feed the architectures shown in figures 13, 14, 15, 16.

```
class DataGenerator(tf.keras.utils.Sequence):
    """Generates data for Keras."""
    def __init__(self, img_files, context_info, labels,
        batch_size=32, dim=(299,299), n_channels=3,
                n_classes=16, shuffle=True):

        self.img_files = img_files
        self.context_info = context_info
        self.labels = labels
        self.batch_size = batch_size
        self.dim = dim
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.shuffle = shuffle
        self.on_epoch_end()
```

```
def __len__(self):
    """Denotes the number of batches per epoch."""
    return int(np.floor(len(self.img_files) /
        self.batch_size))

def __getitem__(self, index):
    """Generate one batch of data."""
    # Generate indexes of the batch
    indexes =
        self.indexes[index*self.batch_size:(index+1)*self.batch_size]
    # Find list of IDs
    img_files_temp = [self.img_files[k] for k in indexes]
    # Generate data
    X, y = self.__data_generation(img_files_temp)
    return X, y

def on_epoch_end(self):
    """Updates indexes after each epoch."""
    self.indexes = np.arange(len(self.img_files))
    if self.shuffle == True:
        np.random.shuffle(self.indexes)

def __data_generation(self, img_files_temp):
    """Generates data containing batch_size samples."""
    X_img = []
    X_context = []
    y = np.empty((self.batch_size), dtype=int)
    # Generate data
    for i , img_file in enumerate(img_files_temp):
        # Read image
        img = Image.open(join('/images/kiel/',img_file))
        img = img.convert(mode='RGB') #convert to 3-channels
        if self.shuffle:
            img = img.rotate(random.uniform(-5,5))
        # Resize image
        im = np.array(img.resize(size=self.dim) ,
            dtype=np.float32)
        # Rescale image
        im = im/255.0
        X_img.append(im)
        X_context.append(self.context_info[img_file])
        y[i] = self.labels[img_file]
    X = [np.array(X_img), np.array(X_context)]

    return X, tf.keras.utils.to_categorical(y,
        num_classes=self.n_classes)
```

4.4 BAYESIAN PRIORS

Prior knowledge can help in improving the performance of a classification system. By taking inspiration from [40], where spatio-temporal prior was successfully used to improve the categorization of bird species in a large-scale fine-grained dataset, we developed a strategy to estimate prior probabilities about the defect classes, and to combine such priors with the predictions of the CNN. Bayesian priors are estimated from the XY coordinates of the defects in the wafer.

To take advantage of the positions of the defects we want to find $P[c|I, x, y]$, which is the probability that a defect belongs to class c given the image of the defect I and the coordinates x and y. For Bayes' rule we have that:

$$\mathsf{P}(c|I, x, y) = \frac{\mathsf{P}(I, x, y|c)\mathsf{P}(c)}{\mathsf{P}(I, x, y)} \tag{3}$$

If we assume that the image and the coordinates are conditionally independent given the defect class:

$$\mathsf{P}(c|I, x, y) = \frac{\mathsf{P}(I|c)\mathsf{P}(x, y|c)\mathsf{P}(c)}{\mathsf{P}(I, x, y)} \tag{4}$$

and by applying Bayes' rule again to $\mathsf{P}(I|c)$ and $\mathsf{P}(x, y|c)$ we have that:

$$\mathsf{P}(c|I, x, y) = \frac{\mathsf{P}(c|I)\mathsf{P}(I)}{\mathsf{P}(c)} \frac{\mathsf{P}(c|x, y)\mathsf{P}(x, y)}{\mathsf{P}(c)} \frac{\mathsf{P}(c)}{\mathsf{P}(I, x, y)} \tag{5}$$

If we drop all the terms which do not depend on c and which do not affect the classification:

$$\mathsf{P}(c|I, x, y) \sim \frac{\mathsf{P}(c|I)\mathsf{P}(c|x, y)}{\mathsf{P}(c)} \tag{6}$$

where $\mathsf{P}(c|I)$ can be estimated from the softmax layer of the network, $\mathsf{P}(c|x, y)$ can be estimated by considering the distribution of the K nearest neighbors of the defect with coordinates (x, y), and $\mathsf{P}(c)$ is a normalization factor.

4.4.1 *Example 1*

Let us consider the i-th defect in a ternary classification task with classes A, B, C and $K = 1000$.
From the coordinates (x_i, y_i) we extract the 1000-closest-defects to (x_i, y_i) and we discover that 500 of them belong to class A, 300 belong

to class B, and 200 to class C. Therefore, $\mathsf{P}(c = A|x_i, y_i) \simeq 0.5$, $\mathsf{P}(c = B|x_i, y_i) \simeq 0.3$, and $\mathsf{P}(c = C|x_i, y_i) \simeq 0.2$.
Let d_i denote the image of the i-th defect and suppose that the output of the softmax layer of the classifier is: $\mathsf{P}(c = A|d_i) \simeq 0.2$, $\mathsf{P}(c = B|d_i) \simeq 0.1$, $\mathsf{P}(c = C|d_i) \simeq 0.7$.
From (6) it follows that:

1. $\mathsf{P}(c = A|d_i, x_i, y_i) \simeq \frac{\mathsf{P}(c=A|d_i)\mathsf{P}(c=A|x_i,y_i)}{N}$
2. $\mathsf{P}(c = B|d_i, x_i, y_i) \simeq \frac{\mathsf{P}(c=B|d_i)\mathsf{P}(c=B|x_i,y_i)}{N}$
3. $\mathsf{P}(c = C|d_i, x_i, y_i) \simeq \frac{\mathsf{P}(c=C|d_i)\mathsf{P}(c=C|x_i,y_i)}{N}$

where $N = \mathsf{P}(c = A|d_i)\mathsf{P}(c = A|x_i, y_i) + \mathsf{P}(c = B|d_i)\mathsf{P}(c = B|x_i, y_i) + \mathsf{P}(c = C|d_i)\mathsf{P}(c = C|x_i, y_i)$.
The result for the proposed example is: $\mathsf{P}(c = A|d_i, x_i, y_i) \simeq 0.37$, $\mathsf{P}(c = B|d_i, x_i, y_i) \simeq 0.11$, $\mathsf{P}(c = C|d_i, x_i, y_i) \simeq 0.52$. Thus, the automatic classifier would keep on predicting class C as defect class but with less confidence.

4.4.2 *Bayesian Priors on unbalanced datasets*

On unbalanced datasets the number of the K-nearest-neighobors of a defect is biased towards the majority classes. Therefore, Bayesian priors should also take into account the cardinality of the classes.
Let n be the total number of defects, and n_X the number of defects belonging to a certain class X.
One way of weighting Bayesian priors is:

$$\mathsf{P}_w(c = X|x, y) = \frac{n - n_X}{n}\mathsf{P}(c = X|x, y) \tag{7}$$

making equation (6) become:

$$\mathsf{P}(c = X|I, x, y) \simeq \frac{\mathsf{P}_w(c = X|x, y)\mathsf{P}(c|I)}{\mathsf{P}(c)} \tag{8}$$

4.4.3 *Example 2*

Let us consider the previous example and suppose that there are $n = 12000$ training defects.
Assume that $n_A = 10000$ of them belong to class A, $n_B = 1000$ to class B, and $n_C = 1000$ to class C.
From (8) it follows that:

1. $\mathsf{P}(c = A|d_i, x_i, y_i) \simeq \frac{\mathsf{P}(c=A|d_i)\mathsf{P}_w(c=A|x_i,y_i)}{N} \simeq 0.1547$

2. $\mathrm{P}(c = B|d_i, x_i, y_i) \simeq \frac{\mathrm{P}(c=B|d_i)\mathrm{P}_w(c=B|x_i,y_i)}{N} \simeq 0.5071$
3. $\mathrm{P}(c = C|d_i, x_i, y_i) \simeq \frac{\mathrm{P}(c=C|d_i)\mathrm{P}_w(c=C|x_i,y_i)}{N} \simeq 0.3381$

where N is a normalization factor such that the probabilities sum up to one. In this case the classifier would change prediction from class C to class B.

4.4.4 *The value of K*

The value of nearest neighbors K can relevantly affect the predictions. The proper value of K should be:

1. Small enough to catch the local dimension of prior distributions;
2. Big enough to get reliable statistics.

The figures below show the selected neighborhoods for some test defects for different values of K. At the left of the figures we can see a zoom of the neighborhood of a test defect represented by a red cross, while at the right we can see how big is the neighborhood compared to the wafer. The represented coordinates have been normalized.

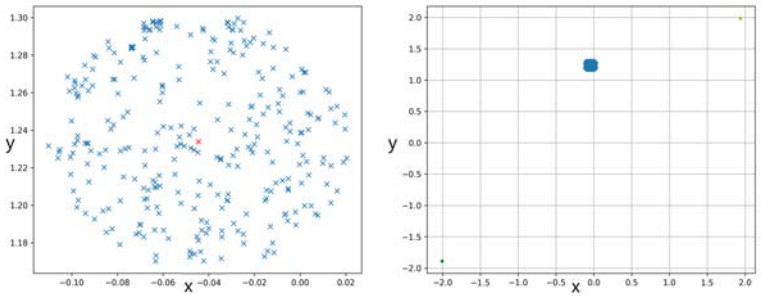


Figure 17: Neighborhood of a test defect for K=300.

