POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAM IN INFORMATION TECHNOLOGY

ADVANCED LEARNING METHODS FOR ANOMALY DETECTION IN MULTIVARIATE DATASTREAMS AND POINT CLOUDS

Doctoral Dissertation of:
Luca Frittoli

Supervisor:
Prof. Giacomo Boracchi

Tutor:
Prof. Nicola Gatti

The Chair of the Doctoral Program:
Prof. Luigi Piroddi

2022 – Cycle XXXIV

This thesis presents research produced during a Ph.D. funded by STMicroelectronics.

Abstract

Anomaly detection is a challenging problem that can be encountered in several application domains, ranging from industrial quality control to cryptographic attacks. In the literature, several statistical and deep learning models have been proposed, each underpinning specific assumptions on the nature of the data to be analyzed. These models are typically configured on a training set to describe data generated by a process in normal conditions, and then assess whether the testing data conforms to the model or not.

This thesis presents new solutions for the anomaly-detection problem in two different settings. In the first part of the thesis, we assume that normal data are realizations of a random vector characterized by a certain probability distribution. In this case, we focus on a change-detection problem, where the goal is to detect permanent changes in the data-generating process by analyzing a sequence of samples acquired over time, namely a datastream. Our most substantial contribution to the research on change detection is *QuantTree Exponentially Weighted Moving Average (QT-EWMA)*, an online and nonparametric change-detection algorithm for multivariate datastreams that can be configured to maintain the target Average Run Length (ARL_0), namely the expected time before having a detection in stationary datastreams. In particular, we employ a QuantTree (QT) histogram to model the initial distribution from a training set and we define a novel change-detection statistic based on the Exponentially Weighted Moving Average (EWMA) monitoring scheme. The properties of the statistics based on QuantTree histograms guarantee that QT-EWMA is completely nonparametric and allow us to compute thresholds that guarantee the ARL_0

independently on the data distribution. Our experiments on synthetic and real-world data confirm that QT-EWMA controls the ARL_0 substantially more accurately than most of the competing methods, while achieving comparable or lower detection delays.

We extend our work to the concept-drift scenario, where the data samples are the object of a classification problem and therefore distribution changes, in this case referred to as concept drifts, require to update an underlying classifier. In the literature, concept drift detection is typically addressed by monitoring either the overall data distribution (thus ignoring class labels) or the error rate of the classifier (thus ignoring distribution changes that have little impact on classification error). We propose *Class Distribution Monitoring (CDM)*, a novel concept drift detection algorithms that combines the information coming from the data distribution and the class labels. In particular, CDM uses multiple instances of QT-EWMA to detect changes in the class-conditional distributions of annotated datastreams. The main advantage of CDM is that it can promptly detect drifts affecting a subset of classes and indicates which class triggered a detection, which might be crucial for diagnostics in practical applications. Most remarkably, we demonstrate that CDM inherits from QT-EWMA the ability to control the ARL_0, which is rarely guaranteed by alternative solutions. Our experiments on synthetic and real-world data show that CDM controls the ARL_0 more accurately than alternative methods that monitor either the overall data distribution or the error rate of a classifier, while achieving lower detection delays in most cases. In particular, CDM achieves the best performance when the drift affects a small subset of classes and when the drift does not substantially increase the classification error, which are the most challenging scenarios.

As a new application of change-detection tests, we address the problem of detecting errors in sequential cryptographic side-channel attacks. The aim of these attacks is reconstructing a private key one bit at a time by using a distinguisher, namely a statistic involving side channel data (e.g. the power consumption of the target device) and some intermediate results of the target algorithm. We cast error detection as a univariate change-detection problem since the distribution of the distinguisher changes after an error in the reconstruction of a key bit. In particular, we propose to detect errors by monitoring the distinguisher sequence by an online and nonparametric change-detection algorithm for univariate datastreams. Then, we propose an error-correction procedure based on a brute-force search over a small key window centered at the detected error, and a statistical test on the distinguisher values corresponding to each combination to select

the correct one. Our experiments on synthetic and real-world side-channel measurements demonstrate that our error detection and correction procedure substantially improves the success rate of different sequential attacks against the RSA-2048 algorithm, outperforming existing techniques, which simply set a threshold on the distinguisher value. Our findings demonstrate that sequential attacks can be substantially strengthened and thus might be more dangerous than previously thought. For this reason, countermeasures such as blinding should be employed even when the low success rate of sequential attacks suggests that the cryptosystem can be considered secure.

In the second part of the thesis, we address anomaly detection in point clouds. Point clouds are lists of the coordinates of points describing, for instance, the surface of an object, and are becoming increasingly popular since they can provide a compact yet detailed representation for 3D data. In this case, our goal is to assess whether individual point clouds belong to a certain normal class or not, a problem that is also known as one-class classification. Point clouds are high-dimensional data (typically a point cloud contains thousands of 3D coordinates) having complicated structures (typically 3D points lie on a locally flat surface). The main challenge of handling point clouds is their lack of a grid structure: in fact, the points do not necessarily lie on a regular grid, so traditional Convolutional Neural Networks (CNNs) cannot be directly applied to this type of data. For this reason, several point-convolutional layers, namely convolutional layers designed to process point clouds, have been proposed in the literature.

We propose the *composite layer*, an original operator that extracts and compresses the spatial information from the coordinates of the points by a Radial Basis Function Network (RBFN) and then combines it with the input features. Compared to the existing layers, our composite layer performs additional regularization by compressing the spatial information, and is substantially more flexible in terms of number of parameters and structure. We use our composite layers to implement *CompositeNets*, deep neural networks achieving excellent classification performance. Most remarkably, we are among the first to address anomaly detection in point clouds by training our CompositeNet in a self-supervised fashion. Our solution achieves state-of-the-art performance in anomaly detection, outperforming the only existing solution (based on a variational autoencoder) and shallow baselines leveraging hand-crafted features.

Finally, we address a relevant anomaly-detection problem in an industrial scenario. In particular, we analyze Wafer Defect Maps (WDMs), namely lists containing the 2D coordinates of the defects found on silicon wafers by an inspection machine. In normal conditions, wafers contain

few, randomly distributed defects, while defects forming specific patterns indicate problems and possibly failures in the production process. Since a few classes of defect patterns have been already identified by production engineers, in this case we cast anomaly detection as an open-set recognition problem, where the goal is to correctly classify WDMs belonging to the known classes (including the normal class) and detect anomalous WDMs, namely those containing defect patterns that do not belong to any known class. Even though the coordinates of a WDM lie on a grid, whose size is determined by the precision of the inspection machine, the grid is huge and prevents any CNN from processing WDMs at full resolution. For this reason, WDMs should be handled as 2D point clouds. To efficiently process WDMs at full resolution, we train a Submanifold Sparse Convolutional Network (SSCN) on the known classes. Then, we propose to detect anomalous patterns by applying an outlier detector based on a Gaussian Mixture Model (GMM) to the latent representation, namely the output of the penultimate layer of the SSCN. Our experiments on a dataset of WDMs acquired at STMicroelectronics show that our solution outperforms several methods from the literature, which we implemented on top of our SSCN for a fair comparison.

Contents

CHAPTER 1

Introduction

Anomaly detection is a challenging problem in machine learning with relevant applications in several domains. Anomaly-detection algorithms analyze the input data with the aim of assessing whether the data-generating process is operating in normal conditions or not. This problem paramount in industrial monitoring, where anomalies might indicate issues in the data-generating process that must be promptly identified and addressed. Crucial challenges of anomaly detection include that typically normal samples are abundant but anomalies are rare and it cannot be assumed that all the possible types of anomalies that might occur during testing are adequately represented in a training set. For this reason, anomaly detection should be conveniently tackled in *unsupervised* settings. In this thesis, we address the anomaly-detection problem under two different sets of data-modeling assumptions that can be encountered in real-world scenarios.

In the first part of the thesis, we consider a scenario where the monitored process produces a virtually unlimited stream of multivariate data samples. In normal conditions, we assume all samples to be drawn from a certain probability distribution, and our goal is to detect any permanent distribution change as soon as possible by monitoring the datastream *online*, i.e. one sample at a time. For this reason, in these settings anomaly-

detection problems are typically formulated as *change-detection* problems. Developing a change-detection algorithm requires addressing three main challenges: *i)* designing a model that can describe an arbitrary multivariate distribution; *ii)* implementing a nonparametric test statistic (based on the distribution model) to monitor the datastream without requiring any assumption on the data distribution; *iii)* defining thresholds for the statistic to control false alarms, which in online monitoring means to maintain a target *Average Run Length* (ARL_0), namely the expected time before a false alarm. In [1, 2], we address change detection in multivariate datastreams by proposing *QuantTree Exponentially Weighted Moving Average* (QT-EWMA) [1], an efficient online change-detection algorithm based on a QuantTree histogram [3], which we use to model the initial data distribution. The theoretical properties of QuantTree [3] enable us to define a nonparametric statistic based on the Exponentially Weighted Moving Average monitoring scheme, and to compute thresholds that control the ARL_0 by Montecarlo simulations. We then extend QT-EWMA by proposing *QT-EWMA-update* [2], where we use the incoming data to incrementally update the histogram, enabling to start monitoring when the training set is extremely small. Our experiments on synthetic and real-world datastreams confirm that QT-EWMA and QT-EWMA-update control the ARL_0 better than competing methods and achieve lower or comparable detection delays. In particular, QT-EWMA-update yields much lower detection delays than the alternatives when the training set is extremely small.

We also address *concept-drift detection*, a change-detection problem associated with a classifier operating on the datastream. In these settings, detecting distribution changes is crucial since it might be necessary to update a classifier to the new data distribution. The vast majority of the existing concept-drift detection methods aim at detecting a distribution change in the datastream or an increase in the error rate of a classifier. The former approach ignores class labels, which might turn out to be key for detecting drifts that affect only a subset of classes. The latter cannot detect drifts that do not significantly increase the classification error but might indicate relevant changes in the data-generating process. In [4], we address concept-drift detection by proposing *Class Distribution Monitoring* (CDM), which combines the information coming from the data distribution and the class labels by monitoring the class-conditional distributions using multiple instances of QT-EWMA. We demonstrate that CDM inherits from QT-EWMA the theoretical guarantees on the ARL_0, which instead is not controlled by the vast majority of concept-drift detection algorithms. Our experiments on synthetic and real-world data confirm that CDM accurately

controls the ARL_0 and outperforms alternative solutions, especially when the concept drift has little impact on the classification error or involves a small subset of classes.

As a new application of change detection, we address the problem of detecting and correcting errors in sequential attacks, a particular class of cryptographic side-channel attacks that recovers a secret key one bit at a time. The key recovery is done by computing, at each step of the attack, a distinguisher, namely a statistic measuring the likelihood of each possible key bit value (0/1) given some related side-channel data, for instance the power consumption of a device using the secret key for decryption. Then, the bit value maximizing the distinguisher is selected as the key bit recovered in that step of the attack. Detecting errors in sequential attacks can be cast as a change-detection problem in the datastream containing the maximum distinguisher values obtained in each step of the attack, since the first error permanently changes the distribution of the distinguisher. This property can also be used to correct the error since the distinguisher follows the same distribution before and after a correctly recovered bit. In [5], we detect errors by monitoring the value of the distinguisher using an online change-detection algorithm for univariate datastreams. Then, we correct errors by a brute-force search over a small window centered at the detected error, using a statistical test on the obtained distinguisher values to select the correct combination of key bits. We are the first to address this problem using sequential monitoring techniques, while existing solutions simply detect an error when the distinguisher gets lower than a threshold. Our experiments on synthetic and real-world side-channel data show that our procedure can improve the success rate of different sequential attacks much more than existing solutions. Our findings reveal that strengthened sequential attacks are more dangerous than expected. Therefore, it might be necessary to implement additional countermeasures even when the cryptosystem is considered secure due to the low success rate of the attacks.

In the second part of the thesis, we address anomaly detection in point clouds, a particular data format that has been recently attracting more and more interest since it allows to compactly represent 3D data. Our goal is to assess whether an individual point cloud belongs to the normal class, which is provided for training, or instead must be considered anomalous. This problem is also known in the literature as *one-class classification*. Point clouds are lists of coordinates – representing, for instance, the surface of a 3D object – that are sometimes associated with additional features (e.g. colors). Point clouds lack the matrix structure of signals and images, and therefore traditional machine learning and deep learning processing tools

(e.g. convolutions) cannot be applied directly, requiring the design of ad-hoc layers. The vast majority of the machine learning research on point clouds is focused on supervised tasks such as classification and semantic segmentation. In contrast, unsupervised tasks such as anomaly detection have been rarely addressed in the literature. In [6], we address anomaly detection in point clouds by proposing the *composite layer*, a new operator for point cloud processing in deep neural networks. Compared to the existing convolutional layers for point clouds, ours guarantees a higher degree of flexibility in terms of number of parameters and structure. We implement a *convolutional composite layer*, which operates similarly to convolutional layers for images, and an *aggregate composite layer*, which instead combines spatial information and features in a nonlinear manner. We employ our composite layers as building blocks for deep neural networks called *CompositeNets*. We successfully train our CompositeNets for classification and, most remarkably, anomaly detection, following a self-supervised approach based on geometric transformations [7]. Our experiments on synthetic and real-world datasets show that our CompositeNets achieve a similar classification accuracy to competing methods despite having a much simpler architecture. Most remarkably, our self-supervised CompositeNets substantially outperform deep and shallow anomaly-detection algorithms from the literature.

Finally, we tackle the relevant industrial problem of detecting anomalous patterns in Wafer Defect Maps (WDMs), which are point clouds containing the 2D coordinates of defects in silicon wafers. In particular, we consider the WDMs acquired by inspection machines at the STMicroelectronics production site in Agrate Brianza, Italy. In normal conditions, defects are few and randomly distributed in the wafer, while defect patterns indicate problems in the production process. Production engineers at STMicroelectronics have identified 12 classes of defect patterns related to known issues, but anomalous patterns might appear in the future due to production problems that have never been observed so far. Therefore, anomaly detection in WDMs can be cast as an *open-set recognition* problem, namely a classification problem in which anomalous samples (i.e. not belonging to any known class) must be detected during testing. Needless to say, addressing this problem is of paramount importance to build effective and automatic quality-inspection tools to monitor the production of silicon wafers. In [8], we address the problem of open-set recognition for WDM monitoring by training a Submanifold Sparse Convolutional Network (SSCN) [9] to accurately classify WDMs into the *Normal* class and the 12 known classes of defect patterns. Then, we detect anomalous patterns by analyzing the

latent representation of the SSCN (i.e. the output of its penultimate layer). In fact, the SSCN embeds the WDMs from each known class in the same region of the latent space, so that the last linear layer can distinguish them. Since anomalous WDMs do not belong to any known class, we employ an anomaly-detection algorithm based on a Gaussian Mixture Model (GMM) fitted on the latent representation of the WDMs from the training set. Our experiments on a real-world dataset of WDMs acquired at the STMicroelectronics plant in Agrate Brianza, Italy, show that our SSCN yields better classification performance than traditional CNNs (in particular, VGG16 and ResNet50) trained on WDMs pre-processed to obtain low-resolution images. In the task of detecting anomalous patterns, our open-set recognition algorithms outperforms several alternatives from the literature, which we apply on top of our SSCN for a fair comparison.

We have also addressed the problem of detecting anomalous regions in texture images, which is strongly related to the material presented in this thesis. However, we do not treat this topic in this thesis, where we focus on point clouds rather than images. In [10], we propose a new loss function to train autoencoders for detecting anomalous regions in images containing a texture. Our loss, which is based on a structural similarity metric, can capture the patterns characterizing the textures substantially better than traditional loss functions for autoencoders, such as the mean squared error. Our experiments on well-known benchmarks show that a simple autoencoder trained with our loss function often outperforms more sophisticated anomaly-detection methods leveraging pre-trained CNNs to extract features from the training images.

The vast majority of the material presented in this thesis appears in the following publications:

- L. Frittoli, M. Bocchi, S. Mella, D. Carrera, B. Rossi, P. Fragneto, R. Susella, and G. Boracchi. "Strengthening sequential side-channel attacks through change detection". *IACR Transactions on Cryptographic Hardware and Embedded Systems* (3), pp. 1–21. 2020.

- L. Frittoli, D. Carrera, and G. Boracchi. "Change detection in multivariate datastreams controlling false alarms". *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML–PKDD)*, pp. 421–436. Online, September 13–17, 2021.

- L. Frittoli, D. Carrera, and G. Boracchi. "Nonparametric and online change detection in multivariate datastreams using QuantTree". *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–14. 2022.

- L. Frittoli, D. Carrera, B. Rossi, P. Fragneto, and G. Boracchi. "Deep open-set recognition for silicon wafer production monitoring". *Pattern Recognition* (124) 108488. 2022.
- A. Bionda, L. Frittoli, and G. Boracchi. "Deep autoencoders for anomaly detection in textured images using CW-SSIM". *International Conference on Image Analysis and Processing (ICIAP)*, pp. 669–680. Lecce, Italy, May 23–27, 2022. Best paper award (NVIDIA prize).
- D. Stucchi, L. Frittoli, and G. Boracchi. "Class-distribution monitoring for concept drift detection". *IEEE International Joint Conference on Neural Networks (IJCNN)*. Padova, Italy, July 18–23, 2022.
- A. Floris, L. Frittoli, D. Carrera, and G. Boracchi. "Composite layers for deep anomaly detection on 3D point clouds". *Submitted to IEEE Transactions on Image Processing*, under review.

1.1 Structure of the Thesis

The structure of the thesis is illustrated in Figure 1.1. In Chapter 2 we formally introduce the main problems we address in this thesis. Then, we start the first part of the thesis (Chapters 3–6), where we address the problem of change detection in datastreams. In Chapter 3 we survey the literature regarding change and concept-drift detection. In Chapter 4 we illustrate the QT-EWMA algorithm, prove its theoretical properties and experimentally show its practical advantages. In Chapter 5 we present the CDM algorithm and the properties it inherits from QT-EWMA. Finally, in Chapter 6 we illustrate our error detection and correction procedure for sequential side-channel attacks based on a change-detection algorithm.

In the second part of the thesis (Chapters 7–9) we address the problem of anomaly detection in point clouds. In Chapter 7 we present the literature on anomaly detection and open-set recognition. In Chapter 8 we focus on 3D point clouds, illustrating our composite layer to process point clouds in Deep Neural Networks, its theoretical properties and applications in unsupervised anomaly detection. Finally, in Chapter 9 we present our open-set recognition algorithm for WDM monitoring. In Chapter 10 we conclude the thesis with some final remarks and possible future research directions.

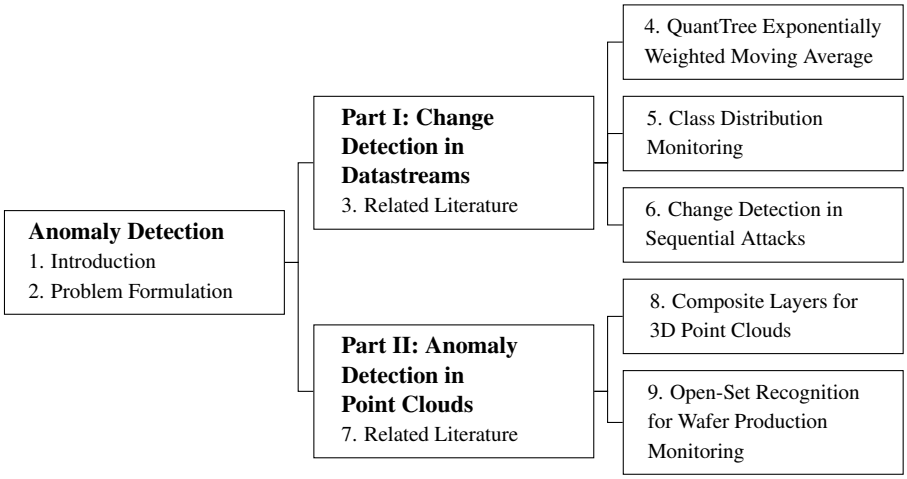


Figure 1.1: *A diagram illustrating the structure of the thesis. After the Introduction (Chapter 1), in Chapter 2 we formulate the anomaly-detection problems we address. The rest of the thesis is divided in two parts. In the first part, we focus on change detection in datastreams. In particular, in Chapter 3 we review the literature on change and concept-drift detection. In Chapter 4 we introduce QT-EWMA, a statistically sound change-detection algorithm for multivariate datastreams. In Chapter 5 we present CDM, an effective concept-drift detection algorithm based on QT-EWMA. Finally, in Chapter 6, we apply a change-detection algorithm to detect and correct errors in sequential side-channel attacks. In the second part of the thesis, we focus on anomaly detection in point clouds. In Chapter 7 we review the literature on anomaly detection and open-set recognition. In Chapter 8 we introduce the composite layer, a flexible operator to process 3D point clouds in Deep Neural Networks for classification and anomaly detection. Finally, in Chapter 9 we illustrate our open-set recognition algorithm for the monitoring of WDMs in semiconductor manufacturing.*

CHAPTER 2

Problem Formulation

Let us consider a data sample $x \in \mathbb{R}^d$ that, in normal conditions, is generated by a certain stochastic process ϕ_0. Anomaly detection is the problem of assessing whether x has been generated by ϕ_0 or by an alternative process ϕ_1, which is completely unknown and represents anomalous conditions. We assume that also the process ϕ_0 is unknown, but a training set TR containing samples drawn from ϕ_0 is available to configure an anomaly-detection algorithm. As stated in Chapter 1, in this thesis we address anomaly detection under two different sets of data-modeling assumptions. In the first part of the thesis, we assume that the data dimension is relatively low, so that each observation can be modeled as a realization of a random vector. We also assume that the samples are acquired in a virtually unlimited *datastream* $x_1, x_2, \ldots$ and that the anomalous condition to be detected is a distribution change $\phi_0 \rightarrow \phi_1$ occurring at an unknown time τ. In these settings, the anomaly-detection problem is usually referred to as *change detection*.

In the second part of the thesis, we consider high-dimensional data, in particular point clouds, that cannot be conveniently modeled as realizations of random vectors. In this case, the problem consists in detecting whether a single instance x has been generated by ϕ_0 or not. Since the

size and peculiar structure of point clouds prevents from directly applying traditional machine-learning techniques, we address anomaly detection on point clouds by deep neural networks.

Although anomaly detection is an unsupervised problem, change-detection and anomaly-detection techniques have often been employed in combination with supervised-learning methods, typically to handle anomalies in classification problems. A traditional application of change-detection algorithms is *concept-drift detection*, a classification problem in which test samples are provided as a datastream. The distribution of the test samples can change over time, requiring to update the classifier to maintain the pre-change performance. A comparatively more recent combination of anomaly detection and classification is *open-set recognition*, a classification problem in which anomalous samples that do not belong to any of the classes provided for training must be identified at test time.

2.1 Change Detection in Datastreams

We address the change-detection problem in a virtually unlimited multivariate datastream $x_1, x_2 \ldots \in \mathbb{R}^d$. We assume that, as long as there are no changes, all the data samples are i.i.d. realizations of a random variable having unknown distribution ϕ_0. We define the *change point* τ as the unknown time instant when a change $\phi_0 \to \phi_1$ takes place:

$$x_t \sim \begin{cases} \phi_0 & \text{if } t < \tau \\ \phi_1 & \text{if } t \geq \tau \end{cases} . \tag{2.1}$$

We assume that both ϕ_0 and $\phi_1 \neq \phi_0$ are unknown, and that a training set TR containing N realizations of ϕ_0 is provided to fit a model $\widehat{\phi}_0$. After fitting $\widehat{\phi}_0$, an online change-detection algorithm assesses, for each new incoming sample x_t, whether the sequence $\{x_1, \ldots x_t\}$ contains a change point. Typically, a statistic $\mathcal{T}_t$ based on $\widehat{\phi}_0$ is computed at each incoming x_t, then a decision rule is applied. Usually, the rule consists in controlling whether $\mathcal{T}_t > h_t$ for a certain threshold h_t, and the detection time t^* is defined as the first time instant when there is enough statistical evidence to claim that the datastream $\{x_1, \ldots x_{t^*}\}$ contains a change point, namely:

$$t^* = \min\{t : \mathcal{T}_t > h_t\}. \tag{2.2}$$

As in any statistical test, the sequence of thresholds $\{h_t\}_t$ employed in change detection should be defined to control the probability of having a false alarm, namely a detection on data drawn from ϕ_0. In online settings,

we measure the amount of false alarms by the Average Run Length [11], defined as $\text{ARL}_0 = \mathbb{E}_{\phi_0}[t^*]$, where the expectation is taken assuming that the whole datastream is drawn from ϕ_0. Thus, the ARL_0 is the average time before a false alarm. Ideally, the *target* ARL_0 of an online change-detection method should be set *a priori*, similarly to Type I error probability in hypothesis testing. The goal is to detect a distribution change as soon as possible, i.e., to minimize the *detection delay* $t^* - \tau$, while *controlling the* ARL_0, i.e. having an *empirical* ARL_0 that approaches the target ARL_0 set before monitoring.

2.1.1 Concept-drift detection

We also consider the problem of concept-drift detection in a virtually unlimited datastream $\{(x_t, l_t)\}_t$, where each sample $x_t \in \mathbb{R}^d$ is associated to a class label $l_t \in \mathcal{L}$. We assume that the observations x_t are independent realizations of a random vector that follows an initial distribution ϕ_0, and we denote by ϕ_0^ℓ the *class-conditional* distribution, i.e., the distribution of instances belonging to class ℓ, defined by

$$\mathbb{P}_{\phi_0^\ell}(x_t) = \mathbb{P}_{\phi_0}(x_t | l_t = \ell), \tag{2.3}$$

for each $\ell \in \mathcal{L}$. In other words, we say that $x_t \sim \phi_0$ if and only if $x_t \sim \phi_0^\ell$, where $l_t = \ell$. A concept drift is defined as a change $\phi_0^\ell \to \phi_1^\ell$ affecting at least one class-conditional distribution $\ell \in \mathcal{L}$, occurring at an unknown time τ. We assume that a training set TR containing labeled samples drawn from the initial distribution ϕ_0 is provided before monitoring, to configure the concept-drift detector.

2.2 Anomaly Detection in Point Clouds

In the second part of the thesis, we address the anomaly detection problem in high-dimensional data that cannot be modeled as realizations of random vectors. In this case, our goal is to analyze a single instance x and assess whether it is normal or anomalous. This is typically done by computing an *anomaly score* $\mathcal{T}(x)$ and comparing it to a threshold h. Then, the sample x is considered anomalous if $\mathcal{T}(x) > h$. When dealing with high-dimensional data such as signals and images, $\mathcal{T}(x)$ is typically the output of an unsupervised *Deep Neural Network (DNN)* exclusively trained on normal instances. This setup is also known as *semi-supervised* since only normal data samples are provided for training.

In particular, we address anomaly detection in *Point Clouds (PCs)*, a challenging data format that, unlike signals and images, cannot be represented by a regular grid. A point cloud is defined as a pair $x = (P, \mathcal{F})$, where $P \subset \mathbb{R}^s$ is a set of points in an s-dimensional space (typically $s = 2$ or $s = 3$) and $\mathcal{F} : P \to \mathbb{R}^I$ is a function associating a feature vector $\mathcal{F}(\mathbf{p}) \in \mathbb{R}^I$, e.g. a color, to each point $\mathbf{p} \in P$. Point clouds are a compact format for 3D data acquired by LiDAR sensors or depth cameras, and for this reason deep learning on point clouds is a challenge that has been attracting more and more interest in the computer vision community.

2.2.1 Open-set recognition

Open-set recognition is an anomaly-detection problem embedded in a classification scenario. In particular, it is assumed that a training set $TR = \{(x_i, l_i)\}_{i=1}^N$ containing labelled samples from a set of known classes $\mathcal{L}$. By contrast, anomalous instances (i.e., samples from *novel* or *unknown* classes) might appear during testing. The goal is to train an *open-set classifier* $\mathcal{K}$ that associates to each test sample x either a known class label or the *Anomalous* label. This is typically done by training a traditional classifier to address the *closed-set classification* problem, i.e., to associate each test sample x with its most likely class label $\widehat{\ell}(x) \in \mathcal{L}$, and an anomaly-detection algorithm to provide an anomaly score $\mathcal{T}(x)$. Then, $\mathcal{K}$ returns the *Anomalous* label when $\mathcal{T}(x) > h$, where h is the threshold, and the known class label $\widehat{\ell}(x)$ otherwise, i.e., the output of $\mathcal{K}$ is:

$$\mathcal{K}(x) = \begin{cases} \textit{Anomalous} & \text{if } \mathcal{T}(x) > h \\ \widehat{\ell}(x) & \text{otherwise} \end{cases}. \tag{2.4}$$

The open-set recognition problem has been widely studied on images, while the literature on open-set recognition in point clouds is rather scarce.

Part I

Change Detection in Datastreams

CHAPTER 3

Related Literature

In this chapter we survey the literature regarding change detection in datastreams. First, in Section 3.1 we review the most relevant methods designed for univariate datastreams. Then, in Section 3.2 we explore the multivariate case, a more challenging problem that has been addressed following different approaches. First, we introduce *one-shot* change-detection tests (Section 3.2.1), which separately analyze batches of data to assess whether they follow the initial distribution ϕ_0 or not. Then, we present *online* change-detection tests (Section 3.2.2), which are designed to be executed at each time t to assess whether the datastream $x_1, \ldots, x_t$ contains a change point or not. Online change-detection tests are typically more powerful since they leverage a larger amount of data compared to one-shot tests, however they require stronger guarantees to control false alarms (Section 3.2.3). Finally, in Section 3.3 we briefly survey the literature on concept drift detection.

3.1 Univariate Datastreams

Change-detection in univariate datastreams was originally addressed in statistical process control [12], where the observations are modeled as i.i.d. realization of a random variable. The first algorithms, such as the Shewhart

chart [13], *Cumulative Sum (CuSum)* [14], and *Exponentially Weighted Moving Average (EWMA)* [15], are designed to be executed online to detect changes in the mean of ϕ_0. The main drawback of these very well-known solution is that they are parametric, i.e., they assume that ϕ_0 belongs to a known family of probability distributions such as Gaussian [13, 14] or Bernoulli [15].

Since in general real-world data cannot be assumed to follow a distribution from a known family, nonparametric tests have been developed to guarantee the control of false alarms when monitoring datastreams generated by an arbitrary distribution ϕ_0. There are two main approaches to obtain nonparametric change-detection tests: the first is to pre-process the datastream in such a way that the samples follow a known distribution. For instance, in [16] the Box-Cox transformation [17] is employed to make the data approximately Gaussian. The second approach is to use nonparametric statistics based on ranks such as the Mann-Whitney [18], Mood [19], Lepage [20], and Kolmogorov-Smirnov [21] test statistics. Another nonparametric approach to change detection consists in permutation tests [22, 23], which however are quite expensive from a computational point of view.

A relevant approach to change detection is the *Change Point Model (CPM)* [12], which is based on a test statistic $\mathcal{T}$ specifically designed to compare the distributions of two sets of observations and assess "how different" they are. To analyze the datastream $x_1, \ldots, x_t$, the CPM tests each tentative change point $k < t$ by splitting the datastream in two sets $\mathbf{A}_k = \{x_1, \ldots, x_{k-1}\}$ and $\mathbf{B}_k = \{x_k, \ldots, x_t\}$, and computing the test statistic $\mathcal{T}_{k,t} = \mathcal{T}(\mathbf{A}_k, \mathbf{B}_k)$. When the maximum of the test statistic $\mathcal{T}_t = \max_k \mathcal{T}_{k,t}$ exceeds a specific threshold h_t, a change point is reported at the location $\hat{\tau}$ maximizing $\mathcal{T}$:

$$\hat{\tau} = \arg\max_k \mathcal{T}_{k,t}. \tag{3.1}$$

The CPM was originally introduced in parametric settings to detect changes either in the mean [12] or in the variance [24] of a Gaussian ϕ_0. In [25, 26] the CPM is modified using nonparametric statistics based on ranks [18–20]. The main advantage of this approach is its ability to estimate the position of the change point by (3.1), while the other online change-detection tests [13–15] only assess whether a distribution change has occurred or not. The main drawback is that, whenever a new sample x_t is acquired, the test statistic $\mathcal{T}$ has to be computed for each tentative change point $k \in \{1, \ldots, t\}$, thus the computational cost of CPM is typically higher than alternative algorithms and increases over time.

3.2 Multivariate Datastreams

Change detection in multivariate datastreams is a challenging problem since statistics designed for univariate datastreams, especially nonparametric statistics based on ranks [18–20], cannot be directly applied to multivariate data. In Section 3.2.1 we describe *one-shot* change-detection algorithms, which operate on fixed-size batches of data rather than online. In Section 3.2.2 we survey the existing approaches to online change detection in multivariate datastreams, and in Section 3.2.3 we focus on the problem of controlling the ARL_0. Finally, in Section 3.2.4 we summarize the properties of the most relevant change-detection algorithms for multivariate datastreams.

3.2.1 One-shot detectors

Change detection in multivariate datastreams has often been addressed using one-shot statistical tests, which analyze individual batches of data to assess whether they follow ϕ_0 or not. A relevant example is the parametric Hotelling test statistic [27], which is designed to detect changes in the mean of a Gaussian distribution. When ϕ_0 is unknown, as often occurs when dealing with real-world data, the typical approach consists in fitting a semiparametric model $\widehat{\phi}_0$, such as a Gaussian [28] or a Gaussian Mixture [29], on a training set drawn from ϕ_0. Then, the likelihood or log-likelihood with respect to $\widehat{\phi}_0$ can be used to assess whether a new batch follows ϕ_0 [29]. Although these models are relatively flexible, they might not be able to adequately approximate any ϕ_0.

Histograms are very flexible nonparametric models to describe any initial distribution ϕ_0 [30, 31]. The main drawback of regular-grid histograms is that the number of bins does not scale well with the data dimension. Moreover, in [31] it has been shown that uniform-density partitions yield better detection performance than regular grids. A remarkable example of a change-detection test based on uniform-density histogram is QuantTree [3]. After constructing a histogram over a training set following a stochastic process, the Pearson χ^2 test statistic [32] can be used to assess the goodness-of-fit of a batch of data.

Another nonparametric approach consists in comparing the batches to reference data drawn from ϕ_0 by the *Maximum Mean Discrepancy (MMD)* statistic [33]. The main drawbacks of MMD are the following: *i)* it requires a large amount of reference data [34], and *ii)* its computational complexity does not scale well with the data dimension. This latter problem has been addressed in *Neural Tangent Kernel MMD (NTK-MMD)* [35], where

a neural network is trained on the reference data to approximate the MMD statistic, reducing its computational overhead.

3.2.2 Online detectors

Several approaches have been proposed to address online change-detection in multivariate datastreams, typically based on one-shot statistical tests or on online methods originally designed for univariate data.

Dimensionality reduction. A common solution consists in reducing the dimensionality of the datastream so that the problem boils down to monitoring a univariate datastream. Dimensionality reduction can be achieved by computing the likelihood of each observation with respect to a Gaussian mixture model fitted on a training set [36] or by Principal Component Analysis (PCA) [37, 38]. Then, the resulting univariate datastream can be monitored using state-of-the-art online change-detection methods designed for univariate datastreams, such as the nonparametric CPM [25].

Multi-stream monitoring. Change detection in multivariate datastreams has often been addressed following the *multi-stream* (*multi-channel*) approach, i.e., by separately analyzing each component of a multivariate datastream [39–41]. However, the assumptions underpinning multi-stream monitoring are stronger than those of change-detection in multivariate datastreams. In fact, [39–41] assume that the components are generated by independent 1-dimensional random variables, thus ignoring the possible correlation among them. Moreover, in multi-stream settings, changes typically affect the distribution of a subset of these random variables [39], while, in multivariate settings, more general distribution changes are considered [42]. In particular, subtle changes affecting the correlation between components might be hard to detect by separately analyzing the components.

Extending univariate tests. Several extensions of online change-detection tests originally designed for univariate datastreams have been proposed, for instance in [43–46], which extend the Shewhart chart, CuSum, and EWMA. Also the CPM framework has been extended to operate on multivariate datastreams [47] using the Hotelling test statistic. As in the univariate case, the main drawback of these solutions is that they are parametric and thus assume that ϕ_0 belongs to a given family of probability distributions, typically a multivariate Gaussian [47].

Unfortunately, the extension of nonparametric statistical tests to multivariate data is not straightforward since these rely on the natural order of

real numbers, which is not well defined in multiple dimensions. For this reason, only a few works follow this direction: for instance, [48] propose a multivariate extension of the Mann-Whitney test statistic by ranking the data points over each dimension, and [49] extend permutation tests to monitor multivariate datastreams. The main drawback of these solutions is that their computational cost does not scale well with the data dimension, making it infeasible to apply them in online settings.

Extending one-shot tests. One-shot statistical tests for multivariate data can in principle be extended to online monitoring by simply applying them to consecutive, fixed-size batches of data. However, as we show in the following Chapter, online tests that consider, at each time t, the whole datastream $x_1, \dots, x_t$ are substantially more powerful since they have access to more data samples. A relevant online generalization of histograms-based change-detection tests is the *Binned Generalized CuSum (BG-CuSum)* test [50]. However, this algorithm has been tested only on univariate datastreams, and it is infeasible to extend to multivariate data because the number of bins scales exponentially with the data dimension. Moreover, it requires to know the analytical expression of ϕ_0, or an accurate approximation, to effectively control false alarms. This implies that, when ϕ_0 is unknown, a large training set is required for parameters estimation, especially when the data dimension d is high [50].

Recently, the MMD statistic has been employed for online change detection [34, 35, 51]. In particular, *No-prior-knowledge EWMA (NEWMA)* [51] analyzes the relation between two EWMA statistics with different forgetting factors based on MMD to detect changes online. Unfortunately, this solution requires the analytical expression of ϕ_0 [51], which limits the applicability of this solution. *Scan-B* [34] and NTK-MMD [35] respectively compute the MMD statistic and approximate it by a neural network on a sliding window of the datastream. A major limitation of the algorithms based on MMD is that they require a large amount of reference data [34,35].

3.2.3 Control of False Alarms

As any statistical test, change-detection tests compute a statistic $\mathcal{T}$ of the input data, and detect a change when the statistic exceeds a threshold h defined to control the false alarm probability. In one-shot change-detection tests, the threshold is defined to set the probability of a batch drawn from ϕ_0 to trigger a detection to the desired value α, and is typically computed by bootstrap on the training set [29] or by Montecarlo simulations on synthetic

data if the test is completely nonparametric [3].

In online change-detection tests, in which a test statistic $\mathcal{T}_t$ is computed whenever a new sample x_t is acquired, a sequence of thresholds $\{h_t\}$ should be defined to set the Average Run Length (ARL_0) [11], i.e., the expected time before a false alarm. There are two main main strategies to define thresholds $\{h_t\}$ that guarantee the desired ARL_0. The first consists in setting a constant false alarm probability, namely

$$\mathbb{P}_{\phi_0}(\mathcal{T}_t > h_t \mid \mathcal{T}_k \leq h_k \; \forall k < t) = \alpha \quad \forall t \geq 1. \tag{3.2}$$

As shown in [52], when this property is satisfied, the detection time under ϕ_0 is a Geometric random variable with parameter α, hence its expected value is $\text{ARL}_0 = 1/\alpha$. Since it is usually infeasible to exactly compute the conditional probabilities in (4.13), [25, 26, 47] perform Montecarlo simulations by computing the statistics $\mathcal{T}_t$ on a large number of synthetic datastreams. The, the first threshold h_1 is defined as the empirical $(1-\alpha)$-quantile of all the values of $\mathcal{T}_1$, where $\alpha = 1/\text{ARL}_0$. Similarly, the thresholds h_t with $t > 1$ are defined as the $(1-\alpha)$-quantiles of the values $\mathcal{T}_t$, using only those datastreams whose statistics $\mathcal{T}_k$ have never exceeded any of the previous thresholds h_k for $k = 1, \ldots, t-1$. Computing the thresholds $\{h_t\}$ in this way guarantees that, for each time t, the empirical quantiles of $\mathcal{T}_t$ are conditioned to $\mathcal{T}_k \leq h_k \; \forall k < t$, which in turn implies (4.13), hence the target ARL_0 is preserved [52]. The main drawback of this approach is that these Montecarlo simulations can be applied only when the test is parametric [47], thus the distribution ϕ_0 is assumed to be known, or when the distribution of the statistics $\mathcal{T}_t$ does not depend on ϕ_0, as in [25, 26]. Due to the large amount of data and computations required for simulations [25], computing the thresholds $\{h_t\}$ by bootstrap is usually infeasible.

The second approach consists in approximating a lower bound for the ARL_0 at a fixed value of the threshold $h_t \equiv h$ [34, 45, 50, 51], for instance studying the asymptotic behavior of the ARL_0 when $h \to \infty$ [34]. Among these, the only method in which the threshold for a given ARL_0 can be computed independently on ϕ_0 is Scan-B [34]. In contrast, [45] assumes that ϕ_0 is a Gaussian distribution, and other solutions that in principle can operate on any ϕ_0 [50,51], require the analytical expression of ϕ_0, or at least a very accurate approximation [50], to compute these threshold. These requirements limit the applicability of these solutions since ϕ_0 is typically unknown, especially when dealing with real-world data, and only a relatively small training set drawn from ϕ_0 is usually assumed to be available.

Table 3.1: *Main properties of the most relevant change-detection algorithms for multivariate datastreams.*

	algorithm	nonparametric	online	control ARL_0	update
Gaussian	Hotelling CPM [47]		✓	✓	✓
	SS-CPD [45]		✓	✓	
	SPLL [29]	semiparametric	with modifications	with modifications	
PCA	PCA-SPLL [37]	✓			
	PCA-CD [38]	✓	✓		
	Martingales [22, 23]	✓	✓		
MMD	Scan-B [34]	✓	✓	✓	✓
	NEWMA [51]	✓	✓	iff ϕ_0 is known	
	NTK-MMD [35]	✓	✓		
Histograms	BG-CuSum [50]	✓	✓	iff ϕ_0 is known	
	QuantTree [3]	✓	with modifications	with modifications	
	QT-EWMA	✓	✓	✓	
	QT-EWMA-update	✓	✓	✓	✓

3.2.4 Summary of the main properties

Table 3.1 summarizes the main properties of the most relevant change-detection algorithms designed to monitor multivariate datastreams. In particular, we consider the following properties: being nonparametric, being executed online, controlling the ARL_0, and being able to incrementally update the model using the incoming data. To the best of our knowledge, Scan-B [34] is the only nonparametric and online change-detection algorithm from the literature in which the target ARL_0 can be set independently of ϕ_0. As we show in Section 4.5, one-shot change-detection methods such as QuantTree [3] and *Semi-Parametric Log-Likelihood (SPLL)* [29] can be modified to operate online while controlling the ARL_0. All the other methods that control the ARL_0 are either parametric [45, 47], or require the analytical expression of ϕ_0 [50, 51].

3.3 Concept-drift Detection

Concept-drift detection [53] is a challenging problem in datastream learning, and has been addressed in different settings and by different approaches, which we summarize here. A more comprehensive survey on concept-drift detection can be found in [54]. Since in this thesis we focus only on concept-drift detection, we do not review the literature on *concept-drift adaptation*. We refer to [55] for a survey on this subject.

The most popular concept-drift detection methods assume that the label

l_t of each sample x_t is revealed after classifying it by a previously trained classifier $\mathcal{K}$ [56]. This assumption enables to detect concept drifts by analyzing the binary stream $\{e_t\}$ defined by the classification errors of $\mathcal{K}$:

$$e_t = \mathbb{1}(\mathcal{K}(x_t) \neq l_t). \tag{3.3}$$

Typically, a concept drift is reported when the error rate increases. In particular, *Drift Detection Method (DDM)* [57] and its variants [58–61] compute a cumulative average and standard deviation of $\{e_t\}$ during monitoring, and employ at each time t a statistical test to assess whether the error rate has changed significantly, approximating the distribution of the error rate by a Gaussian. *ADaptive WINdowing (ADWIN)* [62] and its variants [63–65] analyze $\{e_t\}$ in a sliding-window fashion by comparing the average of two consecutive sub-windows, for all the possible splits of the current window (similarly to the CPM described earlier). A concept drift is detected when the absolute difference between the average values of two sufficiently large sub-windows exceeds a threshold. Recently, it has been proposed to monitor the classification performance on individual classes to handle imbalanced datastreams and drifts affecting only some classes [66, 67]. Unfortunately, none of these solutions is designed to control the ARL_0.

By contrast, *EWMA for Concept Drift Detection (ECDD)* [56] analyzes $\{e_t\}$ online by the EWMA chart [15]:

$$U_t = (1-r)U_{t-1} + \lambda e_t, \quad U_0 = \hat{p}_{0,0}, \tag{3.4}$$

where $\hat{p}_{0,t}$ indicates the average error rate of $\mathcal{K}$ up to time t, and λ is the EWMA parameter, which in [56] is set to $\lambda = 0.2$. A concept drift is detected when $U_t > \hat{p}_{0,t} + \Lambda\sigma_t$, where Λ is a control limit and σ_t is the estimated standard deviation of U_t:

$$\sigma_t = \sqrt{\hat{p}_{0,t}(1-\hat{p}_{0,t})\frac{\lambda}{2-\lambda}(1-(1-\lambda)^{2t})}. \tag{3.5}$$

Since U_t is an incremental estimate of the error rate of $\mathcal{K}$, which gives exponentially larger weights to the latest elements of e_t compared to older elements, and since a one-sided decision rule is applied, ECDD can only detect drifts that increase the classification error. Contrarily to the vast majority of concept drift detection methods, ECDD allows to control false alarms by setting the ARL_0 by tuning the control limit Λ, and [56] provides polynomial approximations to compute Λ for different values of the target ARL_0 as a function of $\hat{p}_{0,t}$.

A second relevant class of concept-drift detection methods monitors the distribution of the input data by applying a change-detection algorithm [54]

such as those described earlier. The change-detection algorithm monitors the input samples x_t, overlooking the corresponding class labels l_t, and any distribution change is reported as a concept drift. On the one hand, this approach ignores class information that might be relevant to detect a concept drift. On the other hand, it can operate also when few or no labels are available, as might be the case in a real-world monitoring scenario.

CHAPTER 4

QuantTree Exponentially Weighted Moving Average

In this chapter, we present *QuantTree Exponentially Weighted Moving Average (QT-EWMA)*, a nonparametric online change-detection algorithm that can effectively monitor multivariate datastreams controlling false alarms. QT-EWMA combines a QuantTree (QT) histogram [3], used as a model for the initial distribution, and a novel online statistic based on Exponentially Weighted Moving Average (EWMA) [15] that monitors the proportion of incoming samples falling in each bin of the histogram. The theoretical properties of QuantTree [3] guarantee that QT-EWMA is completely nonparametric and that its thresholds controlling the ARL_0 can be set *a priori*.

In Section 4.1 we introduce the construction and use of QuantTree histograms, and we extend the theoretical results in [3] by fully characterizing the distribution of the bin probabilities. In Section 4.2 we present the QT-EWMA algorithm, and in Section 4.3 we illustrate our Montecarlo procedure to compute thresholds controlling the ARL_0. In Section 4.4 we extend QT-EWMA by incrementally updating the estimated bin probabilities, improving the detection performance when the training set is small. Then, in Section 4.5 we show how to modify one-shot detectors (Section 3.2.1) for

online monitoring controlling the ARL_0, and in Section 4.6 we analyze the computational complexity of QT-EWMA and alternative algorithms. Finally, in Section 4.7 we present and discuss the results of our experiments.

4.1 QuantTree Histograms

Histograms are a flexible model for multivariate distributions over $\mathbb{R}^d$ [3,30, 31] defined by $\{(\mathcal{S}_j, \tilde{\pi}_j)\}_{j=1}^K$, where the K bins $\mathcal{S}_j \subset \mathbb{R}^d$ form a partitioning of $\mathbb{R}^d$, i.e. $\bigcup_{j=1}^K \mathcal{S}_j = \mathbb{R}^d$ and $\mathcal{S}_j \cap \mathcal{S}_i = \emptyset$ for $j \neq i$, and $\tilde{\pi}_j$ is an estimate of the bin probability $p_j = \mathbb{P}_{\phi_0}(\mathcal{S}_j)$. QuantTree [3] is an algorithm that takes as input a training set TR containing N samples drawn from ϕ_0 and returns a histogram $\mathcal{Q} = \{(\mathcal{S}_j, \tilde{\pi}_j)\}_{j=1}^K$ whose bins $\mathcal{S}_j$ are constructed by splitting $\mathbb{R}^d$ along random directions to contain $\pi_j N$ samples from TR, where $\{\pi_j\}_{j=1}^K$ is a given set of target bin probabilities. Since in [31] it has been shown that histograms based on partitionings in which all bins have the same probability under ϕ_0 yield superior detection performance, the most natural choice for the target probabilities is $\pi_j = 1/K$ for $j = 1, \ldots, K$. The construction of a QuantTree histogram requires to order K components of the N points in the training set TR, thus resulting in a computational cost of $\mathcal{O}(KN \log N)$ operations [3]. Further details on QuantTree – including how to define the bins when TR cannot be exactly split to match target probabilities – can be found in [3].

The rationale behind change-detection tests based on histograms is comparing the proportion of test samples falling in each bin $\mathcal{S}_j$ of the histogram representing ϕ_0 against the actual bin probabilities p_j under ϕ_0. Since ϕ_0 is assumed to be unknown, the bin probabilities $(p_1, \ldots, p_K)$ are the unknown realization of a random vector, and in QuantTree histograms [3] they can be approximated by:

$$\tilde{\pi}_j := \begin{cases} \dfrac{\pi_j N}{N+1} & \text{if } j = 1, \ldots, K-1 \\ \dfrac{\pi_j N + 1}{N+1} & \text{if } j = K \end{cases} . \tag{4.1}$$

The most important property of QuantTree histograms is that the distribution of any statistic defined by the number of test samples falling in each bin $\mathcal{S}_j$ does not depend on ϕ_0 nor on the data dimension d, as demonstrated in [3], thus yielding nonparametric statistical tests. Here we extend the theoretical results in [3] by fully characterizing the probability distribution of $(p_1, \ldots, p_K)$ in the following Proposition.

Proposition 4.1. *Let $\{\mathcal{S}_j\}_{j=1}^K$ be a partitioning built by the QuantTree algorithm with target probabilities $\{\pi_j\}_{j=1}^K$ on a training set $TR \sim \phi_0$ of size N. Then, the bin probability vector $(p_1, \dots, p_K)$ is drawn from the Dirichlet distribution:*

$$(p_1, \dots, p_K) \sim \text{Dir}\big(\pi_1 N, \pi_2 N, \dots, \pi_K N + 1\big). \tag{4.2}$$

Proof. We leverage the result in [68] linking the Dirichlet distribution to the *stick-breaking process*. In particular, the stick-breaking process generates a sequence of K random variables $q_1, \dots, q_K$ as

$$q_j = \prod_{k=1}^{j-1}(1 - \tilde{q}_k) \cdot \tilde{q}_j,\ j < K, \quad q_K = 1 - \sum_{j=1}^{K-1} q_j, \tag{4.3}$$

where $\tilde{q}_j$ for $j = 1, \dots, K-1$ are defined as

$$\tilde{q}_j \sim \text{Beta}\left(\gamma_j, \sum_{k=j+1}^{K} \gamma_j\right), \tag{4.4}$$

and $\gamma_1, \dots, \gamma_K$ are the parameters that define the stick-breaking process. In [68] it has been shown that

$$(q_1, \dots, q_K) \sim \text{Dir}\big(\gamma_1, \dots, \gamma_K\big). \tag{4.5}$$

To prove the proposition it is enough to show that there exists a specific configuration of γ_j such that the bin probabilities p_j of a QuantTree histogram can be expressed as q_j in (4.3). To this purpose, we recall the result in [3] where it has been shown that p_j can be written as

$$p_j = \prod_{k=1}^{j-1}(1 - \tilde{p}_k) \cdot \tilde{p}_j,\ j < K, \quad p_K = 1 - \sum_{j=1}^{K-1} p_j, \tag{4.6}$$

where $\tilde{p}_j$ are independent and follow Beta distributions:

$$\tilde{p}_j \sim \text{Beta}\left(\pi_j N, \left(1 - \sum_{k=1}^{j} \pi_k\right) N + 1\right)\ j = 1, \dots, K-1. \tag{4.7}$$

Now, we only need to find a suitable choice of $\gamma_1, \dots, \gamma_K$ to express the $\tilde{p}_j$ as the $\tilde{q}_j$ in (4.4). If we define $\gamma_j = \pi_j N$ for $j < K$ as in (4.7) and $\gamma_K = \pi_K N + 1$, we obtain that:

$$\sum_{k=j+1}^{K} \gamma_k = \sum_{k=j+1}^{K} \pi_k N + 1 = \left(1 - \sum_{k=1}^{j} \pi_k\right) N + 1, \tag{4.8}$$

where the last equality follows from the fact that $\sum_{j=1}^{K} \pi_j = 1$. Equation (4.8) ensures the correspondence between $\tilde{p}_j$ in (4.7) and $\tilde{q}_j$ in (4.4), which implies the thesis. □

Proposition 4.1 means that, whenever we construct a QuantTree over a training set, we are partitioning $\mathbb{R}^d$ into K bins in such a way that the bin probabilities $(p_1, \ldots, p_K)$ under ϕ_0, namely $p_j = \mathbb{P}_{\phi_0}(\mathcal{S}_j)$, are drawn from the Dirichlet distribution in (4.2). Since the expected value of the j-th component of a random vector drawn from the Dirichlet distribution $\text{Dir}(\gamma_1, \ldots, \gamma_K)$ is $\gamma_j / \sum_{k=1}^{K} \gamma_k$ [69], by simple algebraic manipulation of the Dirichlet parameters in (4.2) we have that $\mathbb{E}_{\phi_0}[p_j] = \tilde{\pi}_j$, where $\tilde{\pi}_j$ are defined as in (4.1). Therefore, the values $\tilde{\pi}_1, \ldots, \tilde{\pi}_K$ defined in (4.1) can be used as estimates of the bin probabilities $p_1, \ldots, p_K$. Moreover, thanks to the properties of the Dirichlet distribution, we have that $\text{var}[p_j] \to 0$ as $N \to \infty$, thus $\tilde{\pi}_j$ is a good estimator of p_j when the training set TR is sufficiently large. We remark that the statistics employed in QuantTree [3] estimate the bin probability p_j by its target value π_j since it is assumed that a large training set is provided, and $\tilde{\pi}_j \to \pi_j$ when $N \to \infty$ by definition (4.1). Here we also consider cases where N is small, thus in the QT-EWMA statistics (4.11) and (4.12) we employ $\tilde{\pi}_j$, which is a more accurate estimate of p_j.

4.2 The QT-EWMA Algorithm

Here we introduce *QuantTree Exponentially Weighted Moving Average (QT-EWMA)*, which extends to online monitoring the QuantTree monitoring scheme [3] that was originally designed for one-shot change detection. In particular, we define a novel online statistic $\mathcal{T}_t$ to monitor the proportion of samples in the datastream belonging to each bin $\mathcal{S}_j$ of a QuantTree histogram.

The QT-EWMA monitoring scheme is illustrated by Algorithm 4.1. First, we construct a QuantTree histogram on a training set TR drawn from ϕ_0 (line 2). When a new sample x_t is acquired at time t, we define K binary statistics from the indicator functions of each bin $\mathcal{S}_j$, namely

$$y_{j,t} = \mathbb{1}(x_t \in \mathcal{S}_j), \quad j = 1, \ldots, K, \tag{4.9}$$

to track in which bin x_t falls. If $x_t \sim \phi_0$, the expected value of each variable $y_{j,t}$ is, by definition,

$$\mathbb{E}_{\phi_0}[y_{j,t}] = p_j, \ j = 1, \ldots, K, \tag{4.10}$$

Algorithm 4.1 QuantTree Exponentially Weighted Moving Average (QT-EWMA)

Input: datastream $x_1, x_2, \ldots$, target probabilities $\{\pi_j\}_{j=1}^K$, thresholds $\{h_t\}_t$, TR
Output: detection flag `ChangeDetected`, detection time t^*

1: `ChangeDetected` $\leftarrow$ False, $t^* \leftarrow \infty$
2: construct QuantTree histogram $\mathcal{Q} = \{(\mathcal{S}_j, \tilde{\pi}_j)\}_{j=1}^K$ from TR
3: $Z_{j,0} \leftarrow \tilde{\pi}_j \; \forall j = 1, \ldots, K$
4: **for** $t = 1, \ldots$ **do**
5: $y_{j,t} \leftarrow \mathbb{1}(x_t \in \mathcal{S}_j)$
6: $Z_{j,t} \leftarrow (1-\lambda) Z_{j,t-1} + \lambda y_{j,t}, \quad j = 1 \ldots, K$
7: $\mathcal{T}_t \leftarrow \sum_{j=1}^K (Z_{j,t} - \tilde{\pi}_j)^2 / \tilde{\pi}_j$
8: **if** $\mathcal{T}_t > h_t$ **then**
9: `ChangeDetected` $\leftarrow$ True, $t^* \leftarrow t$
10: **break**;
11: **end if**
12: **end for**
13: **return** `ChangeDetected`, t^*

where the expected value $\mathbb{E}_{\phi_0}$ is computed under the assumption that $x_t \sim \phi_0$. Since ϕ_0 is unknown, so are the bin probabilities $(p_1, \ldots, p_K)$, which can be approximated by $\tilde{\pi}_j \approx p_j$, where $\tilde{\pi}_1, \ldots, \tilde{\pi}_K$ are defined in (4.1). After evaluating the statistics $y_{j,t}$ for the incoming sample x_t (line 11), we compute the EWMA statistic [15] $Z_{j,t}$ (line 6), to monitor the proportion of data in each $\mathcal{S}_j$:

$$Z_{j,t} = (1-\lambda) Z_{j,t-1} + \lambda y_{j,t} \quad \text{where} \quad Z_{j,0} = \tilde{\pi}_j \,. \tag{4.11}$$

Finally, we define the QT-EWMA change-detection statistic:

$$\mathcal{T}_t = \sum_{j=1}^{K} \frac{(Z_{j,t} - \tilde{\pi}_j)^2}{\tilde{\pi}_j} \,, \tag{4.12}$$

which is similar to the Pearson statistic [32]. In fact, $\mathcal{T}_t$ measures the overall deviation of the proportion of samples $x_1, \ldots, x_t$ falling in each bin $\mathcal{S}_j$, represented by $Z_{j,t}$, from their estimated expected values $\tilde{\pi}_j$ under ϕ_0. This difference naturally increases when $t > \tau$ as a consequence of a change $\phi_0 \to \phi_1$, which can be expected to modify the probability of some bin $\mathcal{S}_j$. The QT-EWMA statistic is computed at each incoming sample (line 7) and then compared against the corresponding threshold h_t to detect changes (line 14).

Impact of the choice of K. The number of bins K of the QuantTree histogram is a fundamental parameter that influences the change-detection

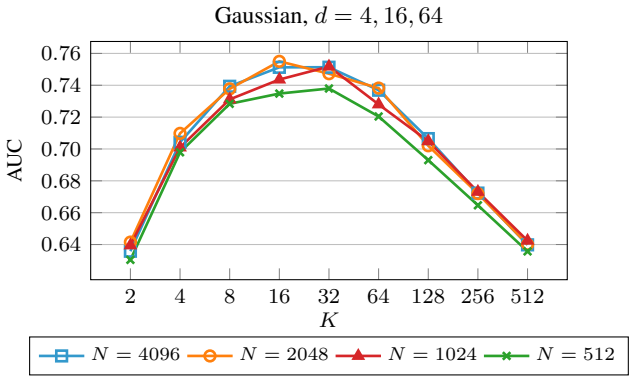


Figure 4.1: *Detection power of QT-EWMA on Gaussian datastreams with different training set size N when varying the number of bins K. The results, which are averaged over $d = 4, 16, 64$, show that histograms with a small K cannot describe ϕ_0 accurately and yield low detection performance. Also setting a very large K harms detection performance since, at a fixed N, increasing K yields inaccurate estimates $\{\hat{\pi}_j\}_j$.*

performance of QT-EWMA. To analyze the impact of the choice of K, we test QT-EWMA with $K = 2, 4, 8, \ldots, 512$. In particular, we compute the QT-EWMA statistic over 5000 stationary Gaussian datastreams and 5000 datastreams containing a change point at $\tau = 500$, and we measure the detection power by the *Area Under the ROC Curve (AUC)* given by the statistic values at $t = 1000$, namely T_{1000}. The distribution changes $\phi_0 \to \phi_1$ consist in random roto-translations of ϕ_0 generated by CCM [42] (see Section 5.3.1). Since the detection performance depends also on the training set size N, we employ different values of $N = 512, 1024, 2048, 4096$.

In Figure 4.1 we report the average results obtained on datastreams with $d = 4, 16, 64$. Setting $K = 2, 4$ yields low AUC because histograms having such few bins cannot describe ϕ_0 well. Increasing the number of bins ($K = 128, 256, 512$) while keeping N fixed increases the variance of the bin probabilities p_j due to the properties of the Dirichlet distribution (4.2), harming the detection performance. Intermediate values – especially $K = 16, 32$ for the considered values of N – yield the best results, thus we in our experiments we select $K = 32$ as in [3]. As expected, the detection performance increases with N since $\text{var}[p_j] \to 0$ when $N \to \infty$. However, the improvement is substantial only when increasing N from 512 to 1024, while using a larger N yields a marginal improvement, see Figure 4.1.

4.3 Control of False Alarms

In online monitoring, the thresholds $\{h_t\}_t$ should guarantee a given $\text{ARL}_0 = \mathbb{E}_{\phi_0}[t^*]$, where t^* is the detection time, as defined in Chapter 2. Following Margavio et al. [52], we compute the thresholds $\{h_t\}_t$ to guarantee a fixed false alarm probability α at each time instant t, namely

$$\mathbb{P}_{\phi_0}(\mathcal{T}_t > h_t \mid \mathcal{T}_k \leq h_k \; \forall k < t) = \alpha \quad \forall t \geq 1, \tag{4.13}$$

where $\alpha = 1/\text{ARL}_0$. This is a standard approach described more in detail in Section 3.2.3. We remark that controlling false alarms by (4.13) implies that the target ARL_0 is an upper bound on the average detection delay since the detection probability can be expected to increase after a distribution change, i.e., when $t > \tau$.

Since it is infeasible to exactly compute the conditional probabilities in (4.13), we resort to Monte Carlo simulations as in [25]. Leveraging Proposition 4.1, we simulate the construction of a QuantTree histogram on a training set $TR \sim \phi_0$ of size N by drawing its bin probabilities $(p_1, \ldots, p_K)$ from the Dirichlet distribution (4.2). Then, for each probability vector, we simulate the binary statistics $(y_{1,t}, \ldots, y_{K,t})$ in (4.9) of a stationary datastream of length $T = 5000$ by drawing them from the following multinomial distribution $\mathcal{M}$:

$$(y_{1,t}, \ldots, y_{K,t}) \sim \mathcal{M}(p_1, \ldots, p_K). \tag{4.14}$$

Then, we use these values $\{(y_{1,t}, \ldots, y_{K,t})\}_{t=1}^{5000}$ to compute the QT-EWMA statistics $\{\mathcal{T}_t\}_{t=1}^{5000}$ by (4.11)–(4.12). To compute the thresholds $\{h_t\}_t$ yielding the desired ARL_0, we repeat the procedure above 1,000,000 times, and define h_1 as the empirical $(1-\alpha)$-quantile of all the values of $\mathcal{T}_1$, where $\alpha = 1/\text{ARL}_0$. Similarly, we define h_t with $t > 1$ as the $(1-\alpha)$-quantiles of the values $\mathcal{T}_t$, using only those sequences $\{(y_{1,k}, \ldots, y_{K,k})\}_{k=1}^{t}$ whose statistics $\mathcal{T}_k$ have never exceeded any of the previous thresholds h_k for $k = 1, \ldots, t-1$. Computing the thresholds $\{h_t\}_t$ in this way guarantees that, for each time t, the empirical quantiles of $\mathcal{T}_t$ are conditioned to $\mathcal{T}_k \leq h_k \; \forall k < t$, which in turn implies (4.13), hence the target ARL_0 is preserved [52].

We compute the thresholds h_t for $t = 1, \ldots, 5000$ and then fit a polynomial in powers of $1/t$ to these values that returns h_t for a given t, as suggested in [25]. This allows to both estimate h_t for $t > 5000$ and to improve the estimates $\{h_t\}_{t=1}^{5000}$ by leveraging correlation among thresholds. In our code we provide the polynomial expressions of the thresholds main-

taining $\text{ARL}_0 = 500, 1000, 2000, 5000, 10000, 20000$, which can be very useful to control false alarms in high-throughput applications.

This procedure based on Proposition 4.1 is substantially more efficient than an equivalent procedure inspired by [3], which consists in computing the QT-EWMA statistics $\{\mathcal{T}_t\}_{t=1}^{5000}$ from synthetic univariate Gaussian datastreams, i.e. $\phi_0 = \mathcal{N}(0, 1)$, after constructing a QuantTree histogram on a synthetic training set $TR \sim \phi_0$. Directly generating the bin probabilities $(p_1, \ldots, p_K)$ from the Dirichlet distribution (4.2) and the sequences $\{(y_{1,t}, \ldots, y_{K,t})\}_{t=1}^{5000}$ from the multinomial distribution (4.14) replaces the construction of a QuantTree histogram on each training set and the computation of the binary statistics $(y_{1,t}, \ldots, y_{K,t})$ (4.9) for each synthetic datastream. In particular, we have observed a 25% reduction in the average runtime of the Montecarlo simulations.

Control over False Alarm Rates. An important consequence of setting a constant false alarm probability in (4.13) is that our thresholds can also control the false alarm rate at any time instant t. In fact, being t^* a Geometric random variable [52] with parameter α, the probability of having a false alarm before t corresponds to the following geometric sum:

$$\mathbb{P}_{\phi_0}(t^* \leq t) = \sum_{k=1}^{t} \alpha(1-\alpha)^{k-1} = 1 - (1-\alpha)^t. \tag{4.15}$$

This property enables us to assess the control of false alarms on datastreams containing a change point at τ by computing the proportion of datastreams in which $t^* \leq \tau$. This can then be compared to the target false positive rate in (4.15), which depends on the target ARL_0.

4.4 Updating the QuantTree Histogram

Here we present QT-EWMA-update (Section 4.4.1), evaluate the impact of the updating speed on the detection performance (Section 4.4.2), and discuss stopping the update to avoid including post-change samples (Section 4.4.3).

4.4.1 The QT-EWMA-update Algorithm

In QT-EWMA we model the distribution ϕ_0 by means of a QuantTree [3] histogram constructed over TR. Then, during monitoring, we use the QT-EWMA statistic $\mathcal{T}_t$ (4.12) to compare the proportion of the samples

$x_1, \dots, x_t$ falling in each bin $\mathcal{S}_j$, measured by $Z_{j,t}$ (4.11), with the estimated bin probabilities $\tilde{\pi}_j$. Being ϕ_0 unknown, the actual bin probabilities $p_j = \mathbb{P}_{\phi_0}(\mathcal{S}_j)$ are unknown but can be approximated $p_j \approx \tilde{\pi}_j$ since $\mathbb{E}[p_j] = \tilde{\pi}_j$ thanks to Proposition 4.1. However, the random vector $(p_1, \dots, p_K)$ follows the Dirichlet distribution (4.2), thus $\text{var}[p_j] \to 0$ as $N \to \infty$, which means that $\tilde{\pi}_j$ is an accurate estimate of p_j when N is sufficiently large. In contrast, when N is small, the variance is higher, and in the extreme case in which $N = K$ the histogram becomes completely uninformative because each bin contains only one training sample.

As a result, when TR is small, the values $\{\tilde{\pi}_j\}_{j=1}^K$ used to compute $\mathcal{T}_t$ yield inaccurate estimates of the actual bin probabilities $\{p_j\}_{j=1}^K$ and these harm both the offline and online change-detection performance. However, in online settings it is possible to update the model $\widehat{\phi_0}$ as new observations arrive [25,47]. To do so, we propose a new algorithm based on QT-EWMA, called *QT-EWMA-update*, where in (4.12) we replace each $\tilde{\pi}_j$ with a more accurate estimate $\hat{p}_{j,t}$ of the bin probability p_j:

$$\mathcal{T}_t = \sum_{j=1}^{K} \frac{(Z_{j,t} - \hat{p}_{j,t})^2}{\hat{p}_{j,t}} \,. \tag{4.16}$$

The estimate $\hat{p}_{j,t}$ is incrementally updated when a new observation x_t becomes available, as long as no changes are detected:

$$\hat{p}_{j,0} = \tilde{\pi}_j, \quad \hat{p}_{j,t} = (1 - \omega_t)\hat{p}_{j,t-1} + \omega_t y_{j,t} \quad t > 0, \tag{4.17}$$

where $y_{j,t} = \mathbb{1}(x_t \in \mathcal{S}_j)$, $\omega_t = 1/\beta(N + t)$ is a parameter representing the *updating speed*, i.e., the weight of the latest sample in the average, and $\beta \geq 1$ is a tuning parameter.

We remark that all the quantities involved in the QT-EWMA-update statistic (4.16), including $\hat{p}_{j,t}$ defined in (4.17), are computed from a QuantTree histogram. Thus, thanks to Proposition 4.1, we can compute the thresholds controlling the ARL_0 by the same Montecarlo procedure presented in Section 4.3.

4.4.2 Setting the Updating Speed

The parameter β allows tuning the updating speed ω_t of QT-EWMA-update, which has a crucial impact on the detection performance. Setting $\beta = 1$ guarantees that $\hat{p}_{j,t} \to p_j$ when $t \to \infty$, as long as $x_t \sim \phi_0$, since (4.17) becomes the cumulative average of $y_{j,t}$, whose expected value is $\mathbb{E}_{\phi_0}[y_{j,t}] = p_j$ by definition. However, samples $x_t \sim \phi_1$ acquired after the change τ introduce a severe bias that harms the detection performance. Therefore, we

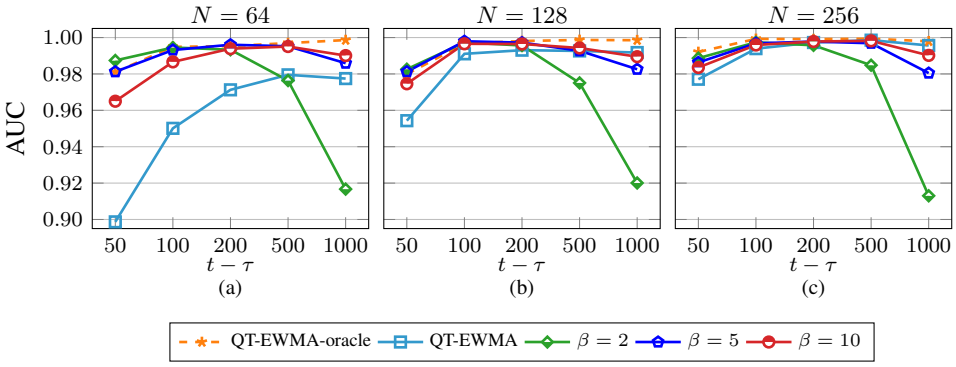


Figure 4.2: *Detection power of QT-EWMA-update ($\beta = 2, 5, 10$) compared to QT-EWMA ($\beta = \infty$) and QT-EWMA-oracle over univariate datastreams with $\phi_0 = \mathcal{U}(0, 1)$, $\phi_1 = \mathcal{N}(0.5, 0.5)$, and $\tau = 1000$, setting $N = 64, 128, 256$. In particular, we compute the AUC given by the statistic $\mathcal{T}_t$ at different times $t > \tau$. QT-EWMA-update outperforms QT-EWMA right after τ, especially when $N = 64$, but its performance gets worse over time since the update includes $x_t \sim \phi_1$ when $t > \tau$. This is evident when the updating speed is high ($\beta = 2$).*

propose to set $\beta > 1$, as this reduces the contribution of the most recent samples when updating $\hat{p}_{j,t}$. Setting $\beta > 1$ slightly biases the estimate $\hat{p}_{j,t}$ in stationary conditions, but turns out to be very beneficial in terms of detection delay, as shown in our experiments. We remark that QT-EWMA corresponds to the case $\beta = \infty$ since $\hat{p}_{j,t} \equiv \tilde{\pi}_j$.

To illustrate the trade-off regulated by β, we perform a simple experiment. First, we generate 1000 univariate training sets from a uniform distribution $\phi_0 = \mathcal{U}(0, 1)$. Then, we generate 500 stationary univariate datastreams of length $T = 2000$ from ϕ_0, and 500 datastreams with initial distribution ϕ_0 containing a change point at $\tau = 1000$. The post-change distribution is a Gaussian $\phi_1 = \mathcal{N}(0.5, 0.5)$.

We monitor each datastream by the QT-EWMA-update algorithm setting $\beta = 2, 5, 10$, and QT-EWMA (corresponding to $\beta = \infty$). We measure the detection power of these algorithms by the *Area Under the ROC Curve (AUC)* of the statistics $\mathcal{T}_t$ computed at different times t after the change such that $t - \tau = 50, 100, 200, 500, 1000$. Since in this case we set $\phi_0 = \mathcal{U}(0, 1)$, we can easily compute the bin probabilities p_j of a given QuantTree histogram, so we also apply the "oracle" QT-EWMA algorithm, which uses p_j instead of $\tilde{\pi}_j$ in (4.11) and (4.12) and is never updated. This solution cannot be adopted in general, since ϕ_0 is unknown, thus it represents an upper bound in terms of detection power.

In Figure 4.2 we show the results of this experiment, using training set size $N = 64, 128, 256$. We observe that the QT-EWMA algorithm steadily

improves its detection power over time thanks to the larger amount of samples drawn from ϕ_1, which increase the statistical evidence of the change. We also observe that the oracle QT-EWMA yields much better results than QT-EWMA when the training set is small, while the gap reduces when N is sufficiently large, as $\tilde{\pi}_j$ becomes an accurate estimate of p_j.

In all cases QT-EWMA-update yields a higher detection power compared to QT-EWMA right after τ, but shows a substantial decrease of the AUC over time due to the fact that $\hat{p}_{j,t}$ is updated using samples from ϕ_1. Using a larger β slows down the decrease, at the cost of slightly reducing the detection power right after τ. This experiment suggests that setting $\beta = 5$ yields the best detection performance.

4.4.3 Stopping the Update

All in all, QT-EWMA-update outperforms QT-EWMA when the training set is small ($N = 64, 128$), but the update yields only a marginal advantage when $N = 256$ because $\tilde{\pi}_j$ is already a good estimate of p_j. Hence, when a large training set is available, QT-EWMA is preferable since it avoids the risk of updating the model using samples from the post-change distribution. QT-EWMA-update is preferred when N is small, but the update of $\widehat{\phi}_0$ should stop as soon as a sufficient number S of samples have been acquired without detecting a change, i.e., when $N + t = S$. This allows to reduce the risk of updating using samples $x_t \sim \phi_1$ once the estimated bin probabilities $\hat{p}_{j,t}$ are sufficiently accurate. Since stopping the update does not change the the fact that the distribution of the statistic is independent from ϕ_0 and d, which is guaranteed by the properties of QuantTree [3], we compute thresholds controlling the ARL_0 for given values of β and S using the same Monte Carlo scheme illustrated in Section 4.3. The only difference with respect to QT-EWMA-update is that we update the bin probabilities $\hat{p}_{j,t}$ by (4.17) only for $t < S - N$, using $\hat{p}_{j,S-N}$ when $t \geq S - N$.

4.5 Baselines controlling the ARL_0

Unfortunately, the vast majority of online change-detection methods for multivariate datastreams does not control the ARL_0, or does so only when ϕ_0 belongs to a known family of distributions [47] or is a known arbitrary distribution [50, 51]. These are strong assumptions that do not hold in general, especially when ϕ_0 represents the distribution of real-world data. Since the lack of methods that effectively control false alarms makes it difficult to fairly assess the detection performance of our solutions, we de-

sign two simple yet effective strategies to adapt one-shot detectors to online change detection controlling the ARL_0. In particular, we focus on algorithms that operate in a batch-wise (Section 4.5.1), and element-wise fashion (Section 4.5.2). These strategies allow us to define baseline methods based on QuantTree [3] and Semi-Parametric Log-Likelihood [29]. In this section we also present Scan-B [34], an online change-detection method based on the MMD statistic that, to the best of our knowledge, is the only one in which the target ARL_0 can be set independently on ϕ_0.

4.5.1 Datastream Monitoring by Batch-wise Detectors

Several change-detection algorithms process the datastream $x_1, x_2, \ldots$ by separately analyzing non-overlapping batches W_t of ν samples:

$$W_t = [x_{(t-1)\nu+1}, \ldots, x_{t\nu}]. \tag{4.18}$$

In particular, these algorithms compute for each incoming batch W_t a test statistic $\mathcal{T}(W_t)$ based on a model $\widehat{\phi}_0$ fit over TR. For example, in QuantTree [3] the model $\widehat{\phi}_0$ is a histogram, while Semi-Parametric Log-Likelihood (SPLL) [29] employs a Gaussian mixture model $\widehat{\phi}_0$. These algorithms detect a change as soon as $\mathcal{T}(W_t) > h^\nu$, where the threshold h^ν does not depend on t and is defined to control the false alarm probability over each batch W_t.

In what follows we show how to employ these batch-wise monitoring schemes for online change detection maintaining the target ARL_0 by setting the correct threshold h^ν. This result is based on the following proposition:

Proposition 4.2. *Let W_t be any batch of ν samples drawn from ϕ_0 and let the detection threshold h^ν be such that*

$$\mathbb{P}_{\phi_0}(\mathcal{T}(W_t) > h^\nu) = \alpha. \tag{4.19}$$

Then, the monitoring scheme $\mathcal{T}(W_t) > h^\nu$ yields $\text{ARL}_0 \geq \nu/\alpha$.

Proof. Let t_b^* the first time instant such that $\mathcal{T}(W_{t_b^*}) > h$ and let us compute $\text{ARL}_0 = \mathbb{E}[t_b^*]$. To this purpose, we follow a strategy similar to that in Section 4.2. At first we observe that since the batches does not overlap, the variables $\{\mathcal{T}(W_t)\}$ are independent if we condition w.r.t. the specific training set realization (thus the model used to compute $\mathcal{T}$). Therefore, we obtain that:

$$\mathbb{P}(\mathcal{T}(W_t) > h \mid TR, \mathcal{T}(W_k) \leq h \; \forall k < t) = \mathbb{P}(\mathcal{T}(W_t) > h \mid TR). \tag{4.20}$$

Let us define the random variable $p = \mathbb{P}(\mathcal{T}(W) > h \mid TR)$, where W is a batch of ν samples drawn from ϕ_0. Following [52], the random variable t_b^* is distributed as a geometric random variable w.r.t. the conditional probability $\mathbb{P}(\cdot|TR)$, and its expected value is

$$\mathbb{E}[t_b^* \mid TR] = \frac{1}{p}. \tag{4.21}$$

To compute the ARL_0 we only have to evaluate the expectation of $1/p$ w.r.t. to the training set realizations since the law of total expectation implies that

$$\mathbb{E}[t_b^*] = \mathbb{E}[\mathbb{E}[t_b^* \mid TR]] = \mathbb{E}\left[\frac{1}{p}\right]. \tag{4.22}$$

We observe that Jensen's inequality implies that

$$\mathbb{E}[t_b^*] = \mathbb{E}\left[\frac{1}{p}\right] \geq \frac{1}{\mathbb{E}[p]}, \tag{4.23}$$

since the function $1/p$ is convex for $p > 0$. Finally, we have to compute $\mathbb{E}[p]$. To this purpose we rewrite

$$p = \mathbb{P}(\mathcal{T}(W) > h^\nu \mid TR) = \mathbb{E}[\mathbb{1}(\mathcal{T}(W) > h^\nu) \mid TR], \tag{4.24}$$

where $\mathbb{1}$ denotes the indicator function. Then:

$$\begin{aligned}
\mathbb{E}[p] &= \mathbb{E}[\mathbb{E}[\mathbb{1}(\mathcal{T}(W) > h^\nu)|TR]] = && (4.25)\\
&= \mathbb{E}[\mathbb{1}(\{\mathcal{T}(W) > h^\nu\})] = && (4.26)\\
&= \mathbb{P}(\mathcal{T}(W_t) > h^\nu) = \alpha, && (4.27)
\end{aligned}$$

where the equality between (4.25) and (4.26) is due to the law of total expectation. Combining (4.23) and (4.27) we obtain that

$$\mathbb{E}[t_b^*] \geq \frac{1}{\alpha}. \tag{4.28}$$

To obtain the thesis we observe that, since the monitoring is performed in a batch-wise manner, change detected after the t_b^* batch translates in a detection made after $\nu \cdot t_b^*$ samples of the datastream, so $\text{ARL}_0 \geq \nu/\alpha$. □

Hence, setting $\alpha = \nu/\text{ARL}_0$, we obtain a conservative online change-detection algorithm, guaranteeing that the ARL_0 is greater than or equal to the target ARL_0. A slightly different result holds when the threshold h^ν is computed from the training set, e.g. through a bootstrap procedure. The following Proposition shows that, in this case, setting $\alpha = \nu/\text{ARL}_0$ enables achieving the target ARL_0 exactly.

Proposition 4.3. *Let W_t be any batch of ν samples drawn from ϕ_0 and let the detection threshold h^ν be such that*

$$\mathbb{P}_{\phi_0}(\mathcal{T}(W_t) > h^\nu \mid TR) = \alpha, \tag{4.29}$$

Then, the monitoring scheme $\mathcal{T}(W_t) > h^\nu$ yields ARL$_0 = \nu/\alpha$.

Proof. Following the same strategy we pursued to prove Proposition 4.2, we have that the random variable $p = \mathbb{P}(\mathcal{T}(W_t) > h^\nu | TR)$ is a constant equal to α. Therefore, the equality holds in (4.23), from which we derive $\text{ARL}_0 = \nu/\alpha$. □

Leveraging the results presented above, we adapt two well-known batch-wise change-detection methods to monitor datastreams online while controlling the ARL$_0$: QuantTree [3] and Semi-Parametric Log-Likelihood [29]. Thanks to the properties of QuantTree [3] it is possible to set h^ν for (4.19) to hold for any α, independently from the initial distribution ϕ_0. Hence, Proposition 4.2 guarantees that it is possible to set a lower bound on the ARL$_0$ in the original QuantTree monitoring scheme [3].

In contrast, the distribution of the SPLL statistic depends on ϕ_0, so the thresholds to set the false positive rate have to be computed by bootstrap over the training set TR. In particular, we use a portion of TR to fit the Gaussian mixture model $\widehat{\phi}_0$, and the rest to form independent batches for bootstrap. For this reason, this algorithm requires a relatively large training set to maintain a target ARL$_0$. In this case, it is Proposition 4.3 guaranteeing the control over the ARL$_0$, since the false positive probability is conditioned on the training set realization. Therefore, when h^ν is set to guarantee $\mathbb{P}_{\phi_0}(\mathcal{T}^\nu(W) > h^\nu | TR) = \nu/\text{ARL}_0$, the SPLL monitoring scheme maintains the target ARL$_0$.

4.5.2 Datastream Monitoring by Element-wise Detectors

As pointed out in Section 3.2.1, a popular approach to change detection in multivariate datastreams consists in reducing the data dimension, constructing a univariate datastream that can be monitored by standard change-detection algorithms. Here we consider a dimensionality reduction method based on SPLL [29]. In particular, we reduce the dimension of each incoming sample x_t by computing the log-likelihood $-\log(\widehat{\phi}_0(x_t))$, where $\widehat{\phi}$ is a Gaussian mixture model fit on the entire TR. Then, we monitor the resulting univariate sequence by a nonparametric online CPM [25] leveraging the Lepage test statistic [20]. This algorithm, which we call SPLL-CPM, maintains the desired ARL$_0$ thanks to the CPM, which controls the ARL$_0$ on any univariate datastream [25].

Table 4.1: *Computational complexity and memory requirement to process each sample x_t of QT-EWMA and QT-EWMA-update compared to the other methods, depending on the configuration.*

algorithm	complexity	memory
QT-EWMA	$\mathcal{O}(K)$	K
QT-EWMA-update	$\mathcal{O}(K)$	$2K$
QuantTree [3]	$\mathcal{O}(K)$	K
SPLL [29]	$\mathcal{O}(md)$	1
SPLL-CPM	$\mathcal{O}(md + w \log w)$	w
Scan-B [34]	$\mathcal{O}(nBd)$	$(n+1)Bd$

4.6 Computational Complexity

In this section we analyze the computational complexity and memory requirements of QT-EWMA and QT-EWMA-update, since efficiency is crucial in online change detection [25]. We perform the same analysis on the modified one-shot detectors QuantTree [3], SPLL [29] and SPLL-CPM (discussed in Section 4.5), and on Scan-B [34] (presented in Section 3.2.2). The results are summarized in Table 4.1.

QT-EWMA, QT-EWMA-update and QuantTree. These algorithms are extremely efficient and require a constant amount of memory over time that does not depend on the data dimension d. During monitoring, these three algorithms find the bin of the QuantTree histogram where each incoming sample x_t falls, resulting in $\mathcal{O}(K)$ operations [3], where K is the number of bins. Then, QT-EWMA and QT-EWMA-update compute the test statistics (4.9), (4.11), (4.12) and these operations have a constant cost that falls within $\mathcal{O}(K)$. QT-EWMA-update also updates the bin probabilities of the QuantTree histogram by (4.17), requiring K additional operations that fall within $\mathcal{O}(K)$. The QuantTree algorithm instead computes the Pearson statistic at the end of each batch, and this does not increase the order of computational complexity either, resulting in $\mathcal{O}(K)$ operations as in QT-EWMA and QT-EWMA-update.

In terms of memory requirement, QT-EWMA has to store only the K values of the statistics $Z_{j,t-1}, j = 1, \dots, K$ to update them (4.11) at each new sample x_t. QT-EWMA-update stores also the K estimated bin probabilities $\hat{p}_{j,t-1}, j = 1, \dots, K$, hence it requires to store $2K$ values in total. Similarly to QT-EWMA, QuantTree has to store only the proportions of points from the current batch that belong to each of the K bins to compute the Pearson statistic.

SPLL and SPLL-CPM. Both these algorithms compute the log-likelihood of each sample with respect to each of the m components of the Gaussian mixture model $\widehat{\phi}_0$ fit on TR. In the modified batch-wise monitoring scheme SPLL, the log-likelihood computation over an entire batch requires $\mathcal{O}(md)$ operations per sample [29], since the log-likelihood can be computed incrementally. Hence, only 1 value has to be stored in memory, namely the log-likelihood computed in the previous step. In contrast, the SPLL-CPM algorithm leverages the less efficient CPM framework [25] to monitor the log-likelihood $-\log(\widehat{\phi}_0(x_t))$ of each incoming observation x_t. In particular, the Lepage test statistic [20] used in the CPM requires to sort the whole log-likelihood sequence obtained until time t, resulting in $\mathcal{O}(t \log t)$ operations on top of the $\mathcal{O}(md)$ operations required to compute the log-likelihood $-\log(\widehat{\phi}_0(x_t))$. In this case, all the t values of the log-likelihood sequence have to be processed and stored at each time t, thus the computational complexity and memory requirement steadily increase over time. Since this is not desirable in online change detection, the ranks of older observations can be discretized through a histogram, and used to approximate the Lepage statistic [25], so only the most recent $w = 500$ samples are processed and stored [25].

Scan-B. The Scan-B algorithm [34] operates in a sliding-window fashion with window size B, using n reference windows of size B sampled from the training set. For each incoming sample x_t, Scan-B updates n Gram matrices by computing B times the MMD statistic, resulting in $\mathcal{O}(nBd)$ operations [51]. The n reference windows from the training set and the current window have to be stored, yielding $(n + 1)Bd$ values in memory [51] since each sample is d-dimensional. Hence the computational cost and memory requirements of Scan-B increase with the data dimension.

4.7 Experiments and Discussion

The goal of our experiments is to empirically demonstrate the advantages of QT-EWMA in terms of control of false alarms and detection power, both on synthetic and real-world data. Moreover, we want to show that QT-EWMA-update has the same control over false alarms as QT-EWMA, but better detection power when the training set is small, and that stopping the update after acquiring a sufficient number of samples can further improve the detection performance.

4.7.1 Considered Datasets

We simulate Gaussian datastreams in different dimensions $d = 4, 16, 64$, choosing an initial Gaussian distribution ϕ_0 with random mean and covariance matrix, and roto-translating ϕ_0 to obtain the post-change distribution $\phi_1 = \phi_0(Q \cdot + v)$. We randomly select the roto-translation parameters Q and v using the CCM framework [42] to guarantee a target *symmetric Kullback-Leibler divergence* [70] $\text{sKL}(\phi_0, \phi_1) = 0.5, 1, 1.5, 2, 2.5, 3$, which is useful to compare detection performances in different dimensions [36].

We also perform our experiments on seven well-known multivariate datasets: Credit Card Fraud Detection ("credit", $d = 28$) from [71], Sensorless Drive Diagnosis ("sensorless", $d = 48$), MiniBooNE particle identification ("particle", $d = 50$), Physicochemical Properties of Protein Ternary Structure ("protein", $d = 9$), El Niño Southern Oscillation ("niño", $d = 5$), and two of the Forest Covertype datasets ("spruce" and "lodgepole", $d = 10$) from the UCI Machine Learning Repository [72]. As in [3], we standardize the datasets and jitter the instances of "sensorless", "particle", "spruce" and "lodgepole" by adding uncorrelated Gaussian noise to avoid repeated values, which harm the construction of QuantTree histograms. We use noise with variance 0.001 for "sensorless" and "particle", and 0.1 for "spruce" and "lodgepole". We randomly sample datastreams from these datasets, whose distribution can be considered stationary, and we introduce a change by shifting the post-change data points by a random vector drawn from a standard d-dimensional Gaussian distribution, scaled by the total variance of the dataset, as in [3, 37]. For brevity, here we report only the average results over the seven datasets (denoted by "UCI+credit"), leaving the results over each individual dataset in Appendix A.

We also test our method on the recently published INSECTS dataset [73] ($d = 33$), which contains features describing the wing-beat frequency of different species of flying insects, extracted from high-dimensional signals acquired by optical sensors. This dataset is meant as a classification benchmark in datastreams affected by concept drift. A concept refer to data acquired under different environmental conditions affecting the insects' behavior. The dataset contains six concepts referring to different distributions. We assemble data from different concepts to form datastreams that include 30 realistic changes: we start sampling observations from one concept (ϕ_0) and switch to another (ϕ_1) introducing a change point.

In all our experiments we make sure that the monitored datastreams do not contain any sample from the training set. To do so, we generate synthetic training and testing data from different seeds, and we sample real-

world datastreams after removing TR from the datasets [71–73].

4.7.2 Figures of Merit

Empirical ARL$_0$. To assess whether QT-EWMA and the other considered methods maintain the target ARL$_0$, we compute the empirical ARL$_0$ as the average time before raising a false alarm. To this purpose, we run the considered methods on 1000 datastreams drawn from ϕ_0, setting the target ARL$_0$ $= 500, 1000, 2000, 5000$. In this experiment, we consider 5000 datastreams of length $T = 6 \cdot$ ARL$_0$ to make sure that, with high probability, we have a detection in each datastream. In fact, the detection time t^* of our method under ϕ_0 is a Geometric random variable with parameter $\alpha = 1/$ARL$_0$ by construction, thus (4.15) indicates that the probability of having a false alarm before T is $\mathbb{P}_{\phi_0}(t^* \leq T) \approx 0.9975$.

Detection delay. We evaluate the detection performance of QT-EWMA and the other methods by their detection delay, i.e. ARL$_1$ $= \mathbb{E}[t^* - \tau]$, where the expectation is taken assuming that a change point τ is present [11]. We run the methods configured with ARL$_0$ $= 500$, 1000, 2000, 5000 on 1000 datastreams of length $T = 10000$, each containing a change point at $\tau = 500$. We compute the empirical ARL$_1$ as the average difference $t^* - \tau$, excluding false alarms.

False alarm rate. To assess whether the considered methods yield the desired false alarm probability, we compute the percentage of false alarms over the datastreams used to evaluate the detection delay, i.e., those in which a detection occurs at $t^* < \tau$. Setting the the target ARL$_0$ $= 500$, 1000, 2000, 5000 should yield a false alarm in, respectively, 63%, 39%, 22% and 9.5% of the datastreams according to (4.15).

4.7.3 Results and Discussion

Empirical ARL$_0$. The comparison between the empirical and target ARL$_0$ on simulated Gaussian datastreams are reported in Figure 4.3 (a,c,e,g) for $d = 4$, in Figure 4.4 (a,c,e,g) for $d = 16$, and in Figure 4.5 (a,c,e,g) for $d = 64$. These plots show that QT-EWMA, QT-EWMA-update and SPLL-CPM control the ARL$_0$ very accurately, independently from the data dimension d and the training set size N. This can be seen from lines that are close to the diagonal (note that axis units are different). The empirical ARL$_0$ of QuantTree is higher than the target, as we expected from Proposition 4.2, while Scan-B cannot maintain high target ARL$_0$. Figure 4.6 (a,b,c,d)

and Figure 4.7 (a,b,c,d) show that we obtain the same results on datastreams sampled from, respectively, the UCI+credit and INSECTS datasets, confirming the nonparametric nature of QuantTree and the limitations of Scan-B. In fact, Scan-B cannot control the ARL_0 accurately because its thresholds are defined by an asymptotic approximation that depends on the window size B [34]. In particular, this approximation is more accurate for higher target ARL_0 when B is large, which increases the computational complexity and memory usage (Table 4.1). Thus, Scan-B with a fixed window size B can only maintain a low target ARL_0. Despite Proposition 4.3, we observe that also SPLL cannot maintain the target ARL_0 accurately, and this is due to inaccurate estimate of its thresholds, which are computed by bootstrap.

Detection delay vs false alarms. We plot the average detection delay against the percentage of false alarms setting target $\text{ARL}_0 = 500, 1000, 2000, 5000$, to assess the trade-off between these two quantities. The performance of the considered methods on simulated Gaussian datastreams with a change point at $\tau = 500$ are shown in Figure 4.3 (b,d,f,h) for $d = 4$, in Figure 4.4 (b,d,f,h) for $d = 16$, and in Figure 4.5 (b,d,f,h) for $d = 64$.

In terms of detection delay, QT-EWMA-update is the best nonparametric method when the training set is small ($N = 64, 128, 256$), being outperformed only by SPLL-CPM, which is expected since its parametric assumptions are met (ϕ_0 is a Gaussian). When the training set is large ($N = 4096$) both SPLL and Scan-B outperform QT-EWMA on Gaussian datastreams, which can also be expected because the parametric assumptions of SPLL are met, and statistics defined on histograms (such as that of QT-EWMA) are known to be less powerful than those based on MMD (such as that of Scan-B), as they perceive only changes affecting bin probabilities, and are for instance totally blind to distribution changes inside each bin. However, our experiments show that both QT-EWMA and QT-EWMA-update yield higher detection power than Scan-B when the training set is small. All methods achieve higher detection delays as d increases, which is also expected due to detectability loss [36].

On the UCI+credit datasets (Figure 4.6 (b,d,f,h)), QT-EWMA-update is the second-best method (slightly outperformed by SPLL-CPM) when the training set is small, and QT-EWMA is clearly the best method when $N = 4096$. On the INSECTS dataset (Figure 4.7 (b,d,f,h)), QT-EWMA-update achieves the best detection delays when $N = 64, 128, 256$, and QT-EWMA approaches the performance of Scan-B when the training set is large ($N = 4096$). Moreover, QT-EWMA and QT-EWMA-update substan-

tially outperform SPLL and SPLL-CPM, meaning that the nonparametric QuantTree can model the distribution of the INSECTS datasets much better than the Gaussian mixture model used in SPLL and SPLL-CPM.

Remarkably, QT-EWMA and QT-EWMA-update consistently outperform QuantTree in all the considered scenarios, which indicates that our sequential statistics are more powerful than the original QuantTree statistic (designed for batch-wise monitoring) when operating online.

As expected, the detection power of the methods based on QuantTree and Scan-B increases significantly with the training set size N. In contrast, SPLL-CPM has similar performances with different values of N since the Gaussian mixture model $\widehat{\phi}_0$ fit on TR is sufficiently accurate even when N is small, and the CPM does not require a training set [25]. Most remarkably, QT-EWMA and QT-EWMA-update outperform Scan-B when N is small, meaning that in these settings our online statistics based on QuantTree histograms have higher detection power than comparing a sliding datastream window with reference data by the MMD statistic. Finally, we observe that QT-EWMA-update substantially outperforms QT-EWMA when the training set is extremely small ($N = 64, 128$), while it yields only a marginal improvement when $N = 256$, meaning that when $N \geq 256$ the values $\tilde{\pi}_j$ are sufficiently good estimates of p_j.

In terms of false alarm rate, QT-EWMA, QT-EWMA-update and SPLL-CPM approach the target values computed by (4.15). In contrast, QuantTree and SPLL have, respectively, fewer and more false alarms than expected in all the considered monitoring scenarios. This can be explained by the fact that the empirical ARL_0 of QuantTree is higher than the target due to Proposition 4.2, while the empirical ARL_0 of SPLL is lower than the target due to inaccurate threshold estimation. The false alarms of Scan-B, instead, exhibit a completely different behavior, which also depends on the data distribution since its thresholds do not yield a constant false alarm probability.

Since on Gaussian data we can control the change magnitude by setting $\text{sKL}(\phi_0, \phi_1)$ [42], in Figure 4.8 we show the detection delays of the considered methods depending on the change magnitude $\text{sKL}(\phi_0, \phi_1) = 0.5, 1, 1.5, 2, 2.5, 3$. In this experiment we use large training sets ($N = 4096$) and set target $\text{ARL}_0 = 1000$, which is maintained by all methods (see Figure 4.3(d)), to make the comparison fair. As expected, the detection delays of all methods decrease when the change magnitude increases, i.e., when the distribution change becomes more evident. All methods achieve higher detection delays as d increases, which is also expected due to *detectability loss* [36]. In particular, in Figure 4.8 (a) we observe that when

$d = 4$ QT-EWMA is on par with the best-performing methods, i.e., the parametric SPLL and SPLL-CPM and the nonparametric Scan-B. In contrast, in Figure 4.8 (b,c) we can see that SPLL and Scan-B have lower detection delays than QT-EWMA when $d = 16, 64$, confirming that statistics based on verified parametric assumptions (SPLL) and on Maximum Mean Discrepancy (Scan-B) are very powerful and yield change-detection algorithms that are more robust to detectability loss.

Stopping the update. As discussed in Section 4.4.2, we can stop the update of the QuantTree histogram after acquiring a sufficient amount of data. This mitigates the problem of updating the estimated bin probabilities $\hat{p}_{j,t}$ when $t > \tau$, i.e., when $x_t \sim \phi_1$. To demonstrate this, we measure the detection delay of QT-EWMA-update where we stop the update after analyzing S samples, i.e., when $N + t = S$. In this experiment, we compare QT-EWMA-update stopping at $S = 512, 1024$ against QT-EWMA and QT-EWMA-update. Figure 4.9 shows the detection delays achieved on Gaussian datastreams with $d = 16$ and length $T = 10000$ containing a change point at $\tau = 250, 500, 750, 1000$. We consider different training set sizes $N = 64, 128, 256$, and set target $\text{ARL}_0 = 2000$. As observed in the previous experiments, when $N = 64, 128$, QT-EWMA-update performs better than QT-EWMA, while the two algorithms have similar results when $N = 256$, confirming that updating the histogram is not necessary when N is sufficiently large. In all cases, the detection delay of QT-EWMA-update decreases when the change occurs later in the datastream since more samples $x_t \sim \phi_0$ are used to update $\hat{p}_{j,t}$.