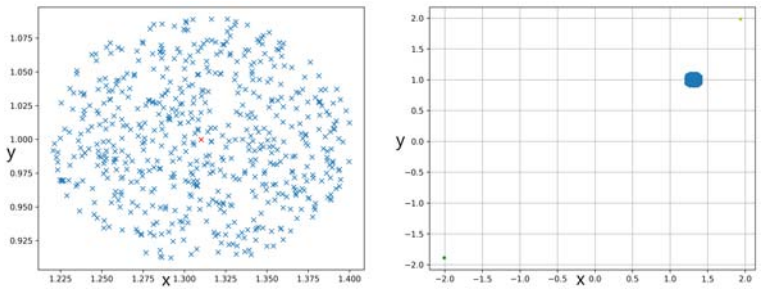


Figure 18: Neighborhood of a test defect for K=600.

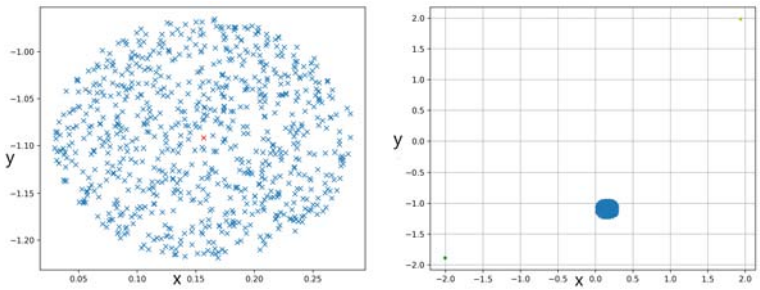


Figure 19: Neighborhood of a test defect for K=800.

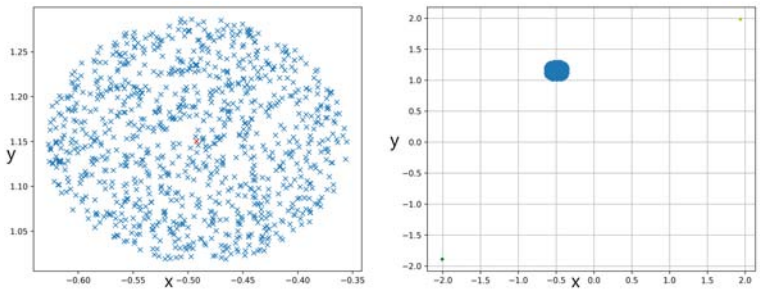


Figure 20: Neighborhood of a test defect for K=1000.

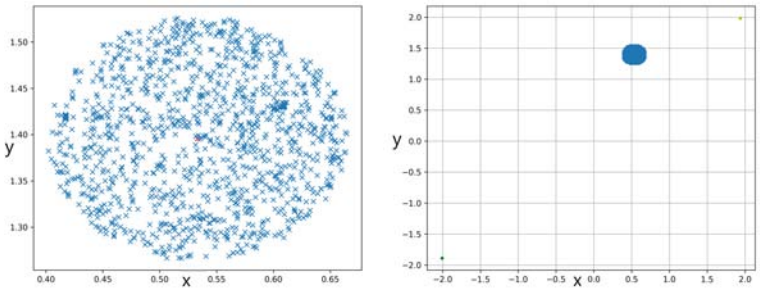


Figure 21: Neighborhood of a test defect for K=1250.

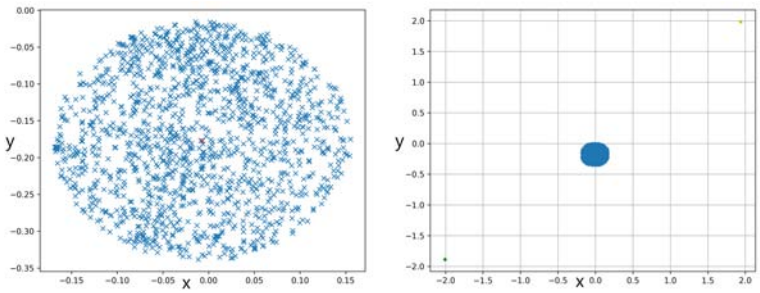


Figure 22: Neighborhood of a test defect for K=1500.

4.4.5 *Priors by lot*

The Bayesian method discussed so far is useful to derive some prior knowledge about general local properties like "a certain defect is more likely to belong to a class X if most of its closest neighbors belong to class X" and general global properties such that "some kind of defects are more likely to lie at the edge of the wafers". However, this method gives us no insights on particular events which may occur on specific wafers or lots.
To catch event-related priors we thought to estimate Bayesian priors at wafer level. More precisely, given a defect of a test wafer, we wished to compute Bayesian priors for that defect by solely considering all the other neighboring defects of that test wafer. However, this approach arises a problem: the number of defects per wafer is not enough to get statistically reliable Bayesian priors. One solution may be to estimate Bayesian priors at lot level; that is, given a defect of a test lot, we calculate Bayesian priors for that defect by considering all the neighboring defects of that specific test lot.
This approach requires a new train/validation/test split where defects of the same lot must belong to the same set. Therefore, similarly to what was done for the previous split, the lots have been randomly divided into 7 folders and then the defects of the lots belonging to the first five folders were used as training set, and the defects of the lots belonging to the sixth and seventh folders were used as validation and test sets respectively.
Figure 23 shows defect annotations for a test lot and the neighborhood of a random defect for $R = 10000$, $R = 20000$, $R = 30000$, $R = 40000$, where R is the radius of the circular neighborhood.

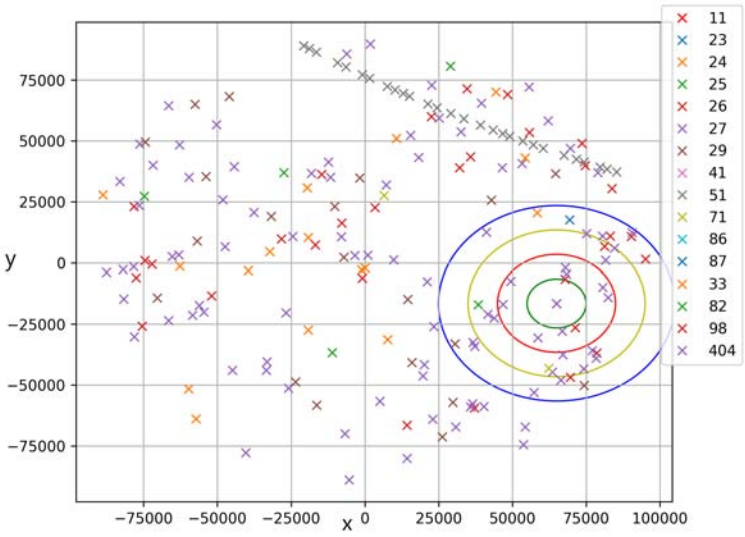


Figure 23: Defect annotations and examples of neighborhoods for a test lot.

Algorithmus 1 : Priors by lot with true labels

```
defects = emptyList();
oldPredictions = emptyList();
newPredictions = emptyList();
for defect in lot do
    prediction = predict(defect);
    oldPredictions.append(prediction);
    newPredictions.append(prediction);
    neighbors = 0;
    for otherDefect in defects do
        d = computeDistance(defect,otherDefect);
        if d < R then
            neighbors += 1;
        end
    end
    if neighbors > threshold then
        prior = computePriors(defect);
        newPredictions = updatePrediction(prediction,priors);
    end
    defects.append(defect);
end
```

Algorithm 1 describes the procedure to derive Bayesian priors for defects of a test lot. This algorithm is applicable only to a scenario in which the deep learning model is used as auxiliary tool for defect classification (and therefore the classification process is not completely automatised). In this scenario, the test defects are fed to the deep learning model and then to an expert human operator one by one. When the first test defect is fed to the automatic classification system, the model tries to predict it. Afterwards, a domain expert checks the prediction and assigns to the defect a "true label". Such label can be then exploited to derive priors for future test defects belonging to the same test lot in the following way. Suppose we are at test time and we have a new test lot with new defects to categorize. For the first defects that our model tries to predict it is not possible to derive Bayesian priors because there are not enough defects in the neighborhood. After some iterations, however, the lot starts "populating" of defects and we can therefore exploit priors for new unseen defects.
The algorithm is sensitive to two hyper-parameters: R, which determines the width of the neighborhood to be analysed, and *threshold*, which tunes the minimum number of defects that must be in a neighborhood to estimate prior probabilities.