The fact that the detection delays of QT-EWMA-update with stopping rule are lower than those of QT-EWMA-update confirms that reducing the amount of samples $x_t \sim \phi_1$ used to update $\hat{p}_{j,t}$ is beneficial. The detection performance of QT-EWMA-update with stopping rule improves when the change point τ occurs later in the datastream, unless the change occurs after having stopped the update, i.e. when $\tau > S - N$. In Figure 4.9(a,b) we observe that setting $S = 512$ yields similar detection delays when $\tau = 500, 750, 1000$ since these changes occur after having stopped the update, thus all the $S - N$ samples used to update $\hat{p}_{j,t}$ are drawn from ϕ_0. In contrast, when $\tau = 250$ the detection delay is higher since samples from ϕ_1 might bias the estimates $\hat{p}_{j,t}$. We observe the same effect in Figure 4.9(c) for $S = 1024$, where the detection delays for $\tau = 750, 1000$ are very similar and lower than those obtained when $\tau = 250, 500$.

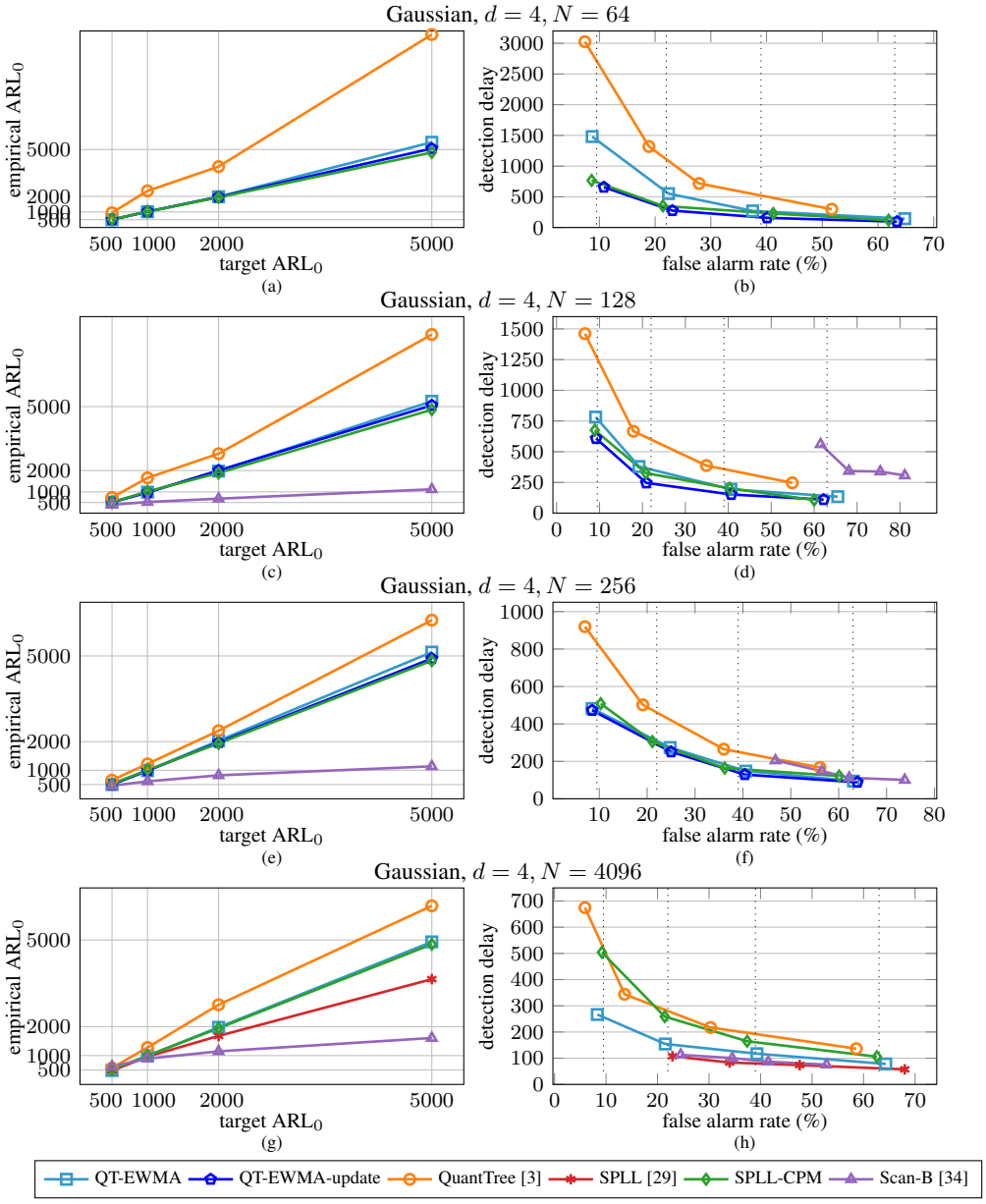


Figure 4.3: *Experimental results over Gaussian datastreams ($d = 4$).* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best-performing methods are QT-EWMA-update and SPLL-CPM when using small training sets ($N = 64, 128, 256$) and SPLL and Scan-B when using large training sets ($N = 4096$). We observe that QT-EWMA-update clearly outperforms QT-EWMA when $N = 64, 128$, and that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (4.15)*, which are represented in the plots by vertical dotted lines.*

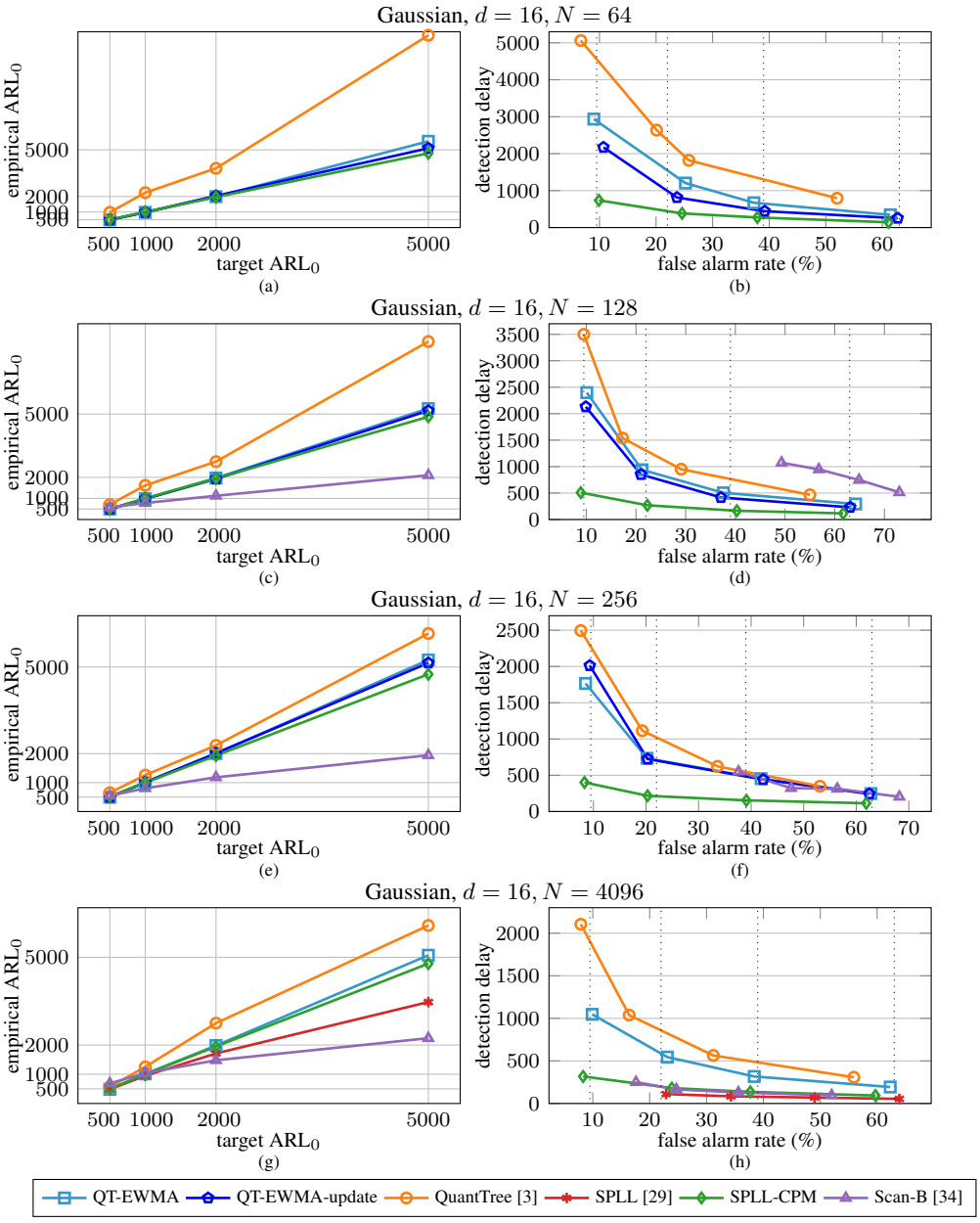


Figure 4.4: *Experimental results over Gaussian datastreams ($d = 16$).* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while the other methods do not maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best-performing method is SPLL-CPM when using small training sets ($N = 64, 128, 256$) and SPLL when using large training sets ($N = 4096$). We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (4.15)*, which are represented by the vertical dotted lines.*

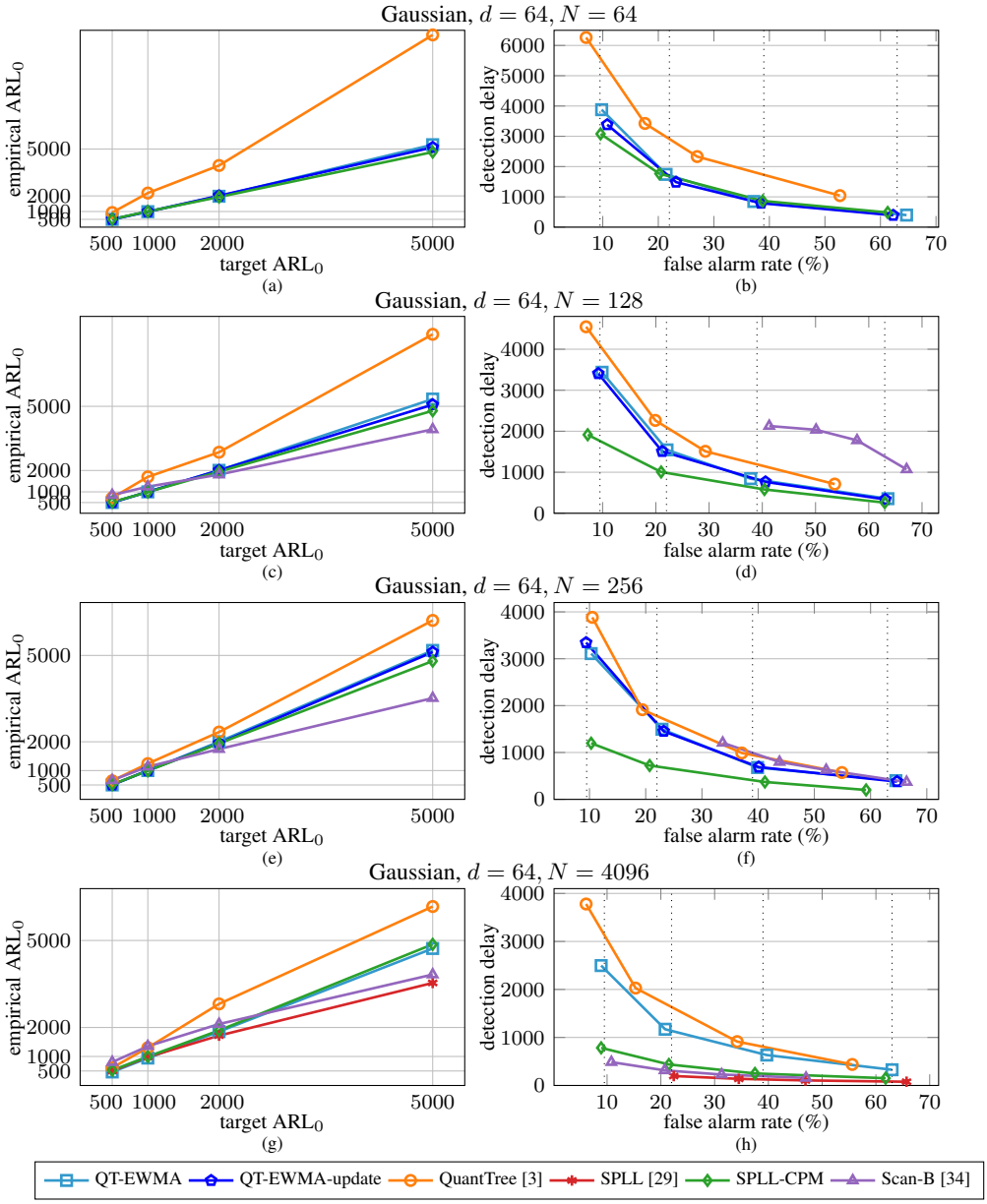


Figure 4.5: *Experimental results over Gaussian datastreams ($d = 64$).* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best-performing method is SPLL-CPM when using small training sets ($N = 64, 128, 256$) and SPLL when using large training sets ($N = 4096$). We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (4.15)*, which are represented by vertical dotted lines.*

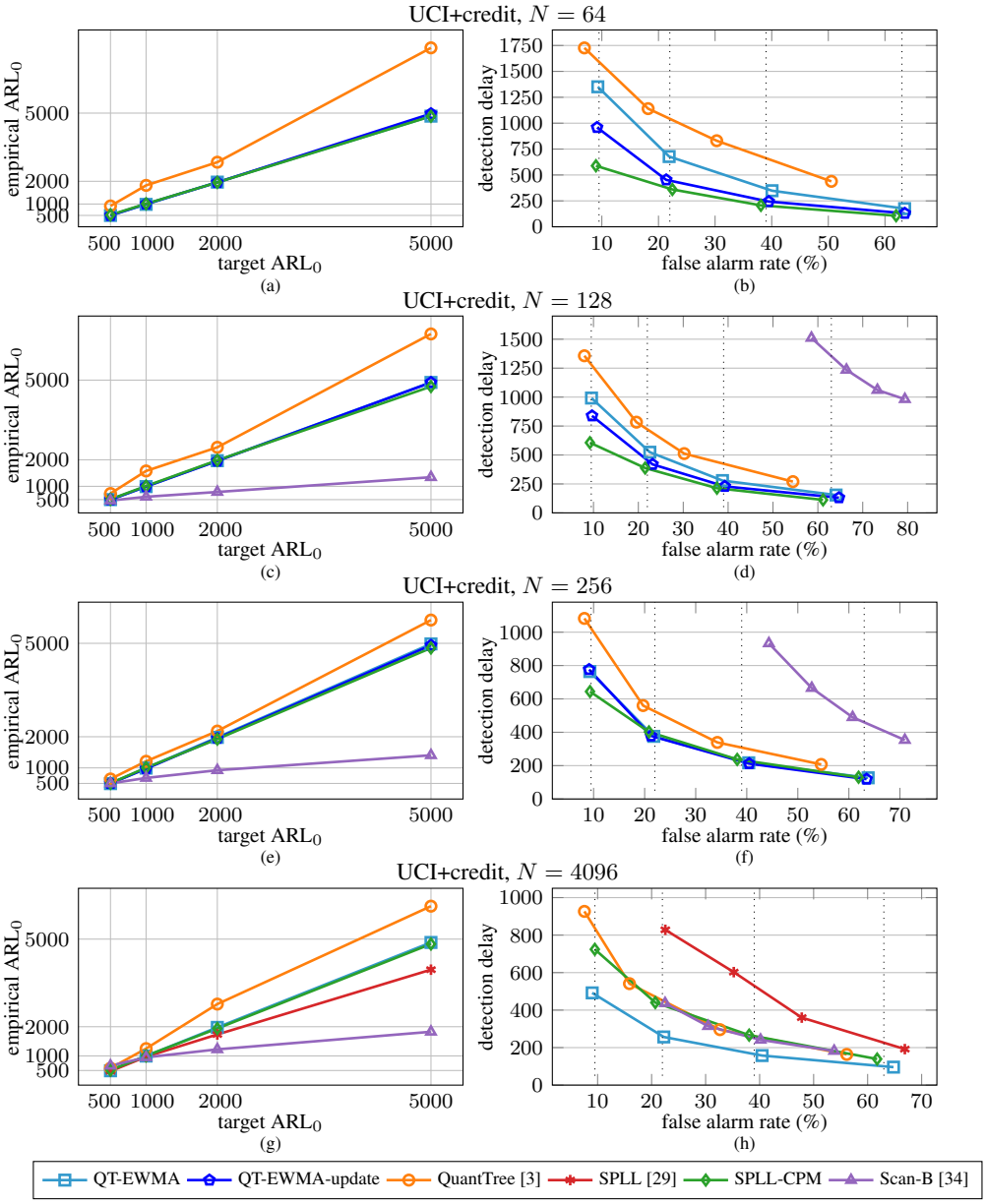


Figure 4.6: *Experimental results averaged over the UCI+credit datasets [71, 72].* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best-performing methods are QT-EWMA-update and SPLL-CPM when using small training sets ($N = 64, 128, 256$) and QT-EWMA when using large training sets ($N = 4096$). We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (4.15)*, which are represented by vertical dotted lines.*

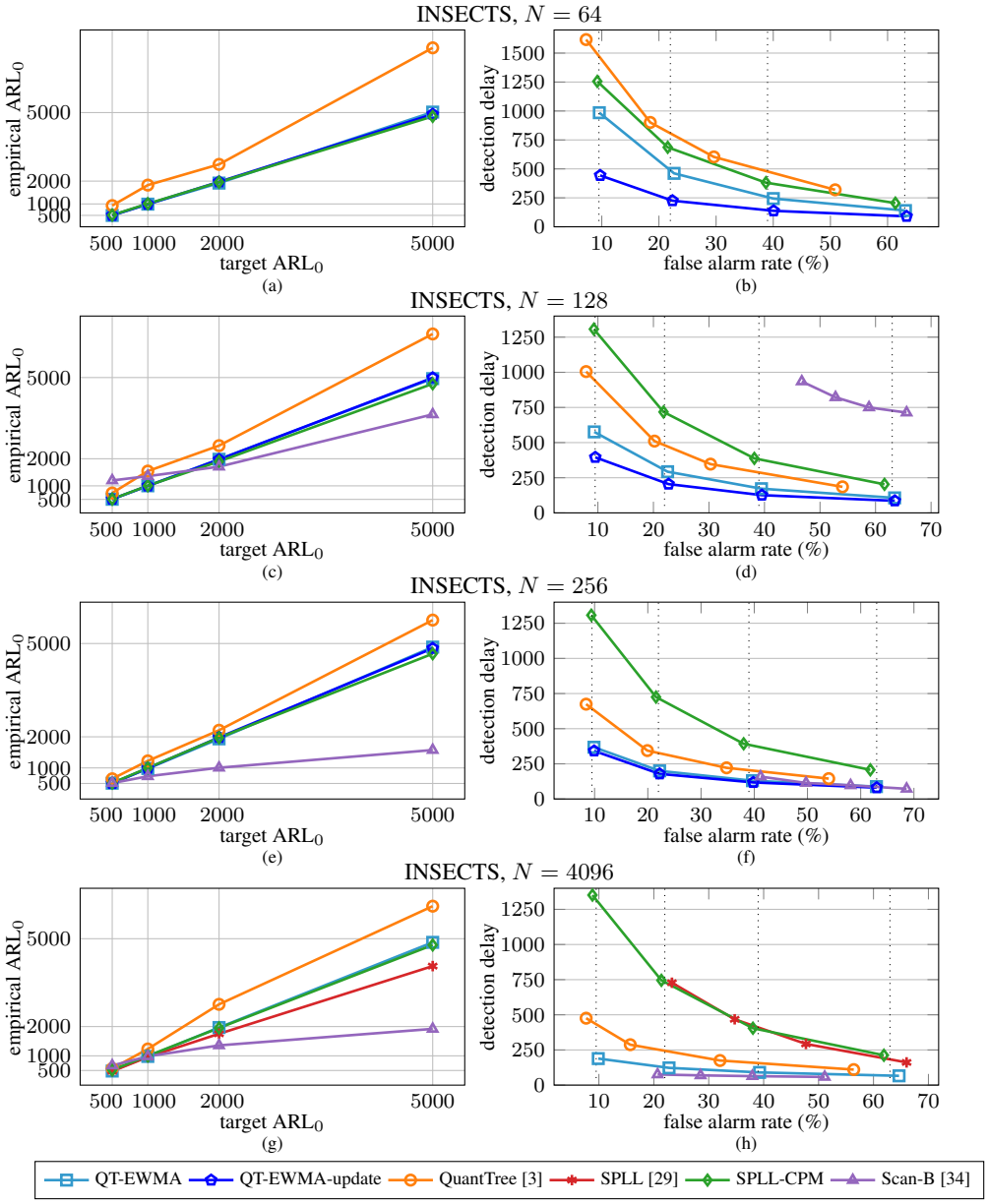


Figure 4.7: *Experimental results averaged over the INSECTS dataset [73]. (a,c,e,g) show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best-performing method is QT-EWMA-update when using small training sets ($N = 64, 128, 256$) and Scan-B when using large training sets ($N = 4096$). We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (4.15)*, which are represented by vertical dotted lines.*

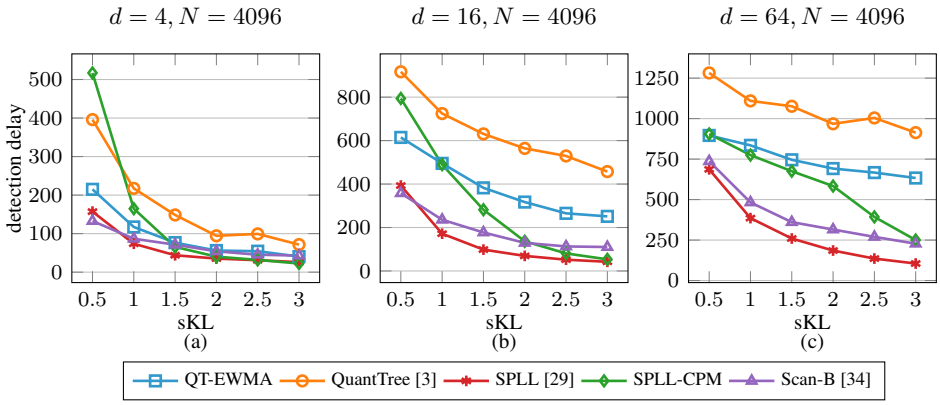


Figure 4.8: *Detection delays of the considered methods over Gaussian datastreams ($d = 4, 16, 64$) with different change magnitudes sKL(ϕ_0, ϕ_1) $= 0.5, 1, 2.5, 2, 2.5, 3$ and training set size $N = 4096$. For a fair comparison, we set target ARL$_0$ $= 1000$, which is maintained by all methods (see for instance Figure 4.3(g)). As expected, all methods improve their performance when increasing sKL(ϕ_0, ϕ_1) and achieve higher detection delays increasing d. We observe that when $d = 4$ QT-EWMA is on par with SPLL, SPLL-CPM and Scan-B, which outperform QT-EWMA when $d = 16, 64$, meaning that Scan-B seems more robust with respect to detectability loss [36].*

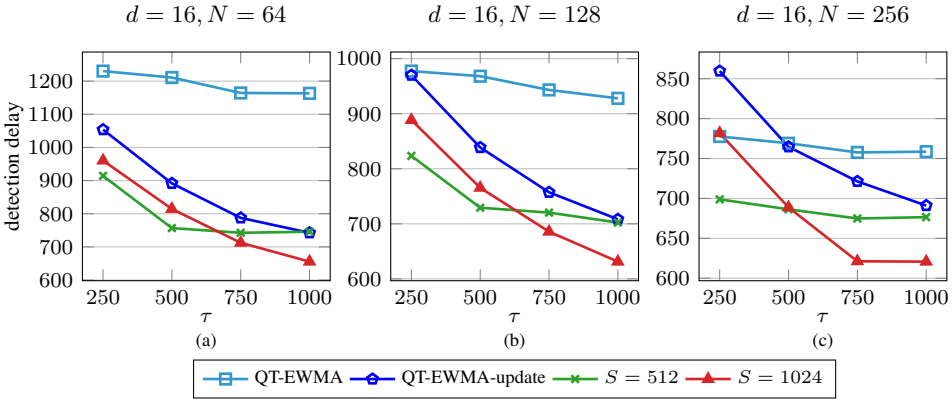


Figure 4.9: *Detection delays of QT-EWMA-update ($\beta = 5$) stopping the update after acquiring $S = 512, 1024$ samples, compared to QT-EWMA-update, QT-EWMA over Gaussian datastreams ($d = 16$) with change points at $\tau = 250, 500, 750, 1000$ with sKL(ϕ_0, ϕ_1) $= 2$. We set initial training set sizes $N = 64, 128, 256$ and target ARL$_0$ $= 2000$. We observe that QT-EWMA-update outperforms QT-EWMA, and that stopping the update improves the performance by reducing the risk of updating when $t > \tau$.*

CHAPTER 5

Class Distribution Monitoring

In this chapter, we introduce *Class Distribution Monitoring (CDM)*, a novel concept-drift detection algorithm that monitors a multi-class datastream to detect changes in the class-conditional distributions. In particular, we employ multiple instances of QT-EWMA (presented in Chapter 4) to monitor samples from different classes, and report a concept drift after detecting a change in at least one of the class-conditional distributions. Thanks to the properties of QT-EWMA, we demonstrate that CDM is nonparametric and guarantees the same ARL_0 as the individual instances of QT-EWMA used to monitor each class.

In Section 5.1 we present the CDM algorithm, and in Section 5.2 we discuss its theoretical properties, in particular the control of the ARL_0. Then, in Section 5.3 we illustrate the experimental evaluation of CDM, and in Section 5.4 we discuss the advantages and limitations of this approach, providing some insights on possible future work.

5.1 The CDM Algorithm

Most concept-drift detection methods that monitor the data distribution simply apply a change-detection algorithm to the datastream $\{x_t\}$ [54].

Algorithm 5.1 Class Distribution Monitoring (CDM)

Input: datastream $\{(x_t, l_t)\}_t$, target probabilities $\{\pi_j\}_{j=1}^K$, thresholds $\{h_t\}_t$, $TR = \{(x, l)\}$
Output: detection flag `ChangeDetected`, detection time t^*, drifted class ℓ^*

// Configuration:
`ChangeDetected` $\leftarrow$ False, $t^* \leftarrow \infty, \ell^* \leftarrow 0$
for $\ell \in \mathcal{L}$ **do**
    $TR^\ell \leftarrow \{x : (x, l) \in TR, l = \ell\}$
    construct QuantTree histogram $\mathcal{Q}^\ell = \{(\mathcal{S}_j^\ell, \pi_j)\}_{j=1}^K$ [3] from TR^ℓ
    initialize $t_\ell \leftarrow 0$, $Z_{j,0}^\ell \leftarrow \tilde{\pi}_j$ for $j \in \{1, \dots, K\}$
end for
// Monitoring:
for $t = 1, \dots$ **do**
    if the label l_t is provided **then**
        $\ell \leftarrow l_t,\ t_\ell \leftarrow t_\ell + 1,\ y_{j,t_\ell} \leftarrow \mathbb{1}(x_t \in \mathcal{S}_j^\ell)$
        compute $Z_{j,t_\ell}^\ell \leftarrow (1-\lambda) Z_{j,t_\ell-1}^\ell + \lambda y_{j,t_\ell-1}, \quad j = 1 \dots, K$ as in (4.11)
        compute QT-EWMA statistic $\mathcal{T}_{t_\ell}^\ell \leftarrow \sum_{j=1}^K (Z_{j,t_\ell}^\ell - \tilde{\pi}_j)^2 / \tilde{\pi}_j$ as in (4.12)
        if $\mathcal{T}_{t_\ell}^\ell > h_{t_\ell}$ **then**
            `ChangeDetected` $\leftarrow$ True
            $t^* \leftarrow t,\ \ell^* \leftarrow \ell$
            break
        end if
    end if
end for
return `ChangeDetected`, t^*, ℓ^*

Therefore, these algorithms compute at each time t a test statistic $\mathcal{T}_t$, and report a drift when $\mathcal{T}_t > h_t$, where h_t is a threshold defined to control the probability of having a false alarm. The detection time t^* is defined as the first time t in which the statistic exceeds the threshold. As in change detection, the goal is to detect a concept drift as soon as possible while controlling the ARL_0.

In the concept-drift detection literature, it is usually assumed that the labels l_t associated with the data samples x_t are regularly provided during monitoring [54]. However, change-detection algorithms are designed to operate in unsupervised settings, and therefore ignore this additional information. As a result, concept drifts affecting only a subset of classes can be hard to detect following this approach.

To exploit class labels, we propose Class Distribution Monitoring (CDM), a novel concept-drift detection algorithm designed to detect changes in any

class-conditional distribution ϕ_0^ℓ. We recall that each ϕ_0^ℓ is defined by

$$\mathbb{P}_{\phi_0^\ell}(x_t) = \mathbb{P}_{\phi_0}(x_t | l_t = \ell), \tag{5.1}$$

and we refer to Section 2.1.1 for further details on the concept-drift detection problem. In Algorithm 5.1 we illustrate the CDM procedure. First, we divide the training set TR into $\#\mathcal{L}$ subsets TR^ℓ and use these to construct $\#\mathcal{L}$ QuantTree histograms $\mathcal{Q}^\ell = \{(\mathcal{S}_j^\ell, \pi_j)\}$ [3], modeling the class-conditional distribution ϕ_0^ℓ for $\ell \in \mathcal{L}$ (lines 4–5). When an input sample x_t is provided with its label l_t, we find the histogram bin such that $x_t \in \mathcal{S}_j^\ell$ in the QuantTree $\mathcal{Q}^\ell$ corresponding to its label $\ell = l_t$ (line 11). Then, we compute the QT-EWMA statistics Z_{j,t_ℓ}^ℓ (4.11) and $\mathcal{T}_{t_\ell}^\ell$ (4.12) (lines 12–13), where t_ℓ is the number of samples of class ℓ observed until time t. We report a concept drift as the first time t when $\mathcal{T}_{t_\ell}^\ell > h_{t_\ell}$, where h_{t_ℓ} is the QT-EWMA threshold defined as in Section 4.3 (lines 14–18).

CDM successfully combines the advantages of monitoring the data distribution and of using the available labels, overcoming the main drawbacks of both approaches. In particular, CDM can detect any type of concept drift, including those that would not significantly increase the error rate of a classifier and therefore ignored by algorithms that monitor the classification error. Moreover, CDM separately monitors the class-conditional distribution, and therefore can promptly detect drifts affecting a small number of classes, which might be hard to identify by looking at the overall data distribution. We also remark that, contrarily to most of the other concept-drift detectors, CDM returns, on top of the detection time t^*, the class ℓ^* that triggered the detection (line 21), which might be extremely useful for diagnostics in practical applications.

5.2 Theoretical Analysis

Here we illustrate the most important properties of CDM. In particular, in Section 5.2.1 we discuss the online and nonparametric nature of CDM. Then, in Section 5.2.2 we demonstrate that CDM can control the ARL_0, and finally in Section 5.2.3 we analyze its computational complexity.

5.2.1 Online and Nonparametric Monitoring

Consistently with the notation introduced in Section 2.1, we can see CDM as an online change-detection test with statistic $\tilde{\mathcal{T}}_t$ defined as

$$\tilde{\mathcal{T}}_t = \mathcal{T}_{t_\ell}^\ell \quad \text{where } \ell = l_t, \tag{5.2}$$

and thresholds $\tilde{h}_t = h_{t_\ell}$. In fact, even though different statistics are computed depending on the label l_t associated to the sample x_t, the datastream is still processed one sample (x_t, l_t) at a time, as is evident from Algorithm 5.1. The nonparametric nature of the CDM test statistic is inherited from QT-EWMA because the distribution of each class-conditional statistic $\mathcal{T}^\ell$, like any other statistic defined on a QuantTree histogram, does not depend on the class-conditional distribution ϕ_0^ℓ [3]. Since $\tilde{\mathcal{T}}_t$ is defined in (5.2) using the nonparametric statistics $\mathcal{T}^\ell$, then its distribution does not depend on ϕ_0.

5.2.2 Control of the ARL_0

CDM also inherits from QT-EWMA the control of false alarms, and in particular the possibility to define thresholds that guarantee a target ARL_0. Here we demonstrate that, since in QT-EWMA the target ARL_0 is guaranteed by

$$\mathbb{P}_{\phi_0}(\mathcal{T}_t > h_t \mid \mathcal{T}_k \leq h_k \; \forall k < t) = \alpha \quad \forall t \geq 1, \tag{5.3}$$

where $\alpha = 1/\text{ARL}_0$ [52], then CDM yields the same ARL_0 as the QT-EWMA monitoring each class-conditional distribution.

Proposition 5.1. *Let $\tilde{\mathcal{T}}$ be the test statistic of CDM defined in* (5.2)*, and let $\{h_t\}$ be the QT-EWMA thresholds used to monitor the class-conditional distributions, which are defined to maintain the target ARL_0 (Section 4.3). Then, the change-detection test defined by $\tilde{\mathcal{T}}$ yields the same ARL_0.*

Proof. To prove the Proposition, we need to show that (5.3) holds for $\tilde{\mathcal{T}}$. By the definition of $\tilde{\mathcal{T}}$ in (5.2) and the law of total probability we have that

$$\begin{aligned}\mathbb{P}_{\phi_0}(\tilde{\mathcal{T}}_t > \tilde{h}_t \mid \tilde{\mathcal{T}}_k \leq \tilde{h}_k \; \forall k < t) = \\ = \sum_{\ell \in \mathcal{L}} \mathbb{P}_{\phi_0}(\mathcal{T}_{t_\ell}^\ell > h_{t_\ell} \mid \mathcal{T}_k^\ell \leq h_k \; \forall k < t_\ell, l_t = \ell) \cdot \\ \cdot \mathbb{P}_{\phi_0}(y_t = \ell \mid \mathcal{T}_k^\ell \leq h_k \; \forall k < t_\ell). \end{aligned} \tag{5.4}$$

Since all the samples (x_t, l_t) in the datastream are assumed to be independent, the label l_t associated with x_t is independent from the values of the statistic $\mathcal{T}_k^\ell$ for $k < t_\ell$, so we can drop the conditioning in the second factor of the second term of (5.4). Moreover, the probability under ϕ_0 in the first factor is conditioned on the event "$l_t = \ell$", so it coincides with the probability under the class-conditional distribution ϕ_0^ℓ defined in (5.1). Hence,

(5.4) becomes:

$$\begin{aligned}\mathbb{P}_{\phi_0}(\tilde{\mathcal{T}}_t > \tilde{h}_t \mid \tilde{\mathcal{T}}_k \leq \tilde{h}_k \; \forall k < t) = \\ = \sum_{\ell \in \mathcal{L}} \mathbb{P}_{\phi_0^\ell}(\mathcal{T}_{t_\ell}^\ell > h_{t_\ell} \mid \mathcal{T}_k^\ell \leq h_k \; \forall k < t_\ell) \cdot \mathbb{P}_{\phi_0}(l_t = \ell) = \\ = \sum_{\ell \in \mathcal{L}} \alpha \cdot \mathbb{P}_{\phi_0}(l_t = \ell) = \alpha, \quad (5.5)\end{aligned}$$

where the penultimate equality derives from the fact that (5.3) holds for the QT-EWMA test statistics $\mathcal{T}^\ell$ monitoring each class-conditional distribution. The last equality in (5.5) derives from the assumption that, under ϕ_0, each sample x_t has a label $l_t = \ell \in \mathcal{L}$, so the events $\{l_t = \ell\}_{\ell \in \mathcal{L}}$ represent a partition of the probability space, thus their probabilities sum to 1. Equation (5.5) demonstrates that (5.3) holds for $\tilde{\mathcal{T}}$, showing that CDM yields $\text{ARL}_0 = 1/\alpha$ [52], which is exactly the thesis. □

We remark that Proposition 5.1 holds for any CDM defined by an online change-detection algorithm that can be configured to yield the desired ARL_0 by setting a constant false alarm probability over time as in (5.3). This means that, in principle, we can define CDM using other change-detection tests, as long as their thresholds can be set to satisfy (5.3). However, to the best of our knowledge, QT-EWMA is the only nonparametric and online change-detection test for multivariate datastreams that satisfies this condition, which is not guaranteed even by algorithms in which the target ARL_0 can be set before monitoring, such as Scan-B [34].

5.2.3 Computational Complexity

CDM is extremely efficient both in terms of computational complexity and memory overhead because it employs $\#\mathcal{L}$ instances of QT-EWMA, whose efficiency is analyzed in detail in Section 4.6. Like QT-EWMA, CDM requires placing each sample x_t in its bin in the QuantTree histogram $\mathcal{Q}^\ell$ corresponding to its class label $\ell = l_t$, resulting in $\mathcal{O}(K)$ operations [3]. Then, CDM updates the corresponding EWMA statistics Z_{j,t_ℓ}^ℓ (4.11) for $j \in \{1, \ldots, K\}$, thus requiring to store in memory $\#\mathcal{L} \cdot K$ values, namely K statistics per class.

5.3 Experiments

Here we illustrate our experiments, which we designed to demonstrate that CDM outperforms mainstream concept-drift detection methods that

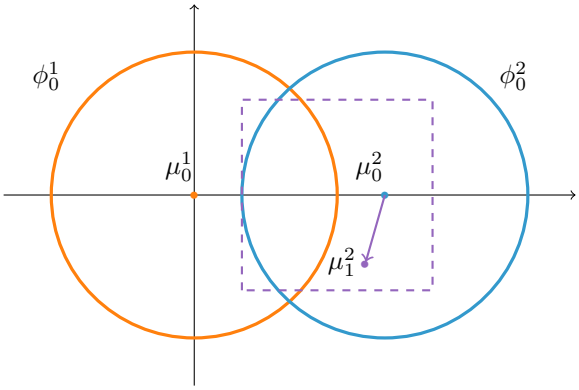


Figure 5.1: *Illustration of the Gaussian class-conditional distributions we use to generate synthetic data. Each class-conditional distribution ϕ_0^ℓ is represented by its mean μ_0^ℓ and 3σ ellipsoid. We consider changes $\phi_0 \to \phi_1$ defined by translating the mean of ϕ_0^2 inside the dashed rectangle, as shown in this example.*

monitor either the error rate of a classifier or the overall data distribution. First, we present the real-world and synthetic datasets on which we test our solution (Section 5.3.1), then we formally define the figures of merit we use (Section 5.3.2) and the reference methods from the literature (Section 5.3.3). Finally, we present and discuss our experiments and their results (Sections 5.3.4,5.3.5).

5.3.1 Considered Datasets

Real-world data. The INSECTS dataset [73] is a well-known benchmark for classification and concept-drift detection. It contains feature vectors ($d = 33$) extracted from sensor measurements describing the wing-beat frequency of six (annotated) species of flying insects. The dataset contains six concepts, each representing measurements acquired at a different temperature, which influences the flying behavior of the insects. This allows us to introduce realistic concept drifts by sampling the datastream from different concepts before and after the change point τ, as in Chapter 4. In our experiments, the stationary condition ϕ_0 is characterized by the class-conditional distributions $\{\phi_0^\ell\}_{\ell \in \mathcal{L}}$ describing the features of $\#\mathcal{L} = 4$ different insect species from each of the six concepts. We consider multiple drifts $\phi_0 \to \phi_1$ that consist in a temperature change affecting one or more classes, namely $\phi_0^\ell \to \phi_1^\ell \neq \phi_0^\ell$. In this setting, for each stationary distribution ϕ_0, the change $\phi_0 \to \phi_1$ is defined among 5 potential temperature changes affecting one of $2^{\#\mathcal{L}} - 1 = 15$ different subsets of the $\#\mathcal{L}$ classes, for a total of

75 distribution changes per initial concept ϕ_0. In our experiments we consider training sets containing 256 instances of each class, sampled without replacement from each class-conditional distribution ϕ_0^ℓ.

Synthetic data. To interpret the results obtained on real-world data, we synthetically generate various distribution changes $\phi_0 \to \phi_1$ and assess their impact on the classification error. In particular, we define the stationary distribution ϕ_0 as a mixture of $\#\mathcal{L} = 2$ Gaussians (one per class) $\phi_0^1 = \mathcal{N}(\mu_0^1, I)$ and $\phi_0^2 = \mathcal{N}(\mu_0^1, I)$ in $\mathbb{R}^2$, where I denotes the identity matrix, $\mu_0^1 = [0, 0]^T$, and $\mu_0^2 = [\delta, 0]^T$ for some $\delta > 0$. Post-change distribution ϕ_1 is defined by shifting $\phi_0^2 \to \phi_1^2 = \mathcal{N}(\mu_1^2, I)$, while keeping ϕ_0^1 fixed. Changes are thus regulated by μ_1^2, which we move over a grid around μ_0^1 (see Figure 5.1). Also in this case, we consider training sets containing 256 samples drawn from each ϕ_0^ℓ.

This setup was designed to assess when CDM is a better option than ECDD. The classification error varies when μ_1^2 moves along the horizontal direction, which is the line connecting μ_0^1 and μ_0^2: these changes can be promptly detected by ECDD when they increase the error rate. In contrast, changes translating μ_1^2 vertically (thus orthogonal to the line joining μ_0^1 and μ_0^2), do not change the error rate but only the input distribution. These changes cannot be detected by ECDD, but are perceivable by CDM, whose performance only depends by the change magnitude, which we measure by the symmetric Kullback-Leibler distance $sKL(\phi_0^2, \phi_1^2)$ [70], that in this case is equal to $\frac{1}{2}\|\mu_1^2 - \mu_0^2\|_2$.

5.3.2 Figures of Merit

As in Chapter 4, we assess the control of false alarms by computing the *empirical* ARL_0, i.e., the average detection time in datastreams distributed as ϕ_0. Thanks to Proposition 5.1, we expect the empirical ARL_0 of CDM to approach the target ARL_0 set before monitoring. Then, we measure the detection power by the *average detection delay* (or ARL_1), namely the average difference between the detection time t^* and the actual change point τ. The detection delay is computed considering only datastreams where no false alarms were reported before the change, thus $t^* > \tau$.

In our experiments on the INSECTS dataset, we measure the detection delay obtained over datastreams sampled from different initial distributions ϕ_0 and containing concept drifts affecting different classes. For an overall evaluation of these results, we also rank the considered methods by their average detection delay in each setting (i.e. rank $= 1$ for the best-performing

method, rank $= 2$ for the second-best, and so on) and compute the average rank of each method, as suggested in [74]. Moreover, we apply the Wilcoxon [75], Nemenyi [76], and Dunn [77] tests to assess whether the difference between the best-performing method in terms of average rank and the others is statistically significant, as suggested in [74].

5.3.3 Considered Methods

To ensure a fair comparison, we only consider methods that *i)* are nonparametric and *ii)* control the false alarms by setting a target ARL_0 before monitoring. In particular, we consider ECDD [56], which monitors the error rate of a classifier by means of an EWMA test statistic with thresholds defined to maintain the target ARL_0 (see Section 3.3 for more details on the ECDD algorithm). To the best of our knowledge, this is the only concept-drift detection algorithm that monitors the classification error while controlling the ARL_0. ECDD is also nonparametric since its test statistic only depends on the probability distribution of the classification error, which is assumed to be a Bernoulli distribution with unknown parameter p_0 for any ϕ_0 [56].

To demonstrate the advantages of monitoring the class-conditional distributions instead of the overall data distribution, we also compare CDM against two nonparametric change-detection algorithms that control the ARL_0, which we apply to the datastream $x_1, x_2, \ldots$ (ignoring class labels). In particular, we consider Scan-B [34] and QT-EWMA (Chapter 4). In QT-EWMA, we set the number of bins of the QuantTree histogram to $\#\mathcal{L} \cdot K$, to have a model of ϕ_0 comparable to that of CDM, which employs $\mathcal{L}$ QuantTree histograms having K bins each.

In terms of computational complexity, in these settings CDM is more efficient than QT-EWMA, which performs $\mathcal{O}(\#\mathcal{L}K)$ operation for each incoming sample x_t (Section 4.6). CDM is also more efficient than Scan-B, since in Section 4.6 we have shown that the computational cost of QT-EWMA favorably compares to that of Scan-B. In contrast, the ECDD test statistic is extremely efficient, but the algorithm also requires to classify each incoming sample x_t, so the computational cost of ECDD strongly depends on that of the classifier $\mathcal{K}$.

5.3.4 Real-world Data

In this Section, we discuss the empirical ARL_0 and the detection delay achieved on the INSECTS dataset by the considered models in the settings described in Section 5.3.1.

Table 5.1: *Empirical ARL_0 of the considered methods on the 6 concepts of the INSECTS dataset [73].*

Method		ECDD [56]	Scan-B [34]	QT-EWMA	CDM
target ARL_0		400	300	375	375
Concept	A	376.51	382.08	379.10	375.44
	B	371.07	384.56	361.78	374.47
	C	373.16	381.65	371.66	365.32
	D	374.14	387.17	367.18	369.94
	E	371.82	376.28	375.10	374.64
	F	377.67	374.22	375.58	371.87

Empirical ARL_0. We compute the empirical ARL_0 of the considered methods on the six concepts of the INSECTS dataset [73], which we denote by A, B, C, D, E, F. We consider each concept as a stationary distribution ϕ_0, and we sample without replacement 5000 training sets and 5000 datastreams of length $T = 8000$ from each ϕ_0. Then, we configure the considered methods on the training sets, and compute the empirical ARL_0 as the average detection time over these stationary datastreams.

We report the results of this experiment in Table 5.1, which shows that ECDD fails at accurately controlling the target $ARL_0 = 400$. In contrast, the empirical ARL_0 of CDM and QT-EWMA approaches their target, which we set to $ARL_0 = 375$ to match the empirical ARL_0 of ECDD. Similarly to ECDD, Scan-B does not accurately control the ARL_0, and this is consistent with the experiments in Chapter 4. For this reason, we set the target $ARL_0 = 300$ in Scan-B to yield approximately the same empirical ARL_0 as the other methods. Table 5.1 indicates that, in these settings, it is possible to fairly compare the detection delays of the considered methods, since they all yield approximately the same empirical ARL_0. Most importantly, the results in Table 5.1 empirically demonstrate that CDM effectively controls the ARL_0, confirming the theoretical results in Proposition 5.1.

Detection delay. For each of the 450 changes $\phi_0 \to \phi_1$ (75 for each of the 6 initial concepts) described in Section 5.3.1, we sample without replacement 1000 training sets and 1000 datastreams to be monitored. Each datastream is the concatenation of $\tau = 160$ points drawn from ϕ_0 and 7000 points drawn from ϕ_1. Table 5.2 reports the average detection delays of the considered methods depending on the drifted classes. As suggested in [74], we rank the considered methods according to their average detection delay obtained on each of the 450 changes (rank $= 1$ for the method with the

Table 5.2: *Average detection delays on the 15 possible subsets of classes of the INSECTS dataset [73] affected by concept drift, averaged over the 30 possible combinations of initial and post-change concepts. We also report the average rank over all the 15·30 experiments, and the p-values of the Wilcoxon, Nemenyi, and Dunn tests.*

Drifted class(es)	ECDD [56]	Scan-B [34]	QT-EWMA	CDM
1	207.98	212.53	267.73	**195.45**
2	245.85	162.58	195.44	**124.92**
3	264.27	224.99	278.57	**204.00**
4	224.91	235.87	265.96	**196.74**
1, 2	198.17	131.71	174.80	**114.44**
1, 3	172.62	169.87	223.50	**160.98**
1, 4	165.77	163.63	221.66	**145.82**
2, 3	163.66	126.56	167.55	**112.18**
2, 4	176.53	119.41	154.95	**106.49**
3, 4	210.04	169.88	218.90	**153.51**
1, 2, 3	139.29	115.01	152.91	**103.60**
1, 2, 4	148.03	103.24	141.09	**98.89**
1, 3, 4	144.81	134.83	183.41	**131.38**
2, 3, 4	132.36	**96.92**	136.90	98.57
1, 2, 3, 4	122.38	**88.86**	128.04	91.44
Avg. rank	2.400	2.382	3.516	**1.698**
Wilcoxon-p	$5.57{\cdot}10^{-15}$	$2.76{\cdot}10^{-20}$	$1.10{\cdot}10^{-75}$	–
Nemenyi-p	$5.75{\cdot}10^{-6}$	$1.36{\cdot}10^{-1}$	$4.36{\cdot}10^{-17}$	–
Dunn-p	$1.99{\cdot}10^{-7}$	$1.87{\cdot}10^{-2}$	$5.36{\cdot}10^{-19}$	–

lowest detection delay, etc.), and report their average rank. We also report the p-values of the Wilcoxon [75], Nemenyi [76], and Dunn [77] post-hoc tests, to assess whether the differences between the best-ranking method and the others are statistically significant.

We observe that CDM turns out to be the best method in 13 out of the 15 considered changes, and the best in terms of average rank. The Wilcoxon, Nemenyi, and Dunn tests show that the gap of CDM over ECDD and QT-EWMA is statistically significant (p-value < 0.05). The gap between CDM and Scan-B is less remarkable, but still significant according to the Wilcoxon and Dunn tests. As expected, all the methods tend to yield lower detection delays when the change affects more classes. In particular, the difference between the detection delays of CDM and QT-EWMA is larger when the change affects only one class rather than when it affects all of them, showing that monitoring the class-conditional distributions can indeed improve the detection performance in these cases. This effect is even more apparent in the comparison between CDM and Scan-B.

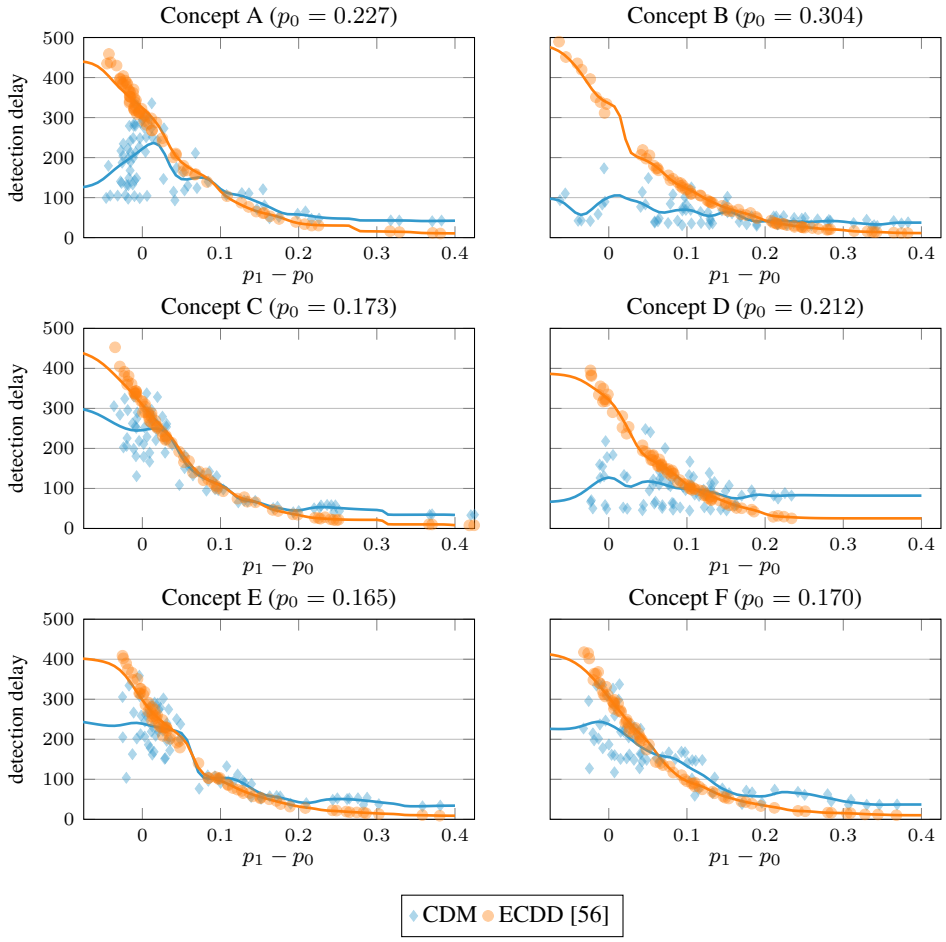


Figure 5.2: *Detection delay achieved by ECDD and CDM on the INSECTS dataset [73] for each of the* 6 *stationary concepts, plotted against the difference* $p_1 - p_0$*, where* p_0, p_1 *are the error rates of* $\mathcal{K}$ *before and after the drift. Each dot is the average of* 1000 *realizations of the same change* $\phi_0 \to \phi_1$ *i.e., thus with the same affected classes.*

Most remarkably, CDM substantially outperforms ECDD in terms of average detection delay in all the considered settings. This is due to the fact that ECDD can only detect concept drifts that increase the classification error, while the considered drifts in the INSECTS dataset might have little impact on the error rate of a classifier. To further analyze the relation between detection delay and classification error, we plot in Figure 5.2 the average detection delays of CDM and ECDD against the difference between the classification error after (p_1) and before the change (p_0). Each plot reports the results obtained on the 75 drifts we consider for each ini-

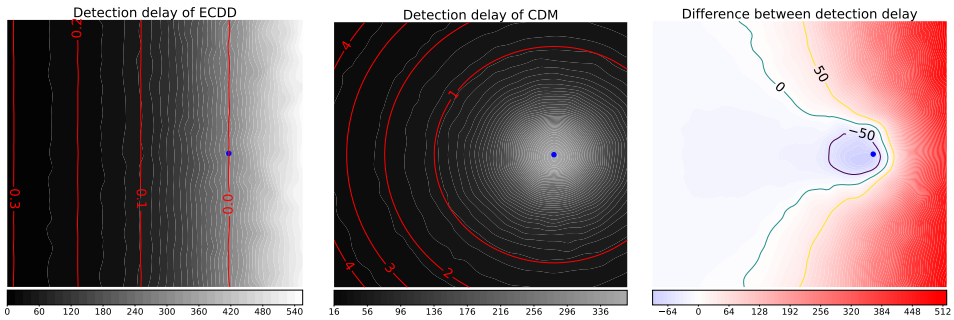


Figure 5.3: *Results of the experiment on synthetic data, represented by heatmaps defined on the feature space $\mathbb{R}^2$. The blue dot represents the pre-change mean $\mu_0^2 \in \mathbb{R}^2$.* (a,b) *report, at each coordinate $\overline{\mu} \in \mathbb{R}^2$, the average detection delay achieved by ECDD and CDM when $\mu_0^2 \to \overline{\mu}$. In* (a) *the contour lines indicate the difference in classification error $p_1 - p_0$ before and after the change, and in* (b) *the change magnitude $sKL(\phi_0^2, \phi_1^2)$.* (c) *report the difference between the detection delay achieved over synthetic data by ECDD and CDM, with contour lines.*

tial concept A, B, C, D, E, F. We highlight the relation between $p_1 - p_0$ and the detection delay by plotting the moving average (weighted by a Gaussian kernel) of the detection delay as a function of $p_1 - p_0$. These results qualitatively show that the performance of ECDD only depends on $p_1 - p_0$, which is often small and sometimes even negative. In contrast, CDM can detect any change in the class-conditional distributions, thus yielding a lower detection delay in most cases.

5.3.5 Synthetic Data

Concept drifts might not always heavily impact the classification performance, as we have shown in Figure 5.2 on the INSECTS dataset. Here we further analyze the fundamental difference between monitoring the classification error by ECDD and the class-conditional distributions by CDM in the synthetic scenario described in Section 5.3.1, where we can control both $p_1 - p_0$ and the change magnitude $sKL(\phi_0^2, \phi_1^2)$. We configure ECDD and CDM to maintain the same ARL_0 as in Section 5.3.4.

The results of this experiment are illustrated in Figure 5.3. In particular, Figures 5.3(a,b) report the detection delays respectively achieved by ECDD and CDM as heatmaps. The color-coded value at a point $\overline{\mu} \in \mathbb{R}^2$ represents the detection delay achieved by the each algorithm when $\mu_1^2 = \overline{\mu}$, averaged over 5000 experiments. Moreover, in the same figures we plot, respectively, the difference between the post- and pre-change error rates $p_1 - p_0$, and the change magnitude $sKL(\phi_0^2, \phi_1^2)$ as level curves.

As expected, ECDD cannot detect virtual drifts, as can be seen by the large detection delays on the right side of Figure 5.3(a), but it achieves excellent detection performance when the translation reduces the distance between the two class-conditional distributions, increasing the classification error ($p_1 - p_0 > 0$). In contrast, the detection delay of our CDM only depends on the distance $sKL(\phi_0^2, \phi_1^2)$, as can be seen in Figure 5.3(b), where the level curves of the detection delays are circles centered at μ_0^2.

Figure 5.3(c) reports the difference between the detection delays of ECDD and CDM, together with the level curves for the values -50, 0 and 50. ECDD outperforms CDM when μ_1^2 falls inside a relatively small triangular portion of $\mathbb{R}^2$, corresponding to drifts that significantly increase the error rate, even though the change magnitude is relatively low. However, the difference is substantial (> 50) only in a small region where the change is nearly imperceptible and the performance of both algorithms is poor. Otherwise, CDM yields lower detection delays than ECDD, and the difference is quite large, especially when the drift reduces the error rate.

5.4 Discussion and Future Work

The results of our experiments on synthetic data, which are summarized in Figure 5.3, visually confirm our intuition that monitoring individual class-conditional distributions is more effective than monitoring the classification error since drifts can cause only a little increase of the error rate of a classifier, and can even be virtual drifts that do not increase it at all. Our results on the INSECTS dataset [73] (see Table 5.2 and, most remarkably, the plots in Figure 5.2), indicate that drifts with little or no impact on the classification error (and even drifts that reduce it) can occur quite often in real-world datastreams, and not only in our rather simplistic synthetic scenario.

Future work will extend CDM by applying other change-detection algorithms whose thresholds are defined to satisfy (5.3), using for instance SPLL-CPM (Section 4.5), which operates in semiparametric settings using an online CPM [25]. Moreover, other change-detection algorithms can be used in combination with QT-EWMA, since Proposition 5.1 holds even when different algorithms are employed to monitor different class-conditional distributions, as long as their thresholds satisfy (5.3). Therefore, different algorithms might be selected based on prior knowledge about each distribution. For instance, SPLL-CPM might be employed to monitor class-conditional distributions that can be assumed to be approximately Gaussian, and QT-EWMA on classes where such assumption cannot be made and thus a nonparametric algorithm is preferable.

CHAPTER 6

Change Detection in Sequential Attacks

In this chapter we address the problem of detecting and correcting errors in a particular class of cryptographic attacks. We cast this as a change-detection problem, and propose an error-detection and correction strategy based on a state-of-the-art online and nonparametric change-detection algorithm. Our experiments on synthetic and real-world data demonstrate that our strategy substantially improves the success rate of the considered attacks, ourperforming the existing solutions adding a relatively small computational overhead. These results show that these attacks can be easily strengthened, making it necessary to implement countermeasures even when the cryptosystems are considered secure due to the low success rate of these attacks.

In Section 6.1 we introduce *sequential side-channel* cryptographic attacks and the existing strategies to deal with errors in these attacks. In Section 6.2 we formally define sequential attacks and cast error-detection and correction as a change-detection problem. In Section 6.3 we illustrate our error-detection and correction strategy, and in Section 6.4 we present two sequential attacks strengthened by our strategy. Finally, in Section 6.5, we describe our experiments and discuss the results.

6.1 Background

Side-Channel Analysis (SCA) is a class of cryptographic attacks that exploit the physical information leaked by a target device during the execution of a cryptographic algorithm, and retrieve the secret key by means of specific statistical tools. Examples of side-channel information include execution time [78], power consumption [79] and electromagnetic radiation [80]. Since its introduction in 1996 [78], SCA has become a major threat, as any statistical dependence between physical leakages and secret information might weaken cryptosystems that are considered secure from a mathematical point of view.

Here we consider sequential attacks, a particular type of side-channel attacks made of consecutive steps. The aim of each step is to recover a small portion of the secret key using a *distinguisher* [81], namely a statistic of the side-channel data that can identify the most likely among all the possible values of the target key portion. In sequential attacks, the distinguisher also depends on the intermediate results computed by the target cryptosystem, which can be calculated using the key portion recovered in the previous step of the attack. As an illustrative example, we consider the *Horizontal Correlation Power Analysis* (H-CPA) attack [82], where the distinguisher is the correlation coefficient between the power consumption and the Hamming weight [83], namely the number of 1-valued bits of the intermediate results of a multiplication. When a key portion is wrongly recovered, e.g. due to noise in the power measurements, the intermediate results predicted in the subsequent steps no more correspond to those that were actually involved in the computations. Therefore, in H-CPA, the correlation coefficients between the predicted Hamming weights and the power consumption will not allow to recover the remaining key bits, since the Hamming weight of the wrongly predicted operands are not correlated to the power consumption, as shown in Figure 6.1(a). At the end of an attack, the recovered key can be verified by checking a digital signature, but when the attack fails this procedure cannot locate the errors.

The fact that sequential attacks propagate the first error introduces a distribution change in the distinguisher values (as shown in Figure 6.1) has been referred to as an *error-detection property* [78] since it can be used to determine the position of the first error. Error propagation was first observed in a Timing Attack against modular exponentiation presented in [78]. The authors also provide a sketch of an error-correction procedure, but do not disclose any implementation details or experimental results.

Error propagation has been first exploited for error detection in a Timing

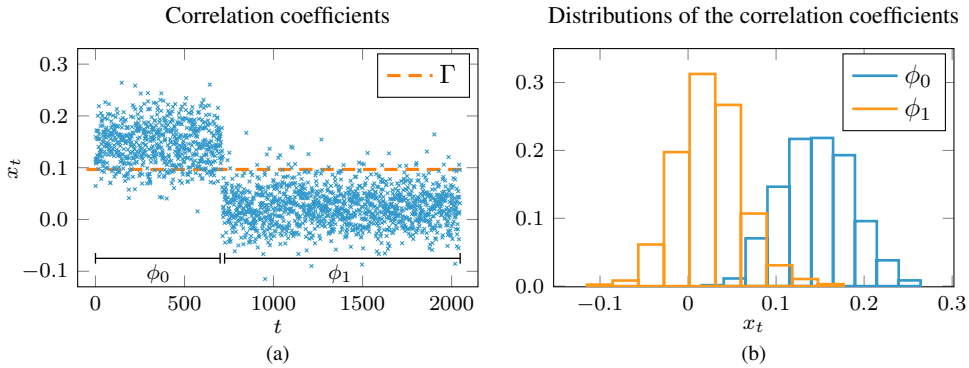


Figure 6.1: (a) *The distinguisher values (correlation coefficients) of the key bits chosen by an unsuccessful H-CPA attack. The first error occurred at step* $\tau = 713$*, causing a distribution change. The horizontal dashed line indicates the optimal threshold* Γ *separating the distinguisher values before and after* τ*, as discussed in Section 6.5.3.* (b) *Two histograms representing the empirical distribution of the distinguisher before (blue) and after (orange)* $\tau = 713$*: the two distributions are significantly different.*

Attack against RSA-512 [84]. In particular, the authors observe that, after the first error, the considered attack is more likely to predict key-bits valued 0, while in the target key the distribution of the bit values can be assumed to be uniform. For this reason, after each step, the authors employ a statistical test to determine whether the distribution of the latest recovered key bit values is uniform or not. When that is not the case, an error is reported and a correction procedure is activated. The same authors extend their work in [85, 86], introducing error-detection strategies for some "divide and conquer attacks", i.e. attacks that target a portion of key at a time, including the Timing Attack presented in [84]. The main drawback of these strategies is the lack of generality. Indeed, each attack is presented together with a specific error-detection strategy that applies only to that attack.

A more general error-detection strategy is proposed in [87–89] for Timing Attacks targeting, respectively, RSA-512, RSA-CRT and a GPU implementation of RSA, all subject to error propagation. These works leverage the fact that, before the first error, the distinguishers corresponding to the correct and wrong key values follow different distributions. After an error, the distinguisher follows the same distribution both for correct and wrong key values. For this reason, the authors monitor the distinguishers and/or the difference between the distinguishers of the two candidate key bit values, and detect an error when that quantity is "low" (i.e. below a threshold) for a certain number of consecutive steps of the attack. Although these error-detection methods can in principle be applied also to other attacks,

Algorithm 6.1 Target algorithm (decryption)

Input: ciphertext C, secret key D
Output: original message M

1: M_1 is initialized
2: **for** $t = 1 : T$ **do**
3: $\quad M_{t+1} \leftarrow O(D[t], M_t, C)$
4: **end for**
5: **return** M_{T+1}

Algorithm 6.2 Sequential attack

Input: target algorithm, ciphertext C, side channel $\{\mathbf{L}_t\}_{t=1}^{T}$, distinguisher $\mathcal{D}$
Output: estimated secret key $\widehat{D}$

1: $\widehat{M}_1$ is initialized as in Algorithm 6.1
2: **for** $t = 1 : T$ **do**
3: $\quad \widehat{D}[t] \leftarrow \arg\max_{\mathbf{v} \in V} \mathcal{D}(\mathbf{v}, \widehat{M}_t, \mathbf{L}_t)$
4: $\quad \widehat{M}_{t+1} \leftarrow O(\widehat{D}[t], \widehat{M}_t, C)$
5: **end for**
6: **return** $\widehat{D} \leftarrow (\widehat{D}[1], \ldots, \widehat{D}[T])$

they require the attacker to set a threshold to decide whether the distinguisher or their difference is "high" or "low". This threshold might be difficult to set *a priori*, and typically relies on heuristics that do not allow to control the false alarm rate. Controlling false alarms is crucial, because any detection triggers a correction procedure that might be time-consuming. Therefore, a large number of false alarm might make the attack impractical from a computational point of view.

6.2 Sequential Attacks

In this section we formally define sequential attacks (Section 6.2.1), and cast error-detection and correction in sequential attacks as a change-detection problem (Section 6.2.2).

6.2.1 The Sequential Attack Procedure

Among side-channel attacks we consider the class of sequential attacks. A sequential attack recovers a portion of the secret key at a time, by reconstructing the intermediate steps of the target algorithm. In particular, the attacker computes the possible intermediate results of each step depending on the value of a portion of the secret key. Then, the attacker leverages a distinguisher, namely a statistic of the side-channel data, to select the most likely value of the portion of secret key involved in that step. Note that, to compute these intermediate results, the attacker needs to know the implementation of the target cryptosystem, as well as its input.

Let us consider the prototype of a decryption function (Algorithm 6.1) as the target of a sequential attack. The t-th step of the decryption typically combines a portion $D[t]$ of the key (one or a few bits), the output M_t of the

previous step and sometimes the ciphertext C to compute the next output $M_{t+1} = O(D[t], M_t, C)$, where O denotes the set of instructions executed in each step of the algorithm. It is assumed that M_1 is initialized to some fixed value, and that the algorithm has T steps in total.

In the t-th step of a sequential attack (Algorithm 6.2), the attacker recovers $D[t]$ by evaluating a *distinguisher* $\mathcal{D}(\mathbf{v}, \widehat{M}_t, \mathbf{L}_t)$, a statistic that takes as input a possible key value $\mathbf{v}$, the output $\widehat{M}_t$ computed in the previous step of the attack, and the side-channel data $\mathbf{L}_t$. The distinguisher is expected to take a high value when $\mathbf{v}$ coincides with the true key value $D[t]$, and a lower value otherwise. Therefore, the attacker selects $\widehat{D}[t]$ as the value $\mathbf{v} \in V$ that maximizes $\mathcal{D}$ (line 3):

$$\widehat{D}[t] = \arg\max_{\mathbf{v} \in V} \mathcal{D}(\mathbf{v}, \widehat{M}_t, \mathbf{L}_t). \tag{6.1}$$

Then, the output $\widehat{M}_{t+1}$ is computed depending on the recovered key value $\widehat{D}[t]$ (line 4), and the whole procedure is repeated in the next step.

By design, sequential attacks are subject to error propagation. In fact, when the maximization of the distinguisher function leads to an error, i.e. $\widehat{D}[t] \neq D[t]$, also the predicted output $\widehat{M}_{t+1}$ is different from the actual output M_{t+1} computed in the target algorithm. All the key portions recovered after the first error are based on a wrong prediction, and therefore are not reliable. An error occurs every time the distinguisher is maximized by a wrong key value rather than the correct one, e.g. due to noise in the side-channel measurements.

6.2.2 Problem Formulation

To strengthen sequential side-channel attacks we need to solve two major problems:

- **Error Detection:** estimating the first attacking step τ where the recovered key portion obtained by (6.1) is different from the true one, formally

  $$\tau = \min\{t : \widehat{D}[t] \neq D[t]\}. \tag{6.2}$$

  Error detection consists in determining that an error has occurred, and providing an estimate $\hat{\tau}$ of its location. Error detection is preferably executed *online*, i.e. during the execution of the attack. As observed in Figure 6.1, the distinguisher values $x_t = \max_{\mathbf{v} \in V} \mathcal{D}(\mathbf{v}, \widehat{M}_t, \mathbf{L}_t)$, for $t < \tau$, corresponding to the correct key values $\{\widehat{D}[t]\}_{t<\tau}$ can be considered i.i.d. realizations drawn from an unknown distribution ϕ_0,

namely $x_t \sim \phi_0$. In contrast, after the first error at τ, the distinguisher values $\{x_t\}_{t>\tau}$ follow another, unknown distribution $\phi_1 \neq \phi_0$, namely $x_t \sim \phi_1$. Therefore, error detection can be addressed as an online change-detection problem (formulated in Chapter 2) on the univariate datastream formed by distinguisher values $\{x_t\}_t$.

- **Error Correction:** correcting the first error using its estimated location $\hat{\tau}$. Note that a detection does not necessarily correspond to an error, being just a false alarm of the change-detection algorithm. Moreover, the estimated error location $\hat{\tau}$ might not be accurate. Therefore, errors cannot be simply corrected by changing the decision made at the step identified by the error-detection procedure.

6.3 Strengthening Sequential Attacks

Here we present our error detection and correction strategy to strengthen a generic sequential attack (Algorithm 6.3). In what follows we provide an overview of our solution (Section 6.3.1), then we describe in detail our error-detection (Section 6.3.2) and correction (Section 6.3.3) procedures. Finally, we discuss more in depth the assumptions on which our methodology is based (Section 6.3.4).

6.3.1 Overview

The core ingredient of our error-detection procedure is a change-detection algorithm that we execute at each step of the attack (Algorithm 6.3, line 6) to monitor the sequence of distinguisher values $\{x_t\}_t$:

$$x_t = \max_{\mathbf{v} \in V} \mathscr{D}(\mathbf{v}, \widehat{M}_t, \mathbf{L}_t) = \mathscr{D}(\widehat{D}[t], \widehat{M}_t, \mathbf{L}_t). \tag{6.3}$$

Therefore, $\{x_t\}_t$ contains all the distinguisher values corresponding to the key portions recovered by the attack. At each time t, the online change-detection algorithm indicates whether the sequence $x_1, \ldots, x_t$ contains a distribution change and, in this case, an estimate of the change-point location $\hat{\tau}$ (lines 1–5). This estimate is usually rather close to the location of the first error τ occurred during the attack, but it does not necessarily correspond to τ, as we show in Section 6.3.4.

Every time we detect a change during the attack, we activate our error-correction procedure (lines 7–19). This consists in a brute-force search over all the possible values of the secret key in a reasonably small window $\mathbf{W}_{\hat{\tau}}$, which is centered around the estimated change point $\hat{\tau}$. Our idea is to use

Algorithm 6.3 Strengthened sequential attack

Input: target algorithm, ciphertext C, side channel $\{\mathbf{L}_t\}_{t=1}^{K}$, distinguisher $\mathcal{D}$, set of possible window lengths $\mathbf{S}$
Output: estimated secret key $\widehat{D}$

1: $\widehat{M}_1$ is initialized as in Algorithm 6.1
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad\widehat{D}[t] \leftarrow \arg\max_{\mathbf{v} \in V} \mathcal{D}(\mathbf{v}, \widehat{M}_t, \mathbf{L}_t)$ // Algorithm 6.2, line 3
4: $\quad\widehat{M}_{t+1} \leftarrow O(\widehat{D}[t], \widehat{M}_t, c)$ // Algorithm 6.2, line 4
5: $\quad x_t \leftarrow \max_{\mathbf{v} \in V} \mathcal{D}(\mathbf{v}, \widehat{M}_t, \mathbf{L}_t)$ // save the distinguisher value x_t
6: $\quad\mathcal{T}_t \leftarrow \max_k \mathcal{T}_{k,t}$ // compute change-detection statistic
7: $\quad$**if** $\mathcal{T}_t > h_t$ **then**
8: $\quad\quad\hat{\tau} \leftarrow \arg\max_k \mathcal{T}_{k,t}$
9: $\quad\quad$align $\hat{\tau}$ and t
10: $\quad\quad$**for each** $w \in \mathbf{S}$ **do**
11: $\quad\quad\quad$`SuccCorr`, $\mathbf{v}_{\text{best}} \leftarrow$ correction$(\hat{\tau}, w, \{x_t\}_{k=1}^{t})$ // Algorithm 6.4
12: $\quad\quad\quad$**if** `SuccCorr` **then**
13: $\quad\quad\quad\quad$**break**
14: $\quad\quad\quad$**end if**
15: $\quad\quad$**end for**
16: $\quad\quad$$(\widehat{D}[\hat{\tau} - u], \ldots, \widehat{D}[\hat{\tau} + u]) \leftarrow \mathbf{v}_{\text{best}}$
17: $\quad\quad$$t \leftarrow \hat{\tau} + u + 1$
18: $\quad\quad$remove $x_{\hat{\tau}-u}, \ldots, x_{\hat{\tau}+u}$
19: $\quad$**end if**
20: **end for**
21: **return** $\widehat{D} \leftarrow (\widehat{D}[1], \ldots, \widehat{D}[T])$

a statistical test to determine whether any of the tested combinations can correct the potential error. In particular, the statistical test assesses whether the distinguisher in a window $\mathbf{W}_{<}$, cropped before the brute-force window $\mathbf{W}_{\hat{\tau}}$, and in a window $\mathbf{W}_{>}$, opened after $\mathbf{W}_{\hat{\tau}}$, follow the same distribution (see Figure 6.2). In fact, assuming that $\mathbf{W}_{<}$ have been computed from correct key values, the statistical test should identify the correct combination as the one yielding the same distribution in $\mathbf{W}_{>}$. It is important to remark that, to deal with false alarms, the window $\mathbf{W}_{\hat{\tau}}$ should not to be considered during the correction procedure, nor in the following steps of the attack (line 18). In fact, the detection at $\hat{\tau}$ suggests that – in case of a false alarm – the distribution of $\mathbf{W}_{\tau}$ is non-stationary even when the key is correctly reconstruted (see Figure 6.2(c)). Note that our methodology is very general, thus it can be applied to any sequential attack, as defined in Section 6.2.1.

6.3.2 Error Detection

To automatically determine whether the distribution in $\{x_t\}_t$ has changed or not and, in the former case, to estimate the exact location of the first wrong guess in the secret key, we monitor the sequence $\{x_t\}_t$ by an online CPM [25]. We recall that the CPM is based on a test statistic $\mathcal{T}_{k,t}$ that compares the distribution of the two consecutive windows $x_1, \ldots, x_k$ and $x_{k+1}, \ldots, x_t$. The statistic $\mathcal{T}_{k,t}$ is computed for all $k \in \{1, \ldots, t-1\}$, and the online change-detection statistic $\mathcal{T}_t$ is defined as $\mathcal{T}_t = \max_k \mathcal{T}_{k,t}$. We compute the statistic $\mathcal{T}_t$ at each time t, and report a change at time $t = t^*$ if $\mathcal{T}_t > h_t$, where h_t is a threshold. After detecting a change, the change point location τ is estimated by

$$\hat{\tau} = \arg\max_k \mathcal{T}_{k,t}. \tag{6.4}$$

We refer to Section 3.1 for a more detailed description of the CPM. In particular, here we use the Lepage statistic [20], which is meant to identify both changes in scale and location [25], as we expect that a wrong guess might affect any of these. A fundamental advantage of the Lepage statistic is that, like other rank-based statistics, it does not depend on ϕ_0 and ϕ_1, thus the thresholds $\{h_t\}_t$ have been pre-computed through Montecarlo simulations [25], to achieve the target ARL_0, which we set to $\text{ARL}_0 = 50,000$.

To apply our error-detection method, we first execute the same operations performed during each step of the sequential attack (Algorithm 6.3, lines 1-4), then append the distinguisher value x_t to the sequence (line 5) and finally monitor $x_1, \ldots, x_t$ by the CPM (line 6). Typically, the online CPM requires a few samples after the change point to gather enough statistical evidence for a detection, which occurs at $t^* > \hat{\tau}$. All the computations performed in the steps between the estimated change point $\hat{\tau}$ and the detection time t^* are pointless due to error propagation, making detection promptness a crucial aspect in attacks that are computationally expensive.

Note that monitoring $\{x_t\}_t$ by an online change-detection test is substantially different from using the elementary error-detection schemes described in [87–89], which estimate τ as the first index where $x_t < \Gamma$, where Γ is a fixed threshold. These methods disregard whether the detection is due to a permanent distribution change (introduced by an error), or rather by an outlier (i.e., a spurious distinguisher value). Moreover, using a CPM allows us to control the false positives by setting the ARL_0.

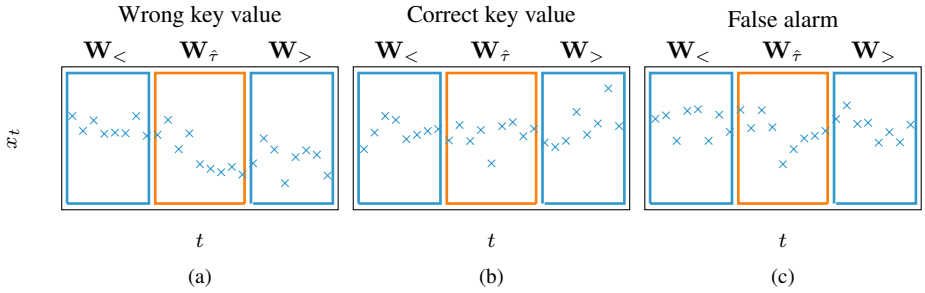


Figure 6.2: *Examples of distinguisher values in $\mathbf{W}_{\hat{\tau}}$, $\mathbf{W}_<$, $\mathbf{W}_>$ and the rationale behind our correction procedure. In* (a) *the key tested in $\mathbf{W}_{\hat{\tau}}$ is wrong, thus $\mathbf{W}_<$ and $\mathbf{W}_>$ have different distributions. In* (b)*, the tested key is correct, thus $\mathbf{W}_<$ and $\mathbf{W}_>$ follow the same distribution. In* (c)*, $\hat{\tau}$ corresponds to a false alarm, thus the distributions of $\mathbf{W}_<$ and $\mathbf{W}_>$ coincide, but differ around the detection, i.e. in $\mathbf{W}_{\hat{\tau}}$. This indicates why it is necessary not to include $\mathbf{W}_{\hat{\tau}}$ in the error correction, and also to remove it from $\{x_t\}_t$ when the monitoring restarts.*

6.3.3 Error Correction

After detecting a change at step t^* with estimated location $\hat{\tau}$, we activate our error-correction method (Algorithm 6.3, line 11), which is detailed in Algorithm 6.4. The aim of this procedure is to correct the error that might have occurred at $\hat{\tau}$ or at a nearby index. More formally, we expect $\hat{\tau}$ to be:

- the location of the first error, i.e. $\hat{\tau} = \tau$ as defined in (6.2), or
- a correct but inaccurate detection, i.e. an error occurred at τ, which is close but does not coincide with $\hat{\tau}$, or
- a false alarm, i.e. a change point is detected at step t^* even though no errors occurred at any $\tau \leq t^*$.

To handle all these cases, our error-correction procedure implements a brute-force search over a reasonably small window $\mathbf{W}_{\hat{\tau}} = \{\hat{\tau} - u, \ldots, \hat{\tau} + u\}$ of size $w = 2u + 1$, centered at the detected change point $\hat{\tau}$. During the brute-force search over $\mathbf{W}_{\hat{\tau}}$, we test each possible value $\mathbf{v} \in \{0, 1\}^w$ as follows: first, we compute the outputs of the operations executed over $\mathbf{W}_{\hat{\tau}}$, using $\mathbf{v}$ (lines 2-3). Then, the attack continues after step $\hat{\tau} + u + 1$ (line 4) for a few steps to compute the distinguisher values of the window $\mathbf{W}_>$. When $\mathbf{v}$ is the correct combination, assuming that no errors occur for a few steps, the distinguisher values in $\mathbf{W}_>$ are expected to follow the distribution ϕ_0, as shown in Figure 6.2(b). A very practical way to assess this hypothesis is to test whether the distribution in $\mathbf{W}_>$ is the same as that in $\mathbf{W}_<$, which is assumed to contain distinguisher values computed before the first error. In contrast, when $\mathbf{v}$ is a wrong combination, the distinguisher

Algorithm 6.4 Correction procedure

Input: target algorithm, ciphertext C, side channel $\{\mathbf{L}_t\}_{t=1}^{T}$, distinguisher $\mathcal{D}$, change point $\hat{\tau}$, $\mathbf{W}_{\hat{\tau}}$ with size $w = 2u + 1$, distinguisher sequence $\{x_t\}_t$, predicted output $\widehat{M}_{\hat{\tau}-u}$

Output: correction goodness (`SuccCorr`), best estimated key $\mathbf{v}_{\text{best}}$ over $\mathbf{W}_{\hat{\tau}}$

1: **for** $\mathbf{v} \in \{0,1\}^w$ **do**
2: set $(\widehat{D}^{\mathbf{v}}[\hat{\tau}-u], \ldots, \widehat{D}^{\mathbf{v}}[\hat{\tau}+u]) = \mathbf{v}$ // initialization
3: compute $\widehat{M}^{\mathbf{v}}_{\hat{\tau}-u+1}, \ldots, \widehat{M}^{\mathbf{v}}_{\hat{\tau}+u+1}$ using O // as in Algorithm 6.2, line 4
4: restart the attack from step $t = \hat{\tau} + u + 1$
5: select the two windows $\mathbf{W}_< \leftarrow \{x_t\}_{t<\hat{\tau}-u}$, $\mathbf{W}_> \leftarrow \{x^{\mathbf{v}}_t\}_{t>\hat{\tau}+u}$
6: run the statistical test $\mathcal{T}(\mathbf{W}_<, \mathbf{W}_>)$
7: **if** the test yields enough statistical evidence **then**
8: **return** true, $\mathbf{v}$
9: **end if**
10: **end for**
11: **return** false, the $\mathbf{v}$ maximizing the statistic in line 6

values in $\mathbf{W}_>$ should follow a different distribution ϕ_1 due to error propagation, (see Figure 6.2(a)). The choice of the hypothesis test and the sizes of the windows $\mathbf{W}_<, \mathbf{W}_>$ (which we have not defined here for the sake of generality) depend on several factors, mainly the computational cost of the attack. In Section 6.4 we provide two detailed examples of correction procedures for different attacks.

As any other statistical test, the CPM might detect a change point in $\{x_t\}_t$ even though no errors occurred. The correction procedure has to cope with this crucial problem: if $\hat{\tau}$ is a false alarm, a statistical test might find that the distinguisher follows different distributions before and after τ even when the correct combination is found by the brute-force search. This is the reason why we do not include $\mathbf{W}_{\hat{\tau}}$ in the windows $\mathbf{W}_<, \mathbf{W}_>$. For the same reason, we remove all the windows $\mathbf{W}_{\hat{\tau}}$ from $\{x_t\}_t$ in the following steps of the attack (Algorithm 6.3, line 18). Since we exclude the analyzed brute-force windows from $\{x_t\}_t$, the indexes of the monitored sequence $\{x_t\}_t$ are not aligned with the attacking steps, and the detected change points must be adjusted (Algorithm 6.3, line 9) by a suitable shift.

The size of the brute-force window determines a trade-off between effectiveness and efficiency: larger brute-force windows allow us to handle inaccurate estimates of the change point locations $\hat{\tau}$, improving the correction performance. On the other hand, larger brute-force windows mean that an exponentially larger number of combinations $\mathbf{v} \in \{0,1\}^w$ must be tested, increasing the computational cost of the correction.

To reduce the average computation time while keeping the effectiveness

of our correction procedure, we propose a greedy strategy (Algorithm 6.3, lines 10-15) that performs the brute-force search over increasingly larger windows, until the test yields a strong statistical evidence that the correction was successful (Algorithm 6.3, line 12). When such evidence is found (Algorithm 6.4, lines 7-8), we select the corresponding value $\mathbf{v}$ as the correct one, stop the brute-force search and restart the attack. For instance, the correction might stop when the p-value of the statistical test exceeds a certain threshold. When none of the possible values $\mathbf{v} \in \{0,1\}^w$ is selected, we repeat the search over a larger window.

This strategy reduces on average the time required by our correction procedure, which is very important when the considered attack is computationally expensive. In fact, in most cases the estimated change point $\hat{\tau}$ is very close to the location of the first error τ and thus a small window is enough to correct the error. The window size (and consequently the computational cost) is increased only in the rare cases where the detected change point is far from the first error.

6.3.4 Assumptions

Here we discuss some of the assumptions on which our methodology is based, and how our error detection and correction methods can cope with violations of these hypotheses.

Distribution of the distinguisher. A key assumption of our solution is that the distinguisher values corresponding to the correct key values are i.i.d. realizations of a random variable, which seems reasonable from our observations (see Figure 6.1). This is not always guaranteed in practice, and violations of this assumption might lead to a higher number of false alarms than expected, which we experience in our experiments. However, our correction procedure (Section 6.3.3) can deal with false alarms, thus our methodology works even when this hypothesis is violated.

It is also possible that some sequential attacks provide non-stationary distributions, or even multiple distinguisher sequences. These cannot be directly employed by our error-detection procedure, which assumes that the distinguisher sequence follows a stationary, univariate distribution. Here we show how the proposed methodology can be modified to be applied also in a variety of these cases:

- when the sequence $\{x_t\}_t$ exhibits a trend, e.g. an increase over time, $\{x_t\}_t$ can be pre-processed to remove the trend by means of specific

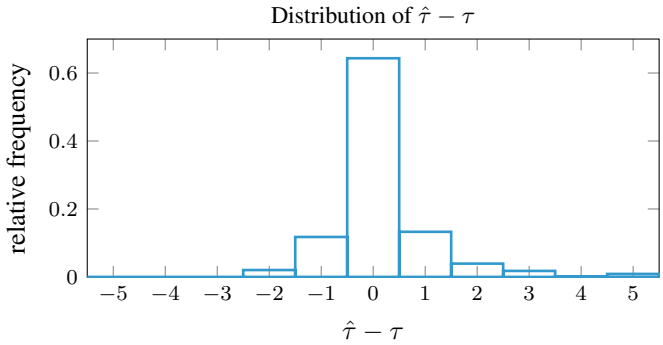


Figure 6.3: *Histogram representing the empirical distribution of the difference between the detected change points $\hat{\tau}$ and the corresponding first error locations τ (excluding false alarms), computed over 2000 H-CPA attacks on simulated data (see Section 6.4.1).*

statistical tools [25, 90]. Then, our error detection procedure can be applied.

- It might happen that the distinguisher in $\{x_t\}_t$ follows different distributions depending on the value of the correct key value, hence the distribution ϕ_0 becomes multimodal. Since the CPM is nonparametric, and therefore can handle data following any distribution [25], our error-detection method can be applied without changes.

- When multiple pieces of hardware or multiple side channel data are analyzed at the same time, there might be several distinguisher sequences at play. In this case, our error detection procedure should be modified to monitor all these sequences in parallel, and to flag a potential error as soon as the CPM finds a change in at least one of the sequences, as in multi-stream monitoring [39–41].

Location of the detected change points. A fundamental assumption of our error-correction procedure is that an estimated change point $\hat{\tau}$ is close to the first error location τ. In case of an inaccurate estimate, we expect $\hat{\tau}$ to be either a few samples before or after the actual first error τ. In Figure 6.3 we illustrate the empirical distribution of the difference $\hat{\tau} - \tau$, which we computed by applying our error-detection procedure to the H-CPA attack [82]. First, we observe that our error-detection procedure is extremely accurate. Figure 6.3 also confirms our intuition that the brute-force windows should contain samples from both sides of $\hat{\tau}$, and confirms the validity of our greedy strategy. In fact, it is very convenient to perform the brute-force search

Algorithm 6.5 Square and multiply always exponentiation (left-to-right)

Input: ciphertext C, key D, modulus n
Output: $M = C^D \bmod n$

```
1: M ← 1
2: for t = 1 : T do
3:     M ← M^2 mod n
4:     if D[t] = 1 then
5:         M ← M · C mod n
6:     else
7:         aux ← M · C mod n
8:     end if
9: end for
10: return M
```

Algorithm 6.6 Multiply and square exponentiation (right-to-left)

Input: ciphertext C, key D, modulus n
Output: $M = C^D \bmod n$

```
1: M ← 1, Q ← C
2: for t = 1 : T do
3:     if D[t] = 1 then
4:         M ← M · Q mod n
5:     else
6:         aux ← M · Q mod n
7:     end if
8:     Q ← Q^2 mod n
9: end for
10: return M
```

over increasingly larger windows, since in most cases a small window is sufficient, and only in rare cases it is necessary to search larger windows. Finally, we remark that the distribution of $\hat{\tau} - \tau$ might depend on the attack, thus this kind of study can be used to define the optimal window size.

6.4 Two Strengthened Sequential Attacks

We applied our error detection and correction methods to strengthen two sequential SCAs targeting the RSA-2048 exponentiation [91]. The first one is a Horizontal Correlation Power Analysis (H-CPA) attack [82] (Section 6.4.1), while the second is a vertical Timing Attack [87] (Section 6.4.2).

6.4.1 Power-analysis attacks

This horizontal attack was introduced in [82] and formalised in [81]. The peculiarity of the *horizontal modus operandi*, which was first proposed in [92], is the use of side-channel data referring to a single execution of the target algorithm. Horizontal attacks are thus very practical, as they require a single execution of the target algorithm and, most importantly, they are by design robust against key blinding, a countermeasure that prevents the attacker from using multiple executions.

As distinguisher, the H-CPA attack uses the correlation between an array $\mathbf{L}_t$ containing the power consumption measured when specific operations are performed at the t-th step of the target algorithm, and the array $\mathbf{H}_t$ containing the Hamming weights of the input and/or output of those

operations [93]. To obtain the array $\mathbf{L}_t$, the attacker needs to perform a pre-processing phase to locate and extract all the relevant power consumption samples from a single power trace, which requires to know exactly the target algorithm and its implementation. We remark that this pre-processing might be extremely hard to perform also when the target implementation is known, because the traces might be jittering and/or the operations within the multiplication algorithm might not produce a visible structure in the power consumption. Nevertheless, we assume a pre-processed power trace to be available, as it is customary in the literature.

In what follows, we describe how to apply the H-CPA attack proposed in [82] to the square and multiply always exponentiation (Algorithm 6.5) and how strengthen it with our error detection and correction methodology. In particular, the attacked operations are the products of the square and multiply always exponentiation (Algorithm 6.5, lines 5,7), whose factors depend on the secret key. The side-channel data $\mathbf{L}_t$ used at the t-th step of the attack consist of the power consumption of the $(t+1)$-th product.

At the t-th step, the attacker computes the possible values of the result variable R, which depend on $\widehat{D}$: when $D[t] = 0$, $M = M_t$, i.e. the output of the square (line 3) and, when $D[t] = 1$, $M = P_t$, i.e. the output of the product (line 5), which define the pair $\widehat{M_t} = (M_t, P_t)$. Then, the attacker simulates, for each value of $D[t]$, the square and the product performed at step $t+1$ of Algorithm 6.5, as shown in Table 6.1, and computes the Hamming weights of the 64-bit digits of the first operand for each simulated product. These are saved in two arrays $\mathbf{H}_t^0, \mathbf{H}_t^1$, associated with the two possible values $\mathbf{v} \in \{0,1\}$ of $D[t]$. Then, the attacker computes the Pearson correlation coefficient, denoted by ϱ, between each array $\mathbf{H}_t^{\mathbf{v}}$ and the array $\mathbf{L}_t$ containing the side-channel data:

$$\mathscr{D}(\mathbf{v}, \widehat{M_t}, \mathbf{L}_t) = \varrho(\mathbf{H}_t^{\mathbf{v}}, \mathbf{L}_t). \tag{6.5}$$

As in (6.1), $\widehat{D}[t]$ is selected as the bit value yielding the highest correlation (Algorithm 6.3, line 3). At step $t+1$, the whole procedure is repeated using the outputs $\widehat{M_{t+1}} = (M_{t+1}(\widehat{D}[t]), P_{t+1}(\widehat{D}[t]))$ predicted at step t (Table 6.1), as in Algorithm 6.3, line 4. After detecting an error by the CPM (Algorithm 6.3, line 6), we start our correction procedure (Algorithm 6.4) with a symmetric brute-force window $\mathbf{W}_{\hat{\tau}}$ of size $w = 3$, which is increased symmetrically by two whenever none of the possible combinations $\mathbf{v} \in \{0,1\}^w$ is selected, up to a maximum window size $w = 9$.

The correction procedure considers windows $\mathbf{W}_<, \mathbf{W}_>$ of size 30 (Algorithm 6.4, line 5) and compares them by the Mann-Whitney test statistic [18], which is designed to detect a shift in the median value of the data

Table 6.1: *The input to the operations computed in step $t+1$ depending on $D[t]$.*

step t		step $t+1$
$M_t \leftarrow (M_t)^2 \bmod n$ $P_t \leftarrow M_t \cdot C \bmod n$	if $D[t]=1$	$M_{t+1}(1) \leftarrow (P_t)^2 \bmod n$ $P_{t+1}(1) \leftarrow M_{t+1}(1) \cdot C \bmod n$
	if $D[t]=0$	$M_{t+1}(0) \leftarrow (M_t)^2 \bmod n$ $P_{t+1}(0) \leftarrow M_{t+1}(0) \cdot C \bmod n$

distribution. We choose the Mann-Whitney test because we observe an evident shift in the median of the distinguisher when an error occurs during H-CPA attacks (see Figure 6.1). When the p-value obtained by the Mann-Whitney test is larger than a fixed threshold h, we assume that there is not enough statistical evidence to claim that the elements of $\mathbf{W}_<$ and $\mathbf{W}_>$ have different distributions, thus $\mathbf{v}$ can be selected as the correct combination. Hence, we stop the brute-force search (Algorithm 6.4, line 7). In case at the end of the brute-force search over the largest window none of the key values $\mathbf{v} \in \{0,1\}^w$ yields a sufficiently large p-value, the combination with the largest p-value is selected as the correct one (Algorithm 6.4, line 11). As an early stopping criterion, in our experiments we set an extremely low value of h, namely $h = 10^{-7}$. Even though unusual in hypothesis tests, we have observed that wrong combinations yield p-values that are orders of magnitude smaller than the correct ones.

Note that computing the Mann-Whitney test statistic is more efficient than repeating the online CPM for each combination, but requires to carry out the attack for at least 30 steps for each combination, to obtain a sufficiently large number of distinguisher values to achieve a statistically significant outcome of the Mann-Whitney test. Performing 30 steps of the attack for each combination tested during the brute-force search is possible because the H-CPA attack is not computationally demanding.

For the sake of simplicity, here we have only described the H-CPA attack targeting the square and multiply always exponentiation (Algorithm 6.5), which can be easily adapted to the multiply and square exponentiation routine (Algorithm 6.6) by changing the pipeline illustrated by Table 6.1 according to the operations performed by Algorithm 6.6. The error detection and correction procedure is exactly the same.

We remark that our strengthened H-CPA attack requires the same preprocessing and hypotheses (in particular, the knowledge of the ciphertext C and of the multiplication algorithm) as the original attack [82]. Hence, it cannot overcome additional countermeasures such as blinding. The aim of our error detection and correction methodology is to improve the suc-

Algorithm 6.7 Sliding window exponentiation

Input: ciphertext C, key D, modulus n
Output: $M = C^D \bmod n$

1: **for** $j = 0 : 7$ **do**
2: $\quad Q[j] \leftarrow (j + 8) \cdot c \bmod n$
3: **end for**
4: $M \leftarrow 1$
5: **for** $t = 1 : T$ **do**
6: $\quad$ **if** $D[t] = 1$ **then**
7: $\quad\quad M \leftarrow M^{16} \bmod n$
8: $\quad\quad i \leftarrow (D[t + 1]D[t + 2]D[t + 3])$
9: $\quad\quad M \leftarrow M \cdot Q[i] \bmod n$
10: $\quad\quad t \leftarrow t + 3$
11: $\quad$ **else**
12: $\quad\quad M \leftarrow M^2 \bmod n$
13: $\quad$ **end if**
14: **end for**
15: **return** M

Algorithm 6.8 Montgomery multiplication

Input: operands A, B, Montgomery parameter r, modulus n, n' s.t. $r \cdot r^{-1} - n \cdot n' = 1$
Output: $P = A \cdot B \cdot r^{-1}$

1: $Q \leftarrow A \cdot B$
2: $P \leftarrow (Q + (Q \cdot n' \bmod r) \cdot n)/r$
3: **if** $P \geq n$ **then**
4: $\quad P \leftarrow P - n$
5: **end if**
6: **return** P

cess rate of the H-CPA attack when the noise in the power measurements introduces errors, preventing the key recovery.

6.4.2 Timing attacks

Timing Attacks were introduced in [78], where the execution time was first used as side-channel data to recover the secret key of a cryptosystem. Here we describe a Timing Attack against the RSA cryptosystem inspired by [87], where the authors break a 512-bit modular exponentiation using a set of execution times. This attack is *vertical* since it analyzes side-channel data gathered from multiple executions of the target algorithm.

The attack exploits the variable-time Montgomery modular multiplication [94] (Algorithm 6.8), which is used as multiplication and squaring algorithm in the sliding-window exponentiation (Algorithm 6.7). The main loop of the Montgomery multiplication creates a temporary result that might be greater than the modulus, depending on the key. A subtraction is executed when that is the case (Algorithm 6.8, line 4), increasing the overall execution time. Therefore, the difference between the execution times of different exponentiations can be a powerful distinguisher.

The attacker selects a set of ciphertexts $\{C^i\}_{i=1}^N$, computes the modular exponentiation with the secret key on the target device, and measures as

side-channel data the set of execution times $\{ET^i\}_{i=1}^N$ corresponding to the selected ciphertexts. The attacker also needs an oracle function that, given the ciphertext, an intermediate result and a hypothetical part of the secret key, simulates the exponentiation algorithm and determines whether, at a particular step, the subtraction (Algorithm 6.8, line 4) has been computed or not. We denote this function by `oracle`. Since the running time might be influenced also by other factors, these attacks usually employ a large number of timing measurements to make the differences due to the conditional subtractions statistically evident. Hence, the effectiveness of the distinguisher grows with the considered number of measurements N.