Algorithmus 2 : Priors by lot with predictions

```
oldPred = emptyList();
for defect in lot do
    oldPred.append(predict(defect));
end
newPred = oldPred;
for defect in lot do
    if neighbors(defect) > threshold then
        priors = computePriors(defect);
        newPred = updatePrediction(oldPred,priors);
    end
end
```

As we saw, with algorithm 1 we can exploit priors only for a restricted number of defects. Instead of using true labels to derive priors, we can use the network's predictions themselves as explained in algorithm 2. Let us suppose that we are at test time and we want to classify defects of a new test lot. First, all the defects in the lot are predicted by the network. Then, for each defect in the lot, if the number of the defect's neighbors is higher than a certain *threshold*, Bayesian priors are computed and the prediction is updated.
Notice that prior probabilities computed with this method may be ineffective because they reflect the distributions of network's predictions and not the real distributions of neighboring defects. Therefore, algorithm 2 may help in improving the overall accuracy of the classification system only if the automatic classifier is already very powerful.
As explained above, both algorithms consider a circular neighborhood of a test defect to derive priors. However there are no guarantees that the choice of a circular neighborhood is the best possible. For this reason, algorithms 1 and 2 were revised by considering the K nearest neighbors instead of circular neighborhoods in order to catch arbitrarily shaped neighborhoods. Figure 24 shows an example of arbitrarily shaped neighborhood for $K = 30$ and $K = 50$.

It is worth mentioning that some lots present a low number of defects. For these lots is not possible to derive any kind of priors. Therefore, algorithms 1 and 2 can be only applied to lots which have enough defects. Now one question arises: when the number of defects per lot is enough? Answering to this question is not easy since the quantity of defects is not all that matter; indeed also the way in which the defects are distributed over the lot matters. However, in practice we ignored the issue of how defects are distributed and we applied algorithms 1 and 2 to lots which have more defects than a certain threshold value.

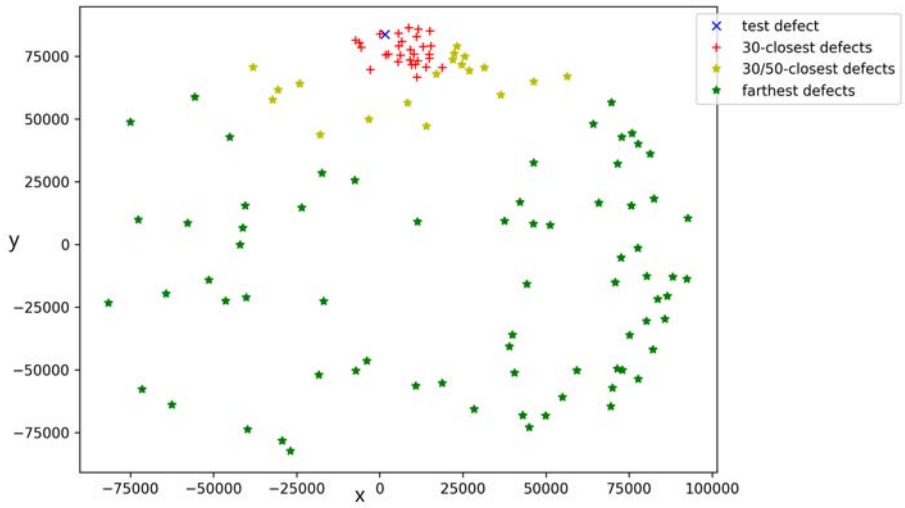


Figure 24: Example of arbitrarily shaped neighborhood for K=30 and K=50.

5

EXPERIMENTAL RESULTS AND EVALUATIONS

The models were mainly evaluated in terms of overall and per-class classification accuracy, precision, recall, and F1 score on actual production data provided by Infineon Technology. Moreover, analogously to what was done in previous work, we simulated how the overall classification improves when the model does not predict the defects on which it is less confident. This can be done by gradually increasing a softmax filter threshold and it's useful to estimate the remaining effort, effort reduction, and productivity gain defined in Chapter 4.

5.1 EXPERIMENTAL SETTINGS

HYPER-PARAMETERS	EXPLANATION	VALUE
lr	learning rate	0.0004
batch size	self-explanatory	32
optimizer	type of stochastic gradient descent algorithm	Adam
patience stop	number of epochs without improvements to wait before early stopping	10
patience lr	number of epochs without improvements to wait before learning rate reduction	6
lr factor	reduction factor by which the learning rate is multiplied when the learning rate reduction procedure is triggered	0.33

Table 2: Overview of training settings.

Xception architecture was trained for at most 50 epochs on Kiel dataset described in Chapter 2. The most relevant hyper-parameter settings are shown in table 2. The deep learning framework we worked on is Keras with Tensorflow backend.

Some Keras callbacks like `Reduce Learning Rate on Plateau` and `Early Stopping` were used during training. Callbacks are procedures which are automatically triggered when specific events occur. For example, in our case, when the validation accuracy has not been improving for 6 epochs the `Reduce Learning Rate on Plateau` callback is triggered and the learning rate is reduced by a factor of 0.33. If the validation accuracy has not been improving for 10 epochs, then the `Early Stopping` callback is triggered and the training ends. In all the experiments we run the training finished due to the `Early Stopping` callback. Figure 25 represents the model history by plotting the training and validation accuracy and loss as functions of the number of epochs. The two green spots highlight the epochs at which we get the highest validation accuracy and lowest validation loss respectively.

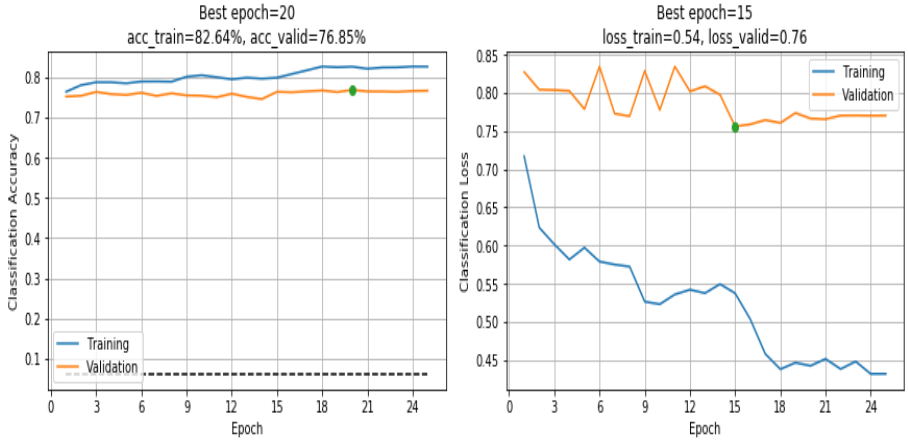


Figure 25: Model history.

5.2 MODELS' COMPARISON

Table 3 compares the Xception architecture, which we will denote as the baseline model, with the architectures described in section 4.2 in terms of overall accuracy and per-class F1 score. We can notice that, apart from model 2, which slightly surpasses the baseline model, combining context features in the way we did is not beneficial. This may be due to the fact that the information contained in context features' vector is negligible (either in quantity or in importance) with respect to the information embedded in the feature vector extracted by Xception. Moreover, in previous work it was estimated by means of a cross-validation strategy that the variance in overall accuracy due to the fact

of choosing a different train/validation/test split is about 0.03. Therefore, the effects of combining context features with the image feature vector are within the noise level of picking a certain split instead of another.

	Baseline	**Model 1**	**Model 2**	**Model 3**	**Model 4**
Accuracy	0.7928	0.7924	**0.7952**	0.7878	0.7859
F1 Score - Cl. 11	0.83	0.84	0.84	0.81	0.82
F1 Score - Cl. 23	0.75	0.74	0.75	0.74	0.74
F1 Score - Cl. 24	0.78	0.78	0.78	0.78	0.77
F1 Score - Cl. 25	0.83	0.83	0.83	0.82	0.81
F1 Score - Cl. 26	0.83	0.83	0.83	0.82	0.83
F1 Score - Cl. 27	0.85	0.85	0.85	0.84	0.84
F1 Score - Cl. 29	0.70	0.70	**0.72**	0.71	0.70
F1 Score - Cl. 41	0.82	0.81	0.82	0.81	0.81
F1 Score - Cl. 51	0.89	0.89	0.89	0.87	0.88
F1 Score - Cl. 71	0.95	0.95	0.95	0.94	0.94
F1 Score - Cl. 86	0.80	0.80	0.80	0.79	0.80
F1 Score - Cl. 87	0.87	0.86	0.87	0.86	0.86
F1 Score - Cl. 33	0.73	0.73	0.73	0.72	0.72
F1 Score - Cl. 82	0.55	0.53	**0.56**	0.55	0.54
F1 Score - Cl. 98	0.78	0.79	0.79	0.79	0.78
F1 Score - Cl. 404	0.52	0.52	0.52	0.49	0.52

Table 3: Models' comparison.

5.3 EFFECTS OF BAYESIAN PRIORS

This section analyses the effects of weighting the network's predictions with Bayesian priors as explained in section 4.4. Table 4 shows how the overall accuracy and per-class F1 score improve with respect to the baseline model when using Bayesian priors for different values of K. We can notice that the use of Bayesian priors is always beneficial for these values of K; in particular for $K = 1500$ the overall accuracy improves of the 0.9% with respect to the baseline (about 300 out of 39000 more defects are correctly classified).