The attack replicates the structure of Algorithm 6.7, where the bits of the secret key are processed either singularly (when $D[t] = 0$) or in chunks of 4 (when $D[t] = 1$). When $D[t] = 0$, a square is computed (Algorithm 6.7, line 12) and the next 4 operations are certainly squares because a subsequent 1-valued bit in D implies the computation of 4 squares (line 7). An extra square is computed for each 0-valued bit in between, so there are at least five consecutive squares. Therefore, to determine whether $D[t] = 0$, the attacker computes the five squares and uses the oracle to find out whether the final subtraction has been executed at the end of the 5-th square, for each ciphertext in $\{C^i\}_{i=1}^N$. In contrast, when $D[t] = 1$, only four squares (and a multiplication) are computed (lines 7, 9). Therefore, the attacker computes the four squares and a product for each possible combination of $(D[t+1], D[t+2], D[t+3])$, and uses the oracle to find out whether, for each ciphertext in $\{C^i\}_{i=1}^N$, the final subtraction has been executed at the end of the first square after the multiplication (which is computed in any case). We analyze the subsequent square instead of the multiplication since products by constant values (i.e., the elements of Q) have been shown to be less informative [87]. For each admissible value $\mathbf{v} \in \{0, 8, \ldots, 15\}$, of $D[t]$ or of the chunk $(D[t], D[t+1], D[t+2], D[t+3])$, the output of the oracle is used to divide the execution times into two sets $\mathbf{U}_t^{\mathbf{v}}, \mathbf{V}_t^{\mathbf{v}}$, depending on whether the subtraction is computed in the last operation or not:

$$\begin{aligned} \mathbf{U}_t^{\mathbf{v}} &= \{ET^i : \texttt{oracle}(\mathbf{v}, \widehat{M}_t^i, C^i) = \text{true}\}, \\ \mathbf{V}_t^{\mathbf{v}} &= \{ET^i : \texttt{oracle}(\mathbf{v}, \widehat{M}_t^i, C^i) = \text{false}\}. \end{aligned} \tag{6.6}$$

The distinguisher is the difference between the averages of the two sets, denoted by Δ:

$$\mathscr{D}(\mathbf{v}, \widehat{M}_t, \mathbf{L}_t) = \Delta(\mathbf{U}_t^{\mathbf{v}}, \mathbf{V}_t^{\mathbf{v}}). \tag{6.7}$$

Note that the side-channel data $\mathbf{L}_t$ is the same for each step t, and coincides with the whole set $\{ET^i\}_{i=1}^N$. As in (6.1), the value of $\widehat{D}[t]$ (or of

$(\widehat{D}[t], \widehat{D}[t+1], \widehat{D}[t+2], \widehat{D}[t+3]))$ is chosen by maximizing Δ, because the averages of the two sets are expected to be significantly different for the correct guess $\mathbf{v}$. Indeed, the exponentiations corresponding to the times in $\mathbf{U}_t^{\mathbf{v}}$ compute, on average, one more subtraction than the ones corresponding to the times in $\mathbf{V}_t^{\mathbf{v}}$. In contrast, the information provided by the oracle in case of a wrong guess does not correspond to what happens in the actual computations, so the difference of the averages of the two sets is not significant. Before starting a new step, if $\widehat{D}[t] = 1$, t is incremented by 3 (plus the increment of the for loop), as in Algorithm 6.7 (line 10).

When applying our error detection and correction methodology to this Timing Attack, we analyze the distinguisher sequence $\{x_t\}_t = \{\Delta_t\}$ with the online CPM (Algorithm 6.3, line 6). When a change point is detected at $\hat{\tau}$, the correction procedure (Algorithm 6.4) starts with the smallest possible brute-force window $\mathbf{W}_{\hat{\tau}} = \{\hat{\tau}\}$. When the correction procedure is not considered successful, the window size is increased by 1 (alternatively, once to the left and once to the right), up to a maximum window $w = 3$, which means that not all the searched brute-force windows are symmetric.

The correction procedure defines $\mathbf{W}_<$ as the whole sequence $\{x_t\}_t$ obtained before $\mathbf{W}_{\hat{\tau}}$, while $\mathbf{W}_>$ is an open window, including a new sample at each step. The two windows are compared by monitoring their concatenation $(\mathbf{W}_<, \mathbf{W}_>)$ with the same CPM used for error detection. When the CPM finds a new change point that is larger than $\hat{\tau}$, we assume that there is enough statistical evidence to claim that the elements of $\{x_t\}_t$ follow the same distribution before and after the window, thus we stop the brute-force search (Algorithm 6.4, line 7). The correction procedure is then repeated on the new change point. When no change point is detected, the attack is considered successful.

Note that the same statistical test employed by the correction procedure on the H-CPA attack cannot be used for the Timing Attack due to its computational cost. In fact, we have observed that monitoring $(\mathbf{W}_<, \mathbf{W}_>)$ online with the CPM is much more convenient than performing 30 attacking steps and then computing the Mann-Whitney test statistic for each key value tested during the brute-force search.

6.5 Experiments

The goal of our experiments is to demonstrate that our error-detection and correction procedures can effectively and efficiently strengthen different sequential attacks, improving their success rate more than existing error-detection techniques based on thresholds. In particular, we test the H-CPA

attack (Section 6.4.1) on a hardware implementation of the square and multiply always exponentiation (Algorithm 6.5) based on a schoolbook multiplication with an underlying 64-bit multiplier, and on a software implementation of the multiply and square exponentiation (Algorithm 6.6) based on Montgomery multiplication [94] with an underlying 32-bit multiplier. In the Timing attack (Section 6.4.2), we target a software implementation of the sliding window exponentiation with window size 4 (Algorithm 6.7) based on Montgomery multiplication. Since Algorithms 6.5,6.7 parse the private key D *left-to-right*, i.e starting from the most significant bit, in those algorithms we denote by $D[t]$ the bit processed at the t-th step of the algorithm rather than the t-th least significant bit of D, to be consistent with Algorithm 6.2.

6.5.1 Datasets

Power consumption (H-CPA attack). The power traces are noisy signals that can be modeled as

$$\mathbf{L} = \mathbf{p} + \boldsymbol{\eta}, \tag{6.8}$$

where $\mathbf{p}$ is the signal (i.e. the power consumption) and $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \sigma I)$ is uncorrelated Gaussian noise with standard deviation σ. The power of the signal with respect to the noise is usually measured using the *signal-to-noise ratio* (SNR), which is defined as $10 \log_{10}(\mathbb{E}[\mathbf{p}^2]/\sigma^2)$, where $\mathbb{E}[\mathbf{p}^2]$ is the expected value of the squared signal. Since we obtain the power traces for our experiments in ideal conditions, they are virtually noiseless. For this reason, we estimate $\mathbb{E}[\mathbf{p}^2]$ as the average of the squared elements of the power trace, and we synthetically generate the Gaussian noise, adjusting the standard deviation σ to achieve the desired SNR.

In our experiments we consider two different datasets:

- **Simulated dataset.** We simulate the power consumption of two executions of the RSA-2048 exponentiation using the square and multiply always algorithm (Algorithm 6.5) based on a hardware implementation of the schoolbook multiplication (with an underlying 64-bit multiplier). To simulate the two power traces, we started from a *Register-Transfer Level* (RTL) description of the algorithm, and performed a logic synthesis in 40nm at 100MHz (using Synopsys Design Compiler) to obtain a gate-level description. We simulated the power consumption by Synopsys PrimeTime using the maximum allowed resolution (100ps), which yields 100 samples per clock cycle. We simulated measurement noise by generating uncorrelated Gaussian noise and adding it to the traces to achieve SNR $\in \{10, 9, \ldots, -1, -2\}$. We

tested our attack on 250 noisy versions of each power trace for each considered SNR value.

- **Measured dataset.** We acquired the power consumption of ten RSA-2048 exponentiations using a ChipWhisperer®-Pro (CW-1200) mounting, as target microcontroller, an ARM® Cortex®-M4 CPU. The exponentiation was computed with a software multiply-and-square algorithm (Algorithm 6.6) based on Montgomery multiplication [94] and leveraging the CPU 32-bit multiplier. We estimated that measurement noise yields SNR $= 24$, which is unrealistically high, so we add uncorrelated Gaussian noise to the traces to achieve SNR $\in \{7, 6, \ldots, -1, -2\}$, similarly to the simulated dataset. We tested our attack on 50 noisy versions of each power trace per SNR value.

Timing measurements (Timing Attack). The dataset consists in different sets of ciphertexts $\{C^i\}_{i=1}^N$ (with $N = 80,000$, $100,000$, $120,000$, $140,000$, $160,000$) that are the input of an RSA-2048 exponentiation (always using the same private key), along with their respective timing measurements $\{ET^i\}_{i=1}^N$. Each timing measurement refers to the whole execution time of the software sliding window exponentiation on a Cortex®-M7 microcontroller. The more measurements are analyzed, the more effective the attack is, so in this attack N plays a similar role as the SNR in the H-CPA attack.

6.5.2 Figures of Merit

We employ the following figures of merit to evaluate the effectiveness of our error detection and correction methodology in the two considered sequential attacks:

- The *success rate* measures the effectiveness of an attack as the percentage of successful key recoveries over the total number of independent attempts.
- The *number of change points* is the average number of errors detected during a successful attack (including false alarms). This number indicates how often the correction procedure has been successfully activated.
- The *runtime* measures the average time required to carry out a successful attack equipped with our error detection and correction procedure. In our experiments on the H-CPA Attack we used an AMD Ryzen™ Threadripper™ 1950X CPU @ 3.4 GHz, while for the Timing Attack

we used an Intel® Xeon™ E5-2697A v4 CPU @ 2.6 GHz, both with a x86_64 architecture.

6.5.3 Considered Methods

We compare our strengthened attacks against their original counterparts and a variant that adopts an error-detection method based on thresholds inspired by [87–89]. This latter comparison was done only for the H-CPA attack. Here are the details of the three methods we considered.

Original attacks. These are the original H-CPA attack [82] and Timing Attack [87]. These attacks are not equipped with any error detection/correction techniques.

Threshold-based error detection. We compare our solution with the standard error-detection technique based on thresholds presented in several works [87–89]. This rather heuristic procedure monitors the sequence $\{x_t\}_t$ using a fixed threshold Γ, and detects an error as soon as $x_t < \Gamma$. The correction simply consists in flipping the corresponding key bit $D[t]$. To deal with the large number of false alarms of such a simple scheme, error correction is performed only when the distinguisher remains smaller than the threshold for at least 10 steps, as in [88]. Since none of the existing works [87–89] provides insights on how to define the threshold Γ, we assume that the attacker can compute an optimal threshold to separate the distinguisher values before and after the first error. In the H-CPA attack, we define Γ as the optimal separation between ϕ_0 and ϕ_1, which we assume to be Gaussian. While this is certainly not guaranteed in general, preliminary tests show that the Gaussian distribution seems to fit the distinguisher values obtained in our experiments.

Strengthened attacks. These are the H-CPA attack and Timing Attack equipped with our error detection and correction methodology, which are described in Section 6.4.1 and Section 6.4.2 respectively.

6.5.4 Results and Discussion

H-CPA (simulated dataset). As shown in Figure 6.4(a), our methodology can successfully improve the success rate of the H-CPA attack on simulated power traces. In particular, our solution achieves substantially higher success rates than the original counterpart and the attack equipped with the threshold-based error-detection method. This is particularly true when the

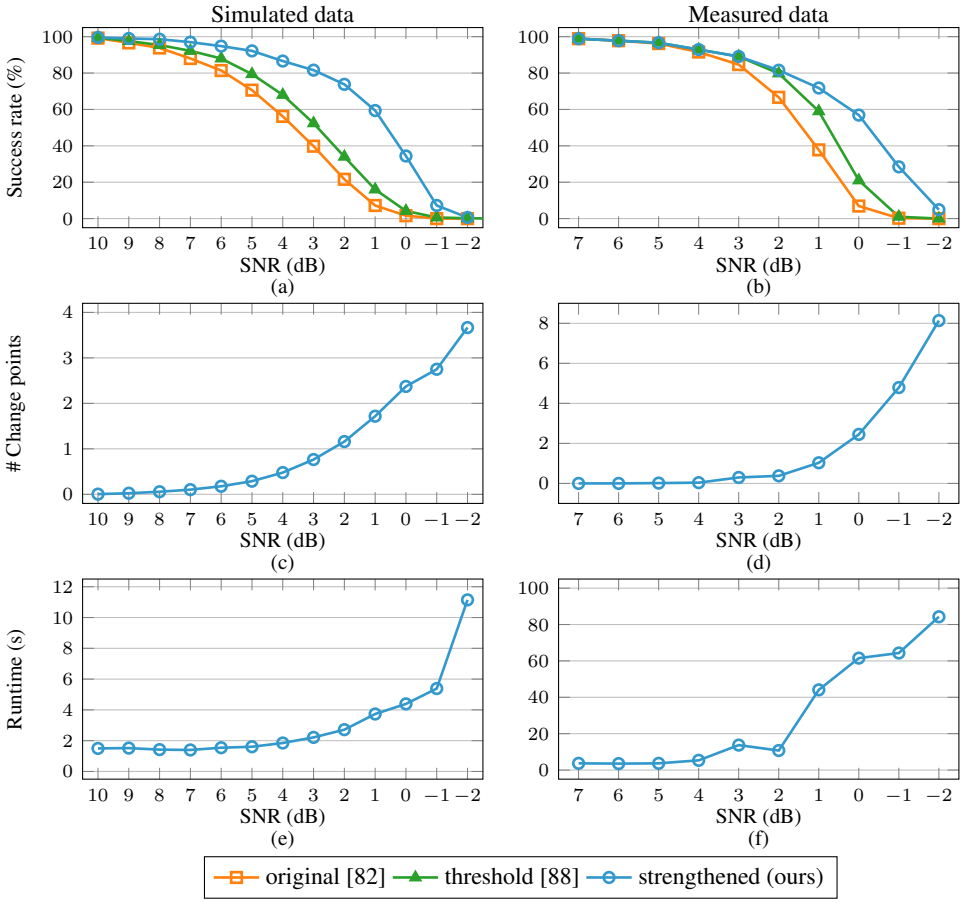


Figure 6.4: (a,b) *Success rate of the original H-CPA attack, the attack equipped with threshold-based error-detection [88], and strengthened by the proposed methodology.* (c,d) *Average number of change points detected by our methodology.* (e,f) *Average runtime of the successful attacks with our error detection and correction methodology. All these quantities are presented as a function of the SNR of the input power traces, which have been tuned by adding white Gaussian noise.* (a,c,e) *show the results obtained by the H-CPA attack on simulated data, while* (b,d,f) *show the results of the attack on the dataset measured by means of ChipWhisperer®.*

SNR is low. For instance, when SNR $= 1$, the original attack succeeds only 7.2% of the times, the attack equipped with the threshold-based error-detection succeeds 16% of the times, while our strengthened attack achieves 59.4% success rate. The success rate of all the considered attacks decreases with the SNR, but our error detection and correction method guarantees a slower decay. Figure 6.4(c) shows that reducing the SNR increases the number of errors to be corrected, and this in turns increases the runtime (Figure 6.4(e)) since the correction procedure must be activated more of-

ten. Comparing the average runtime of the attack at SNR $= 10$ (where the average number of detected change points is ≈ 0.004) with those obtained at lower SNR, we observe that our error detection and correction procedure does not significantly increase the computational cost of the attack.

H-CPA (measured dataset). The results on measured power traces are in line with those obtained over the simulated dataset: Figure 6.4(b) shows that our strengthened attack outperforms the two alternatives, especially when the SNR is low. For instance, when SNR $= 0$, the success rate of the original attack is 6.9%, the attack equipped with the threshold-based error-detection technique succeeds 21% of the times, while our strengthened attack succeeds 56.9% of the times. Consistently with the results discussed above, the success rate of the three attacks decreases with the SNR, but the decay is slower for our strengthened attack. Also in this case, at lower SNR values, the number of errors increases (Figure 6.4(d)), together with the runtime since the correction procedure has to be executed more often (Figure 6.4(f)). We remark that the results of the H-CPA attack on the measured and simulated datasets obtained with the same SNR values cannot be directly compared, since measured and simulated traces might provide different correlation levels with the Hamming weights due to the different implementations (hardware/software) of the RSA exponentiation. For this reason, the two experimental settings yield different success rates, numbers of change points and, consequently, runtime, even though the attacking procedures are very similar.

Timing Attack The success rate of the strengthened Timing Attack (Figure 6.5(a)) confirms the effectiveness of our error detection and correction procedure, which makes the attack possible at a relatively low number of measurements. The strengthened attack achieves 96% success rate when $N = 160,000$, while the original attack could never recover the entire secret key for any considered value of N (0% success rate). As expected, the success rate tends to decrease with the number of timing measurements, as this plays a similar role as the SNR in the H-CPA. Figure 6.5(b) shows that, in line with the H-CPA attack, the number of change points grows when N decreases. Figure 6.5(c) also shows that the overall runtime rapidly increases when reducing the value of N. This is due to the large number of times the correction procedure must be executed when few measurements are used (see Figure 6.5(b)).

The original Timing Attack never succeeds in the considered experimental settings. As a comparison, we tested the original Timing Attack on

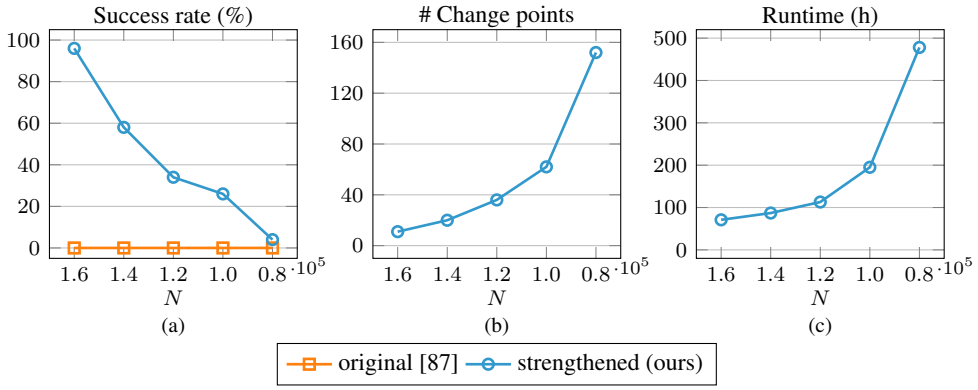


Figure 6.5: (a) *Success rate of the original Timing attack and of the attack strengthened by the proposed methodology.* (b) *Average number of change points detected.* (c) *Average runtime of the successful strengthened attacks. All these quantities are presented as a function of the number N of execution time measurements available in the training set TR. We observe that the original Timing attack never succeeds at the considered values of N.*

RSA-1024, and we found out that more than $N = 370,000$ measurements were required to succeed. Since the number of measurements required to recover a secret key grows with the length of the key, a 2048-bit key would need an even higher number of timing measurements, making the attack impractical. Thanks to our error detection and correction methodology, the strengthened attack achieves high success rates even with a limited number of measurements.

In conclusion, our experiments indicate that our error detection and correction procedure can significantly strengthen the considered attacks, increasing the robustness with respect to noise in H-CPA attacks, and reducing the number of measurements required by the Timing Attack. The only alternative solution that can be applied to any sequential attack simply sets a threshold on the distinguisher, and yields only marginal improvements over the original attacks. These findings confirm that an error detection and correction procedure based on a sound change-detection test perform substantially better than heuristic solutions based on thresholds, which cannot cope with false alarms and inaccurate estimates of the first error location. We finally remark that our methodology does not resolve intrinsic limitations of the attack to be strengthened, and therefore viable countermeasures (e.g. message blinding for the H-CPA) should be employed even when the device is considered safe due to measurement noise or the large number of side-channel data required.

Part II

Anomaly Detection in Point Clouds

CHAPTER 7

Related Literature

In this chapter we survey the literature on unsupervised anomaly detection and open-set recognition. In contrast with the first part of the thesis, here the goal is to determine whether a single instance x is normal or anomalous. First, in Section 7.1 we review the most relevant anomaly detection methods based on traditional machine learning (Section 7.1.1) and deep learning (Section 7.1.2). Then, in Section 7.2 we present the most recent works addressing open-set recognition.

7.1 Anomaly Detection

Unsupervised anomaly detection is a challenging machine learning problem that has been addressed both by traditional machine learning and deep learning models. A more comprehensive survey on anomaly detection algorithms can be found in [95].

7.1.1 Traditional Machine Learning

Early approaches assume normal data to be realizations of random vectors following a certain distribution ϕ_0, and construct a model $\widehat{\phi}_0$, for instance

by *Kernel Density Estimation (KDE)* [96], or by fitting a Gaussian [97] or Gaussian Mixture model [98] on normal data from the training set using the *Expectation Maximization* algorithm [99]. During testing, these methods compute the likelihood of new samples with respect to $\widehat{\phi}_0$, and identify as anomalous those that can be considered outliers according to their likelihood.

More recently, anomaly detection has been addressed as a *one-class classification* problem, where the aim is to find a boundary in the input space to enclose the normal data from the training set. This is typically done by solving an optimization problem, as in *One-Class Support Vector Machine (OC-SVM)* [100], which uses a linear boundary, and *Support Vector Data Descriptor (SVDD)* [101], which uses a spherical boundary.

Other approaches measure the distance between a new sample and the normal elements from the training set, and detect anomalous samples by counting the number of samples from the training set within a given distance from the test sample [102, 103]. By doing so, these methods aim at capturing the region of the input space in which normal samples are more concentrated, leveraging the mutual distance between training samples. Another solution consists in constructing an ensemble of binary-trees partitioning the input space, as in *Isolation Forest (IFOR)* [104], and detecting anomalies by measuring the average height of the trees at which a test sample gets "isolated", i.e. belongs to a subset of the input space in which there are no training samples.

The main drawback of these solutions is that they cannot be directly applied to high-dimensional data such as signals, images, or point clouds due to the *curse of dimensionality* [105]. To employ these machine learning methods to high-dimensional data, it is necessary to extract lower-dimensional features from the raw data, for instance by PCA [71] or *sparse representations* [106], and then use these features as input.

7.1.2 Deep Learning

Contrarily to traditional machine learning methods, the increasingly popular deep learning methods for anomaly detection can directly be applied also on high-dimensional data. The most popular anomaly detection solutions for signals and images are probably *autoencoders*, in which a first neural network, called the *encoder*, extracts a low-dimensional *latent representation* from the input signal, and a second neural network, called the *decoder*, takes as input the latent representation and reconstructs a signal having the same size as the original input. The two networks are jointly

trained on normal samples to minimize the difference between the input and the output of the decoder. Then, a test sample is identified as anomalous if the reconstruction error of the autoencoder on that sample is larger than a threshold [107–109], since the autoencoder will be able to accurately reconstruct only normal samples. In [110], the autoencoder is trained to produce a latent representation that follows a Gaussian Mixture distribution, and anomalies are detected by computing the likelihood with respect to that distribution.

Some works address anomaly detection using deep versions of traditional machine learning methods such as OC-SVM [111] and SVDD [112–114]. This is typically done by training a deep neural network to produce a low-dimensional latent representation of the input instances such that normal samples seen during training have latent representations that are all enclosed, for instance, in a hypersphere [112]. In particular, in [112] the network is trained to minimize the distance between the latent representation of the normal samples in the training set and a center defined as the output obtained by randomly initializing the network. The main drawback of these solutions is *mode collapse*, namely the fact that the network might learn a trivial solution that assigns all samples the same latent representation, which coincides with the center of the hypersphere. By definition, this solution would minimize the loss, even though it would be completely useless to distinguish normal and anomalous samples. To mitigate this problem, it is possible to add noise to the loss [115]. As an alternative, more sophisticated methods [114] define the boundary enclosing normal samples using both the training data and anomalies generated by adding noise to normal samples. Somewhat similarly, some works [116–118] address anomaly detection using a *Generative Adversarial Network (GAN)* [119], which jointly trains an autoencoder and another neural network, called the *discriminator*, to distinguish between samples from the training set and samples generated by feeding the decoder with random latent representations.

Another way to learn the characteristics of normal data is to train a deep neural network to solve a classification problem using *surrogate labels* and, for this reason, this approach is usually referred to as *self-supervised*. These surrogate labels are typically generated by applying n geometric transformations $f_0, \ldots, f_{n-1}$ (e.g., rotations and translations) to normal data (typically, images) [7, 120], and the classification problem consists in recognizing which transformation has been applied. The same transformations are then applied to each test sample x, which is identified as anomalous if the network cannot accurately classify which transformations have been applied. In particular, in [7] the network is a CNN, and the anomaly score

is defined as

$$\mathcal{T}(x) = \frac{1}{n}\sum_{i=0}^{n-1}\left[\text{CNN}\big(f_i(x)\big)\right]_i , \tag{7.1}$$

where $\text{CNN}(\cdot)$ indicates the maximum classification score of the network. Although very effective, these self-supervised methods rely on transformations that are specifically designed for normal data (typically images), and might not be suitable for other types of data. For instance, some rotations might be hard to recognize when the data samples are characterized by certain symmetries. This issue can be addressed by training the network to learn domain-specific transformations to extract meaningful features from normal data, as in [121, 122].

A challenging problem that is strongly related to anomaly detection is the identification of anomalous regions within images. Typically, this problem has been addressed by extending anomaly-detection algorithms to produce a *local anomaly score*, i.e. an anomaly score defined on each pixel of the image. A simple yet effective solution involves autoencoders [10, 123], whose reconstruction error can be directly used as a local anomaly score. Other solutions leverage pre-trained CNNs as feature extractors. A prominent example is the *Student-Teacher* approach [124], where a CNN (pre-trained on natural images), referred to as the Teacher, is used to extract features from image patches, and a set of Student networks are trained on anomaly-free patches to estimate the Teacher's output in a regression problem. During testing, the local anomaly score is computed by combining the regression error and the prediction variance of the Student networks. The intuition is that the Students can extract similar features compared to the Teacher only from normal patches. It is also worth mentioning the solutions based on *inpainting* such as [125], where the images are transformed by taking out some patches, and a CNN is trained to reconstruct the missing parts.

The problem of detecting anomalous regions is out of the scope of this thesis. Nevertheless, it is an interesting anomaly detection problem with relevant applications that has been widely studied on images. However, this challenge is starting to attract some interest also in the point-clouds research community [126].

7.2 Open-Set Recognition

The open-set recognition problem has been introduced by Scheirer et al. [127] to address classification in a realistic scenario where only part of

the classes have been already identified and included in the training set, and anomalous samples from unknown classes must be detected during testing. A comprehensive survey on open-set recognition methods can be found in [128]. The first open-set recognition methods are based on traditional machine-learning algorithms. Scheirer et al. [127] use modified Support Vector Machines with decision boundaries designed to reject unknown samples. Other methods detect novelties using the distance of test samples from the centroids of the known classes [129], or the reconstruction error of sparse representations [130].

7.2.1 Classification Scores and Latent Representations

More recently, deep open-set recognition methods started to gain more and more attention due to the outstanding results achieved by deep learning in most classification and pattern-recognition tasks. A straightforward way to extend a CNN trained for closed-set classification to open-set recognition consists in identifying as unknown those test samples on which the CNN is uncertain. These samples are typically identified by analyzing the maximum classification score [131], or the Shannon entropy of the classification scores vector [132]. Bendale et al. [133] propose the OpenMax function to replace SoftMax as the last layer of a CNN trained on the known classes at test time. In particular, during testing, OpenMax evaluates the distance between the CNN score vectors and the *mean activation vectors* (MAVs), computed using the score vectors of training samples. Each MAV represents a known class, and a test sample is detected as a novelty when the likelihood of its distance from all the MAVs with respect to a Weibull distribution model fitted using the training set is below a certain threshold. Cevikalp et al. [134] propose deep classifiers using polyhedral conic boundaries to separate instances from different known classes, instead of the traditional linear boundaries. This makes the acceptance regions of the known classes more compact, thus easing anomaly detection.

Another approach consists in applying an outlier-detection method to the latent representation of a deep classifier trained on known classes [135, 136]. Instances from known classes are expected to be mapped in the same region of the latent space, following a multimodal distribution. Therefore, it is possible to detect novelties from their latent representations as outliers with respect to this distribution. To this purpose, Zhu et al. [135] employ a variant of IFOR [104], while Zhang et al. [136] define confidence intervals over each component of the latent space. Socher et al. [137] obtain a different latent representation by embedding the images into a semantic word

space associated with the class labels. Then, they fit an isometric Gaussian model to represent each known class in the semantic space and use the likelihood as novelty score.

7.2.2 Reconstruction and Generative Models

Similarly to anomaly detection (Section 7.1.2), autoencoders have been gaining more and more attention also for deep open-set recognition in images [138–140]. The idea is to train a deep autoencoder to extract a compact latent representation from the input images and then reconstruct them. Moreover, a second branch of the network is jointly trained to classify the input images from known classes starting from the latent representation of the autoencoder. Thus, these solutions provide both classification scores for the known classes and an anomaly score based on the reconstruction error of the autoencoder. As in anomaly detection (Section 7.1.2), GANs can be used to generate anomalous (counterfactual) samples during training [141–143]. These generated samples are assumed to be "just outside of the known class boundaries" [142], and the discriminator is trained to classify known and generated samples. During testing, the discriminator is directly used for open-set recognition.

Although a variety of sophisticated open-set recognition methods can be found in the literature, a recent study by Vaze et al. [144] suggests that the performance of a simple baseline such as [131] can be substantially increased by improving the underlying closed-set classifier. In light of this, a fair comparison between the performance of different open-set recognition methods would require that their closed-set classification accuracy is approximately the same.

CHAPTER 8

Composite Layers for 3D Point Clouds

In this chapter, we address the anomaly-detection problem in 3D point clouds. In particular, we define the *composite layer*, a novel operator replacing point-convolutions, namely convolutions defined on point clouds. Compared to existing point-convolutional layers, the composite layer guarantees more flexibility in terms of number of parameters and design, encompassing both convolutional and non-convolutional operators. Our experiments on synthetic and real-world point clouds show that our *CompositeNets*, deep neural networks based on composite layers, outperform ConvPoint [145] and achieve similar classification accuracy to KPConv [146] despite having a much simpler architecture. Most remarkably, we are among the first to train deep neural networks for anomaly detection in point clouds, training CompositeNets in a self-supervised fashion [7]. Our experiments demonstrate that the self-supervised approach outperforms deep and shallow baselines. Also in this case, our CompositeNets outperform ConvPoint and achieve similar results to KPConv, trained in the same fashion, despite having a much simpler architecture.

In Section 8.1 we review the literature on machine learning for 3D point clouds, with a particular focus on unsupervised models, and in Section 8.2 we illustrate point-convolutional layers and how they have been im-

plemented in the literature. In Section 8.3 we present our composite layers and in Section 8.4 we demonstrate their superior design flexibility compared with existing point-convolutional layers. Finally, in Section 8.5 we illustrate and discuss our experiments.

8.1 Machine Learning on Point Clouds

3D deep learning has recently become one of the most studied branches of computer vision, and the design of deep neural networks (DNNs) able to process point clouds is a challenge that has been attracting more and more interest. Point clouds are unordered sets of points in $\mathbb{R}^3$, typically acquired by LiDAR sensors or depth cameras, which are sometimes paired with additional features such as colors or the normal vector to a surface. Point clouds are very informative and provide a compact yet detailed representation of a 3D object. Not surprisingly, they have been widely used in autonomous driving [147], topography [148], architecture and heritage preservation [149], and are now a popular 3D data format. Starting from *PointNet* [150], several DNNs processing point clouds have been proposed. Such a flourishing literature is motivated by the intrinsic challenges of training machine learning models on point clouds, which are primarily due to the peculiar structure of point clouds. Unlike images, point clouds cannot be arranged over the regular grids where traditional convolutional layers are defined, thus require special layers such as the *composite layer* we present in this chapter.

The earliest deep learning solutions coping with the scattered nature of point clouds project 3D input on several planes [151, 152], generating 2D images to be processed by traditional CNNs. In *volumetric CNNs* [151, 153], 3D point clouds are instead mapped to a voxel grid, and then processed by 3D convolutional filters. Both approaches imply a loss of information. Moreover, the computational cost of volumetric convolutions scales very poorly with the grid resolution. *Submanifold sparse convolutional networks* [9] represent an efficient alternative that can handle higher-resolution voxel grids by leveraging the sparse nature of point clouds.

In what follows we overview deep learning techniques that operate directly on 3D point clouds without projections or voxelization, with a focus on operators that extend convolutions to point clouds. A more comprehensive review on deep learning for 3D point clouds can be found in [154]. We refer to Chapter 7 for an overview of the most relevant algorithms for anomaly detection, which is the main task we address in this thesis.

8.1.1 Deep Learning on Point Clouds

Early approaches. *PointNet* [150] was the first DNN directly processing 3D point clouds using Multi-Layer Perceptrons (MLPs) and permutation-invariant pooling. The main advantage of this approach is that the output of the network does not depend on the order of the points. This is crucial since two point clouds containing the same points in a different order should be considered equivalent. The major drawback is that PointNet feeds the entire point cloud to the same MLP and therefore fails at recognizing local structures inside the point cloud. A straightforward improvement implemented in *PointNet++* [155] is to hierarchically apply PointNet to nested partitions of the point cloud.

Point convolutions. Due to the success of CNNs, recent works have defined convolutional layers operating like filters on 2D images. In this direction, volumetric convolutions [156] are defined by interpolating a 3D function with the input point cloud. Other methods approximate continuous convolutional filters by MLPs [157, 158] or polynomials [159].

A wide variety of point-convolutional layers have been proposed in the literature. The vast majority of these layers define convolution using either *Radial Basis Function Networks* (RBFNs) or *Multi-Layer Perceptrons* (MLPs). The most representative solution following the first approach is *KPConv* [146], which uses a RBFN to define a tensor of weights combining the spatial coordinates of the points in a neighborhood (relatively to the position of the output point) and the input features. KPConv and its extension *AGMMConv* [160] also implement a deformable convolutional kernel based on a learnable shift function that translates the RBFN centers depending on the output point where the kernel is applied. *InterpCNN* [161] is equivalent to KPConv, the only difference being the chosen RBFs. *FPConv* [162] follows a slightly different approach based on the fact that point clouds usually represent locally flat surfaces. In particular, FPConv locally projects 3D points to the estimated tangent plane to the surface, and then applies 2D convolution whose weights depend on the distance between the projected points and centers placed on a regular grid, similarly to KPConv.

A representative layer defined by MLPs is *ConvPoint* [145]. This operator separates feature and spatial operations, the latter being handled by a MLP that takes as input the coordinates of the points in X_y relatively to a set of centers, which play a similar role to the RBFN centers in the layers described above. Then, the MLP outputs are linearly combined with the input features. Also *RS-CNN* [163], *RandLa-Net* [164], and *RPNet* [165] follow

the same approach, with the difference that the MLP takes as input not only the relative position of the points with respect to the output, but also their absolute position in the point cloud, and thus are not convolutional strictly speaking. RandLa-Net [164] also makes use of attention-based pooling and *ad-hoc* residual blocks to handle large-scale point clouds. *DensePoint* [166] is similar to RS-CNN, the main difference being that the feature vectors associated with the points in the convolution window are transformed by a shared single-layer perceptron, independently from the relative position of the points with respect to the output point.

The structure and the number of parameters of KPConv, ConvPoint, and the other point-convolutional layers depend on the number of centers. In contrast, our composite layer enables more flexibility since spatial and feature operations are defined independently. Moreover, composite layers can aggregate spatial information and features by more general operations than linear combinations. To the best of our knowledge, the only layer that separately processes the coordinates of the points and then shares the spatial information among the output features is *PAConv* [167]. However, we do not consider this in our experiments because its convolutional kernel is defined by a sophisticated neural network and depends on the absolute position of the points, differently from convolutions.

Graph neural networks. Another approach consists in processing the point clouds by *Graph neural networks* (GNNs) [168–173], which can be considered an extension of PointNet. In particular, a GNN transforms a point cloud into a graph whose edges are defined by the Euclidean distance between the points. A relevant example is DGCNN [169], where a graph is defined by connecting a number of randomly selected points in the input point cloud to their nearest neighbors to take into account local patterns. The major difference between GNNs and point-convolutional networks is that, in GNNs, the coordinates of the points are considered as features associated with the graph vertices and thus are not preserved in the network layers. In contrast, point-convolutional networks modify only the features, e.g. the color associated to a point, while preserving the spatial information, i.e. the coordinates of the points.

8.1.2 Unsupervised Learning over Point Clouds

All the operators above have only been employed in DNNs designed for supervised tasks such as classification and semantic segmentation, and much fewer works address unsupervised learning on point clouds. Among these,

the vast majority only tackle very specific industrial monitoring problems such as detecting defective products in additive manufacturing [174]. Other works apply unsupervised clustering to recognize vehicles in autonomous driving [175], or One-Class SVM to identify geometric structures [148, 176] such as poles [176] from point clouds acquired by LiDAR. In general, all machine-learning solutions for anomaly detection in point clouds rely on hand-crafted features also referred to as *point cloud descriptors* [177]. However, as we show in our experiments, these descriptors do not generalize well for anomaly detection. In contrast, the proposed DNNs perform anomaly detection by extracting data-driven features that can effectively describe a wider variety of point clouds.

Recently, unsupervised deep learning on 3D data has been drawing more and more interest for pre-training [178, 179] and domain adaptation [180–182] techniques. In fact, annotated datasets of 3D point clouds are not as large and easy to acquire as 2D image classification datasets. To the best of our knowledge, the only DNN for anomaly detection on point clouds is a variational autoencoder (VAE) [183], while the other autoencoders for point clouds [184–186] have not been employed for anomaly detection yet.

8.2 Point Convolutions

Before introducing the proposed composite layer and illustrating how to implement CompositeNets, we present some more details on point convolutions. We start by defining the output points and the corresponding convolution windows, i.e. the subsets of the input point clouds on which point-convolutions are applied (Section 8.2.1). Then, we formally define the point-convolutional operator and illustrate how it is implemented in ConvPoint [145] and KPConv [146] (Section 8.2.2). We recall that the vast majority of the point-convolutional layers in the literature are very similar to either ConvPoint or KPConv. Therefore, these two layers can be considered as representatives for point convolutions.

8.2.1 Convolution Window and Output Point Cloud

2D convolutional filters operate on *convolution windows*, namely small portions of the input image selected in a sliding window fashion, and return as output a pixel value for each window. While the grid structure of 2D images makes these operations straightforward, when working on point clouds there are multiple ways to define both the output points $\mathbf{q} \in Q$ and the corresponding windows $X_{\mathbf{q}} \subseteq P$ in the input point cloud.

Typically, point-convolutional layers first select each output point $\mathbf{q} \in Q$ and then define the convolution window $X_{\mathbf{q}}$. A popular approach [145,156] to select $\mathbf{q}$ is random sampling with replacement from P, which implies that $Q \subseteq P$. As in [145], we set a lower sampling probability to points $\mathbf{q} \in P$ that have already been drawn or that are close to points in Q. When stacking multiple point convolutions, the cardinality of the point cloud can be reduced by sampling fewer output points than there are in the input point cloud, i.e. $\#Q < \#P$, which is similar to having a stride in 2D convolution. Pooling [146] is an alternative way to reduce the point cloud cardinality.

The convolution window $X_{\mathbf{q}}$ can be defined either as a sphere $X_{\mathbf{q}} := \{\mathbf{p} \in P : \|\mathbf{p} - \mathbf{q}\| < \rho\}$ [146], or as a set containing a fixed number of nearest neighbors of $\mathbf{q}$ in P. We follow this second approach since it produces windows containing a fixed number of points, making point convolution more efficient and easier to implement. The two methods coincide when the density of the points in P is uniform [187].

8.2.2 Point-convolutional Operators

After defining the output $\mathbf{q}$ and the convolution window $X_{\mathbf{q}}$, the point-convolutional operator [146, 158, 159] is defined at each $\mathbf{q} \in Q$ as:

$$\mathcal{G}_j(\mathbf{q}) = (\mathcal{F} * g_j)(\mathbf{q}) = \sum_{\mathbf{p} \in X_{\mathbf{q}}} \sum_{i=1}^{I} \mathcal{F}_i(\mathbf{p}) g_{ij}(\mathbf{p} - \mathbf{q}), \tag{8.1}$$

where $\mathcal{F}_i$ is the i-th input feature and $\mathcal{G}_j$ is the j-th output feature, for $j \in \{1, \dots J\}$. Similarly to traditional CNNs, each convolutional layer stacks J *filters* $g_j : \mathbb{R}^d \to \mathbb{R}^I$, each having I components g_{ij} like the input feature function $\mathcal{F}$. When handling images, the filters are defined as a matrices whose elements correspond to specific pixels of input image. In contrast, point clouds lack such grid structure, and therefore the filter g_j is a function defined over the entire space $\mathbb{R}^d$.

Convolutional filters for images are learnable weight matrices having the same size as the convolution window. Therefore, each weight is associated to a spatial location and is multiplied to the features associated with the pixel in the corresponding location of the convolution window. When operating on images, the spatial locations of weights and pixels is embedded in the grid structure of the image. In contrast, point clouds lack such grid structure, and therefore the filter g_j is a function defined over $\mathbb{R}^d$.

As stated in Section 8.1.1, the vast majority of the point-convolutional layers in the literature operate very similarly to either ConvPoint [145] or

KPConv [146], which we illustrate here more in detail. Both these layers implement g_{ij} as a linear combination:

$$g_{ij}(\mathbf{p}-\mathbf{q}) = \sum_{m=1}^{M} \tilde{w}_{ijm} H_m(\mathbf{p}-\mathbf{q}), \tag{8.2}$$

where $\tilde{w}_{ijm}$ are learnable weights associated to a set of spatial locations $\{\mathbf{c}_m\}_{m=1}^M \subset \mathbb{R}^d$, called *centers*, and H_m are *correlation functions* that depend on the relative position of $\mathbf{p}-\mathbf{q}$ with respect to $\mathbf{c}_m$ [146]. This formulation is inspired by convolution on images, where $\mathbf{p}-\mathbf{q}$ represent the pixel locations in the convolution window, $\{\mathbf{c}_m\}$ represent the spatial positions of the weights, and H_m is the Kronecker function $H_m = \delta(\mathbf{p}-\mathbf{q}, \mathbf{c}_m)$, i.e. $\delta(\mathbf{p}-\mathbf{q}, \mathbf{c}_m) = 1$ if $\mathbf{p}-\mathbf{q} = \mathbf{c}_m$ and 0 otherwise [145]. Due to the irregular disposition of the points in point clouds, in point-convolutional layers H_m must be a continuous function, and the positions of the centers either belong to a regular grid [146] or are learnable parameters [145]. In ConvPoint, the correlation functions H_m are defined by MLPs, namely $H_m(\mathbf{p}-\mathbf{q}) = \text{MLP}_m\left([(\mathbf{p}-\mathbf{q}) - \mathbf{c}_m]_{m=1}^M\right)$, where square brackets denote concatenation. In KPConv, each H_m is an RBF, namely a function $H_m(\mathbf{p}-\mathbf{q}) = \hbar(\|(\mathbf{p}-\mathbf{q}) - \mathbf{c}_m\|)$ that differs from the others only by the parameter $\mathbf{c}_m \in \mathbb{R}^d$.

In Figure 8.2(b) we illustrate the operations of the point-convolutional layers ConvPoint [145] and KPConv [146] in terms of matrix multiplications. The linear combination in (8.2) can be expressed as a multiplication between the weight tensor $\widetilde{W} = (\tilde{w}_{ijm})$ and the matrix H stacking the the correlations $H_m(\mathbf{p}-\mathbf{q})$ for $\mathbf{p} \in X_\mathbf{q}$ and $m \in \{1,\dots,M\}$. Then, each output feature $\mathcal{G}_j(\mathbf{q})$ can be expressed as the Frobenius inner product between the matrix $\mathcal{F}(X_\mathbf{q})$ containing the features $\mathcal{F}_i(\mathbf{p})$ for $\mathbf{p} \in X_\mathbf{q}$ and $i \in \{1,\dots,I\}$ and the j-th slice of $\widetilde{W}H$.

8.3 Composite Layers

Here, we introduce our composite layers for point cloud processing in DNNs. Our composite layer is defined by composing a *spatial function* $\mathcal{s} : \mathbb{R}^s \to \mathbb{R}^K$, which extracts the spatial information from the coordinates of the points in $X_\mathbf{q}$, and a *semantic function* $\mathcal{A}(\mathcal{F}, \mathcal{s}) : \mathbb{R}^s \to \mathbb{R}^J$, which aggregates the output of $\mathcal{s}$ with all the input features $\mathcal{F}(\mathbf{p})$, for $\mathbf{p} \in X_\mathbf{q}$, to produce the output feature vector $\mathcal{G}(\mathbf{q})$ (see Figure 8.1):

$$\mathcal{G}_j(\mathbf{q}) = \mathcal{A}_j(\mathcal{F}, \mathcal{s})(\mathbf{q}). \tag{8.3}$$

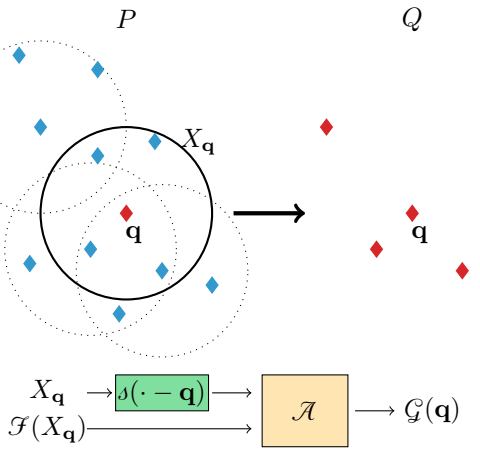


Figure 8.1: *Our composite layer: the spatial function $\mathfrak{s}$ outputs a vector in $\mathbb{R}^K$ for each point $\mathbf{p}$ of the neighborhood $X_{\mathbf{q}}$ of the output point $\mathbf{q}$. The semantic function combines the input features $\mathcal{F}(X_{\mathbf{q}})$ and $\{\mathfrak{s}(\mathbf{p}-\mathbf{q})\}_{\mathbf{p}\in X_{\mathbf{q}}}$ to produce the output features $\mathcal{G}(\mathbf{q})$.*

This formulation provides additional regularization and guarantees superior design flexibility compared to the existing point-convolutional layers, as we empirically show in Section 8.4. Here we first illustrate our convolutional composite layer and compare it to the point-convolutional layers used in KPConv [146] and ConvPoint [145] (Section 8.3.1). Then, we present our aggregate composite layer (Section 8.3.2), which aggregates spatial information and features in a non-convolutional manner.

8.3.1 Convolutional Composite Layer

Our *convolutional composite layer* replicates point convolution (8.1) by our composite formulation (8.3). Following the approach of ConvPoint, we randomly sample the output points $\mathbf{q} \in Q$ from the input point cloud P, and define the convolution windows $X_{\mathbf{q}}$ by nearest neighbors. This choice guarantees that all the windows contain the same number of points, making point convolution easier to implement.

Spatial Function. Inspired by [146, 156], we implement the spatial function $\mathfrak{s}$ using a Radial Basis Function Network (RBFN) with M centers, which in principle has the same approximation capability of MLPs [188] but depends on fewer hyper-parameters. We define the k-th component of $\mathfrak{s}$ as:

$$\mathfrak{s}_k(\mathbf{p}-\mathbf{q}) = \sum_{m=1}^{M} v_{km}\, h(\|(\mathbf{p}-\mathbf{q}) - \mathbf{c}_m\|), \tag{8.4}$$

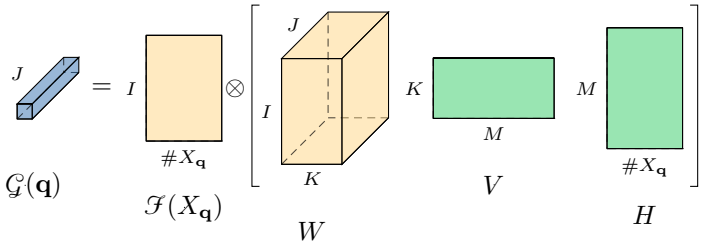
where v_{km} are learnable parameters, for $k \in \{1, ..., K\}$. We remark that the input of each $\mathfrak{s}_k$ is the relative position of each point $\mathbf{p} \in X_\mathbf{q}$ with respect to $\mathbf{q}$, as for the filters g_{ij} of point convolution (8.1). Each RBF term in (8.4) differs from the others only by the parameter $\mathbf{c}_m \in \mathbb{R}^d$, called ***RBF center***, which translates the origin for the RBFs in the d-dimensional space. Here we employ a Gaussian $\hbar(r) = \exp(r^2/2\sigma^2)$, where $r = \|(\mathbf{p} - \mathbf{q}) - \mathbf{c}_m\|$, σ is a hyper-parameter common to all RBFs, as in [156], and we learn the position of the centers $\{\mathbf{c}_m\}$ during training. The spatial function can be expressed as a multiplication between the matrix H, containing the RBF output $\hbar(\|(\mathbf{p}-\mathbf{q})-\mathbf{c}_m\|)$ for $\mathbf{p} \in X_\mathbf{q}$ and $m \in \{1, \ldots, M\}$, and a learnable weight matrix $V = (v_{km})$ (see Figure 8.2(a)). Thus, the spatial function compresses the spatial information by projecting each column of H to a K-dimensional space, where $K < M$ is a hyperparameter.

Semantic Function. To produce an equivalent formulation of point convolution (8.1) from our composite layer (8.3) we define the semantic function:

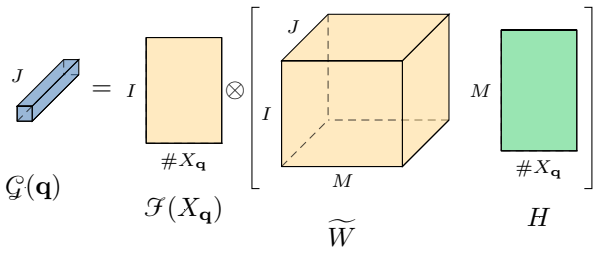
$$\mathcal{G}_j(\mathbf{q}) = \mathcal{A}_j(\mathcal{F}, \mathfrak{s})(\mathbf{q}) = \sum_{\mathbf{p} \in X_\mathbf{q}} \sum_{i=1}^{I} \mathcal{F}_i(\mathbf{p}) \underbrace{\sum_{k=1}^{K} w_{ijk}\, \mathfrak{s}_k(\mathbf{p} - \mathbf{q})}_{g_{ij}(\mathbf{p}-\mathbf{q})}, \tag{8.5}$$

where i, j, k refer to the i-th component of the input feature vector, the j-th output feature vector and the k-th component of the spatial function, and w_{ijk} are learnable parameters. As shown in Figure 8.2(a), the inner sum can be expressed as a multiplication of the projections in VH against the J slices of $W = (w_{ijk})$, which contains the learnable weights of each convolutional filter. Finally, each output feature $\mathcal{G}_j(\mathbf{q})$ can then be expressed as the Frobenius inner product between the matrix $\mathcal{F}(X_\mathbf{q})$ containing the features $\mathcal{F}_i(\mathbf{p})$ for $\mathbf{p} \in X_\mathbf{q}$ and $i \in \{1, \ldots, I\}$ and the j-th slice of WVH.