	Baseline	K=800	K=1000	K=1250	K=1500
Accuracy	0.7928	0.8001	0.8008	0.8017	**0.8018**
F1 Score - Cl. 11	0.83	0.86	0.85	0.86	0.86
F1 Score - Cl. 23	0.75	0.75	0.75	0.75	0.75
F1 Score - Cl. 24	0.78	0.78	0.79	0.79	0.79
F1 Score - Cl. 25	0.83	0.83	0.83	0.83	0.83
F1 Score - Cl. 26	0.83	0.84	0.84	**0.85**	0.84
F1 Score - Cl. 27	0.85	0.85	0.85	0.85	0.85
F1 Score - Cl. 29	0.70	0.75	0.75	0.75	0.75
F1 Score - Cl. 41	0.82	0.82	0.82	0.82	0.82
F1 Score - Cl. 51	0.89	0.89	0.89	0.89	0.89
F1 Score - Cl. 71	0.95	0.95	0.95	0.95	0.95
F1 Score - Cl. 86	0.80	0.80	0.80	0.80	0.80
F1 Score - Cl. 87	**0.87**	0.86	0.86	0.86	0.86
F1 Score - Cl. 33	0.73	0.74	0.74	0.74	0.74
F1 Score - Cl. 82	0.55	0.60	0.60	**0.61**	0.60
F1 Score - Cl. 98	0.78	0.79	0.79	0.79	0.79
F1 Score - Cl. 404	0.52	0.55	0.55	0.55	**0.56**

Table 4: Effects of Bayesian priors for different values of K.

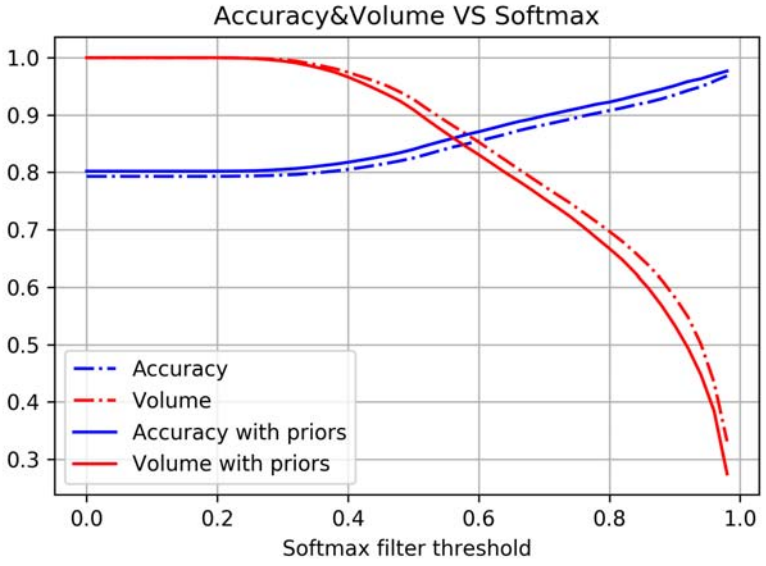


Figure 26: Softmax threshold simulation.

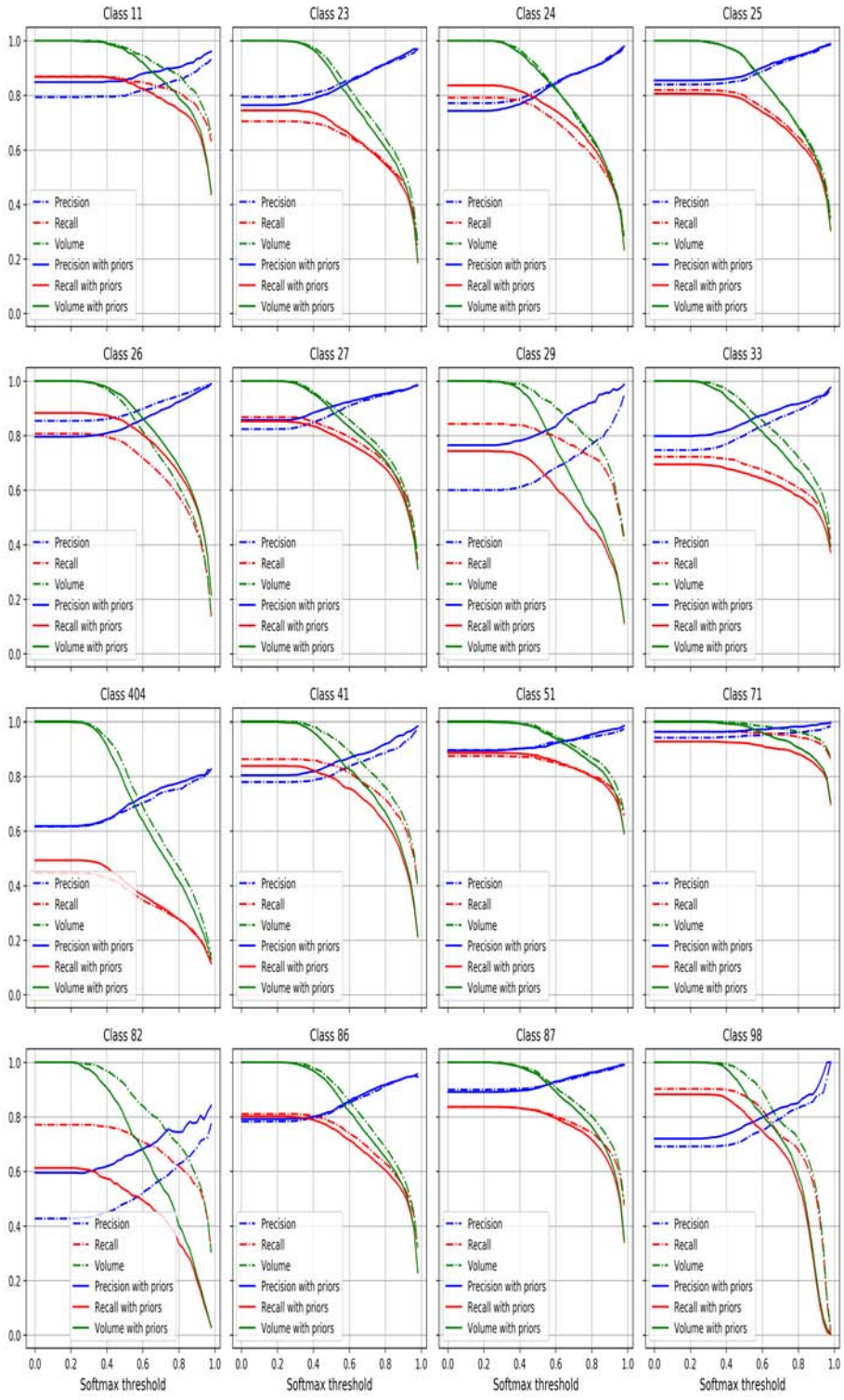


Figure 27: Per-class softmax threshold simulation.

Figures 26 and 27 compare the accuracy, the volume (fraction of defects which are actually classified), the per-class precision and recall as functions of the softmax filter threshold without and with priors ($K = 1500$). From these plots we can visualize that the use of priors is

actually beneficent. Particularly, as we can see from table 4, the classes which benefit most from Bayesian priors are classes '29', '82', and '404'. This does not surprise us since these classes are quite dissimilar from all the other classes (see figure 7). From the analysis of Chapter 3, we also expect that class '33' benefits from Bayesian priors because it has a very different defect density distribution with respect to all the other classes. Indeed, we noticed a small improvement in F1 score for such class, therefore our expectations have not been refuted.

Algorithms 1 and 2 were tested according to the modalities described in subsection 4.4.5. Algorithm 1 led to negligible improvements while algorithm 2 turned out to be ineffective for improving classification performances.

Bayesian priors can be used on their own to derive predictions; indeed they can work as an out-and-out KNN classifier. Such classifier achieves around 15% of accuracy on the test set, which is much worse than CNNs' performances but it outperforms the trivial models which output random predictions or predictions only for the majority class. Moreover, it was observed that this KNN classifier performs relatively well only on classes '26', '33', and '404'. It is noteworthy that for such classes the accuracy, precision and recall always improves when weighting the network's predictions with Bayesian priors.

6

DISCUSSIONS AND FUTURE WORK

As we saw in Chapter 5, some defect classes are hard to distinguish from each other. For instance, classes' pairs '23'-'24', '25'-'26, and '86'-'87' are very similar to each other and therefore the network struggles a bit in discriminating such pairs. Figure 28 visually shows the confusion between the aforementioned pairs; indeed, in the main diagonal of the confusion matrix (at least) three blocks are easily detectable.

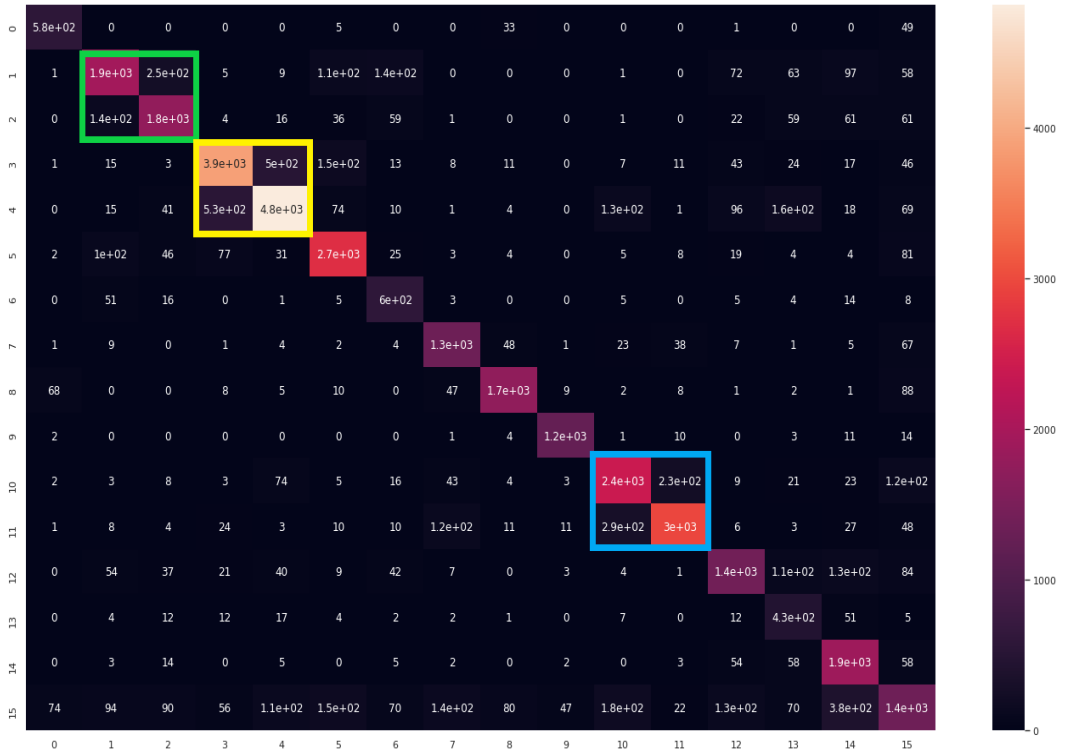


Figure 28: Block structure of the confusion matrix.

Consider now the similarity matrix represented in figure 7; we can notice that the aforementioned pairs not only are hard to distinguish at image level, but also they are very similar in the way they are distributed over the wafer grid. This means that, although some information can be extracted from defects' positions, we are in the unlucky case where similar defect classes are similarly distributed over the wafer.

The block structure of the confusion matrix highlights the need for reducing that confusion. One solution to this problem might be hierarchical classification. Suppose we detect M blocks in the main

diagonal of a confusion matrix C. In the first part of the training every class of a block is given the same label. Then, M specialised models are retrained (where the word retrain refers to the fact that the training data has been already seen and not that the models have been previously trained) on each block to distinguish among intra-block classes. Note that the retraining must not destroy what has been learnt in the previous step. One big issue with hierarchical classification is that M new models need to be trained, which implies higher and higher computational overhead as M grows. A strategy which does not imply the training of new specialised models is cost-sensitive learning and will be discussed in the next section.

In semiconductor manufacturing, labelling microscopic defects is a tedious and time-consuming task. However, many unannotated defects' images are often available and they are not exploited. We will hint a semi-supervised learning framework to deal with this likely situation.

6.1 COST-SENSITIVE LEARNING

Most classification models are cost-blind, that is they treat all the misclassification errors equally. However, in several real-world applications the costs of different misclassification errors are not the same. Cost-sensitive learning aims at making the optimal classification decisions when different misclassification errors incur different penalties [9]. Formally, let C be a cost matrix whose (i, j) entry reflects the cost of classifying a test sample x as i when its true label is j. The optimal prediction for x is the class i^*, where:

$$i^* = \arg\min_i \sum_j \mathrm{P}(j|x)C(i,j) \tag{9}$$

We conjecture that cost-sensitive learning may help in solving the confusion discussed at the beginning of the chapter.
A possible cost-sensitive learning strategy could be the following. At the beginning of the training do not give too much importance to classification errors towards very similar classes. Then, as the model gets more and more specialized, give more importance to errors between similar classes. In this way, in the first phase of the training the network will learn to distinguish among the macro-classes, whereas in the second phase the network will specialise in discriminating very similar sub-classes.
The extreme version of cost-sensitive learning, where given a class i no punishment is given to the model if it confuses i with a very similar class, is often known as one-vs-most classification [40].

The following subsections will describe some cost learning approaches. As we will see, such approaches can be used in combination with the two-step cost-sensitive learning strategy proposed above.

6.1.1 *Rescale approach*

A typical strategy for cost-sensitive learning is rescaling the classes such that their influences during the training are proportional to their costs. This can be done by assigning different weights to training samples of distinct classes, where the weights are in proportion to the misclassification costs.
While classical Rescale approaches are optimal for cost-sensitive binary classification, they are not for multi-class problems. To overcome this limitation, Zhou and Liu [44] proposed the $RESCALE_{\text{new}}$ approach. Let ϵ_{ij} be the cost of misclassifying a sample of class i to class j and let C be cost matrix which can be constructed from such costs. Let c denote the number of classes. Assume that the cost of making correct classifications is always zero, therefore $\epsilon_{ii} = 0$, $i = 1, 2, .., c$. Moreover, suppose that, at least for now, there is no class imbalance. The optimal rescaling ratio of class i against class j can be defined as:

$$\tau_{\text{opt}}(i, j) = \frac{\epsilon_{ij}}{\epsilon_{ji}} \tag{10}$$

Suppose that each class can be assigned with a weight w_i, $i = 1, 2, .., c$. After rescaling, the weights should satisfy the relation:

$$\frac{w_i}{w_j} = \tau_{\text{opt}}(i, j), i, j = 1, 2, .., c \tag{11}$$

which can be expanded into $\binom{c}{2}$ constraints:

$$\begin{array}{cccc} \frac{w_1}{w_2} = \frac{\epsilon_{12}}{\epsilon_{21}}, \ldots, & \frac{w_1}{w_c} = \frac{\epsilon_{1c}}{\epsilon_{c1}} \\ \frac{w_2}{w_3} = \frac{\epsilon_{23}}{\epsilon_{32}}, \ldots, & \frac{w_2}{w_c} = \frac{\epsilon_{2c}}{\epsilon_{c2}} \\ \ldots \ldots \ldots & \\ \frac{w_{c-1,c}}{w_{c,c-1}} = \frac{\epsilon_{c-1,c}}{\epsilon_{c,c-1}} & \end{array}$$

which can be written as an homogeneous system of $\frac{c(c-1)}{2}$ equations in c unknowns:

$$\begin{cases} w_1 \cdot \epsilon_{21} - w_2 \cdot \epsilon_{12} + ... + w_c \cdot 0 = 0 \\ ..... = 0 \\ w_1 \cdot \epsilon_{c1} + w_2 \cdot 0 + ... - w_c \cdot \epsilon_{1c} = 0 \\ w_2 \cdot \epsilon_{32} + ... - w_3 \cdot \epsilon_{23} + w_c \cdot 0 = 0 \\ ..... = 0 \\ w_1 \cdot 0 + w_2 \cdot \epsilon_{32} + ... - w_c \cdot \epsilon_{2c} = 0 \\ ..... = 0 \\ ... + w_{c-1,c} \cdot \epsilon_{c,c-1} - w_c \cdot \epsilon_{c-1,c} = 0 \end{cases}$$

If the rank of the system's coefficient matrix is smaller than c, the system has a non-trivial solution $\mathbf{w}$, and therefore all classes can be rescaled simultaneously. Thus, in this case, the multi-class cost-sensitive learning problem can be solved directly. Instead, if the system's coefficient matrix is full rank, the system has only the trivial solution and we need to decompose the multi-class problem into many binary-class cost-sensitive problems.
Suppose now that the dataset is not balanced. As we mentioned in chapter 4, a strategy to tackle class imbalance can be to adjust a weight vector $\mathbf{b}$ such that more importance is given to under-represented classes during training and less importance is given to classes with higher cardinality. If we wish to solve the problems of class imbalance and cost-sensitive learning simultaneously, we can just multiply vectors $\mathbf{b}$ and $\mathbf{w}$.

6.1.2 *Cost-Sensitive Deep Metric Learning*

Zhao and Peng proposed Cost-sensitive Deep Metric Learning (CDML), an approach which integrates confusion analysis, confusion deep metric learning and weighted sofmax for learning the differences among hard-to-distinguish sub-classes [20]. In the confusion analysis phase the confusion degrees among different sub-classes is estimated from the confusion matrix. In the confusion deep metric learning phase a triplet loss is defined and used to focus on learning the difference among sub-classes with small variance. In this phase, a triplet distribution matrix M is iteratively constructed as well. In the last phase, a weighted softmax loss function is defined in the hope of learning more discriminative features for hard-to-distinguish sub-classes. Such

loss puts more cost on sub-classes with higher misclassification rates as follows:

$$Softmax_w(I, L) = \frac{1}{n}\sum_{i=1}^{n} -W_i * log(s_i^{(\mathrm{L_i})}) \tag{12}$$

where I is the image set, L stands for the sub-classes set, n is the total numbers of samples in L, and:

$$s_i^{(\mathrm{L_i})} = \frac{(1/k) * e(I_i, L_i)}{\sum_{j=1}^{k} M_{\mathrm{ij}} * e(I_i, L_j)} \tag{13}$$