Comparison with point-convolutions. The formulation of our composite convolution (8.5) is equivalent to point-convolution (8.1) since it is possible to interpret the inner sum as the filter function g_{ij}. In Figure 8.2 we compare the matrix operations of our convolutional composite layer with KPConv [146] and ConvPoint [145]. When applied to a given neighbourhood $X_\mathbf{q}$, all these layers combine the input features of the neighbourhood elements, grouped in the matrix $\mathcal{F}(X_\mathbf{q})$, and the spatial information from the coordinates of the neighbourhood elements encoded in the matrix H. While our layer compresses the spatial information in H by the spatial function



(a) Convolutional composite layer



(b) KPConv / ConvPoint

Figure 8.2: *The scheme illustrating the operations in point-convolutional layers expressed in matrix form* (a) *Our convolutional composite layer* (8.5)*, and* (b) *the well-known point-convolutional layers KPConv and ConvPoint. Here* $\otimes$ *indicates the Frobenius inner product. In practice, our composite layer decomposes the matrix* $\widetilde{W}$ *of ConvPoint and KPConv into the product* $W \cdot V$*, enabling more flexibility in terms of number of parameters.*

(8.4) and then shares it among the filters (8.5), KPConv and ConvPoint learn a huge weight tensor $\widetilde{W}$ to combine spatial information and features.

Therefore, our convolutional composite layer has two major advantages over the existing point-convolutional layers: *i)* it performs an implicit regularization by decomposing $\widetilde{W}$ into the product WV, which has lower rank than $\widetilde{W}$, and *ii)* is more flexible since it allows to increase the complexity (and thus the descriptive power) of the spatial function without dramatically increasing the number of parameters of the layer, as we empirically show in Section 8.4. In particular, we can tune the complexity of the spatial function by increasing the number of RBF centers M. In KPConv and ConvPoint, the size of $\widetilde{W}$, which collects the majority of the parameters, grows linearly with M, while in our layer only the size of V, which contains substantially fewer parameters than $\widetilde{W}$, grows with M (see Figure 8.2).

8.3.2 Aggregate Composite Layer

We also define the *aggregate composite layer*, implementing a semantic function $\mathcal{A}_j(\mathcal{F}, \mathfrak{s})$ that aggregates the spatial information extracted by the same spatial function $\mathfrak{s}$ (8.4) and the input features $\mathcal{F}$ in a nonlinear man-

ner (differently from point-convolution). In particular, we compute the component-wise mean $\text{mean}_\mathcal{S}$ and standard deviation $\text{std}_\mathcal{S}$ of the spatial function over the convolution window $X_\mathbf{q}$ as:

$$\begin{aligned}[\text{mean}_\mathcal{S}(\mathbf{q})]_k &= \frac{1}{\#X_\mathbf{q}} \sum_{\mathbf{p}\in X_\mathbf{q}} \mathcal{S}_k(\mathbf{p}-\mathbf{q}), \\ [\text{std}_\mathcal{S}(\mathbf{q})]_k &= \sqrt{\frac{\sum_{\mathbf{p}\in X_\mathbf{q}} (\ \mathcal{S}_k(\mathbf{p}-\mathbf{q}) - [\text{mean}_\mathcal{S}(\mathbf{q})]_k\ )^2}{\#X_\mathbf{q}-1}}.\end{aligned}$$

Here, $\text{mean}_\mathcal{S}$ and $\text{std}_\mathcal{S}$ act as pooling operators with K components, receiving $\#X_\mathbf{q}$ vectors in $\mathbb{R}^K$ (the output of the spatial function $\mathcal{S}$), and returning a single vector in $\mathbb{R}^K$. Analogously, we define $\text{mean}_\mathcal{F}(\mathbf{q}),\ \text{std}_\mathcal{F}(\mathbf{q}) \in \mathbb{R}^I$ as the component-wise mean and standard deviation of the input features over the convolution window $X_\mathbf{q}$. Then, we concatenate these in two vectors $\theta(\mathbf{q}) = [\text{mean}_\mathcal{S}(\mathbf{q}); \text{std}_\mathcal{S}(\mathbf{q})] \in \mathbb{R}^{2K}$ and $\eta(\mathbf{q}) = [\text{mean}_\mathcal{F}(\mathbf{q}); \text{std}_\mathcal{F}(\mathbf{q})] \in \mathbb{R}^{2I}$, and aggregate them as:

$$\mathcal{A}_j(\mathcal{F},\mathcal{S})(\mathbf{q}) = \sum_{i=1}^{2I}\sum_{k=1}^{2K} \theta_k(\mathbf{q}) w_{kji} \eta_i(\mathbf{q}), \tag{8.6}$$

where w_{ijk} are learnable parameters. Since the two vectors $\theta(\mathbf{q}), \eta(\mathbf{q})$ are obtained by concatenating the mean and standard deviation over the window $X_\mathbf{q}$, this semantic function is nonlinear. In terms of the matrix multiplication illustrated in Figure 8.2(a), our aggregate composite layer combines the columns of VH to extract a unique spatial descriptor for the entire $X_\mathbf{q}$, and the same is performed on $\mathcal{F}(X_\mathbf{q})$. Our aggregate layer forces the network to learn from the mean and standard deviation (8.6) of the points coordinates in $X_\mathbf{q}$ and of the features in $\mathcal{F}(X_\mathbf{q})$, introducing an additional form of regularization.

8.3.3 CompositeNet

We implement a simple neural network, called CompositeNet, consisting of 5 composite layers and a final dense layer. A scheme illustrating our architecture is shown in Table 8.1, which also reports the number of output features J, the cardinality of the window $\#X_\mathbf{q}$, and the number of output points $\#Q$ for each layer. We chose these parameters so that our architecture is equivalent to the one of ConvPoint [145]. The number of output features J of each layer is defined relatively to the number of output features J_0 of the first layer, which is a hyper-parameter. Nonlinearities and

Table 8.1: *The architecture employed in all classification experiments for both our convolutional and aggregate CompositeNets.*

Layer Type	J	$\#X_{\mathbf{q}}$	$\#Q$
Composite + BN + ReLU	J_0	32	1024
Composite + BN + ReLU	$2 \cdot J_0$	32	256
Composite + BN + ReLU	$4 \cdot J_0$	16	64
Composite + BN + ReLU	$4 \cdot J_0$	16	16
Composite + BN + ReLU	$8 \cdot J_0$	16	1
Dense	–	–	–

batch normalization are applied only to the features and not to the spatial coordinates of the points, as in image convolution: in fact, the position of the pixels is not modified by the layers of a CNN. The cardinality of the point cloud is reduced throughout the network, and from the last composite layer we obtain a single point with a feature vector. We feed this to a dense layer, discarding its coordinates.

First, we train a convolutional and an aggregate CompositeNets for point cloud classification, to compare our composite layers to existing point-convolutional layers in terms of learning capability and design flexibility (Section 8.4). We also train our CompositeNets for anomaly detection following the self-supervised approach proposed in [7], which we describe in Section 7.1.2. In particular, we form a self-annotated training set by applying n geometric transformations $\{f_0, \dots, f_{n-1}\}$ to each point cloud of a given normal class. Then, we train a CompositeNet to classify the transformations applied to each input point cloud. The rationale is that, as shown in [7], learning to distinguish different geometric transformations applied to normal instances allows to extract features that are useful for anomaly detection. During testing, we apply the same transformations to each point cloud x and classify the resulting point clouds $f_i(x)$. Finally, we use as normality score the average posterior probability (according to the CompositeNet) of the correct transformations applied to x, i.e.

$$\mathcal{T}(x) = \frac{1}{n} \sum_{i=0}^{n-1} \big[\text{CompositeNet}\big(f_i(x)\big)\big]_i. \tag{8.7}$$

This solution is designed for 2D images, so the proposed geometric transformations are rotations, translations and flips [7], which cannot be

straightforwardly adopted on point clouds. For instance, virtually all the neural networks that process point clouds are insensitive to translations, thus cannot distinguish between a point cloud x and any translation of x. Moreover, most real-world objects (e.g. chairs, tables, etc.) have a common vertical orientation, and appear arbitrarily rotated around the vertical axis in the training set. Therefore, we expect rotations around the vertical axis to be impossible to distinguish. Leveraging the prior knowledge about the vertical orientation of the point clouds, we employ a set of $n = 8$ rotations of $0°, 45°, 90°, 135°, 210°, 240°, 300°, 330°$ around a fixed horizontal axis, including the identity (the rotation of $0°$).

8.4 Design Flexibility

In Sections 8.3.1–8.3.2 we show that our composite layer has a flexible structure, encompassing both point-convolutional layers and operators that aggregate spatial information and features in a nonlinear manner. Here we investigate the superior flexibility of our composite layers in terms of number of parameters compared to ConvPoint [145] and KPConv [146]. In particular, we perform two experiments to show that, contrarily to ConvPoint and KPConv, *i)* we can substantially improve the performance without increasing the number of parameters of our CompositeNets and *ii)* we can substantially reduce the number of parameters of our CompositeNets without compromising their performance.

Tuning the complexity of the spatial function. In the first experiment, we train different configurations of our CompositeNets on the ScanNet dataset [189], tuning the number of centers $M \in \{8, 16, \dots, 256\}$ to modify the complexity of the spatial function. We compare the accuracy of our CompositeNet with that obtained by a ConvPoint network having the same architecture as our CompositeNets. In these settings, we cannot directly compare our CompositeNets with the model proposed in KPConv [146] since it has a substantially more sophisticated architecture including residual blocks and thus the overall numbers of parameters are not comparable. For this reason, we also implement a network based on KPConv layers having the same architecture as our CompositeNets, which we call *KPConv-vanilla* to distinguish it from the original model [146]. In both ConvPoint and KPConv-vanilla we tune $M \in \{8, 16, \dots, 256\}$ as in our CompositeNets.

As remarked in Section 8.3.1, in ConvPoint and KPConv-vanilla the number of parameters of the layer increases linearly with M, which determines the size of the weight tensor $\widetilde{W}$. In contrast, when tuning M in

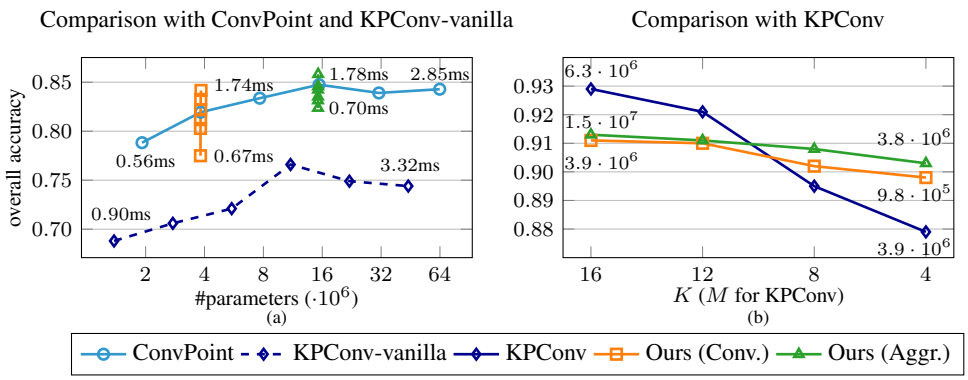


Figure 8.3: (a) *Overall accuracy against the number of parameters, varying the number of centers M. We report the average processing time during training for the least and most complex configurations. The parameters of ConvPoint and KPConv-vanilla increase linearly with M, while in our CompositeNets M can be increased without significantly changing the number of parameters.* (b) *Overall accuracy on ModelNet40 when reducing the number of parameters by tuning K (M for KPConv). We also report the number of parameters of the most and least complex configurations. These results show that we can substantially reduce the parameters of our CompositeNets without compromising the accuracy, while this is not the case in KPConv.*

both our convolutional and aggregate composite layers, we only modify the complexity of the spatial function, i.e. the size of the weight matrix V. The output dimension K of the spatial function is instead fixed, in particular we set $K = 16$, thus the size of the weight tensor W, which contains the majority of the learnable parameters, does not increase with M.

Figure 8.3(a) shows the overall accuracy (OA) of our CompositeNets and ConvPoint against their number of parameters. In ConvPoint the number of parameters doubles with M, while in our CompositeNets it increases less than 1% when passing from $M = 8$ to $M = 256$. In our CompositeNets, the accuracy increases with M, while ConvPoint and KPConv-vanilla achieve their best performance at $M = 64$, and yield worse accuracy at $M = 128, 256$, suggesting that they might be more prone to overfitting due to the increased number of parameters. For the least and most complex configurations of the three networks, we report the mean processing time of a point cloud during training. We observe that, in all cases, the processing time increases with M due to the increasing complexity of the spatial component. However, in ConvPoint and KPConv-vanilla the processing time increases substantially more than in our CompositeNets due to the linear growth of the number of parameters with respect to M. KPConv-vanilla performs substantially worse than our CompositeNets and ConvPoint, suggesting that KPConv layers cannot be successfully employed in simple ar-

chitectures and that the excellent performance of KPConv [146] is probably due to its sophisticated architecture.

We obtain similar results by performing the same experiment on the synthetic ModelNet40 [190] and ShapeNetCore [191] datasets. On these datasets, our CompositeNets achieve excellent results when setting $M = 64$. Further increasing M does not yield a significant improvement, but increases the processing time and thus is not convenient. However, the results in Figure 8.3(a) demonstrate that increasing the complexity of our spatial function by setting a larger M can substantially improve the performance of our CompositeNets on challenging, real-world datasets such as ScanNet [189] without requiring a larger number of parameters.

Tuning the number of parameters. In the second experiment, we train different configurations of our CompositeNets on the ModelNet40 dataset [190] to investigate how the classification accuracy varies after reducing the number of parameters of the network by tuning the output dimension of the spatial function $K \in \{16, 12, 8, 4\}$. We compare the accuracy of our CompositeNets with that achieved by KPConv. Since KPConv lacks a tunable spatial function, we vary the number of centers $M \in \{16, 12, 8, 4\}$, which play a similar role as K in our composite layers (see Figure 8.2).

Figure 8.3(b) shows the overall accuracy of each network. We also report the overall number of parameters of the most and least complex configurations of each network. Although KPConv achieves the best performance overall (when $M = 16$), its accuracy gets substantially worse when reducing the number of parameters by setting $M = 8$ and $M = 4$. On the other hand, both our convolutional and aggregate composite layers retain better performance for low K. These results highlight the flexibility of our composite layers, which allows to substantially reduce the number of parameters (–75% passing from $K = 16$ to $K = 4$) without compromising the accuracy, which decreases by a mere 0.01. In fact, the simpler semantic function given by a small K is counterbalanced by a spatial function whose complexity (determined by the number of centers, which we set to $M = 16$) remains constant in all experiments. In contrast, the performance of KPConv is extremely sensitive to M (–0.05 passing from $M = 16$ to $M = 4$), which is the only hyperparameter we can tune to reduce the number of parameters of a KPConv layer. Moreover, the reduction in the number of parameters is less substantial compared to our CompositeNets (–38% passing from $M = 16$ to $M = 4$) due to the more sophisticated architecture. In fact, tuning M does not change the number of parameters of the residual blocks.

8.5 Experiments

Our experiments are meant to show the effectiveness of composite layers in DNNs for point clouds. First, we illustrate the datasets we employ to test our CompositeNets (Section 8.5.1). Then, we evaluate the performance of our CompositeNets for classification (Section 8.5.2) and anomaly detection following a self-supervised approach [7] (Section 8.5.3).

8.5.1 Benchmarking Datasets

We adopt two well-known synthetic 3D classification benchmarks: *ModelNet40* [190], which contains 12,311 objects from 40 classes, and a subset of *ShapeNet* [191] called *ShapeNetCore*, which contains 51,127 shapes from 55 rather imbalanced classes. We also perform our experiments on real-world data using the *ScanNet-v2* dataset [189], which contains 1513 3D scenes (obtained from RGB-D scans) of 707 indoor environments with instance-level annotations of 608 classes. We extract the objects belonging to the 17 largest classes (excluding planar objects such as windows, walls and doors) as in the classification experiments in [189]. The resulting classification dataset presents severe class imbalance. All these datasets contain 3D meshes, from which we sample point clouds of 1024 points, setting a constant feature function $\mathcal{F} \equiv 1$. This is a standard procedure to build point clouds containing only the spatial information [146, 150].

In our classification experiments, we use the official training split of each dataset. We use the same splits in our anomaly detection experiments, but in that case the DNNs are trained only on instances from a single class that is considered normal. During testing, point clouds from other classes are deemed anomalous. Since training DNNs requires relatively large training sets to extract meaningful features, we consider only the 7 most represented classes from ShapeNet (at least 2000 instances, as in [183]) and the 14 most numerous classes from ScanNet (at least 400 instances).

8.5.2 Classification

We train our CompositeNets for 200 epochs on an NVIDIA RTX A6000 GPU. We use the Adam optimizer [192] and the traditional cross-entropy loss for multiclass classification. In Table 8.2 we report the hyperparameter configurations that maximize the validation accuracy on each dataset.

Considered methods. We compare our CompositeNets against PointNet [150], KPConv [146], and ConvPoint [145], reporting their accuracy on Model-

Table 8.2: *Hyperparameters of our convolutional and aggregate CompositeNets, ConvPoint and KPConv-vanilla maximizing the validation accuracy on each dataset.*

Dataset	Layer	J_0	M	K
ModelNet40 [190]	**Ours (Conv.)**	64	64	16
	Ours (Aggr.)	64	64	16
	ConvPoint	64	16	–
	KPConv-vanilla	64	16	–
ShapeNetCore [191]	**Ours (Conv.)**	32	64	16
	Ours (Aggr.)	32	64	16
	ConvPoint	32	16	–
	KPConv-vanilla	64	32	–
ScanNet [189]	**Ours (Conv.)**	64	256	32
	Ours (Aggr.)	64	256	32
	ConvPoint	64	32	–
	KPConv-vanilla	64	64	–

Net40 from the corresponding papers. Since the performance on ShapeNetCore and ScanNet were not reported, we train and test the competing networks using the software implementations made publicly available. We remark that both our CompositeNets are equivalent to ConvPoint in terms of architecture, and have a comparable number of parameters. In contrast, PointNet uses a custom non-convolutional architecture and KPConv uses a deeper residual network, making both models significantly different from ours. In all our experiments we employ the original architecture and configuration of KPConv ($M = 16$). We also train KPConv-vanilla, namely a network based on KPConv layers having the same architecture as our CompositeNets. In Table 8.2 we report the configuration of ConvPoint proposed in [145] for ModelNet40, and the configurations we selected for ConvPoint and KPConv-vanilla to maximize the validation accuracy also on the other datasets. KPConv layers have an additional hyperparameter with respect to ConvPoint, namely the radius ρ defining the convolution windows [146]. In KPConv-vanilla we have tuned ρ in a $\pm 20\%$ range compared to the value in [146] without obtaining substantial changes in the accuracy.

Figures of merit. We evaluate the classification performance by the overall accuracy (OA), defined as the proportion of correctly classified point clouds, and the average accuracy (AA), namely the average per-class accuracy, which better takes into account the accuracy on under-represented classes. We train each method 5 times on each dataset and compute the average OA and AA, except for ModelNet40, where we report the results from the papers presenting PointNet [150], KPConv [146], and ConvPoint [145].

Experimental results. The results in Table 8.3 indicate that our CompositeNets achieve comparable performance to ConvPoint [145] and KPConv

Table 8.3: *Overall accuracy (OA) and average accuracy (AA). The results of competing methods on ModelNet40 are from the literature (KPConv [146] does not report the AA). The results on ShapeNetCore and ScanNet are obtained using the available code.*

	PointNet [150]		**KPConv** [146]		**KPConv-vanilla**		**ConvPoint** [145]		**Ours (Conv.)**		**Ours (Aggr.)**	
Dataset	OA	AA	OA	AA	OA	AA	OA	AA	OA	AA	OA	AA
ModelNet40 [190]	0.892	0.862	**0.929**	–	0.879	0.848	0.918	**0.885**	0.913	0.871	0.911	0.879
ShapeNetCore [191]	0.830	0.686	0.829	0.671	0.752	0.632	0.837	0.669	0.828	0.653	**0.839**	**0.698**
ScanNet [189]	0.786	0.751	**0.878**	0.835	0.766	0.699	0.847	0.811	0.861	0.824	0.871	**0.839**

[146] on all the considered datasets. Therefore, the point-convolutional layers in these models can be successfully replaced by our composite layers, which are more flexible, as shown in Section 8.4. Moreover, the poor performance of KPConv-vanilla confirms that KPConv requires a sophisticated residual architecture to achieve good classification accuracy.

Here we illustrate these results more in detail. On ModelNet40 [190], our CompositeNets approach ConvPoint [145] both in terms of overall and average accuray, and substantially outperform PointNet [150] in terms of overall accuracy. KPConv [146] yields the best overall accuracy, but the paper does not report the average accuracy. On ShapeNetCore [191], our aggregate CompositeNet yields the best OA, slightly outperforming ConvPoint, while our convolutional CompositeNet performs similarly to PointNet and KPConv. Our aggregate CompositeNet yields also the best AA, which is particularly important on imbalanced datasets such as ShapeNetCore since it better evaluates the accuracy on under-represented classes. On ScanNet [189], our aggregate CompositeNet performs very similarly to KPConv, which has a slightly better OA. Also in this case, our aggregate CompositeNet represents yields the best AA, suggesting that our aggregate CompositeNet is more robust to class imbalance than the other methods. Our convolutional CompositeNet performs worse than the aggregate and KPConv, but yields better overall and average accuracy than ConvPoint. While on ModelNet40 and ShapeNetCore PointNet achieves competitive results, on ScanNet the performance difference with the other models is substantial. Finally, we remark that, on challenging classification benchmarks such as ShapeNetCore and ScanNet, our aggregate CompositeNet outperforms ConvPoint and achieves similar accuracy to KPConv despite having a much simpler architecture.

8.5.3 Anomaly Detection

We train our CompositeNets following the self-supervised approach presented in [7]. Since this approach solves a classification problem (with surrogate labels), we employ the same configurations reported in Table 8.2.

Considered methods. As baselines, we consider some traditional anomaly-detection algorithms trained on hand-crafted features extracted from the point clouds in the training set. In particular, we use the *Global Orthographic Object Descriptor* (GOOD) [193] to compress the salient characteristics of each point cloud into a feature vector. Compared to most of the existing point cloud descriptors [177], the advantage of GOOD is that it can be directly employed on point clouds that contain only the coordinates of the points, and do not require additional features such as normal vectors. To perform anomaly detection, we first use GOOD to extract a feature vector from each normal point cloud in the training set. Then, we use these feature vectors to train some shallow anomaly detectors, namely Isolation Forest (IFOR) [104], SVDD [101], and One-Class SVM [100], which we then employ at test time to identify anomalous point clouds. Since IFOR, SVDD, and One-Class SVM achieve approximately the same performance in our experiments, here we only report only the results obtained by IFOR, which, on average, performs slightly better than the others.

To the best of our knowledge, the only work using deep learning for anomaly detection in point clouds is [183], where the authors leverage a variational autoencoder (VAE) based on FoldingNet [186]. Since the implementation of this method is not publicly available, we only report from [183] the results achieved by this method on the 7 most numerous classes of the ShapeNet dataset. We also compare our self-supervised CompositeNets with KPConv [146] and ConvPoint [145], trained in the same fashion using the same architectures as in classification. We recall that our CompositeNets have a simple architecture similar to ConvPoint, while KPConv has a deeper, residual architecture.

Figures of merit. We assess the anomaly detection performance by the Area Under the ROC Curve (AUC), which measures how well each network can distinguish normal instances (i.e. from the training class) from the anomalous ones (i.e. from any other class in the official test set). Following [74], we report the average rank of each method and the p-values of the one-sided Wilcoxon Signed-Rank test assessing whether the performance difference between the best-ranking method and the others is statistically significant.

Experimental results. In Table 8.4 we report the results of our experiments. On ShapeNet, self-supervised DNNs outperform both the shallow baseline IFOR and the VAE [183] in most of the classes and in terms of average rank. However, the Wilcoxon test [74] comparing our aggregate CompositeNet (which is the best-performing method in terms of average rank) and

Table 8.4: *Anomaly detection performance (AUC) of our self-supervised CompositeNets, two similar networks based on ConvPoint [145] and KPConv [146], and a shallow detector based on the GOOD descriptor [193]. We also report the results obtained by the variational autoencoder in [183] on ShapeNet. Bold denotes the best AUC for each class. Our aggregate CompositeNet is the best-performing method on most classes and in terms of average rank.*

		Baseline	SOTA	Self-supervised deep neural networks			
	Class	IFOR	VAE [183]	KPConv	ConvPoint	Ours (Conv.)	Ours (Aggr.)
ShapeNet [191]	Airplane	0.912	0.747	**0.974**	0.956	0.969	0.970
	Car	0.712	0.757	**0.988**	0.953	0.953	0.972
	Chair	0.571	0.931	0.921	0.910	0.904	**0.941**
	Lamp	**0.962**	0.907	0.372	0.373	0.391	0.424
	Table	0.883	0.839	**0.943**	0.777	0.821	0.854
	Rifle	0.475	0.382	0.967	0.976	**0.978**	0.977
	Sofa	**0.986**	0.777	0.923	0.898	0.936	0.944
	Avg. rank	3.714	4.429	2.857	4.357	3.500	**2.143**
	Wilcoxon-p	$2.89 \cdot 10^{-1}$	$1.09 \cdot 10^{-1}$	$3.44 \cdot 10^{-1}$	$7.81 \cdot 10^{-3}$	$2.34 \cdot 10^{-2}$	–
ScanNet [189]	Backpack	0.677	–	**0.851**	0.778	0.783	0.779
	Book	**0.830**	–	0.734	0.731	0.782	0.782
	Bookshelf	0.745	–	0.796	0.792	**0.824**	0.798
	Box	0.429	–	**0.705**	0.638	0.630	0.659
	Cabinet	0.581	–	0.813	0.818	0.810	**0.827**
	Chair	0.449	–	0.747	**0.842**	0.837	0.838
	Desk	0.482	–	0.869	0.857	0.870	**0.888**
	Lamp	0.540	–	0.639	0.707	0.703	**0.725**
	Pillow	0.647	–	0.693	0.707	0.754	**0.771**
	Sink	0.415	–	0.896	0.892	0.921	**0.940**
	Sofa	0.785	–	**0.876**	0.859	0.852	0.852
	Table	0.318	–	**0.920**	0.912	0.902	0.902
	Towel	**0.801**	–	0.657	0.690	0.640	0.655
	Trash Can	0.426	–	0.881	0.918	0.921	**0.927**
	Avg. rank	4.727	–	2.000	3.227	3.273	**1.773**
	Wilcoxon-p	$1.46 \cdot 10^{-3}$	–	$6.50 \cdot 10^{-1}$	$2.53 \cdot 10^{-3}$	$9.28 \cdot 10^{-3}$	–

these baselines is inconclusive (p-value > 0.05), probably because the number of considered classes is small. Unfortunately, we could not compare our CompositeNets with the VAE [183] on more classes since its implementation is not publicly available.

On ScanNet, all methods tend to perform worse than they do on the synthetic data from ShapeNet, as we expected since the real-world objects in ScanNet are often partially occluded. Also in this case, we observe that self-supervised DNNs outperform IFOR in most of the classes. Our aggregate CompositeNet is the best-performing method in most classes and in terms of average rank, and the Wilcoxon test [74] confirms that the difference with all the other methods, except for KPConv, is statistically significant (p-value ≤ 0.05). These results show that our aggregate CompositeNet substantially outperforms ConvPoint in both the considered datasets, and performs similarly to KPConv despite having a much simpler architecture.

Finally, we observe that applying IFOR to hand-crafted features ex-

tracted by GOOD yields relatively good performance on a small number of classes, outperforming deep solutions on the *Lamp* and *Sofa* classes from ShapeNet, and on the *Book* and *Towel* classes from ScanNet. However, IFOR yields poor performance on most of the other classes, often achieving AUC ≤ 0.5. This suggests that point cloud descriptors cannot characterize well all classes of point clouds, thus deep learning can address anomaly detection in a more general way.

8.6 Discussion and Limitations

Our classification and anomaly detection experiments show that our aggregate CompositeNet achieves excellent results, performing better than ConvPoint and similarly to KPConv despite having a much simpler architecture. The performance of our convolutional CompositeNet is slightly worse, on par with ConvPoint. We believe that these promising results can be further improved adopting more sophisticated architectures.

We speculate that the good performance of our CompositeNets is due to the superior design flexibility of our models, which enables the design of rich spatial functions that boost the layer's capability to recognize local spatial relations between points. In contrast, existing point-convolutional layers such as ConvPoint and KPConv cannot be customized without substantially changing the overall number of parameters of the network (Figure 8.3(a)) or risking affecting its performance (Figure 8.3(b)). The ability to customize the number of parameters can be crucial when training sets are small (as in our anomaly detection experiments), which increases the risk of overfitting. Moreover, the performance difference between our aggregate and convolutional CompositeNets suggests that a non-convolutional aggregation between spatial information and features might be beneficial to the classification and anomaly-detection performance.

We are among the first to address unsupervised deep anomaly detection on point clouds. Although very effective, the self-supervised approach [7] has a fundamental limitation since, as noted in [114], the transformations applied to the training set are domain-dependent, and most transformations used for images are not suited for point clouds. Moreover, the selected set of transformations might not be well suited for some of the possible normal classes, resulting in poor anomaly detection performance. For instance, all the considered self-supervised neural networks achieve AUC < 0.5 on the class *Lamp* from the ShapeNet dataset. In contrast, the VAE [183] and GOOD [193] successfully extract the relevant features from these point clouds, and therefore yield better performance on the class *Lamp* from

ShapeNet [191]. We speculate that these results are due to the fact that said class contains both table and ceiling lamps, thus preventing self-supervised networks from distinguishing the rotations applied to each input, depending on whether the lamp is placed on a table or hangs from the ceiling. All in all, our experiments confirm that the self-supervised approach proposed in [7] is extremely effective as long as the geometric transformations are carefully chosen depending on the considered normal class. When that is not the case, e.g. in our experiments on the class *Lamp* from ShapeNet [191], other algorithms might perform better. In general, the great flexibility of our composite layers and the promising results obtained by our models in the anomaly detection task encourage us to continue our research and employ our CompositeNets also for other unsupervised tasks.

8.6.1 Future Work

Future work will address the implementation of more sophisticated CompositeNets, leveraging deeper, residual architectures that comprise both convolutional and aggregate composite layers, with the aim of improving classification and anomaly-detection performance. To address the intrinsic limitations of the self-supervised approach in [7], we will investigate how to automatically learn suitable transformations directly from the point clouds from the training set as in [121], where such transformations have been successfully learned for signals and images.

In this work, we have only considered the problem of distinguishing between normal and anomalous point clouds, therefore our self-supervised CompositeNets are not designed to detect anomalous regions within the point clouds. This problem has been widely studied in 2D images, and only recently the first dataset designed for localized anomaly detection within point clouds has been released [126]. Therefore, we will investigate how to train CompositeNets to detect anomalous regions within point clouds, to be tested in upcoming real-world datasets such as [126]. Another research direction that combines the design flexibility of our composite layer and the self-supervised approach is unsupervised hyperparameter tuning and pre-training [178, 179] for 3D deep learning, which are of paramount importance since annotated 3D datasets are not as large and widespread as 2D image datasets.

CHAPTER 9

Open-Set Recognition for Wafer Production Monitoring

In this chapter we address the problem of detecting anomalous patterns in Wafer Defect Maps (WDMs), namely point clouds listing the 2D coordinates where defects were found in a silicon wafer by an inspection machine. In normal production conditions, defects are rare and randomly distributed in WDMs, while defects grouped in patterns indicate production problems. Some classes of defect patterns have already been identified, while anomalous patterns might appear due to unknown problems in the manufacturing process. Therefore, we cast this as an open-set recognition problem, and address it by applying an outlier detector based on a Gaussian Mixture Model (GMM) to the latent representation of a Submanifold Sparse Convolutional Network (SSCN) trained on known classes. Our experiments on WDMs acquired at the STMicroelectronics plant in Agrate Brianza, Italy show that our solution can detect anomalous patterns better than alternative methods, which we apply on top of the same SSCN to obtain a fair comparison. Since no anomalous WDMs have been included in our dataset yet, we design a new experimental setup in which we train multiple SSCNs, each time taking out from the training set one of the defect classes, which will then be

considered anomalous during testing.

In Section 9.1 we define WDMs, illustrate the problem of wafer production monitoring and how it has been addressed in the literature. In Section 9.2 we present our WDM monitoring pipeline, including details on our SSCN and the data augmentation procedure we employ to mitigate class imbalance. Finally, in Section 9.3 we present our experimental setup and our classification and anomaly detection results.

9.1 Silicon Wafer Monitoring

Silicon wafers represent the base upon which most electronic components are built, including processors, memories and sensors that are present in any electronic device from smartphones to cars. Producing wafers requires costly, long, and high-tech industrial processes. Each wafer (Figure 9.1(a)) contains hundreds of chips and has to be analyzed by several *inspection machines* at different stages of the production process to locate any defect, which might be related to specific problems in the manufacturing process. In this section, we first describe Wafer Defect Maps (WDMs), which represent the type of data we work on in this chapter (Section 9.1.1), and then we briefly review the literature on automatic wafer production monitoring, focusing mainly on WDM classification (Section 9.1.2).

9.1.1 Wafer Defect Maps

Each wafer inspection returns a Wafer Defect Map (WDM), which we indicate by x, namely a point cloud containing the 2D coordinates of the wafer where defects are found. Defect coordinates belong to a regular grid, namely $x \subset \mathbb{N}^{K \times K}$, determined by the resolution of the inspection machines. In practice, the grid might be huge (in our case $K = 20,000$).

In normal production conditions, defects are rare and randomly distributed in WDMs. In contrast, WDMs containing *patterns* like those shown in Figure 9.1(b) might be symptoms of problems or failures in the production line, which must be promptly identified to solve production failures as soon as possible, preventing the waste of time and resources. Some defect patterns have been studied and labeled by production engineers into a set $\mathcal{L}$ of classes. These patterns are known to be related to specific problems in a particular manufacturing step. However, WDMs might also exhibit anomalous patterns related to unknown production issues, and it is a primary concern of production engineers to detect them as well. For this reason, we cast WDM monitoring as an open-set recognition problem (Section 7.2).

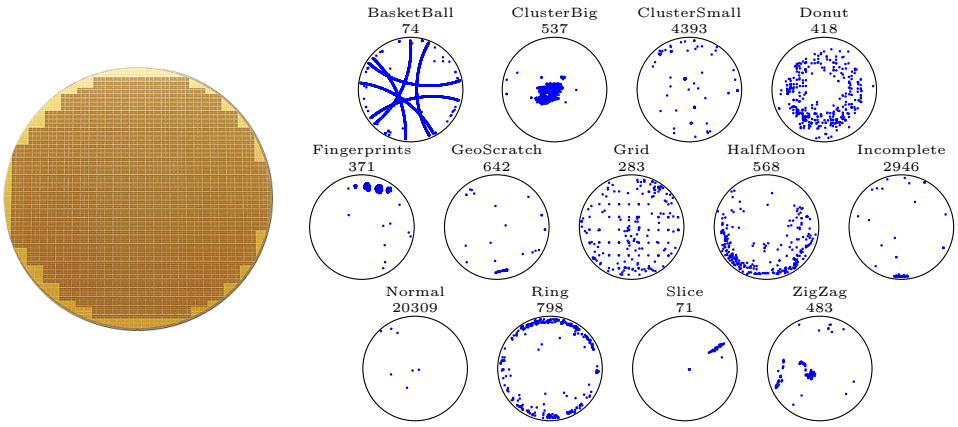


Figure 9.1: (a) *An example of a wafer containing few hundreds of chips (the small cells).* (b) *An example of WDM for each known class in the ST dataset. We also report the number of instances of each class to emphasize the severe class imbalance in the ST dataset.*

9.1.2 Existing solutions

During production, wafers are monitored to identify defect patterns in WDMs [194–201], and also to analyze images of localized defects acquired by electronic microscopes [202, 203]. Here we briefly review the literature on WDM monitoring. We refer to [204] for a more comprehensive survey on wafer monitoring.

The very first WDM monitoring pipelines required production engineers to visually inspect a relatively small number of randomly sampled WDMs. In the last few decades, the production volume and quality standards have been substantially increased, making it necessary to develop automatic WDM classification algorithms. These techniques allow to inspect the whole wafer production and reduce classification errors, which in visual inspection were frequently caused by fatigue.

For automatic classification purposes, a WDM x can be seen as a binary image $\mathcal{I}_x \in \{0,1\}^{K\times K}$, where each pixel (i,j) corresponds to an inspected wafer location and

$$\mathcal{I}_x(i,j) = \begin{cases} 1 & \text{if } (i,j) \in x \\ 0 & \text{otherwise} \end{cases} . \tag{9.1}$$

However, the resolution of WDMs is huge (in our case $K = 20,000$), so a WDM would require almost 3 GB to be loaded in memory in single precision as a grayscale image. For this reason, the vast majority of the

existing solutions preliminary reduce the size of these images by building Wafer Bin Maps, namely smaller images (say 200×200) where pixels are associated with small portions of the wafer, and the value of each pixel indicates whether the corresponding wafer portion contains defects or not.

The first supervised methods [195–197] employ hand-crafted features, including geometric regional features such as area, perimeter, or eccentricity of defect clusters [195], and density-based features such as the location of defect-dense areas [196]. Other informative features can be obtained by analyzing Wafer Bin Maps in a different domain using the Radon or Hough transforms to highlight specific patterns [197]. A few of these features are then stacked in vectors and fed to a classifier such as a support vector machine (SVM) or a decision tree. However, hand-crafted features might not be able to extract meaningful information in some cases, e.g., when defect patterns are rotated, shifted, or cover only part of the WDM. Moreover, hand-crafted features are usually defined to highlight patterns belonging to known classes, and might be meaningless for anomaly detection.

Since Convolutional Neural Networks have achieved impressive results in image classification, the most recent methods [198–201] employ Deep Learning models to classify Wafer Bin Maps. In particular, [198] addresses the simplified problem of distinguishing radial map patterns from non-radial ones. The solution presented in [199] proposes a specific preprocessing where the intensity value of each pixel represents the number of defects in the corresponding wafer portion. Several works [194, 200, 201] show the superiority of deep CNNs over traditional machine-learning methods based on hand-crafted features using the public dataset WM-811K [205], where they achieve excellent classification performance. However, in these works, wafer monitoring is tackled as a closed-set classification problem, ignoring possible anomalous patterns. Moreover, the WM-811K dataset contains small images representing Wafer Bin Maps, so we cannot use it to test the proposed solution, which takes the original WDMs as input.

Although here we focus on WDM monitoring, it is worth mentioning the work by Cheon et al. [203], which addresses open-set recognition on localized defects. In particular, they apply a k-nearest neighbors outlier detector to the features extracted by a CNN trained on the known classes. However, this solution is designed to process traditional images of localized defects, thus it cannot be applied directly to the analysis of WDMs. To the best of our knowledge, this is the only open-set recognition solution for wafer monitoring in the literature.

9.2 Proposed Solution

We address WDM monitoring as an open-set recognition problem and propose a network architecture to efficiently classify full-resolution WDMs belonging to the classes represented in the training set (Section 9.2.1). Then, we extend our network – which is trained to address a traditional multi-class classification problem – to open-set recognition, i.e., to detect WDMs containing anomalous patterns (Section 9.2.2). Here, we also describe the class-specific data augmentation procedure we employ in our open-set recognition method, both at training and test time (Section 9.2.3) and summarize the proposed pipeline to classify WDMs (Section 9.2.4).

9.2.1 Classification

The open-set recognition problem includes a traditional multi-class classification problem with a fixed set of known classes, i.e., those represented in the training set. Traditional CNNs, which represent the state of the art in image classification, cannot be directly applied to WDMs because they take as input relatively small images (e.g., VGG16 [206] and ResNet50 [207] take as input 224×224 RGB images), while images representing full-resolution WDMs are huge and cannot be used to train and test CNNs.

To handle WDMs efficiently, we build a deep network based on Submanifold Sparse Convolution (SSC) [9], a modified convolutional operator designed to process sparse images at arbitrary resolution. The output of an SSC is the same as that of a regular convolution, but only on the active sites of its receptive field, namely, the non-zero locations, i.e.:

$$\text{SSC}(x) = \text{conv}(x) \odot \mathbb{1}(\text{supp}(x)), \tag{9.2}$$

where $\odot$ indicates the Hadamard product, $\mathbb{1}$ the indicator function and $\text{supp}(x)$ the support of x, i.e., the set of its non-zero locations. The main advantage of SSC compared to traditional convolution is that it enables a very efficient processing of sparse images, regardless of their resolution. Thus, a WDM can be processed directly as the list of the coordinates where defects lie within the wafer, which correspond to the active sites. Moreover, (9.2) implies that SSC maintains the input sparsity – thus the shape of the defect patterns in WDMs – throughout the layers and does not increase the number of active sites, while regular convolutions reduce the sparsity of WDMs, as illustrated in Figure 9.2.

The architecture of the proposed SSCN is inspired by the convolutional part of the VGG16 [206], and is made of consecutive building blocks reducing the spatial resolution of the activation maps. Each block (Figure 9.3(a))

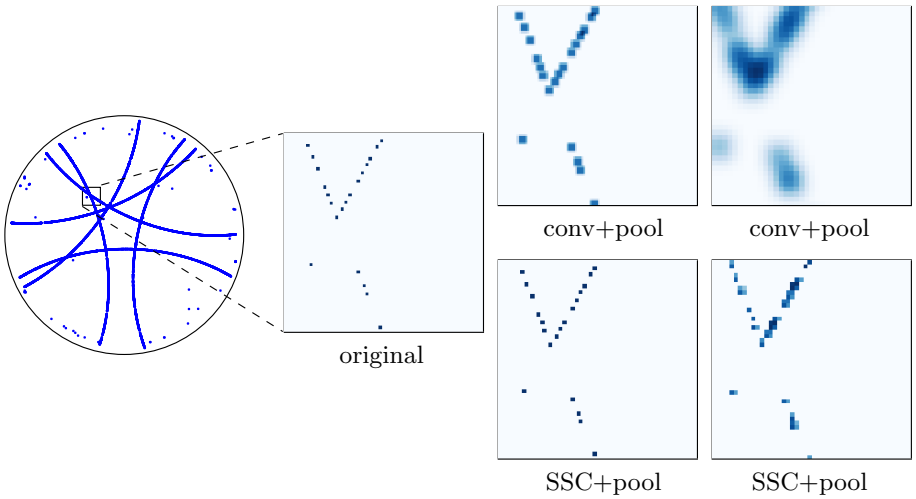


Figure 9.2: *Visual representation of the difference between processing a WDM patch (*200×200*) by regular convolutions (conv) and Submanifold Sparse Convolutions (SSC). All these convolutional layers have uniform filters of size* 7×7 *and are followed by max-pooling with a kernel of size* 2×2*, yielding a downsampling factor 2 on each dimension. For illustration purposes, we represent all the images at the same size. While regular convolutions reduce the original sparsity, SSCs maintain it since they do not increase the number of active locations.*

includes an SSC layer, with Batch Normalization (BN) and ReLu activations, followed by a Max-Pooling layer of stride 2. Thus, a single block reduces the resolution of the feature map by a factor 2 on each dimension. Our architecture, illustrated by Figure 9.3(b), is formed by 13 such blocks followed by a convolutional layer, and yields a 128-dimensional latent representation $L(x)$ of each WDM x, which we also employ to detect anomalies (Section 9.2.2). Eventually, a fully-connected layer with SoftMax activations outputs a vector of $\#\mathcal{L}$ scores, whose maximum determines the predicted class of the input WDM.

We remark that both our solution and those based on traditional CNNs extract a compact representation of the WDMs [199–201]. The main difference is that traditional CNNs for WDM classification require a preliminary binning of the WDMs to reduce their resolution, which we believe might lead to a loss of information. In contrast, our solution is entirely data-driven and does not discard any piece of information contained in WDMs, thanks to Submanifold Sparse Convolutions. Our SSCN yields a spatial downsampling factor 2^{13} on each dimension, way larger than what is customarily obtained by CNNs for image classification: for instance, the CNN proposed

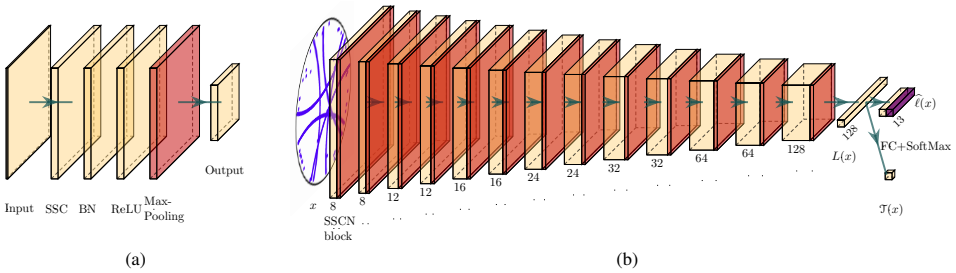


Figure 9.3: (a)*: our SSCN block made of a Submanifold Sparse Convolutional layer (SSC), Batch Normalization (BN), ReLU activations and Max-Pooling layer (Pool) with stride 2.* (b)*: our SSCN architecture, based on our SSCN block, for open-set recognition on WDMs. After 13 SSCN blocks, each WDM x is transformed to a 128-dimensional latent representation $L(x)$, which is fed to a fully connected-layer (FC) with SoftMax activation to identify the known classes, and to the negative log-likelihood function of a GMM (fitted on the latent representations of WDMs from known classes), which is our anomaly score $\mathcal{T}(x)$.*

in [201] for Wafer Bin Maps only achieves a downsampling factor 2^5. Although the network architecture made of convolutional and pooling layers recalls the VGG16 [206], our SSCN has a substantially fewer parameters $(164,077)$ since, on top of the convolutional part, the VGG16 has 3 large fully-connected layers, which we do not include in our SSCN.

9.2.2 Anomaly Detection

The second task in open-set recognition is detecting anomalous instances, namely defect patterns that do not belong to any known class. This is particularly important when monitoring WDMs because any anomalous pattern must be detected as soon as possible and studied to determine whether an unknown failure occurred during production. We follow the simple but effective approach of applying an outlier detector to the latent representation of our classifier – i.e., the output of the penultimate layer of the network (see Figure 9.3). This approach leverages the fact that a classifier maps instances belonging to the same class in the same area of the latent space so that the last, fully connected layer can separate the classes by linear boundaries [135, 136]. Hence, the latent representation of the known classes can be described by a multimodal distribution ϕ, where each mode is associated with a known class. We expect anomalous instances to be mapped in low-density regions of the latent space with respect to ϕ. For this reason, we fit a Gaussian Mixture Model (GMM) with $\#\mathcal{L}$ components, namely the number of known classes, and obtain a density $\widehat{\phi}$ to describe the distribution of the latent representations $L(x)$ extracted by the SSCN trained on

WDMs from known classes, i.e.:

$$\widehat{\phi} = \sum_{i=1}^{\#\mathcal{L}} \widehat{a}_i \cdot \mathcal{N}(\widehat{\mu}_i, \widehat{\Sigma}_i), \tag{9.3}$$

where each Gaussian component $\mathcal{N}(\widehat{\mu}_i, \widehat{\Sigma}_i)$ for $i = 1, \ldots, \#\mathcal{L}$ represents a known class. Then, at test time, we employ as anomaly score the negative log-likelihood of $\widehat{\phi}$, i.e., for each WDM x,

$$\mathcal{T}(x) = -\log\left(\widehat{\phi}(L(x))\right). \tag{9.4}$$

Eventually, a WDM x is detected as *Anomalous* when $\mathcal{T}(x) > h$, where h is a threshold we set such that $\mathbb{P}(\mathcal{T}(x) \geq h \mid \ell \in \mathcal{L}) = \alpha$, where ℓ is the label associated to x and α is the target false positive probability.

To prevent $\widehat{\phi}$ from overfitting the latent representation of the training set, we use 90% of the data to train the SSCN, and compute the latent representation of another 5% of the training set. Then, we use these latent representations to fit $\widehat{\phi}$ by estimating the parameters of the GMM by the well-known *Expectation Maximization* algorithm [99], as in [98]. Finally, we compute the anomaly score $\mathcal{T}(x)$ (9.4) of each instance x of the remaining 5% of the training data, and set the threshold h as the empirical $(1-\alpha)$-quantile of these anomaly scores. We remark that, since we extract the latent representation $L(x)$ by the same SSCN used to classify WDMs from the known classes, our anomaly-detection procedure does not require additional computations compared to closed-set classification, except for the calculation of the anomaly score $\mathcal{T}(x)$.

9.2.3 Data Augmentation

The ST dataset is relatively small compared to the traditional datasets used for image classification and is characterized by an extreme class imbalance. Both these facts increase the risk of overfitting when training a deep classifier, and, in particular, the classification performance is likely to be poor on under-represented classes. To address this problem, we implement an ad-hoc data augmentation procedure based on a set of *label-preserving* transformations to be applied to the WDMs. Let F^ℓ denote the set of label-preserving transformations for WDMs belonging to the class ℓ:

$$F^\ell = \left\{ f^\ell_{\boldsymbol{\theta}} \colon \mathbb{N}^{K\times K} \to \mathbb{N}^{K\times K},\ \boldsymbol{\theta} \in \Theta_\ell \right\}, \tag{9.5}$$

where $\boldsymbol{\theta}$ indicates the parameters that define each transformation $f^\ell_{\boldsymbol{\theta}}$, and Θ_ℓ is the set of transformation parameters specific for each class $\ell \in \mathcal{L}$.

Each $f^{\ell}_{\boldsymbol{\theta}}$ combines different transformations widely used for data augmentation on images, such as rotations around the center, horizontal flips, and small translations of the defect coordinates. Besides these traditional augmentation procedures, we apply two transformations that we specifically designed for WDMs:

Noise injection consists in adding a small number of defects to each WDM at randomly sampled coordinates. This operation does not change the label of any WDM because a few randomly distributed defects are present in every wafer due to natural impurities in the silicon. In particular, the defects in WDMs from the *Normal* class can be seen as pure noise since they are few and randomly spread within the wafer, as can be expected when the process is executed normally. For this reason, we compute the empirical cumulative distribution $\widehat{\psi}$ of the number of defects that are present in *Normal* WDMs in the ST dataset, and use $\widehat{\psi}$ to randomly generate the number D of defects to be injected in a WDM during augmentation. Our study and the experience of production engineers confirm that defects in *Normal* WDMs do not show any specific pattern. Thus we inject these D defects at uniformly sampled polar coordinates in $[0, 2\pi] \times [0, K/2]$.

Random mixing creates new training samples from under-represented classes, such as *BasketBall* and *Slice*, by superimposing randomly cropped parts of WDMs from the same class. We empirically verified that random mixing preserves the label: production engineers at STMicroelectronics could not tell the original WDMs from those generated by this procedure. We remark that random mixing is somewhat similar to *mixup* [208], which builds new training samples as linear combinations of instances of the training set. However, mixup is designed for relatively small images and cannot be directly applied to WDMs. Another difference is that we do not change the label of a WDM after applying random mixing.

We execute our data augmentation procedure in each training epoch to produce new augmented batches, and also when fitting the GMM $\widehat{\phi}$, to obtain a sufficient number of samples from under-represented classes. This is done by generating several versions of each original WDM x using class-specific transformations $f^{\ell}_{\boldsymbol{\theta}}(x)$, where $\boldsymbol{\theta}$ is randomly sampled from Θ_{ℓ}.

Test time augmentation. To stabilize the output of our SSCN and improve the classification performance we employ data augmentation also at test time, averaging the classification scores obtained on different augmented versions of each WDM from the test set, as in [206]. Indeed, even though the features extracted by the network should, in principle, be invariant to the label-preserving transformations in F^{ℓ}, perfect invariance cannot be

achieved in practice, thus combining the predictions obtained from several augmented versions of the same WDM typically improves the classification performance [206]. Since the labels of the test set cannot be assumed to be known, during testing we only apply transformations that preserve all the labels $\ell \in \mathcal{L}$:

$$F = \left\{ f_{\boldsymbol{\theta}} \,:\, \boldsymbol{\theta} \in \Theta = \bigcap_{\ell \in \mathcal{L}} \Theta_\ell \right\}. \tag{9.6}$$

We exclude from F our random mixing transformation, which we only use to construct new training samples from under-represented classes such as *BasketBall* and *Slice*.

To apply data augmentation when testing an open-set recognition method, it is important to verify that the transformations in F preserve not only the known class labels $\ell \in \mathcal{L}$, but also the *Anomalous* class label, which might be encountered during testing. This is certainly guaranteed when F is a group since any $f_{\boldsymbol{\theta}} \in F$ that transforms an anomaly into a sample of a known class would have an inverse $f_{\boldsymbol{\theta}}^{-1} \in F$ that does not preserve the known class labels because it transforms an instance of a known class into an anomaly. Here is a contradiction, therefore every $f_{\boldsymbol{\theta}} \in F$ must preserve the *Anomalous* class label.

In our case, F is not a group because noise injection has no inverse transformation in F. However, noise injection naturally preserves every label, including *Anomalous*, since it simulates the noise affecting all the manufactured wafers. We can obtain the other transformations $f_{\boldsymbol{\theta}} \in F$ by composing, in any order, a rotation around the center of either $0°$, $90°$, $180°$, or $270°$, an optional horizontal flip and a translation in a random direction with maximum distance ν. The inverse $f_{\boldsymbol{\theta}}^{-1}$ of an element of F can be obtained by composing, in the opposite order, the inverses of the rotation, horizontal flip, and translation that define $f_{\boldsymbol{\theta}}$. This implies that also $f_{\boldsymbol{\theta}}^{-1} \in F$, hence F preserves the *Anomalous* class label by the same argument used above for the case in which F is a group. Thus, we can safely employ the transformations in F for data augmentation at test time without influencing the anomaly-detection performance.

9.2.4 WDM monitoring pipeline

Here we summarize the proposed pipeline to classify a test WDM x: first, we generate a set of n augmented maps

$$x_i = f_{\boldsymbol{\theta}_i}(x), \quad i = 1, \ldots, n, \tag{9.7}$$

where each $\boldsymbol{\theta}_i$ is randomly sampled from Θ (Section 9.2.3). Then, we feed all the augmented versions of x to the network, and average both the anomaly score and the classification scores to stabilize the output, as in [206]. Thus, output of our classifier $\mathcal{K}$ is:

$$\mathcal{K}(x) = \begin{cases} \textit{Anomalous} & \text{if } = \frac{1}{n}\sum_{i=1}^{n} \mathcal{T}\left(x_i\right) > h \\ \widehat{\ell}(x) = \arg\max_{\ell \in \mathcal{L}} \frac{1}{n}\sum_{i=1}^{n} \text{SSCN}(x_i) & \text{otherwise} \end{cases}, \quad (9.8)$$

where h is the threshold defined in Section 9.2.2 for the anomaly score $\mathcal{T}$ and $\text{SSCN}(x_i)$ indicates the classification scores of our SSCN (Section 9.2.1) obtained from x_i. Thus, when a WDM is not detected as *Anomalous*, it is classified by taking the label $\widehat{\ell}(x) \in \mathcal{L}$ maximizing the average classification score over the n augmented versions of x.

9.3 Experiments and Discussion

Our experiments show that: *i*) Submanifold Sparse Convolutional Networks handling Wafer Defect Maps at full resolution outperform traditional CNNs trained on low-resolution Wafer Bin Maps, *ii*) our data augmentation procedure is crucial to achieve good classification performance, and *iii*) our open-set recognition solution based on a GMM fitted on the latent representations extracted by our SSCN can detect anomalous patterns better than state-of-the-art open-set recognition methods applied on top of the same SSCN for a fair comparison.

9.3.1 Experimental Setup

Dataset. We test our solution on the *ST dataset*, which contains $N = 31,893$ WDMs acquired at the STMicroelectronics plant in Agrate Brianza, Italy. These WDMs are either annotated as *Normal*, i.e., they do not contain any defect patterns, or belong to one of the 12 defect classes identified by production engineers at STMicroelectronics. Figure 9.1(b) displays a sample WDM for each of these classes.

In the literature, the open-set recognition performance is typically assessed over datasets with numerous classes such as CIFAR-100 [209], and ImageNet [210] so that a certain number of classes can be taken out during training and considered *Anomalous* at test time. However, in our industrial scenario, we have only 12 defect classes, and WDMs containing anomalous patterns have not been included in the dataset. For this reason, we follow a leave-one-out approach by training our model on all the known classes except a single defect class, which we then consider *Anomalous* at test time.

This is quite a realistic scenario in which we assume to have identified a certain number of defect classes (and the *Normal* class, which is always assumed to be known), and *Anomalous* patterns from a previously unseen class appear during testing.

Figures of Merit. We assess the classification performance of the proposed SSCN (trained on all the 13 known classes) by the *confusion matrix*. We also provide an overall evaluation of the classification performance using two multi-class extensions of the *Area Under the ROC Curve* (AUC). The first one is the *1vsRest*-AUC [211], which is a weighted average of the AUC values obtained in all the binary classification problems where each class is selected in turns as the positive class and all the remaining classes are merged in the negative class (1vsRest). We remark that, since the weight given to each positive class is its frequency in the test set [211], the 1vsRest-AUC is influenced by class proportions. The second one is the *1vs1*-AUC [212], which is the average of the AUC values of the binary classification problems between every pair of classes (1vs1). Contrarily to the 1vsRest-AUC, it has been shown that the 1vs1-AUC is not influenced by the class proportions in the test set [212]. Since the ST dataset is extremely imbalanced, it is important to observe both these figures of merit to analyze the classification performance on under-represented classes and also to assess the impact of our data augmentation procedure.

We employ 10-fold cross-validation and average all these figures of merit over the 10 test folds to provide an overall assessment of the classification performance. We also rank the considered methods (rank $= 1$ for the best method, 2 for the second-best, and so on) according to the 1vsRest-AUC and the 1vs1-AUC, and compute their average rank over the 10 test folds to compare their classification performance, as suggested in [74].

Detecting anomalous patterns is, in fact, a binary classification problem where anomalous instances represent the positive class (*Anomalous*), and the known classes used for training are merged in the negative class (*Known*). Therefore, we can directly compute the AUC of these binary classification problems, which does not depend on class proportions, and this is crucial because anomalous instances are rare. In contrast to other metrics such as F-scores, the AUC does not depend on how detection thresholds on the anomaly scores are set, which allows to evaluate the effectiveness of different methods without setting the thresholds.

To provide an overall assessment of the anomaly-detection performance, we rank the considered methods according to the AUC and compute their average rank over the different anomalous classes, as suggested in [74].

Moreover, we perform the one-sided Wilcoxon Signed-Rank test [75] to assess the statistical significance of the difference between the AUC of the best-ranking solution over the 12 *Anomalous* classes and those of the other methods. Moreover, we apply the nonparametric Mann-Whitney statistical test [18] on the anomaly scores of the best and second-best performing methods on each *Anomalous* class to assess whether the difference between their AUC is statistically significant. This test is suggested in [213] to compare the AUC of two methods on the same binary-classification problem.

9.3.2 Classification of known classes

In this experiment, we evaluate the classification performance of our solution over the 13 classes identified so far by STMicroelectronics engineers. We train our SSCN using the Adam optimizer [192] on an NVIDIA Titan Xp GPU. Training the model for 100 epochs requires about 8 hours, while the average time to classify a WDM is 0.061 ± 0.055 seconds. This time includes the generation and processing of $n = 250$ augmented WDMs as in (9.7). The large variability in the classification time can be explained by the fact that the number of operations executed by an SSC layer directly depends on the input sparsity [9], which in our case can vary substantially from class to class (see Figure 9.1(b)).

Considered methods. We compare the classification performance of the proposed SSCN with traditional CNNs for image classification. To this purpose, we reduce the resolution of the WDMs from $20,000 \times 20,000$ to 224×224 by binning, i.e. converting each WDM to a grayscale image where the intensity of each pixel is the number of defects that fall into the corresponding bin, as in [199]. Then, we use the resulting Wafer Bin Maps to fine-tune the *VGG16* [206] and *ResNet50* [207] models, both pre-trained on the ImageNet dataset [210]. We employ the Adadelta optimizer [214] for the VGG16, and the Adam optimizer [192] for the ResNet50, since we observed that these configurations yield the best performance.

We compare our SSCN to these models because fine-tuning CNNs for image classification is rather standard in wafer monitoring [205]. We do not employ custom architectures for wafer classification [200, 201] since the implementation and parameters of these models, pre-trained on the WM-811K dataset [205], are not publicly available. Both traditional architectures (such as VGG16 and ResNet50) and custom architectures produce very deep models with several million parameters, which would risk overfitting when trained from scratch on the relatively small ST dataset. For a

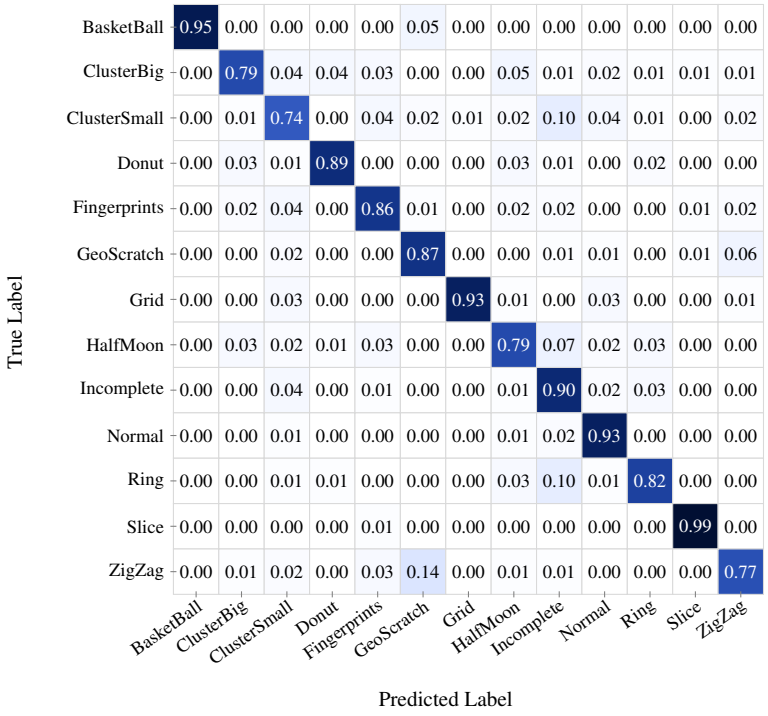


Figure 9.4: *Confusion matrix of our SCNN obtained by 10-fold cross-validation on the ST dataset. Our network achieves very high accuracy on all the classes, and most misclassified samples belong to similar classes such as* ClusterSmall *and* Incomplete*.*

fair comparison with our SSCN, we train and test the VGG16 and ResNet50 using the same data augmentation described in Section 9.2.3 on the original WDMs, before reducing their resolution.

To assess the impact of data augmentation, we also evaluate the performance of our SSCN without data augmentation (both at training and test time), and with traditional augmentation, i.e., using only geometric transformations (translations, rotations, and flips). This experiment evaluates the importance of our custom transformations, namely noise injection and random mixing, in data augmentation.

Results and discussion. Figure 9.4 shows the confusion matrix of our SSCN obtained by 10-fold cross-validation on the ST dataset. Our model achieves excellent classification accuracy, and we observe that the majority of prediction errors occur when our model fails to distinguish two classes containing very similar patterns (e.g., *ClusterSmall* and *Incomplete*, see Figure 9.1(b)). In Table 9.1, we report in the first three columns the accuracy achieved over each class (i.e., the diagonal of the confusion matrix) by our

Table 9.1: *Classification accuracy of the considered methods obtained by 10-fold cross-validation. We also report the average 1vsRest-AUC and 1vs1-AUC, the average rank on the 10 folds according to these metrics, and the p-values of the Wilcoxon test assessing the significance of the difference between the best-ranking method and each alternative [74]. We also evaluate our SSCN without augmentation and all the considered methods with traditional augmentation, and report the performance differences w.r.t. our augmentation procedure. Here the Wilcoxon test assesses the significance of the performance difference compared to the same method with our augmentation procedure.*

	OUR AUGMENTATION			W/O AUG.	TRADITIONAL AUGMENTATION		
Class accuracy	SSCN	VGG16	ResNet50	SSCN	SSCN	VGG16	ResNet50
BasketBall	**0.9545**	0.9091	0.9091	−0.7273	−0.0909	−0.2727	−0.0909
ClusterBig	0.7880	0.7805	**0.7917**	−0.1013	+0.0075	−0.0809	−0.0188
ClusterSmall	0.7439	**0.7844**	0.7216	−0.0221	+0.0143	−0.0436	+0.0492
Donut	**0.8933**	0.8178	0.8133	−0.2311	−0.0356	−0.0221	−0.0133
Fingerprints	**0.8571**	0.7967	0.7857	−0.3764	+0.0110	−0.0773	−0.0110
GeoScratch	0.8694	**0.8774**	0.8678	−0.3264	−0.0287	−0.1284	−0.0239
Grid	0.9261	**0.9545**	0.9375	−0.2841	−0.0227	−0.0352	+0.0000
HalfMoon	**0.7912**	0.7349	0.7470	−0.3855	−0.0221	−0.0064	+0.0000
Incomplete	0.8965	**0.9097**	0.8904	−0.1565	−0.0136	−0.0177	−0.0037
Normal	0.9332	0.9132	**0.9596**	+0.0248	−0.0045	−0.0306	−0.0093
Ring	0.8224	**0.8567**	0.8552	−0.0806	+0.0060	−0.0026	−0.0239
Slice	**0.9859**	0.9437	0.9437	−0.4366	−0.0141	+0.0037	+0.0000
ZigZag	**0.7681**	0.7638	0.7064	−0.3277	+0.0213	−0.1472	+0.0000
1vsRest-AUC	0.9858	0.9818	**0.9868**	−0.0239	−0.0007	−0.0013	−0.0019
Avg. rank	1.8000	3.0000	**1.2000**				
Wilcoxon-p	0.0142	0.0025	–	0.0025	0.0844	0.0844	0.0372
1vs1-AUC	**0.9893**	0.9846	0.9849	−0.0614	−0.0018	−0.0066	−0.0029
Avg. rank	**1.0000**	2.6000	2.4000				
Wilcoxon-p	–	0.0025	0.0035	0.0025	0.0083	0.0025	0.0063

SSCN, the VGG16, and the ResNet50, all trained and tested using our augmentation procedure. Table 9.1 also reports the average 1vsRest-AUC and 1vs1-AUC achieved by the considered methods on the test folds to compare their overall performance. As recommended in [74], we also compute the average rank of the considered method and the p-values of the one-sided Wilcoxon Signed-Rank test [75] assessing whether the difference between the 1vsRest-AUC and the 1vs1-AUC of the best-ranking method and those of the other methods is significant.

First, we observe that methods trained and tested with our data augmentation procedure achieve consistently high accuracy, and that our SSCN achieves the best accuracy on 6 out of 13 classes. Overall, our SSCN is the best in terms of 1vs1-AUC on all the 10 test folds (avg. rank$= 1$), and the Wilcoxon Signed-Rank test confirms that the performance differences with traditional CNNs are statistically significant (p-value ≤ 0.05). Notably, our SSCN outperforms the VGG16, which has a similar architecture to our SSCN, in both 1vsRest-AUC and 1vs1-AUC. The ResNet50 model is the

best in terms of 1vsRest-AUC thanks to its high accuracy on the *Normal* class ($\approx 96\%$), which is by far the most represented in the ST dataset (see Figure 9.1(b)), and the 1vsRest-AUC is sensitive to class proportions [211]. Our SSCN provides a better trade-off between the accuracy on the *Normal* class ($> 93\%$) and the accuracy on under-represented classes, as it outperforms the ResNet50 on 9 out of 12 defect classes, often by a substantial margin. This explains the significant difference between the two in terms of 1vs1-AUC, which is not influenced by class proportions [212].

To assess the impact of our data augmentation procedure on classification, we report in the fourth column of Table 9.1 the performance differences between the proposed solution, which leverages data augmentation both during training and testing, and the same SSCN architecture trained and tested without augmentation. Remarkably, training the VGG16 and ResNet50 without data augmentation leads to overfitting due to the fact that the ST dataset is relatively small and the considered CNNs have several million trainable parameters. In contrast, our SSCN can be effectively trained without augmentation since it has a substantially lower number of parameters. However, without augmentation, our SSCN achieves substantially lower classification accuracy on all classes except the *Normal* class, which is by far the most common in the ST dataset (see Figure 9.1(b)) and therefore can also be learned very accurately ($\approx 96\%$) without augmentation. The SSCN without augmentation also achieves lower 1vsRest-AUC and 1vs1-AUC compared to our SSCN trained using our data augmentation procedure, and the differences are statistically significant (p-value ≤ 0.05) according to the Wilcoxon test.

Similarly, the last three columns of Table 9.1 report the performance differences between the models trained and tested with our augmentation procedure (which includes noise injection and random mixing) and the same models trained and tested using only traditional augmentation, namely geometric transformations. We observe that traditional augmentation yields lower accuracy on most classes, and lower 1vsRest-AUC and 1vs1-AUC compared to the same model with our augmentation. According to the Wilcoxon test, the difference in 1vsRest-AUC is significant (p-value ≤ 0.05) for the ResNet50. Remarkably, the differences in 1vs1-AUC are significant for all the considered models, which confirms that our augmentation procedure improves the robustness to class imbalance, since the 1vs1-AUC is not influenced by class proportions [212].

9.3.3 Detection of anomalous patterns

In this experiment, we assess the performance of our solution in detecting WDMs containing anomalous defect patterns. We train 12 different models, each time taking out one of the defect classes, which we then consider *Anomalous* at test time, as illustrated in Section 9.3.1. Since some of the identified defect classes in the ST dataset (e.g., *ClusterSmall* and *Incomplete*) are very similar, we expect some anomalies to be more difficult to detect than others.

Considered methods. We compare the anomaly-detection performance of our solution against methods implementing the following open-set recognition techniques on the same SSCN:

- *SoftMax* is a widely employed open-set recognition baseline [131]. It relies on the intuition that a classifier trained on known classes would probably classify anomalous instances with low confidence. Let $v = (v_1, \ldots, v_{\#\mathcal{L}})$ be the score vector associated to a WDM x, i.e., the output of the last fully-connected layer of our SSCN (or any neural network). Then, the SoftMax function is defined as:

$$p_j = \text{SoftMax}(v)_j = \frac{\exp(v_j)}{\sum_{i=1}^{\#\mathcal{L}} \exp(v_i)}, \tag{9.9}$$

  which guarantees that the final classification scores are non-negative and sum to 1. The anomaly score is then simply defined as $\mathcal{T}(x) = -\max_j p_j$, which is high when the posterior probability of the selected class is low.

- *PreSoftMax* is based on the same idea as SoftMax, but uses as anomaly score the opposite of the maximum classification score before applying SoftMax, i.e., $\mathcal{T}(x) = -\max_j v_j$. In fact, even when all the scores v of an anomalous instance x are low, SoftMax might still assign x to one of the known classes with extremely high confidence since posteriors are forced to sum to 1 [131], thus preventing SoftMax to detect x as *Anomalous*. The PreSoftMax scores are not subject to this constraint, so they might be more informative than the SoftMax scores in anomaly detection.

- *OpenMax* is a function designed in [133] to replace SoftMax in open-set classifiers at test time. This method employs as anomaly score the likelihood of a Weibull distribution fitted on the distances between the

PreSoftMax score vectors v of a classifier and the Mean Activation Vectors (MAVs) of each known class, computed from the training set.

- We denote by *SME* the open-set recognition method based on SoftMax Entropy proposed in [132]. SME relies of the observation that the posterior probability vectors p of anomalous instances tend to have a larger Shannon entropy compared to instances of known classes, thus the anomaly score is defined as the entropy of the posterior probabilities, i.e., $\mathcal{T}(x) = -\sum_j p_j \log p_j$. This method is part of a broader solution for open-set recognition and one-shot detection [132], but here we only consider the proposed open-set recognition solution.
- We denote by *IFOR* the open-set recognition method proposed in [135] that applies the Isolation Forest [104] outlier detector to the latent representation of the WDMs, i.e., to $L(x)$. The method presented in [135] considers a latent representation including the features extracted by a classifier and an embedding of the class labels to a semantic word space. However, in our industrial setting, there is no meaningful embedding of the labels to a semantic space. Hence we only consider $L(x)$ as input.
- We denote by *CI* the open-set recognition method proposed in [136] that applies an outlier detector based on Confidence Intervals to the latent representation of the classifier, treating each component of $L(x)$ as an independent variable. This method computes a confidence interval $\widehat{\mu} \pm \lambda\widehat{\sigma}$ for each component, where $\widehat{\mu}$ and $\widehat{\sigma}$ are estimated from the training set, and its anomaly score is the number of components of $L(x)$ that exceed their confidence intervals.

Each of these methods produces a anomaly score and, similarly to ours, requires a decision threshold computed on a small part of the dataset, as illustrated in Section 9.2.2. However, we evaluate the anomaly-detection performance by the AUC, which does not depend on the threshold, nor, most importantly, on class proportions. SoftMax, PreSoftMax and SME do not require training other than that of the SSCN. In OpenMax, we follow the procedure presented in [133] to compute the MAVs and fit the Weibull model from the entire dataset used to train the SSCN. For a fair comparison with our solution, we adopt in IFOR and CI the same procedure illustrated in Section 9.2.2, so we fit the Isolation Forest and compute the parameters $\widehat{\mu}$, $\widehat{\sigma}$ from a small portion of the dataset (5%) that was not used for training.

We remark that open-set recognition solutions based on the reconstruction error of autoencoders [138–140] have been designed for images and

Table 9.2: *AUC of the considered methods in detecting each* Anomalous *class. Bold indicates the best values, and the underlined values are significantly higher than the second-best on the same anomalous class (p-value ≤ 0.05) according to the Mann-Whitney test [213]. We also report the average rank over the anomalous classes and the p-values of the Wilcoxon test assessing the significance of the performance differences between our solution and each alternative [74].*

Anomaly	SoftMax [131]	PreSoftMax	OpenMax [133]	SME [132]	IFOR [135]	CI [136]	GMM (ours)
BasketBall	0.3697	0.3234	0.3455	0.3191	0.9812	0.4009	**0.9894**
ClusterBig	0.7022	0.8272	0.7216	0.6776	0.9541	0.6195	**0.9543**
ClusterSmall	0.8672	**0.9003**	0.8775	0.8813	0.7694	0.8169	0.8227
Donut	0.4692	0.8508	0.6614	0.4270	**0.9596**	0.4916	0.9515
Fingerprints	0.8418	0.9222	0.8556	0.8234	0.8710	0.8214	**0.9249**
GeoScratch	0.6377	0.7282	0.6699	0.6177	0.7997	0.8557	**0.8562**
Grid	0.6716	0.5297	0.5454	0.6438	**0.8933**	0.4989	0.8907
HalfMoon	0.7465	0.8566	0.7918	0.7375	0.8234	0.8641	**0.8873**
Incomplete	0.8162	**0.8438**	0.8319	0.8258	0.5752	0.7672	0.7244
Ring	0.4898	0.4928	0.5211	0.4626	0.8590	0.7376	**0.9196**
Slice	0.8313	0.6944	0.7772	0.7723	0.8887	0.7969	**0.9050**
ZigZag	0.6448	0.7251	0.6647	0.6269	0.8802	0.8865	**0.9149**
Avg. rank	4.8333	3.7500	4.2500	5.7500	3.0833	4.4167	**1.9167**
Wilcoxon p	0.0024	0.0061	0.0024	0.0012	0.0024	0.0024	–

cannot be directly applied to full-resolution WDMs, hence we have not considered them. Instead, we have focused on open-set recognition methods built on top of our SSCN, pre-trained to accurately classify WDMs from known classes. This enables a fair comparison between the anomaly detection power of methods that achieve the same classification performance over known classes, which has been shown to strongly influence the open-set recognition performance [144].

Results and discussion. Table 9.2 reports the AUC achieved by the considered methods in the anomaly-detection problem. As suggested in [74], we also report the average rank of the considered methods and the p-values of the one-sided Wilcoxon Signed-Rank test [75] assessing whether the difference between the AUC of our solution, which has the lowest average rank, and those of the other methods is significant. As suggested in [213], we apply the nonparametric Mann-Whitney test on the anomaly scores of the best and second-best performing methods on each *Anomalous* class to assess whether the difference between their AUC is statistically significant.

Table 9.2 indicates that our solution is the best in detecting 8 out of the 12 *Anomalous* classes. In 5 of these, the performance difference with the second-best method is statistically significant (p-value ≤ 0.05) according to the Mann-Whitney test [213]. Most importantly, our solution achieves the best average rank among the considered methods, and the p-values of the Wilcoxon test show that there is enough statistical evidence to claim that

GMM performs significantly better (p-value ≤ 0.05) than the alternatives. As we note in Section 9.3.2, some classes are very similar and might be easily confused by our SSCN (e.g., *ClusterSmall* and *Incomplete*), therefore most of the considered methods find it difficult to detect them as anomalous.

The AUC values achieved by IFOR follow a trend similar to those of GMM, even though inferior in most cases. These results can be explained by observing that $L(x)$ can be expected to follow a multimodal distribution due to the presence of different known classes, and the GMM is explicitly multimodal, while IFOR is a nonparametric outlier detector. In contrast, SME and CI achieve relatively good results only on few specific *Anomalous* classes, such as *ClusterSmall*.

SoftMax performs worse than most of the other methods: as expected, anomalous instances are often classified with high confidence due to the SoftMax operation. OpenMax performs better than SoftMax according to the average ranks but worse than GMM. Perhaps surprisingly, PreSoftMax performs very well, and represents the best method to detect *ClusterSmall* and *Incomplete*, which are quite difficult to distinguish.

The fact that PreSoftMax outperforms OpenMax and SoftMax outperforms SME shows that the highest classification score (before and after applying the SoftMax) is the most informative for anomaly detection in WDMs, confirming the intuition that anomalous instances are likely to be classified with low confidence. By also considering the other scores, OpenMax and SME search for outliers in a 12-dimensional space where most dimensions are not informative for anomaly detection, while PreSoftMax and SoftMax operate on a 1-dimensional space since they only consider the highest score. For this reason, we speculate that OpenMax and SME might suffer from an effect similar to *detectability loss* [36] i.e., the higher the dimensionality, the harder it is to detect a distribution change.

9.4 Discussion

The effective and automatic monitoring of large volumes of WDMs is a crucial challenge to improve the quality and efficiency of industries operating in semiconductor manufacturing. Our results show that a simple deep-learning model based on Submanifold Sparse Convolutions is substantially more robust to class imbalance than traditional CNNs. This suggests that binning WDMs to reduce their size before feeding them to a CNN leads to a relevant information loss, thus an SSCN is the perfect instrument since it allows to efficiently process WDMs at their original resolution.

Moreover, we are the first to address the open-set recognition problem

on WDMs, which is of paramount importance for the industry because anomalous patterns might occur due to production issues that have not been observed yet. Our results show that applying a simple outlier detector based on a GMM to the latent representation of our SSCN yields better anomaly-detection performance than state-of-the-art open-set recognition methods, which we also applied on top of our SSCN to obtain a fair comparison.

Our classifier is currently employed in several STMicroelectronics facilities all over the world. Future work will address the deployment of our open-set recognition pipeline to monitor the production of wafers at the STMicroelectronics plant in Agrate Brianza, Italy. Moreover, we are investigating the possibility to jointly train our SSCN and the GMM, using a custom loss function to produce a latent representation that follows a Gaussian mixture distribution, as in [110].

CHAPTER 10

Concluding Remarks

In this thesis, we address anomaly detection, namely the problem of monitoring data to determine whether the data-generating process operates in normal conditions or not. This problem has relevant applications in several domains, and each domain introduces specific challenges due to the structure of the data at hand. We address anomaly detection under two different sets of modeling assumptions on the data-generating process.

In the first part of the thesis, we model the data as consecutive realizations of a random vector, and focus on the problem of detecting a permanent distribution change in the datastream. The main challenge is designing an online and nonparametric change-detection algorithm for multivariate datastreams that can effectively control false alarms by maintaining the target Average Run Length (ARL_0), namely the expected time before a false alarm. We propose QuantTree Exponentially Weighted Moving Average (QT-EWMA), a change-detection algorithm combining a QuantTree histogram, used to model the initial distribution from a training set, and a new, powerful statistic based on Exponentially Weighted Moving Average (EWMA). To effectively operate also when the training set is small, we propose QT-EWMA-update, where we incrementally update the bin probabilities of the QuantTree histogram using the incoming data.

The theoretical properties of QuantTree guarantee that the distribution of the QT-EWMA and QT-EWMA-update statistics is independent from the data distribution and dimension, and allow us to design an efficient Monte-carlo procedure to compute thresholds that maintain a target ARL_0 on any datastream. Our experiments on synthetic and real-world data empirically demonstrate that our algorithms effectively control the ARL_0 and obtain detection delays that are lower than or comparable to those of state-of-the-art change-detection algorithms based on the Maximum Mean Discrepancy statistic (MMD), especially when the training set is small.

We also address concept-drift detection, namely a change-detection problem where the data samples are the object of a classification problem. We propose Class Distribution Monitoring (CDM), a new concept-drift detection algorithm leveraging multiple instances of QT-EWMA to monitor the class-conditional distributions. Even though it combines multiple change-detection tests, we demonstrate that CDM maintains the same target ARL_0 as the QT-EWMA monitoring each class-conditional distribution. Our experiments on synthetic and real-world data empirically show that CDM effectively controls the ARL_0 and outperforms concept-drift detection algorithms that monitor either the overall data distribution or the error rate of an underlying classifier.

As a relevant application of change detection, we investigate sequential side-channel attacks, a class of cryptographic attacks where a distinguisher, namely a statistic of some side-channel data (e.g. the power consumption of a cryptosystem), is used to recover the private key one bit at a time. To determine the full potential danger posed by these attacks, we address the problem of automatically detecting and correcting errors in these attacks. We propose to monitor the univariate datastream of distinguisher values using a state-of-the art change-detection algorithm to detect errors, and to correct them by a brute-force search over a small window centered at each detected error, using the new distinguisher values to select the correct combination. Our experiments on synthetic and real-world side-channel measurements demonstrate that our error-detection and correction procedure can substantially improve the success rate of sequential attacks, outperforming existing solutions that detect an error whenever the distinguisher falls below a pre-defined threshold.

In the second part of the thesis, we model the data as individually acquired point clouds, namely lists of coordinates describing objects in a 2D or 3D space. In this case, the goal is to assess whether a test point cloud belongs to the normal class, which is the only class represented in the training set, or instead must be considered anomalous. The main challenge of point

cloud processing is the lack of a grid structure, which prevents from using traditional convolutions. We propose the Composite Layer, a new operator to process point clouds in deep neural networks. Differently from the existing convolutional layers for point clouds, our Composite Layer first extracts the spatial information from the coordinates of the points, and then shares this information among the convolutional filters. Compared to the alternatives, our Composite Layer operates more similarly to a convolutional layer on an image, where the spatial information is embedded in the grid structure and thus is implicitly shared among the filters. Moreover, our Composite Layer can be designed with greater flexibility in terms of structure and number of parameters.

We define a convolutional and a non-convolutional Composite Layers, and use them to implement CompositeNets, deep neural networks that we train for classification and, most remarkably, anomaly detection, following a self-supervised approach. Our experiments on synthetic and real-world point clouds show that, in classification, our CompositeNets approach the accuracy of a more sophisticated network based on an alternative convolutional layer. Most remarkably, in anomaly detection our self-supervised CompositeNets outperform shallow baselines using hand-crafted features, a variational autoencoder that is the only deep-learning based anomaly-detection method for point clouds, and similar self-supervised networks based on alternative convolutional layers.

As a relevant application of anomaly detection in point clouds, we address the problem of monitoring Wafer Defect Maps (WDMs), namely point clouds listing the 2D coordinates where defects lie within a silicon wafer. These coordinates belong to a regular grid defined by the precision of the inspection machines, however the resolution is huge and prevents traditional CNNs from directly processing WDMs. In normal conditions, defects are rare and randomly distributed, while defects grouped in patterns indicate problems in the manufacturing process. Some classes of defect patterns have already been studied by production engineers, while anomalous patterns might occur due to unexpected issue in the production line, and therefore must be detected as soon as possible. For this reason, we cast WDM monitoring as an open-set recognition problem, where the goal is to recognize instances from a set of known classes and to detect anomalies, i.e. instances that do not belong to any known class.

We propose to employ a Submanifold Sparse Convolutional Network (SSCN), which can efficiently process high-resolution, sparse images such as WDMs. In particular, we train a custom SSCN on the known classes, and to detect anomalous patterns by applying an outlier detector based on

a Gaussian Mixture Model (GMM) fitted on the latent representation, i.e. the output of the penultimate layer of our SSCN. Since our dataset is extremely imbalanced, we employ a data augmentation procedure combining traditional and custom transformations. Since our dataset does not contain any anomalous pattern, to evaluate the open-set recognition performance we design an experimental setup in which we train our multiple SSCNs, each time taking out a defect class from the training set and considering it anomalous during testing. Our experiments on WDMs acquired at the STMicroelectronics plant in Agrate Brianza, Italy show that, in classification, our SSCN outperforms traditional CNNs trained on low-resolution images obtained by binning the WDMs. In particular, our results suggest that our SSCN is more robust to class imbalance and overfitting. In the anomaly-detection task, our solution outperforms state-of-the-art open-set recognition methods, which we also apply on top of our SSCN to obtain a fair comparison.

10.1 Future Work

The research presented in this thesis can be expanded following different directions. First, we have shown that QT-EWMA is an effective extension of the QuantTree algorithm to online change detection, but it might be worth investigating how to combine QuantTree with other monitoring schemes such as the Change Point Model (CPM) [12] or the Binned Generalized Cumulative Sum (BG-CuSum) [50]. Another possible extension of our work involves CDM. For simplicity, here we employ the same change-detection algorithm (namely, an instance of QT-EWMA) to monitor each class-conditional distribution. However, it might be possible to boost the concept-drift detection performance using QuantTree histograms with different number of bins, or even different change-detection algorithms (as long as they control the ARL_0), leveraging prior knowledge on the class-conditional distributions.

Our work on point clouds might be extended by using our Composite Layers to implement new anomaly-detection networks [114] to overcome the intrinsic limitations of the self-supervised approach, which requires to select domain-dependent geometric transformations. A research direction we do not pursue in this thesis is the design of deep-learning methods to detect anomalous regions within point clouds. This problem has been widely studied on images, but not on point clouds, mainly due to the lack of suitable datasets, which are now starting to be released [126].

Besides the deployment of our WDM monitoring pipeline in the STMi-

croelectronics plant in Agrate Brianza, our open-set recognition solution might be extended by jointly training our SSCN and the GMM, using a custom loss function to learn a latent representation that follows a Gaussian mixture distribution, following the approach presented in [110] to train a deep autoencoder for anomaly detection.

APPENDIX $\mathcal{A}$

Additional Results on QT-EWMA

Here we report and comment the results of our change-detection experiments on the following datasets: Credit Card Fraud Detection ("credit", $d = 28$) from [71], Sensorless Drive Diagnosis ("sensorless", $d = 48$), MiniBooNE particle identification ("particle", $d = 50$), Physicochemical Properties of Protein Ternary Structure ("protein", $d = 9$), El Niño Southern Oscillation ("niño", $d = 5$), and two of the Forest Covertype datasets ("spruce" and "lodgepole", $d = 10$) from the UCI Machine Learning Repository [72]. For brevity, in Chapter 4 we have reported only the average results.

Empirical ARL$_0$. The comparison between the empirical and target ARL$_0$ on simulated Gaussian datastreams sampled from the credit [71] (see Figure A.1 (a,b,c,d)) and all the considered UCI datasets [72] (see Figures A.2–A.7) confirms that QT-EWMA, QT-EWMA-update and SPLL-CPM control the ARL$_0$ very accurately, independently from the data dimension d and the training set size N, as shown in Chapter 4. The empirical ARL$_0$ of QuantTree is higher than the target, as we expected from Proposition 4.2, while Scan-B cannot maintain high target ARL$_0$.

Detection delay vs false alarms. When the training set is small ($N = 64$, 128, 256) QT-EWMA-update is the best method in terms of detection delay over the particle (Figure A.3 (b,d,f)) and niño (Figure A.5 (b,d,f)) datasets, and is outperformed only by SPLL-CPM over the other datasets, though only slightly over spruce (Figure A.6 (b,d,f)) and lodgepole (Figure A.7 (b,d,f)). This confirms that, overall, SPLL-CPM slightly outperforms QT-EWMA-update over the UCI+credit datasets when $N = 64, 128, 256$, as shown by the average results presented in Chapter 4. In contrast, Scan-B yields higher detection delays compared to QT-EWMA-update on all the considered datasets when $N = 64, 128, 256$.

As observed in Chapter 4, QT-EWMA-update outperforms QT-EWMA in terms of detection delay when N is small (especially when $N = 64, 128$) over real-world datastreams (Figures A.1–A.7). These results confirm that updating the QuantTree histograms improves the detection performance when the initial training set is small. Moreover, over all the considered datasets we have that QT-EWMA and QT-EWMA-update outperform QuantTree, consistently with the experiments illustrated in Chapter 4.

When the training set is large ($N = 4096$), Scan-B substantially improves its detection delays, being the best-performing method over the credit dataset (Figure A.1 (h)) and approaching QT-EWMA over the sensorless (Figure A.2 (h)) and particle (Figure A.3 (h)) datasets. QT-EWMA outperforms all the other methods over the particle (Figure A.3 (h)) and niño (Figure A.5 (h)) datasets, while SPLL-CPM yields the lowest detection delays on the remaining datasets, although it only slightly outperforms QT-EWMA over the protein (Figure A.4 (h)), spruce (Figure A.6 (h)) and lodgepole (Figure A.7 (h)) datasets. These plots show that QT-EWMA is very effective on the UCI+credit datasets in terms of detection delay, confirming the average results reported in Chapter 4. Moreover, these experimental results confirm our observation that Scan-B requires a large training set to achieve good detection performance.

We recall that, when the false alarm probability of a change-detection algorithm is a constant α at each time t, its stopping time t^* is a Geometric random variable having parameter α [52]. Hence, when the ARL_0 is controlled by setting a constant false alarm probability $\alpha = 1/\text{ARL}_0$, the false alarm probability at any time t corresponds to the Geometric sum:

$$\mathbb{P}_{\phi_0}(t^* \leq t) = \sum_{k=1}^{t} \alpha(1-\alpha)^{k-1} = 1 - (1-\alpha)^t, \tag{A.1}$$

thus the change-detection algorithm can control the false alarm rate at any time t, as we show in Chapter 4. The results presented here over datas-

treams sampled from the considered UCI+credit datasets (Figure A.1-A.7) confirm that QT-EWMA, QT-EWMA-update and SPLL-CPM approach the target values computed by (A.1) at $\text{ARL}_0 = 500, 1000, 2000, 5000$. As in Chapter 4, we observe that QuantTree has fewer false alarms than expected, and this is a consequence the fact that its empirical ARL_0 is greater than the target due to Proposition 4.2. In contrast, SPLL has more false alarms than expected since its empirical ARL_0 is slightly lower than the target, which confirms that computing the thresholds by bootstrap over a limited training set yields inaccurate estimates, as we observe also in Chapter 4. The false alarms of Scan-B, instead, exhibit a completely different behavior, which also depends on the data distribution, confirming that its thresholds do not yield a constant false alarm probability. All in all, these results confirm those presented in Chapter 4, showing that QT-EWMA and QT-EWMA-update can control the false alarm rates in all the considered datasets and monitoring scenarios.