Actually, the term "softmax loss" is often misused, because we may use the softmax activation in combination with any other loss function. However, in ML community the softmax activation is often followed by the caterogical cross-entropy loss. For this reason, the terms "softmax loss" and "caterogical cross-entropy loss" are often used interchangeably.
The following lines of code show a vectorized implementation of the weighted categorical cross-entropy loss in Tensorflow.

```
class WeightedCategoricalCrossentropy(CategoricalCrossentropy):

    def __init__(self, cost_mat,
        name='weighted_categorical_crossentropy', **kwargs):
        assert(cost_mat.ndim == 2)
        assert(cost_mat.shape[0] == cost_mat.shape[1])

        super().__init__(name=name, **kwargs)
        self.cost_mat = K.cast_to_floatx(cost_mat)

    def __call__(self, y_true, y_pred):

        return super().__call__(
            y_true=y_true,
            y_pred=y_pred,
            sample_weight=get_sample_weights(y_true, y_pred,
                self.cost_mat),
        )

def get_sample_weights(y_true, y_pred, cost_m):
    num_classes = len(cost_m)

    y_pred.shape.assert_has_rank(2)
    y_pred.shape[1].assert_is_compatible_with(num_classes)
    y_pred.shape.assert_is_compatible_with(y_true.shape)

    y_pred = K.one_hot(K.argmax(y_pred), num_classes)

    y_true_nk1 = K.expand_dims(y_true, 2)
    y_pred_n1k = K.expand_dims(y_pred, 1)
```

```
    cost_m_1kk = K.expand_dims(cost_m, 0)

    sample_weights_nkk = cost_m_1kk * y_true_nk1 * y_pred_n1k
    sample_weights_n = K.sum(sample_weights_nkk, axis=[1, 2])

    return sample_weights_n
```

The proper usage of this loss function is:

```
m.compile(loss=WeightedCategoricalCrossEntropy(cost_matrix),...)
```

which send us back to the problem of choosing an effective cost matrix. Notice that if we choose the cost matrix such that it also considers the class imbalance, we can address the problems of cost-sensitive learning and class imbalance simultaneously.

6.2 SEMI-SUPERVISED LEARNING

Let us consider a multi-class classification task. Assume we have a training set D with N labelled images and a set U with M unlabelled images, where usually $M \gg N$. Semi-supervised learning deals with exploiting both sets in the learning process. In this section we will describe the interesting semi-supervised image classification framework proposed by [14].
In their approach, an hopefully powerful teacher model is first trained on D to label the samples in U. Then, for each target label the top-K examples are selected and are used to construct a new training dataset $\hat{D}$. It is worth noticing that only the top-K examples are selected in order to limit the labelling noise. Afterwards, a new student model is trained on $\hat{D}$. Finally, the student model is fine-tuned on set D.
This approach suits well to our industrial case study since there are many unlabelled images which can be potentially exploited to improve the current automatic classification system.

6.3 TRANSFER LEARNING

The Xception architecture we used was pre-trained on the ImageNet dataset. Therefore, we utilized the weights of ImageNet challenge as initialization and then we trained the network on Kiel dataset. However, the morphology of SEM images is completely different from the morphology of the images of ImageNet dataset, thus the weights initialization we used is very likely to be nonoptimal. Moreover, there are no significant publicly available SEM images datasets to exploit for transfer learning. For this reasons, we believe that future work need

to be done to build a proper dataset on which pre-training the network.

Recall that Kiel dataset is composed of SEM images taken at the second, third, and fourth metal layers. Thus, the information of defect images taken at previous layers is not exploited. Future work should also address the issue of developing a transfer learning strategy to convey knowledge from layer to layer.

7 CONCLUSIONS

In this work we presented our methodological and experimental contributions to deep learning-based automatic classification of microscopic defects in silicon wafers with context information. Although the classification of defect patterns in wafers has been hugely studied in literature, the automatic categorization of microscopic defects has not been adequately addressed yet.
Thanks to the availability of huge amount of data and high-performance computing systems, deep learning models, and especially CNNs, have achieved great results in almost any image recognition task. Several architectures have been proposed to improve the performance and the efficiency of CNNs in some challenges like the ImageNet large scale visual recognition challenge [15]. Some of these well-known architectures, which are described in Appendix A, can be effectively used for defect classification as well.
Canonical deep learning-based microscopic defect classification approaches have the limitation of utilizing only the information contained in the images. For example the information "some defect types can only appear in the memory section of a chip" cannot be discovered by classical deep learning models. This work overcame this limitation by using some context information about the defects, like the position of the defects in the wafer and in the die, to improve the current deep learning-based automatic classification system.
One way of combining context information with the information extracted from the image is to merge context features with the features extracted by the CNN. There can be several ways of merging these two kind of features; some examples of merging procedures were schematically shown in Chapter 4.
Another way to extract information from context attributes is to use the coordinates of the defects to estimate local prior probability distributions. Indeed, the distribution of the closest neighbors of a sample defect can be used to approximate the probability that the sample defect belong to a certain class given its Cartesian coordinates. Such probabilistic framework was formalised in Chapter 4.
These strategies to extract information from context were tested on actual production data provided by Infineon Technologies, and the results were presented in Chapter 5. The peculiarity of some results was discussed in Chapter 6 and some hints for future works were proposed.

Part III

Appendix

A

APPENDIX

CNNs have recently achieved great successes on large-scale image recognition tasks. This has been possible thanks to the availability of large-scale image datasets and high-performance computing systems like GPUs and distributed clusters. This section overviews some of the most famous state-of-the-art architectures based on Convolutional Networks.

A.1 VERY DEEP CONVOLUTIONAL NETWORKS

Historically, CNN architectures were only few layers deep and large kernel sizes, and therefore wide receptive fields, were used in the first convolutional layers.
Simonyan and Zisserman investigated on how the depth of a network impacts on large-scale image recognition tasks [25]. By using small (3x3) convolution filters and by pushing the architecture's depth to 16-19 weight layers, they achieved state-of-the-art results in the ImageNet Challenge 2014.
The input to their network is a fixed-size 224x224 image. After some simple preprocessing, the image is passed through a stack of convolution layers with filter of size 3x3 and stride 1. The padding is done such that the spatial resolution is preserved after convolution (so the padding is 1 for 3x3 convolution layers). Five max-pooling layers perform spatial pooling after some convolution layers. Max-pooling is done over a 2x2 window with stride 2. A stack of convolution layers is then followed by three fully-connected layers. All hidden layers utilize the ReLu activation function.
The architecture described above is widely known as VGG. Figure 29 overviews different VGG configurations.

Figure 30 shows that two stacked 3x3 convolution layers have the same receptive field as a 5x5 convolution layer. Analogously, it can be shown that a stack of three 3x3 convolution layers can replace a 7x7 convolution layer. Replacing a 7x7 convolution layer with three 3x3 convolution layers has mainly two advantages. Firstly, three ReLu non-linearity are used instead of a single one, which makes the model more discriminative. Secondly, the number of parameters is widely reduced. For example, if both the input and output of a stack of three 3x3 convolution layers have C channels, the stack is parametrised by $3(3^2C^2) = 27C^2$ parameters, while a single 7x7 convolution layer

requires $7^2C^2 = 49C^2$ parameters.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 29: VGG configurations (taken from [25]).

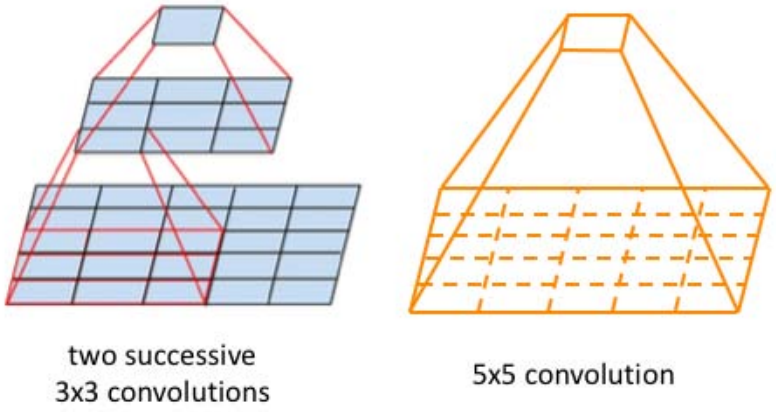


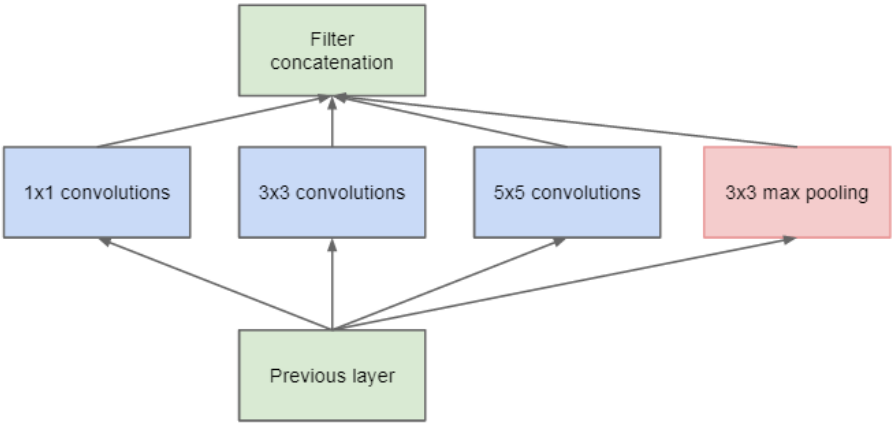
Figure 30: 3x3 convolutions VS 5x5 convolution