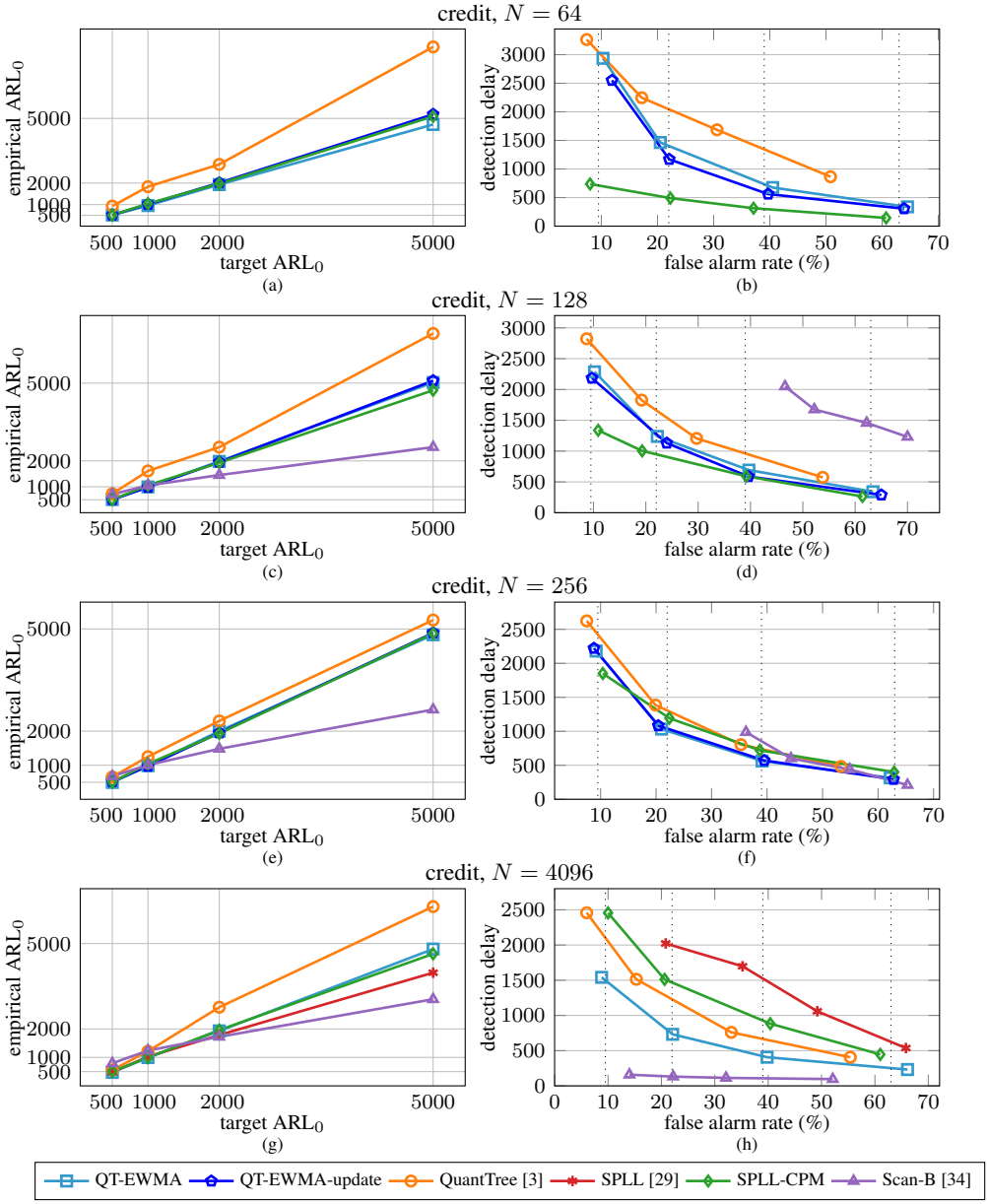


Figure A.1: *Experimental results over the credit dataset ($d = 28$) [71].* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best method is SPLL-CPM when $N = 64, 128$. QT-EWMA and QT-EWMA-update outperform all the other methods when $N = 256$, while Scan-B is the best method when $N = 4096$. Only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (A.1)*, which are represented in the plots by vertical dotted lines.*

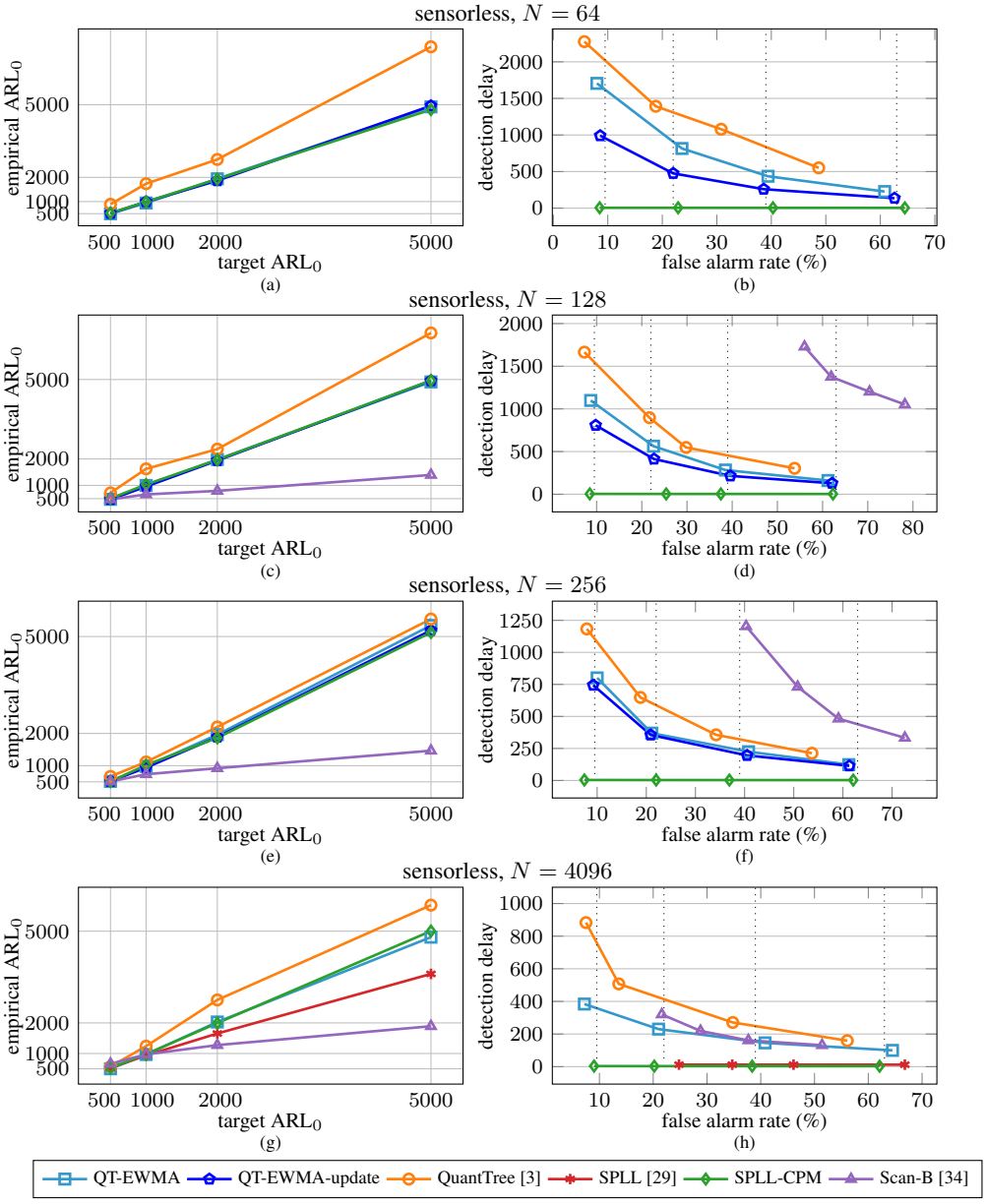


Figure A.2: *Experimental results over the sensorless dataset ($d = 48$) [72].* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, SPLL-CPM is the best method in all the considered scenarios, and SPLL achieves similar results when $N = 4096$. We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (A.1)*, which are represented in the plots by vertical dotted lines.*

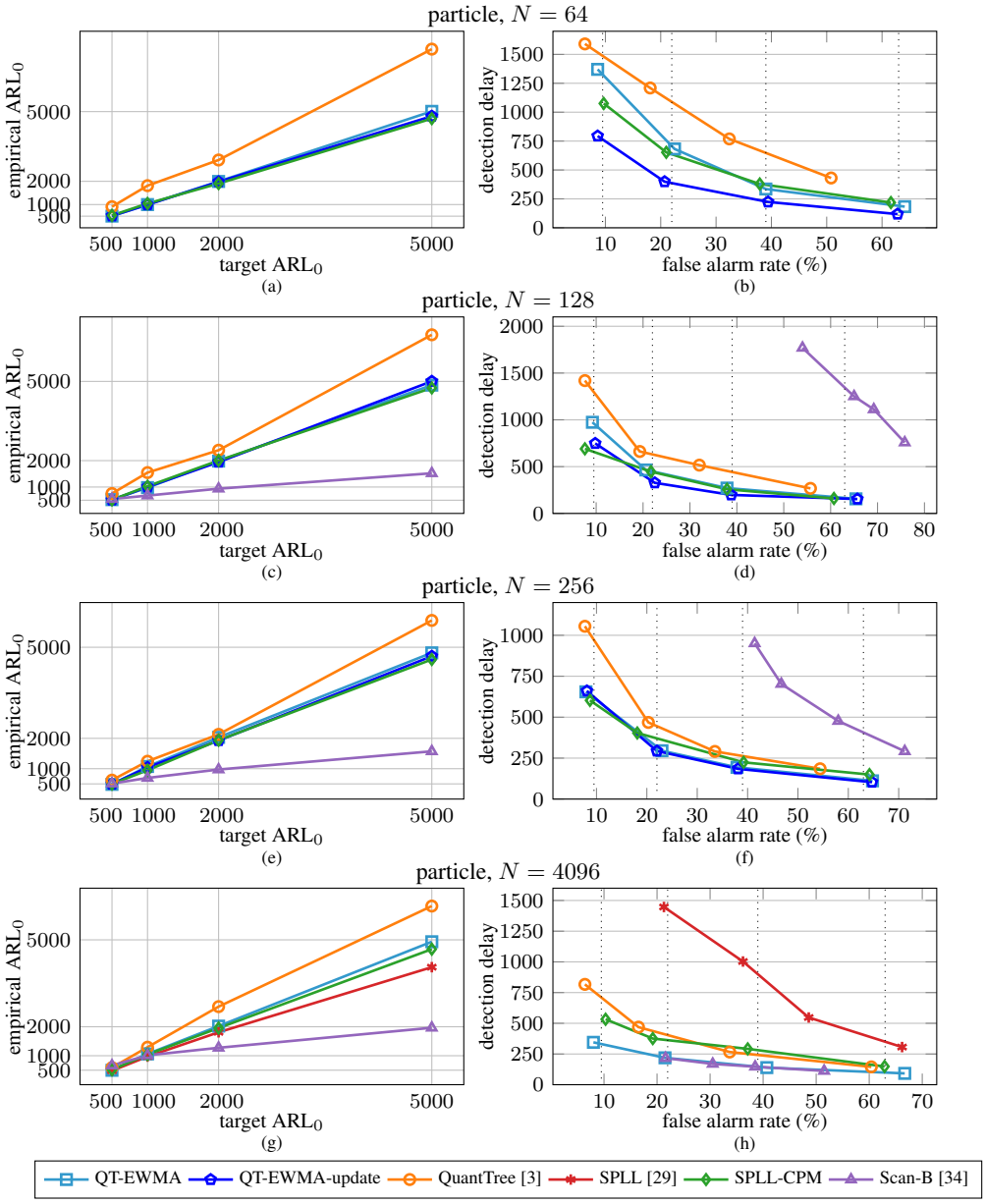


Figure A.3: *Experimental results over the particle dataset ($d = 50$) [72].* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best method is QT-EWMA-update when $N = 64, 128, 256$, and QT-EWMA (on par with Scan-B) when $N = 4096$. We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (A.1)*, which are represented in the plots by vertical dotted lines.*

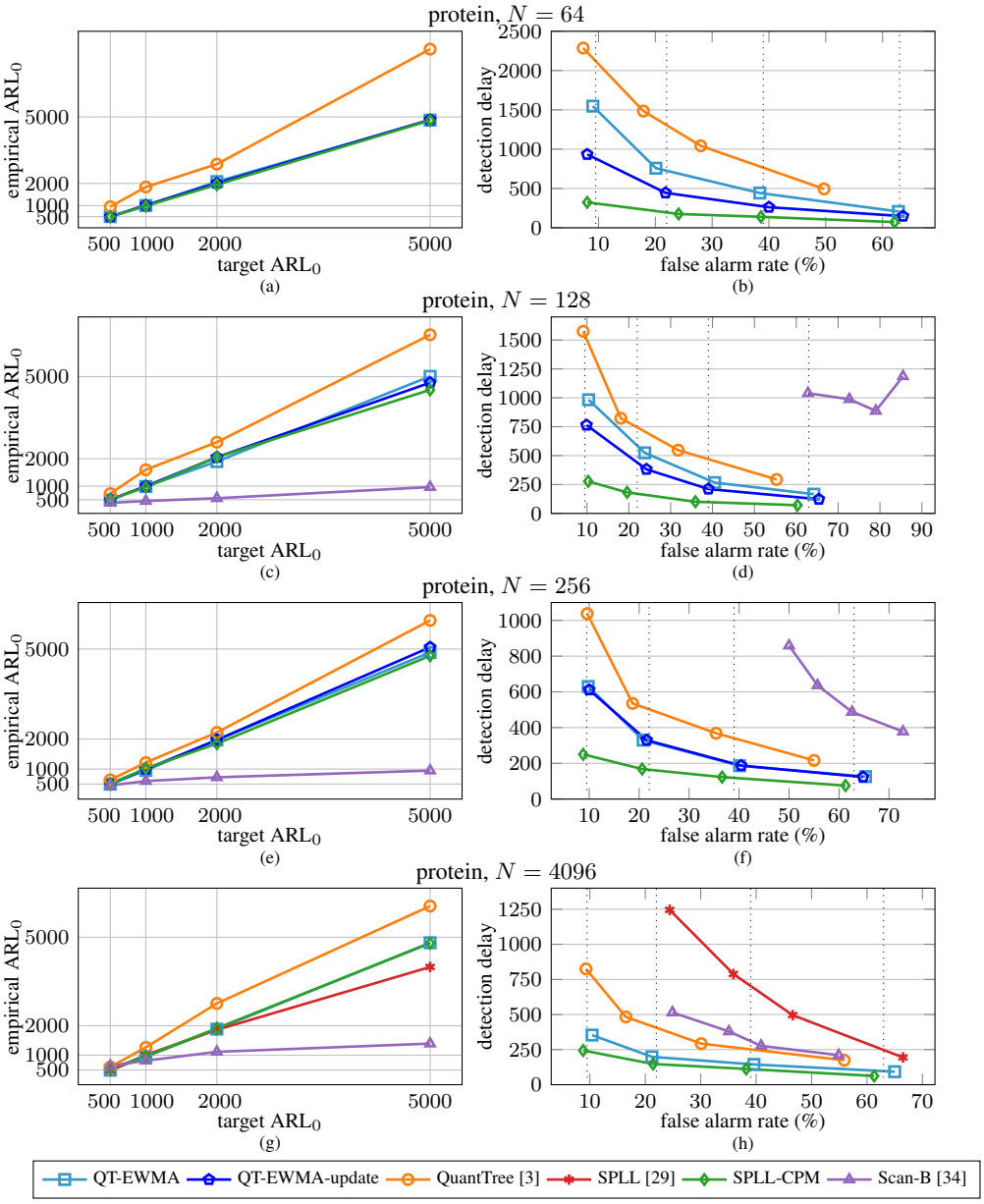


Figure A.4: *Experimental results over the protein dataset ($d = 9$) [72].* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best method is SPLL-CPM for all the considered values of N, approached by QT-EWMA when $N = 4096$. We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (A.1)*, which are represented in the plots by vertical dotted lines.*

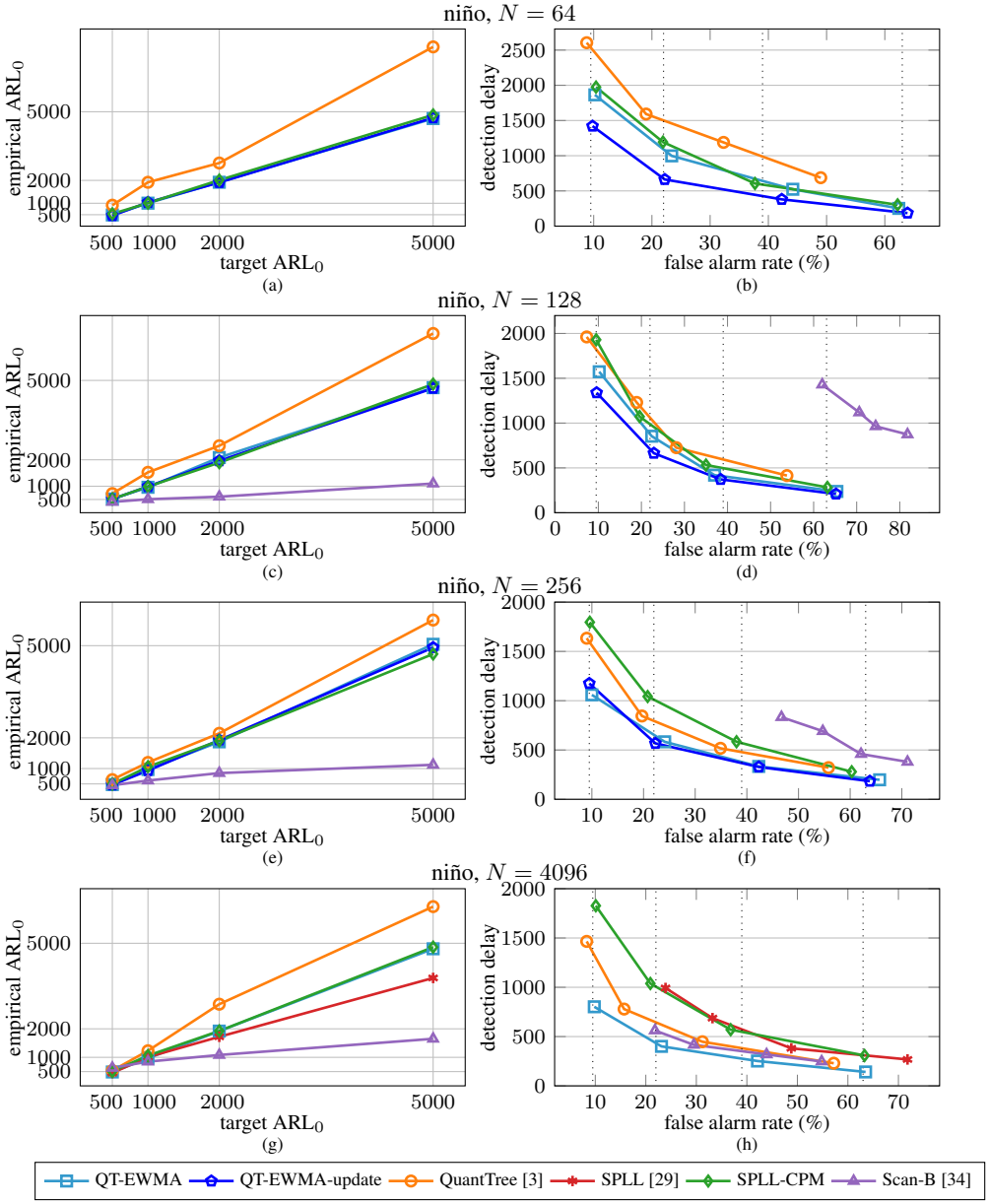


Figure A.5: *Experimental results over the niño dataset ($d = 5$) [72].* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best method is QT-EWMA-update when $N = 64, 128, 256$, and QT-EWMA when $N = 4096$. We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (A.1)*, which are represented in the plots by vertical dotted lines.*

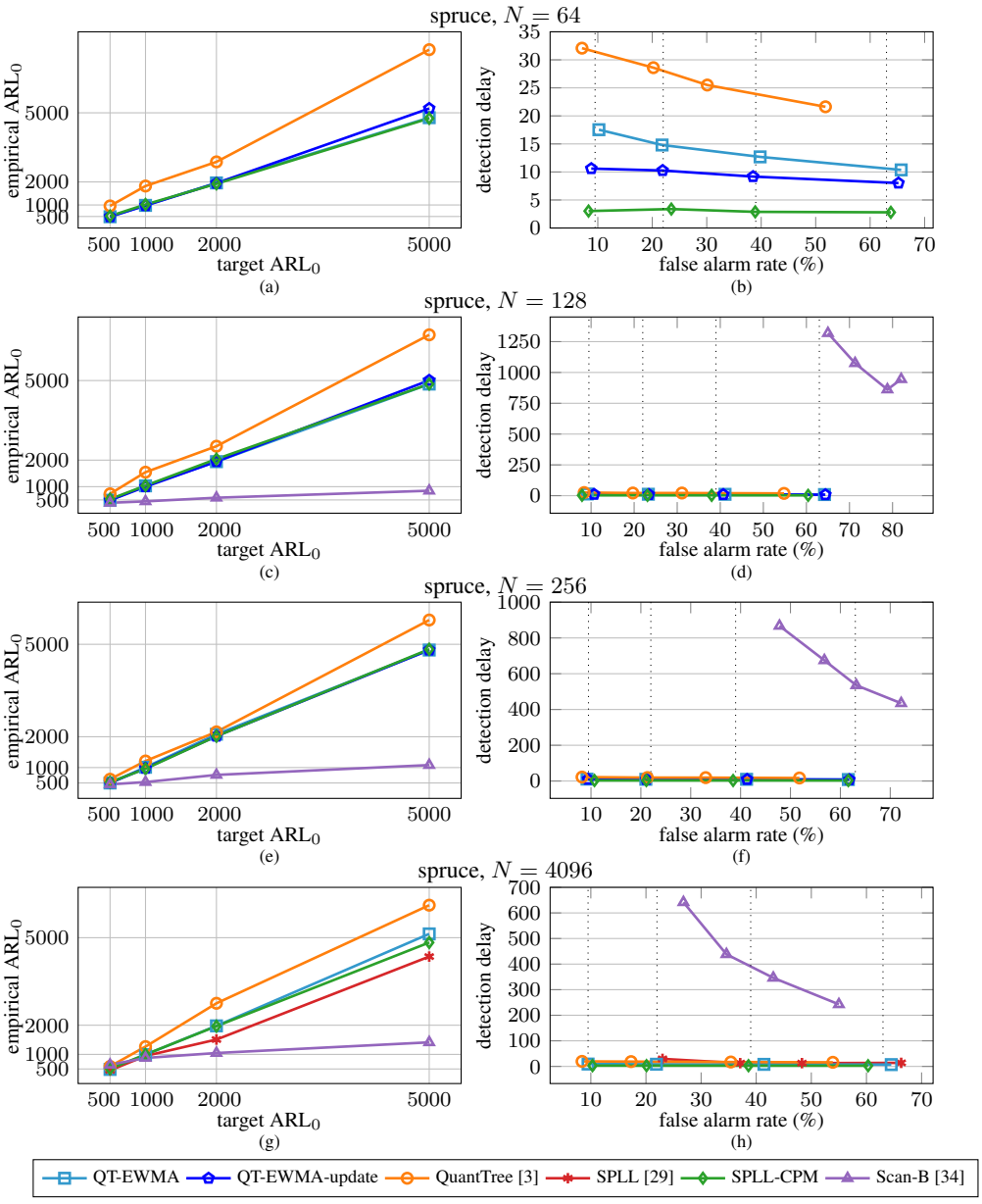


Figure A.6: *Experimental results over the spruce dataset ($d = 10$) [72].* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best method is SPLL-CPM, although all the methods except Scan-B perform similarly. We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (A.1)*, which are represented in the plots by vertical dotted lines.*

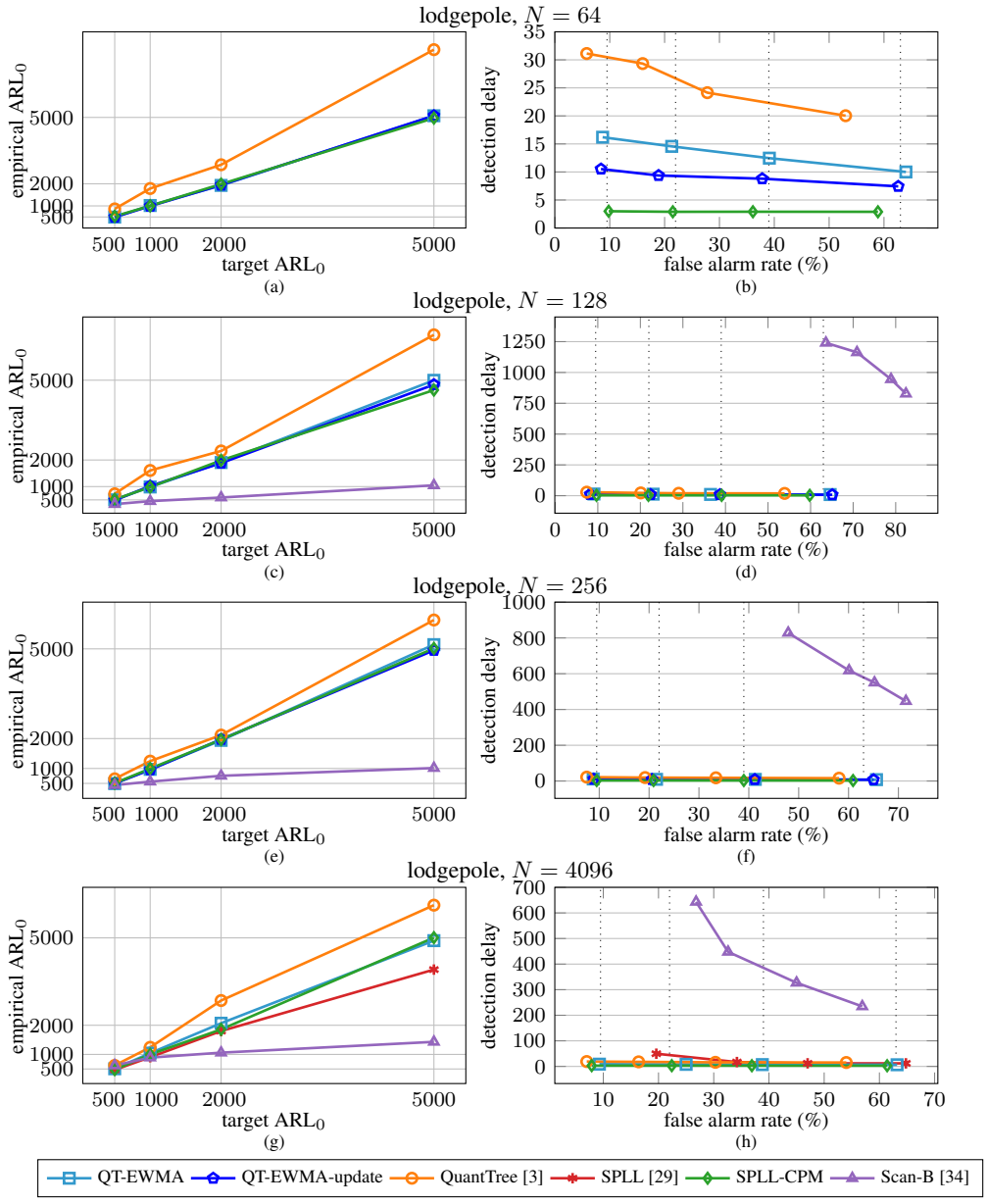


Figure A.7: *Experimental results over the lodgepole dataset ($d = 10$) [72].* (a,c,e,g) *show that the empirical ARL$_0$ of QT-EWMA, QT-EWMA-update and SPLL-CPM approaches the target, while Scan-B and SPLL cannot maintain the target ARL$_0$.* (b,d,f,h) *show that, in terms of detection delay, the best method is SPLL-CPM, although all the methods except Scan-B perform similarly. We observe that only QT-EWMA, QT-EWMA-update and SPLL-CPM achieve the target false alarm rates given by* (A.1)*, which are represented in the plots by vertical dotted lines.*

Bibliography

[1] Luca Frittoli, Diego Carrera, and Giacomo Boracchi. Change detection in multivariate datastreams controlling false alarms. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML–PKDD)*, pages 421–436. Springer, 2021.

[2] Luca Frittoli, Diego Carrera, and Giacomo Boracchi. Nonparametric and online change detection in multivariate datastreams using QuantTree. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–14, 2022.

[3] Giacomo Boracchi, Diego Carrera, Cristiano Cervellera, and Danilo Macciò. QuantTree: histograms for change detection in multivariate data streams. In *International Conference on Machine Learning*, pages 639–648. PMLR, 2018.

[4] Diego Stucchi, Luca Frittoli, and Giacomo Boracchi. Class distribution monitoring for concept drift detection. In *IEEE-INNS International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022.

[5] Luca Frittoli, Matteo Bocchi, Silvia Mella, Diego Carrera, Beatrice Rossi, Pasqualina Fragneto, Ruggero Susella, and Giacomo Boracchi. Strengthening sequential side-channel attacks through change detection. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 3:1–21, 2020.

[6] Alberto Floris, Luca Frittoli, Diego Carrera, and Giacomo Boracchi. Composite layers for deep anomaly detection on 3D point clouds. *arXiv preprint arXiv:2209.11796*, under review for IEEE Transactions on Image Processing, 2022.

[7] Izhak Golan and Ran El-Yaniv. Deep anomaly detection using geometric transformations. In *Advances in Neural Information Processing Systems*, pages 9781–9791, 2018.

[8] Luca Frittoli, Diego Carrera, Beatrice Rossi, Pasqualina Fragneto, and Giacomo Boracchi. Deep open-set recognition for silicon wafer production monitoring. *Pattern Recognition*, 124:108488, 2022.

[9] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2018.

[10] Andrea Bionda, Luca Frittoli, and Giacomo Boracchi. Deep autoencoders for anomaly detection in textured images using CW-SSIM. In *International Conference on Image Analysis and Processing (ICIAP)*, pages 669–680. Springer, 2022.

[11] Michèle Basseville, Igor V Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

[12] Douglas M Hawkins, Peihua Qiu, and Chang Wook Kang. The changepoint model for statistical process control. *Journal of Quality Technology*, 35(4):355–366, 2003.

[13] Walter Andrew Shewhart. *Economic control of quality of manufactured product*. MacMillan And Co Ltd, London, 1931.

[14] Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[15] SW Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 1(3):239–250, 1959.

[16] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. A just-in-time adaptive classification system based on the intersection of confidence intervals rule. *Neural Networks*, 24(8):791–800, 2011.

[17] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[18] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, pages 50–60, 1947.

[19] Alexander M Mood et al. On the asymptotic efficiency of certain nonparametric two-sample tests. *The Annals of Mathematical Statistics*, 25(3):514–522, 1954.

[20] Yves Lepage. A combination of Wilcoxon's and Ansari-Bradley's statistics. *Biometrika*, 58(1):213–217, 1971.

[21] Frank J Massey Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[22] Shen-Shyang Ho. A Martingale framework for concept change detection in time-varying data streams. In *International Conference on Machine Learning*, pages 321–327, 2005.

[23] Niloofar Mozafari, Sattar Hashemi, and Ali Hamzeh. A precise statistical approach for concept change detection in unlabeled data streams. *Computers & Mathematics with Applications*, 62(4):1655–1669, 2011.

[24] Douglas M Hawkins and KD Zamba. A change-point model for a shift in variance. *Journal of Quality Technology*, 37(1):21–31, 2005.

[25] Gordon J Ross, Dimitris K Tasoulis, and Niall M Adams. Nonparametric monitoring of data streams for changes in location and scale. *Technometrics*, 53(4):379–389, 2011.

[26] Gordon J Ross and Niall M Adams. Two nonparametric control charts for detecting arbitrary distribution changes. *Journal of Quality Technology*, 44(2):102–116, 2012.

[27] Harold Hotelling. A generalized t test and measure of multivariate dispersion. In *Berkeley Symposium on Mathematical Statistics and Probability*, pages 23–41. University of California, 1951.

[28] Alexander G Tartakovsky, Boris L Rozovskii, Rudolf B Blazek, and Hongjoong Kim. A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods. *IEEE Transactions on Signal Processing*, 54(9):3372–3382, 2006.

[29] Ludmila I Kuncheva. Change detection in streaming multivariate data using likelihood detectors. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1175–1180, 2011.

[30] Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Symposium on the Interface of Statistics, Computing Science, and Applications*. Citeseer, 2006.

[31] Giacomo Boracchi, Cristiano Cervellera, and Danilo Macciò. Uniform histograms for change detection in multivariate data. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1732–1739. IEEE, 2017.

[32] Erich L Lehmann and Joseph P Romano. *Testing statistical hypotheses*. Springer, 2006.

[33] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[34] Shuang Li, Yao Xie, Hanjun Dai, and Le Song. M-statistic for kernel change-point detection. *Advances in Neural Information Processing Systems*, 28:3366–3374, 2015.

[35] Xiuyuan Cheng and Yao Xie. Neural tangent kernel maximum mean discrepancy. *Advances in Neural Information Processing Systems*, 34:6658–6670, 2021.

[36] Cesare Alippi, Giacomo Boracchi, Diego Carrera, and Manuel Roveri. Change detection in multivariate datastreams: Likelihood and detectability loss. *International Joint Conference on Artificial Intelligence*, 2:1368–1374, 2016.

[37] Ludmila I Kuncheva and William J Faithfull. PCA feature extraction for change detection in multidimensional unlabeled data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):69–80, 2013.

[38] Abdulhakim A Qahtan, Basma Alharbi, Suojin Wang, and Xiangliang Zhang. A PCA-based change detection framework for multidimensional data streams. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944, 2015.

[39] Yao Xie and David Siegmund. Sequential multi-sensor change-point detection. In *2013 Information Theory and Applications Workshop*, pages 1–20. IEEE, 2013.

[40] Georgios Fellouris and Alexander G Tartakovsky. Multichannel sequential detection-part I: Non-iid data. *IEEE Transactions on Information Theory*, 63(7):4551–4571, 2017.

[41] Zhongchang Sun, Shaofeng Zou, Ruizhi Zhang, and Qunwei Li. Quickest change detection in anonymous heterogeneous sensor networks. *IEEE Transactions on Signal Processing*, 70:1041–1055, 2022.

[42] Diego Carrera and Giacomo Boracchi. Generating high-dimensional datastreams for change detection. *Big Data Research*, 11:11–21, 2018.

[43] Harold Hotelling. Multivariate quality control. *Techniques of Statistical Analysis*, 1947.

[44] Ronald B Crosier. Multivariate generalizations of cumulative sum quality-control schemes. *Technometrics*, 30(3):291–303, 1988.

[45] Liyan Xie, Yao Xie, and George V Moustakides. Sequential subspace change point detection. *Sequential Analysis*, 39(3):307–335, 2020.

[46] Cynthia A Lowry, William H Woodall, Charles W Champ, and Steven E Rigdon. A multivariate exponentially weighted moving average control chart. *Technometrics*, 34(1):46–53, 1992.

[47] KD Zamba and Douglas M Hawkins. A multivariate change-point model for statistical process control. *Technometrics*, 48(4):539–549, 2006.

[48] Alexandre Lung-Yut-Fong, Céline Lévy-Leduc, and Olivier Cappé. Robust changepoint detection based on multivariate rank statistics. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3608–3611. IEEE, 2011.

[49] David S Matteson and Nicholas A James. A nonparametric approach for multiple change point analysis of multivariate data. *Journal of the American Statistical Association*, 109(505):334–345, 2014.

[50] Tze Siong Lau, Wee Peng Tay, and Venugopal V Veeravalli. A binning approach to quickest change detection with unknown post-change distribution. *IEEE Transactions on Signal Processing*, 67(3):609–621, 2018.

[51] Nicolas Keriven, Damien Garreau, and Iacopo Poli. NEWMA: a new method for scalable model-free online change-point detection. *IEEE Transactions on Signal Processing*, 68:3515–3528, 2020.

[52] Thomas M Margavio, Michael D Conerly, William H Woodall, and Laurel G Drake. Alarm rates for quality control charts. *Statistics & Probability Letters*, 24(3):219–224, 1995.

[53] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[54] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.

[55] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

[56] Gordon J Ross, Niall M Adams, Dimitris K Tasoulis, and David J Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2):191–198, 2012.

[57] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian Symposium on Artificial Intelligence*, pages 286–295. Springer, 2004.

[58] João Gama and Gladys Castillo. Learning with local drift detection. In *International Conference on Advanced Data Mining and Applications*, pages 42–55. Springer, 2006.

[59] Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and Rafael Morales-Bueno. Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams*, volume 6, pages 77–86, 2006.

[60] Isvani Frias-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yailé Caballero-Mota. Online and non-parametric drift detection methods based on Hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, 2014.

[61] Roberto Souto Maior de Barros, Juan Isidro González Hidalgo, and Danilo Rafael de Lima Cabral. Wilcoxon rank sum test drift detector. *Neurocomputing*, 275:1954–1963, 2018.

[62] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *2007 SIAM International Conference on Data Mining*, pages 443–448. SIAM, 2007.

[63] Albert Bifet and Ricard Gavalda. Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*, pages 249–260. Springer, 2009.

[64] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavalda. New ensemble methods for evolving data streams. In *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 139–148, 2009.

[65] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495, 2017.

[66] Shuo Wang and Leandro L Minku. AUC estimation and concept drift detection for imbalanced data streams with multiple classes. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[67] Łukasz Korycki and Bartosz Krawczyk. Concept drift detection from multi-class imbalanced data streams. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1068–1079. IEEE, 2021.

[68] Bela A Frigyik, Amol Kapila, and Maya R Gupta. Introduction to the Dirichlet distribution and related processes. *Technical Report UWEETR-2010-0006*, 2010.

[69] Samuel Kotz, Narayanaswamy Balakrishnan, and Norman L Johnson. *Continuous multivariate distributions, Volume 1: Models and applications*. John Wiley & Sons, 2004.

[70] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[71] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection: a realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8):3784–3797, 2017.

[72] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[73] Vinicius Souza, Denis M dos Reis, Andre G Maletzke, and Gustavo EAPA Batista. Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery*, 34(6):1805–1858, 2020.

[74] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[75] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1(6):80–83, 1945.

[76] Peter Bjorn Nemenyi. *Distribution-free multiple comparisons*. PhD Thesis, Princeton University, 1963.

[77] Olive Jean Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.

[78] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[79] Paul C Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[80] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: concrete results. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 251–261. Springer, 2001.

[81] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and vertical side-channel attacks against secure RSA implementations. In *Cryptographers' Track at the RSA Conference*, pages 1–17. Springer, 2013.

[82] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal correlation analysis on exponentiation. In *International Conference on Information and Communications Security*, pages 46–61. Springer, 2010.

[83] Richard W Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[84] Jean-Jacques Quisquater, François Koeune, and Werner Schindler. Unleashing the full power of timing attack. In *Universite Catholique de Louvain–Crypto Group, 2001. 120, 130 BIBLIOGRAPHY 159*. Citeseer, 2001.

[85] Werner Schindler, François Koeune, and Jean-Jacques Quisquater. Improving divide and conquer attacks against cryptosystems by better error detection/correction strategies. In *IMA International Conference on Cryptography and Coding*, pages 245–267. Springer, 2001.

[86] Werner Schindler. On the optimization of side-channel attacks by advanced stochastic methods. In *International Workshop on Public Key Cryptography*, pages 85–103. Springer, 2005.

[87] Jean-François Dhem, François Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In *International Conference on Smart Card Research and Advanced Applications*, pages 167–182. Springer, 1998.

[88] CaiSen Chen, Tao Wang, and Junjian Tian. Improving timing attack on RSA-CRT via error detection and correction strategy. *Information Sciences*, 232:464–474, 2013.

[89] Chao Luo, Yunsi Fei, and David Kaeli. GPU acceleration of RSA is vulnerable to side-channel timing attacks. In *International Conference on Computer-Aided Design*, pages 113–120. ACM, 2018.

[90] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. A hierarchical, nonparametric, sequential change-detection test. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 2889–2896. IEEE, 2011.

[91] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[92] Colin D Walter. Sliding windows succumbs to big mac attack. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 286–299. Springer, 2001.

[93] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 16–29. Springer, 2004.

[94] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[95] Lukas Ruff, Jacob R Kauffmann, Robert A Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G. Dietterich, and Klaus-Robert Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109(5):756–795, 2021.

[96] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[97] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[98] Stephen Roberts and Lionel Tarassenko. A probabilistic resource allocating network for novelty detection. *Neural Computation*, 6(2):270–284, 1994.

[99] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[100] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[101] David MJ Tax and Robert PW Duin. Support vector data description. *Machine Learning*, 54(1):45–66, 2004.

[102] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3):237–253, 2000.

[103] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. LOF: identifying density-based local outliers. In *2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104, 2000.

[104] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[105] Gordon Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, 14(1):55–63, 1968.

[106] Diego Carrera, Beatrice Rossi, Pasqualina Fragneto, and Giacomo Boracchi. Online anomaly detection for long-term ECG monitoring using wearable devices. *Pattern Recognition*, 88:482–492, 2019.

[107] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–674, 2017.

[108] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Robust, deep and inductive anomaly detection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 36–51. Springer, 2017.

[109] Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton van den Hengel. Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection. In *IEEE/CVF International Conference on Computer Vision*, pages 1705–1714, 2019.

[110] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018.

[111] Sarah M Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. *Pattern Recognition*, 58:121–134, 2016.

[112] Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *International Conference on Machine Learning*, volume 80, pages 4393–4402. PMLR, 2018.

[113] Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. Deep semi-supervised anomaly detection. In *International Conference on Learning Representations*, 2020.

[114] Sachin Goyal, Aditi Raghunathan, Moksh Jain, Harsha Vardhan Simhadri, and Prateek Jain. DROCC: Deep robust one-class classification. In *International Conference on Machine Learning*, pages 3711–3721. PMLR, 2020.

[115] Penny Chong, Lukas Ruff, Marius Kloft, and Alexander Binder. Simple and effective prevention of mode collapse in deep one-class classification. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2020.

[116] Duc Tam Nguyen, Zhongyu Lou, Michael Klar, and Thomas Brox. Anomaly detection with multiple-hypotheses predictions. In *International Conference on Machine Learning*, pages 4800–4809. PMLR, 2019.

[117] Samet Akcay, Amir Atapour-Abarghouei, and Toby P Breckon. GANomaly: Semi-supervised anomaly detection via adversarial training. In *Asian Conference on Computer Vision*, pages 622–637. Springer, 2018.

[118] Lucas Deecke, Robert Vandermeulen, Lukas Ruff, Stephan Mandt, and Marius Kloft. Image anomaly detection with generative adversarial networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML–PKDD)*, pages 3–17. Springer, 2018.

[119] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 27, 2014.

[120] Siqi Wang, Yijie Zeng, Xinwang Liu, En Zhu, Jianping Yin, Chuanfu Xu, and Marius Kloft. Effective end-to-end unsupervised outlier detection via inlier priority of discriminative network. *Advances in Neural Information Processing Systems*, 32:5960–5973, 2019.

[121] Chen Qiu, Timo Pfrommer, Marius Kloft, Stephan Mandt, and Maja Rudolph. Neural transformation learning for deep anomaly detection beyond images. In *International Conference on Machine Learning*, pages 8703–8714. PMLR, 2021.

[122] Alex Tamkin, Mike Wu, and Noah Goodman. Viewmaker networks: Learning views for unsupervised representation learning. In *International Conference on Learning Representations*, 2021.

[123] Paul Bergmann, Sindy Löwe, Michael Fauser, David Sattlegger, and Steger Carsten. Improving unsupervised defect segmentation by applying structural similarity to autoencoders. In *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP)*, pages 372–380. SciTePress, 2019.

[124] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Uninformed students: Student-teacher anomaly detection with discriminative latent embeddings. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4183–4192, 2020.

[125] Vitjan Zavrtanik, Matej Kristan, and Danijel Skočaj. Reconstruction by inpainting for visual anomaly detection. *Pattern Recognition*, 112:107706, 2021.

[126] Paul Bergmann., Xin Jin., David Sattlegger., and Carsten Steger. The MVTec 3D-AD dataset for unsupervised 3D anomaly detection and localization. In *17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP)*, pages 202–213. SciTePress, 2022.

[127] Walter J Scheirer, Lalit P Jain, and Terrance E Boult. Probability models for open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2317–2324, 2014.

[128] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(10):3614–3631, 2020.

[129] Abhijit Bendale and Terrance Boult. Towards open world recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1893–1902, 2015.

[130] He Zhang and Vishal M Patel. Sparse representation-based open set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(8):1690–1696, 2016.

[131] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*, 2017.

[132] Chuanxing Geng, Lue Tao, and Songcan Chen. Guided CNN for generalized zero-shot and open-set recognition using visual and semantic prototypes. *Pattern Recognition*, 102:107263, 2020.

[133] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1563–1572, 2016.

[134] Hakan Cevikalp, Bedirhan Uzun, Okan Köpüklü, and Gurkan Ozturk. Deep compact polyhedral conic classifier for open and closed set recognition. *Pattern Recognition*, 119:108080, 2021.

[135] Yue Zhu, Kai Ming Ting, and Zhi-Hua Zhou. Multi-label learning with emerging new labels. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1901–1914, 2018.

[136] Yu Zhang, Yin Wang, Xu-Ying Liu, Siya Mi, and Min-Ling Zhang. Large-scale multi-label classification using unknown streaming images. *Pattern Recognition*, 99:107100, 2020.

[137] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *Advances in Neural Information Processing Systems*, pages 935–943, 2013.

[138] Ryota Yoshihashi, Wen Shao, Rei Kawakami, Shaodi You, Makoto Iida, and Takeshi Naemura. Classification-reconstruction learning for open-set recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4016–4025, 2019.

[139] Poojan Oza and Vishal M Patel. C2AE: Class conditioned auto-encoder for open-set recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2307–2316, 2019.

[140] Xin Sun, Zhenning Yang, Chi Zhang, Keck-Voon Ling, and Guohao Peng. Conditional gaussian distribution learning for open set recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13480–13489, 2020.

[141] Zongyuan Ge, Sergey Demyanov, Zetao Chen, and Rahil Garnavi. Generative openmax for multi-class open set classification. In *British Machine Vision Conference*, 2017.

[142] Lawrence Neal, Matthew Olson, Xiaoli Fern, Weng-Keen Wong, and Fuxin Li. Open set learning with counterfactual images. In *European Conference on Computer Vision*, pages 613–628, 2018.

[143] Inhyuk Jo, Jungtaek Kim, Hyohyeong Kang, Yong-Deok Kim, and Seungjin Choi. Open set recognition by regularising classifier with fake data generated by generative adversarial networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2686–2690. IEEE, 2018.

[144] Sagar Vaze, Kai Han, Andrea Vedaldi, and Andrew Zisserman. Open-set recognition: A good closed-set classifier is all you need? In *International Conference on Learning Representations*, 2022.

[145] Alexandre Boulch. ConvPoint: Continuous convolutions for point cloud processing. *Computers & Graphics*, 88:24–34, 2020.

[146] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. KPConv: Flexible and deformable convolution for point clouds. In *IEEE International Conference on Computer Vision*, pages 6411–6420, 2019.

[147] Yaodong Cui, Ren Chen, Wenbo Chu, Long Chen, Daxin Tian, Ying Li, and Dongpu Cao. Deep learning for image and point cloud fusion in autonomous driving: A review. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[148] Zurui Ao, Yanjun Su, Wenkai Li, Qinghua Guo, and Jing Zhang. One-class classification of airborne LiDAR data in urban areas using a presence and background learning algorithm. *Remote Sensing*, 9(10):1001, 2017.

[149] Simone Teruggi, Eleonora Grilli, Michele Russo, Francesco Fassi, and Fabio Remondino. A hierarchical machine learning approach for multi-level and multi-resolution 3d point cloud classification. *Remote Sensing*, 12(16):2598, 2020.

[150] Charles R. Qi, Hao Su, Mo Kaichun, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 77–85, 2017.

[151] Charles R. Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas Guibas. Volumetric and multi-view CNNs for object classification on 3D data. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[152] Liuhao Ge, Hui Liang, Junsong Yuan, and Daniel Thalmann. Robust 3d hand pose estimation from single depth images using multi-view CNNs. *IEEE Transactions on Image Processing*, 27(9):4422–4436, 2018.

[153] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[154] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4338–4364, 2021.

[155] Charles R. Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, 2017.

[156] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Transactions on Graphics*, 37(4), 07 2018.

[157] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2589–2597, 2018.

[158] Wenxuan Wu, Zhongang Qi, and Li Fuxin. PointConv: Deep convolutional networks on 3d point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[159] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. SpiderCNN: Deep learning on point sets with parameterized convolutional filters. In *European Conference on Computer Vision*, pages 87–102. Springer, 2018.

[160] Fei Yang, Huan Wang, and Zhong Jin. Adaptive GMM convolution for point cloud learning. In *British Machine Vision Conference*, 2021.

[161] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated convolutional networks for 3d point cloud understanding. In *IEEE/CVF International Conference on Computer Vision*, pages 1578–1587, 2019.

[162] Yiqun Lin, Zizheng Yan, Haibin Huang, Dong Du, Ligang Liu, Shuguang Cui, and Xiaoguang Han. FPConv: Learning local flattening for point convolution. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4293–4302, 2020.

[163] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019.

[164] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. RandLA-Net: Efficient semantic segmentation of large-scale point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117, 2020.

[165] Haoxi Ran, Wei Zhuo, Jun Liu, and Li Lu. Learning inner-group relations on point clouds. In *IEEE/CVF International Conference on Computer Vision*, pages 15477–15487, 2021.

[166] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. DensePoint: Learning densely contextual representation for efficient point cloud processing. In *IEEE/CVF International Conference on Computer Vision*, pages 5239–5248, 2019.

[167] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. PAConv: Position adaptive convolution with dynamic kernel assembling on point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3173–3182, 2021.

[168] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. SplineCNN: Fast geometric deep learning with continuous b-spline kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[169] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics*, 38(5):1–12, 2019.

[170] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. DeepGCNs: Can GCNs go as deep as CNNs? In *IEEE/CVF International Conference on Computer Vision*, pages 9267–9276, 2019.

[171] Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical kernel for efficient graph convolution on 3d point clouds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(10):3664–3680, 2020.

[172] Mingtao Feng, Syed Zulqarnain Gilani, Yaonan Wang, Liang Zhang, and Ajmal Mian. Relation graph network for 3d object detection in point clouds. *IEEE Transactions on Image Processing*, 30:92–107, 2020.

[173] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual MLP framework. In *International Conference on Learning Representations*, 2022.

[174] Jiaqi Lyu and Souran Manoochehri. Online convolutional neural network-based anomaly detection and quality control for fused filament fabrication process. *Virtual and Physical Prototyping*, 16(2):160–177, 2021.

[175] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[176] Borja Rodríguez-Cuenca, Silverio García-Cortés, Celestino Ordóñez, and Maria C Alonso. Automatic detection and classification of pole-like objects in urban point cloud data using an anomaly detection algorithm. *Remote Sensing*, 7(10):12680–12703, 2015.

[177] Xian-Feng Hana, Jesse S Jin, Juan Xie, Ming-Jie Wang, and Wei Jiang. A comprehensive review of 3d point cloud descriptors. *arXiv preprint arXiv:1802.02297*, 2, 2018.

[178] Saining Xie, Jiatao Gu, Demi Guo, Charles R Qi, Leonidas Guibas, and Or Litany. PointContrast: Unsupervised pre-training for 3d point cloud understanding. In *European Conference on Computer Vision*, pages 574–591. Springer, 2020.

[179] Zaiwei Zhang, Rohit Girdhar, Armand Joulin, and Ishan Misra. Self-supervised pretraining of 3d features on any point-cloud. In *IEEE/CVF International Conference on Computer Vision*, pages 10252–10263, 2021.

[180] Longkun Zou, Hui Tang, Ke Chen, and Kui Jia. Geometry-aware self-training for unsupervised domain adaptation on object point clouds. In *IEEE/CVF International Conference on Computer Vision*, pages 6403–6412, 2021.

[181] Li Yi, Boqing Gong, and Thomas Funkhouser. Complete & label: A domain adaptation approach to semantic segmentation of lidar point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15363–15373, 2021.

[182] Feiyu Wang, Wen Li, and Dong Xu. Cross-dataset point cloud recognition using deep-shallow domain adaptation network. *IEEE Transactions on Image Processing*, 30:7364–7377, 2021.

[183] Mana Masuda, Ryo Hachiuma, Ryo Fujii, Hideo Saito, and Yusuke Sekikawa. Toward unsupervised 3d point cloud anomaly detection using variational autoencoder. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3118–3122. IEEE, 2021.

[184] Maciej Zamorski, Maciej Zieba, Piotr Klukowski, Rafał Nowak, Karol Kurach, Wojciech Stokowiec, and Tomasz Trzciński. Adversarial autoencoders for compact representations of 3d point clouds. *Computer Vision and Image Understanding*, 193:102921, 2020.

[185] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3d point capsule networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1009–1018, 2019.

[186] Siheng Chen, Chaojing Duan, Yaoqing Yang, Duanshun Li, Chen Feng, and Dong Tian. Deep unsupervised learning of 3d point clouds via graph topology inference and filtering. *IEEE Transactions on Image Processing*, 29:3183–3198, 2019.

[187] Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Monte Carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics*, 37(6):1–12, 2018.

[188] Jooyoung Park and Irwin W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, 1991.

[189] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3d reconstructions of indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5828–5839, 2017.

[190] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015.

[191] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. *Technical Report arXiv:1512.03012 [cs.GR]*, 2015.

[192] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[193] S Hamidreza Kasaei, Ana Maria Tomé, Luís Seabra Lopes, and Miguel Oliveira. GOOD: A global orthographic object descriptor for 3D object recognition and manipulation. *Pattern Recognition Letters*, 83:312–320, 2016.

[194] Roberto di Bella, Diego Carrera, Beatrice Rossi, Pasqualina Fragneto, and Giacomo Boracchi. Wafer defect map classification using sparse convolutional networks. In *International Conference on Image Analysis and Processing (ICIAP)*, pages 125–136. Springer, 2019.

[195] Jianbo Yu and Xiaolei Lu. Wafer map defect detection and recognition using joint local and nonlocal linear discriminant analysis. *IEEE Transactions on Semiconductor Manufacturing*, 29(1):33–43, 2016.

[196] Mengying Fan, Qin Wang, and Ben van der Waal. Wafer defect patterns recognition based on optics and multi-label classification. In *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 912–915, 2016.

[197] Cheng-Wei Chang, Tsung-Ming Chao, Jorng-Tzong Horng, Chien-Feng Lu, and Rong-Hwei Yeh. Development pattern recognition model for the classification of circuit probe wafer maps on semiconductors. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 2(12):2089–2097, 2012.

[198] Kouta Nakata, Ryohei Orihara, Yoshiaki Mizuoka, and Kentaro Takagi. A comprehensive big-data-based monitoring system for yield enhancement in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 30(4):339–344, 2017.

[199] Takeshi Nakazawa and Deepak V Kulkarni. Wafer map defect pattern classification and image retrieval using convolutional neural network. *IEEE Transactions on Semiconductor Manufacturing*, 31(2):309–314, 2018.

[200] Naigong Yu, Qiao Xu, and Honglu Wang. Wafer defect pattern recognition and analysis based on convolutional neural network. *IEEE Transactions on Semiconductor Manufacturing*, 32(4):566–573, 2019.

[201] Muhammad Saqlain, Qasim Abbas, and Jong Yun Lee. A deep convolutional neural network for wafer defect identification on an imbalanced dataset in semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing*, 33(3):436–444, 2020.

[202] Wei-Chen Li and Du-Ming Tsai. Wavelet-based defect detection in solar wafer images with inhomogeneous texture. *Pattern Recognition*, 45(2):742–756, 2012.

[203] Sejune Cheon, Hankang Lee, Chang Ouk Kim, and Seok Hyung Lee. Convolutional neural network for wafer surface defect classification and the detection of unknown defect class. *IEEE Transactions on Semiconductor Manufacturing*, 32(2):163–170, 2019.

[204] Szu-Hao Huang and Ying-Cheng Pan. Automated visual inspection in the semiconductor industry: A survey. *Computers in Industry*, 66:1–10, 2015.

[205] Ming-Ju Wu, Jyh-Shing R Jang, and Jui-Long Chen. Wafer map failure pattern recognition and similarity ranking for large-scale data sets. *IEEE Transactions on Semiconductor Manufacturing*, 28(1):1–12, 2015.

[206] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[207] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[208] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[209] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[210] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[211] Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[212] David J Hand and Robert J Till. A simple generalisation of the area under the ROC curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, 2001.

[213] Elizabeth R DeLong, David M DeLong, and Daniel L Clarke-Pearson. Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, pages 837–845, 1988.

[214] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.