A.2 INCEPTION

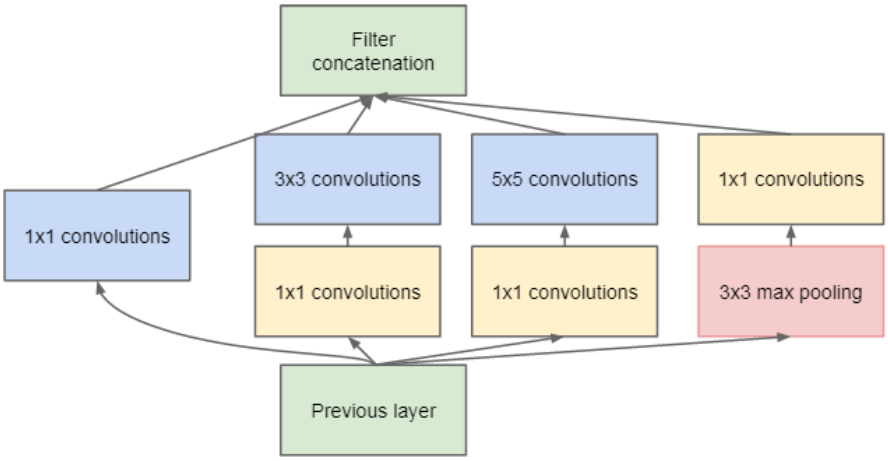
The deployment of VGG can be computationally expensive even in GPUs. Indeed VGG is a densely-connected architecture, where, in a convolution operation, each input channel is connected to each output channel.

Szegedy et al. built Inception based on the idea that the optimal network topology can be constructed layer by layer by analizing the correlations of the activations of the preceding layer and clustering neurons with highly correlated outputs [7]. Thus, they claim that the optimal network topology can be represented by a sparse architecture. However, the available computing infrastructures are inefficient with calculations on sparse data. In order to put in practice this idea, GoogLeNet devised a module, called Inception module, which is able to both approximate a sparse CNN and use the computing tools which are optimised for densely-connected architectures [7].

In its naive version, the Inception module concatenates 1x1, 3x3, and 5x5 convolutions in order to extract abstract features at different scales simultaneously. Moreover, a 3x3 max-pooling operation is added at each module. However, even a small number of 5x5 convolutions can be computationally heavy. Therefore, to reduce the computational burden, they introduced a 1x1 convolution layer, which is often known as bottleneck layer, before applying larger sized kernels.
Figure 31 shows the Inception module.



(a) Inception module, naive version



(b) Inception module with dimension reductions

Figure 31: Inception module (taken from [7])

As we saw in the previous section, convolutions with filters larger than 3x3 can be reduced into a series of 3x3 convolutions. However it turns out that a 3x3 convolution can be replaced by a 3x1 convolution followed by a 1x3 convolution [6]. Assuming that the number of input and output channel is C, the latter configuration is parametrised by $3C^2 + 3C^2 = 6C^2$ parameters instead of the $3^2C^2 = 9^2C^2$ parameters of the former configuration. Theoretically, this reasoning can be generalised to any nxn convolution.
Figure 32 depicts the Inception module after the factorization of the nxn convolution.

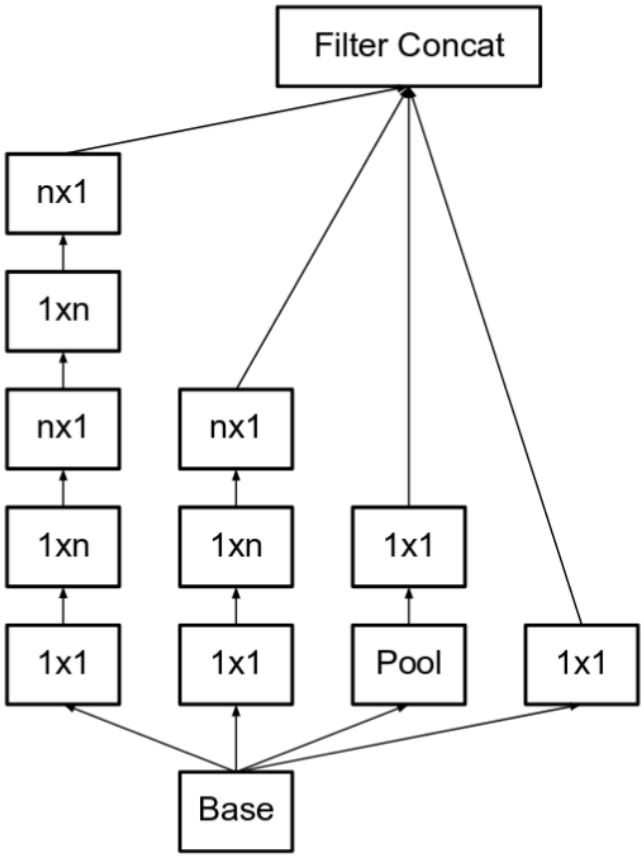


Figure 32: Inception module after the factorization of the nxn convolutions

GoogLeNet also replaced the fully-connected layers at the end with a simple global average pooling which averages over the values of the bidimensional feature maps, after the last convolutional layer. This drastically reduces the total number of parameters.

A.3 RESIDUAL NETWORKS

Consider an architecture and its deeper counterpart obtained by adding identity layers onto it. One may expect that a deeper model should not perform worse of its shallower counterpart. But in practice that does not happen (shallower architectures produce smaller or equal training errors). This issue, which is not due to overfitting, is known as the degradation problem and it was addressed by He et al. in [22] and [23] through deep residual learning.

Let x be the input to few stacked layers and let $H(x)$ be an underlying mapping to be learnt. If multiple non-linear layers can asymptotically approximate any function $H(x)$, then they can asymptotically approximate the residual function $F(x) = H(x) - x$. The only differ-

ence in approximating $H(x)$ and $F(x)$ may be the ease of learning. Figure 33 shows a building block of the residual leaning framework.

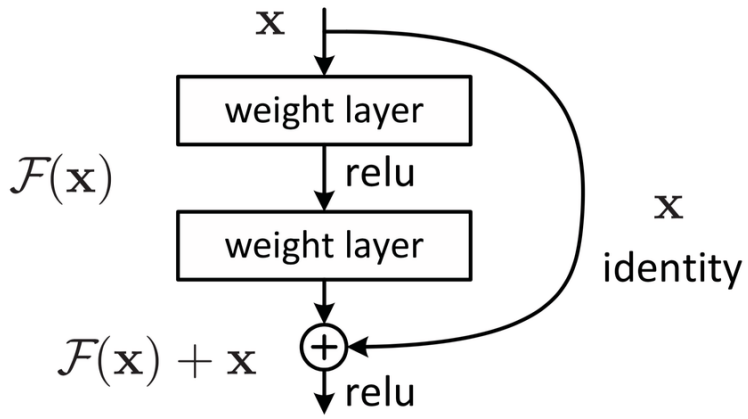


Figure 33: Residual learning building block (taken from [22]).

The degradation problem suggests that multiple non-linear layers struggle in approximating identity mappings. With residual learning, the block of layers can learn the identity mappings by simply pushing their weights to zero. In practice, identity mappings hardly ever are the optimal functions we want to approximate, but it can be shown that they are often close to such optimal functions. Therefore, it turns out that learning residual functions is easier than learning the underlying mappings.
Residual learning can be realised by feed-forward neural networks with shortcut connections. In [22], shortcut connections simply perform identity mappings. It is worth noticing that identity shortcut connections add no extra complexity.

A.4 XCEPTION

As we saw, Inception is based on the hypothesis that cross-channel correlations and spatial correlations are partially decoupled. Indeed, the Inception module first looks at cross-channel correlations through 1x1 convolutions, mapping the input data into 3 or 4 spaces of smaller dimension, and then looks at spatial correlations through regular 3x3 and 5x5 convolutions. In [8], an extreme version of the Inception module is proposed under the assumption that cross-channel correlations and spatial correlations are completely decoupled. This extreme version of the Inception module first uses 1x1 convolutions to map cross-channel correlations, and then separately maps the spatial correlations of each output channel. The proposed module is very similar to depthwise separable convolutions (independent spatial convolutions

over each input channel followed by pointwise convolutions) [8].

The Xception architecture has 36 convolutional layers structured into 14 modules. Each module, except for the first and last ones, have linear residual connection among them.
Figure 34 describes the specifications of the Xception architecture.

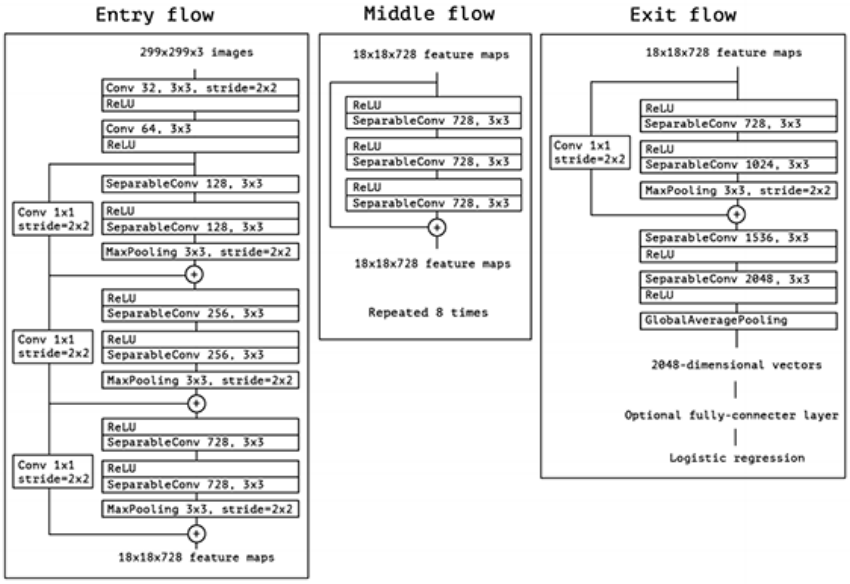


Figure 34: The Xception architecture (taken from [8]).

BIBLIOGRAPHY

[1] M. Retersdorf C. Wooten X. Song A. Hesse A. Drozda-Freeman M. McIntyre. "Recognition of Systematic Spatial Pattern in Silicon Wafers Based on SOM and K-means." In: *IEEE/SEMI Advanced Semiconductor Manufacturing Conference* (2007).

[2] S. C. Hsu C. F. Chen and Y. J. Chen. "A system for online detection and classification of wafer bin map defect patterns for manufacturing intelligence." In: *International Journal of Production Research* 51.8 (2013).

[3] W. C. Wang C. F. Chien and J.-C. Cheng. "Data mining for yield enhancement in semiconductor manufacturing and an empirical study." In: *Expert Systems with Applications* 33.1 (2007).

[4] S.-J. Wang C.-H. Wang and W.-D. Lee. "Automatic identification of spatial defect patterns for semiconductor manufacturing." In: *International Journal of Production Research* 44.23 (2006).

[5] C. Chien C. Liu. "An intelligent system for wafer bin map defect diagnosis: An empirical study for semiconductor manufacturing." In: *Engineering Applications of Artificial Intelligence* (2013).

[6] S. Ioffe J. Shlens Z. Wojna C. Szegedy V. Vanhoucke. "Rethinking the Inception Architecture for Computer Vision." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[7] Y. Jia P. Sermanet S. Reed D. Anguelov D. Erhan V. Vanhoucke A. Rabinovich C. Szegedy W. Liu. "Going deeper with convolutions." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015).

[8] F. Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).

[9] Charles Elkan. "The Foundations of Cost-Sensitive Learning." In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence* 2 (2001), 973–978.

[10] S. F. Liu F. L. Chen. "A neural-network approach to recognize defect spatial pattern in semiconductor fabrication." In: *IEEE Transactions on Semiconductor Manufacturing* 13.3 (2000).

[11] C. Ha G. Choi S.-H. Kim and S. J. Bae. "Multi-step ART1 algorithm for recognition of defect patterns on semiconductor wafers." In: *International Journal of Production Research* 50.12 (2012).

[12] P. D. Yoo Y. Al-Hammadi S. Muhaidat U. Lee G. Tello O. Y. Al-Jarrah. "Deep-Structured Machine Learning Model for the Recognition of Mixed-Defect Pattern in Semiconductor Fabrication Process." In: *IEEE Transactions on Semiconductor Manufacturing* 31.2 (2018).

[13] C. Y. Hsu. "Clustering ensemble for identifying defective wafer bin map in semiconductor manufacturing." In: *Mathematical Problems in Engineering* (2015).

[14] K. Chen M. Paluri D. Mahajan; Facebook AI I.Z. Yalniz H. Jégou. "Billion-scale semi-supervised learning for image classification." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

[15] *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*. URL: `http://www.image-net.org/challenges/LSVRC/`.

[16] *ImageNet*. URL: `http://www.image-net.org/`.

[17] M. D. Ohman J. S. Ellen C. A. Graff. "Improving plankton image classification using context metadata." In: *Limnology and Oceanography (ASLO)* (2019).

[18] J. Luo J. Yu. "Leveraging Probabilistic Season and Location Context Models for Scene Understanding." In: *International Conference on Image and Video Retrieval (CIVR)* (2008).

[19] X. Lu J. Yu. "Wafer Map Ddefect Detection and Recognition Using Joint Local and Nonlocal Linear Discriminant Analysis." In: *IEEE Transactions on Semiconductor Manufacturing* 29.1 (2016).

[20] Y. Peng J. Zhao. "Cost-Sensitive Deep Metric Learning for Fine-Grained Image Classification." In: *24th International Conference on Multimedia Modeling* (2018).

[21] P. Luley A. Almer L. Paletta K. Amlacher G. Fritz. "Geo-Contextual Priors for Attentive Urban Object Recognition." In: *International Conference on Robotics and Automation (ICRA)* (2009).

[22] S. Ren J. Sun; Microsoft Research K. He X. Zhang. "Deep Residual Learning for Image Recognition." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[23] S. Ren J. Sun; Microsoft Research K. He X. Zhang. "Identity Mappings in Deep Residual Networks." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

[24] H. Kim K. Kyeong. "Classification of mixed-type defect patterns in wafer bin maps using convolutional neural networks." In: *IEEE Transactions on Semiconductor Manufacturing* 31 (2018).

[25] Department of Engineering Science University of Oxford K. Simonyan A. Zisserman; Visual Geometry Group. "Very Deep Convolutional Networks for Large-scale Image Recognition." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[26] L. Fei-Fei R. Fergus L. Bourdev Computer Science Department of Standford University Facebook AI Research K. Tang M. Paluri. "Improving Image Classification with Location Context." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[27] D. Wang K. Weiss T. M. Khoshgoftaar. "A survey of transfer learning." In: *Journal of Big Data* (2016).

[28] B. Krawczyk. "Learning from imbalanced data: open challenges and future directions." In: *Progress in Artificial Intelligence* (2016).

[29] B. van der Waal M. Fan Q. Wang. "Wafer Defect Patterns Recognition Based on OPTICS and Multi-Label Classification." In: *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* (2016).

[30] Y. Hiltunen M. Liukkonen. "Recognition of Systematic Spatial Pattern in Silicon Wafers Based on SOM and K-means." In: *International Federation of Automatic Control (IFAC)* 51 (2018), pp. 439–444.

[31] J. Y. Lee J. Byun M. Piao C. H. Jin. "Decision Tree Ensamle-Based Wafer Map Failure Pattern Recognition Based on Radon Transform-Based Features." In: *IEEE Transactions on Semiconductor Manufacturing* 31.2 (2016).

[32] J. Y. Lee M. Saqlain B. Jargalsaikhan. "A Voting Ensemle Classifier for Wafer Map Defect Pattern Identification in Semiconductor Manufacturinf." In: *IEEE Transactions on Semiconductor Manufacturing* (2019).

[33] J. L. Chen Ming-Ju Wu Jyh-Shing R. Jang. "Wafer Map Failure Pattern Recognition and Similarity Ranking for Large-Scale Data Sets." In: *IEEE Transactions on Semiconductor Manufacturing* (2015).

[34] A. More. "Survey of resampling techniques for improving classification performance in unbalanced datasets." In: *Applications (stat.AP) arXiv:1608.06048* (2016).

[35] A. Torroella M. Shah S. Ardeshir A. R. Zamir. "Gis-assisted object detection and geospatial localization." In: *European Conference on Computer Vision (ECCV)* (2014).

[36] C. O. Kim S. H. Lee S. Cheon H. Lee. "Convolutional Nerual Network for Wafer Surface Defect Classification and the Detection of Unknown Defect Class." In: *IEEE Transactions on Semiconductor Manufacturing* (2018).

[37] J. H. Hays A. A. Efros M. Hebert S. K. Divvala D. Hoiem. "An Empirical Study of Context in Object Detection." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2009).

[38] P. Pintelas S. Kotsiantis D. Kanellopoulos. "Handling imbalanced datasets: A review." In: *GESTS International Transactions On Computer Science And Engineering* 30 (), 25–36.

[39] J. Wu L. Cao Q. Meng P.J. Kennedy S. Wang W. Liu. "Training deep neural networks on imbalanced data sets." In: *International Joint Conference on Neural Network* (2016).

[40] S. W. Lee M. L. Alexander D. W. Jacobs P. N. Belhumeur T. Berg J. Liu. "Birdsnap: Large-scale fine-grained visual categorization of birds." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014).

[41] D. V. Kulkarni T. Nakazawa. "Wafer Map Defect Pattern Classification and Image Retrieval Using Convolutional Neural Network." In: *IEEE Transactions on Semiconductor Manufacturing* 31.2 (2018).

[42] Yuan-Fu Yang. "A Deep Learning Model for Identification of Defect Patterns in Semiconductor Wafer Map." In: *Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)* (2019).

[43] Kuo Yuan. "A model-based clustering approach to the recognition of the spatial defect patterns produced during semiconductor fabrication." In: *IIE Transactions* 40 (2007).

[44] X. Liu Z. Zhou. "On Multi-Class of Cost-Sensitive Learning." In: *Proceedings of the 21st national conference on Artificial Intelligence* 1 (2006), 567–572.